White Paper on

w2bill™ REALTIME

CONTEXT
ARCHITECTURE
CAPABILITIES
ROADMAP

INDEX

03 **INTRODUCTION**

04 **CONTEXT**

05 **ARCHITECTURE**
06 PROCESSING ENGINE
07 DATA ACCESSIBILITY
PERSISTENCY
LOGGING/METRICS PLATFORM
METRICS
LOGGING
DISCOVERY SERVICE & CONFIGURATION SERVER
MONITORING
08 FRONTENDS
AUTHENTICATION SERVICE
TECHNOLOGIES

09 **CAPABILITIES**

10 **ROADMAP**

11 **APPENDIX**
PROCESSING ENGINE
JAVA 8
PIVOTAL SPRING FRAMEWORK
PIVOTAL RABBITMQ
12 RESOURCE MANAGEMENT
APACHE MESOS
DATA ACCESSIBILITY
APACHE IGNITE
PERSISTENCY
APACHE CASSANDRA

INTRODUCTION

In the past few years, the needs and wishes of customers have grown extensively, fuelled by technological accomplishments and the evermore present desire to be able to do anything, anywhere, anytime. This has in turn driven enterprises to come up with ways to satisfy these needs, by continuously engaging their customers with innovative products and offers, whilst being capable of quickly adapting without having to constantly change their infrastructures or platforms to meet the demand. In order to accomplish every tangible market expectation, companies must be able to change, transform and scale their supporting activities and infrastructures, all for the sake of the customer experience win-win place at the end of track. Every aspect or dimension supporting the business model, especially the IT infrastructure needs to be flexible, scalable, agile, efficient and have as little implementation time as possible.

Companies seek adaptability, modularity and the ability to account for future changes. This proven fact has been creating the need for integration solutions, which can automate and scale performance in their core business support activities, diminishing the huge and lengthily headache of tying all the loose ends together with every innovation and shifting from the core legacy systems to automation and scalability with little time to do it.

CONTEXT

Market focus is aimed at developing new services and experiences, relying on faster and deeper automation, predictive analytics and security and ever-deepening levels of integrations and partnerships between multiple players. In every market, across every industry, both the producers/merchants and consumer expectations are in constant transformation. Consumers increasingly demand higher and quicker delivery, on-demand interactions and results, while seemingly non-coherently expecting lower and more flexible costs and service fees. From consumers to large corporations, everyone seems to agree that Immediacy is quickly turning into a basic capability rather than a differentiation. While the market demands it, market regulators struggle to keep up with it, and only the agilest and efficient companies thrive.

Technology constantly outpaces itself year after year, with new advances being presented incessantly, and changing the way people and enterprises do things. It continuously grows in its presence everywhere, effectively breeding a dependency like never before. People don't go anywhere without their smartphones, they operate simultaneously on their tablets and laptops, while at the same time enterprises rely on complex infrastructures to operate and effectively manage their core business. Little can be achieved without technology, and any company without a strong digital presence is doomed to oblivion. With so much technology in place, so many different users and such diversity in services and offers, the amount of data and complexity of its management, from a provider perspective, is ever more obvious. The evolution has been marked by moving from monolithic applications, into modular ones; from there, onto discrete domain-specific systems interconnected by complex integration architectures. The model to do businesses nowadays have to be flexible, adaptive, innovative and modern, especially on the pressure put into operating costs. From the startup company to the large enterprise, the key to prosperity seems to be able to accommodate the markets 'mood-swings', endure the constant shifts and to compete being as light weighted as possible.

ARCHITECTURE

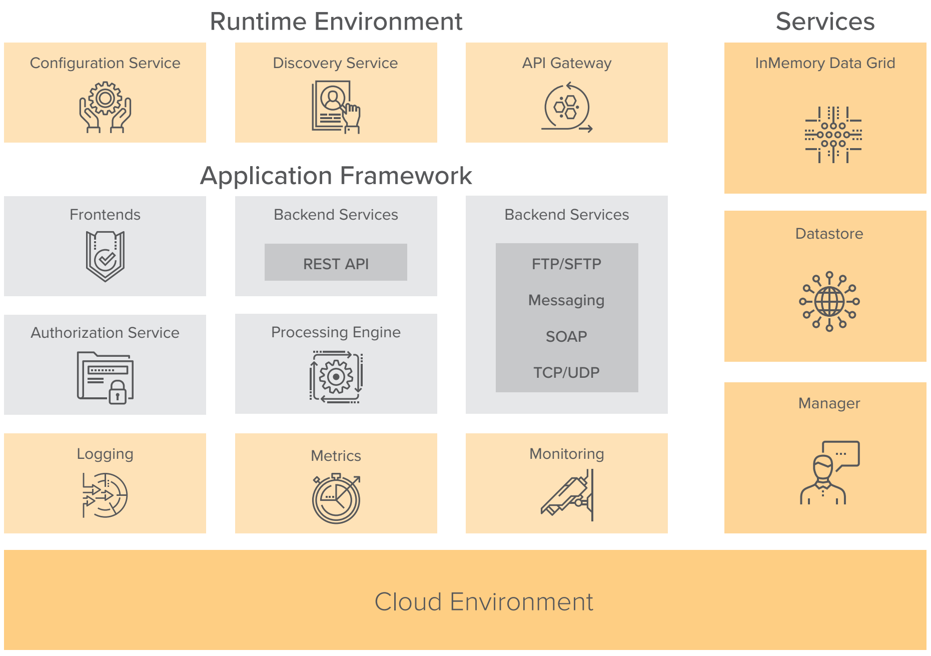

Figure 1. **w2bill**™ Architecture Overview

w2bill™ Real-time Task Management Framework was designed, from its earlier stages, with the future in mind. Following the old path made little sense in a world marked by constant evolution, and with the growing certainty that today's solutions will most likely be outdated soon. With such concerns in focus, different ways of doing things were considered, where scalability, fault-tolerance, reliability, resilience and configurability where key. What is true today will soon be outdated and outpaced. The current accepted amount of data will soon be surpassed. More processing power, in different geographies, through public clouds or private data centers, must be supported, with all the challenges therein considered and handled.

The three core concepts behind **w2bill**™:

1. PROCESSING ENGINE

The Processing Engine is the heart of the **w2bill**™ solution. It is a package of micro-services targeted at executing flows of tasks across a variety of platforms. Each flow can be fully configured, allowing sequential or parallel tasks to be executed. It features three main classes of components: Receptor, Orchestrator and Executor.

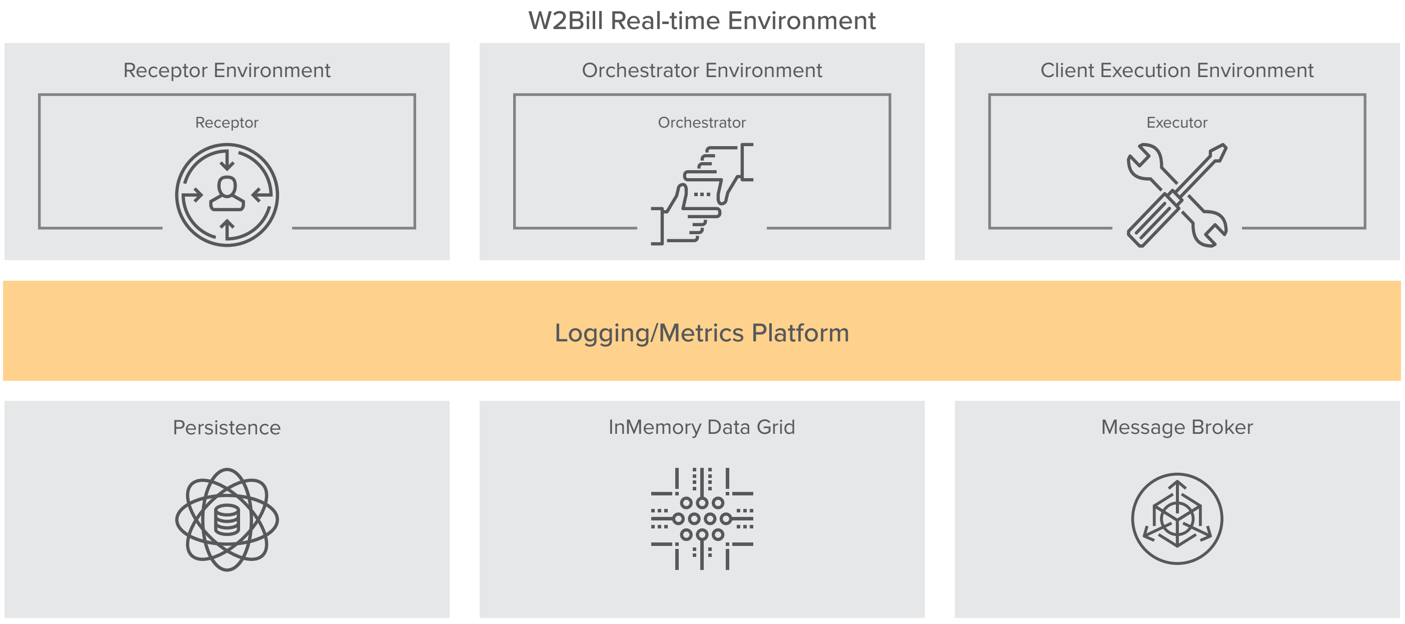

Figure 2. **w2bill**™ Processing Engine Overview

1.1 RECEPTOR

The Receptor component is the main entry point of any request, enabling its initial processing, which includes technical validations as well as configurable business rules to be applied to ascertain the coherence of information, as well as derive which flow is to be executed based on the input data. Should all steps be valid, an order is placed in the queuing system and a reply is returned to the calling platform. On the other hand, should it fail, the calling platform is informed of the failure for its correction and eventual resubmission. A Receptor can provide several services (endpoints) to be invoked, along with multiple versions of each endpoint, which are dynamically generated and exposed based on user configuration. It further supports multiple instances running over various machines, each with its own set of services for finer granularity and control.

1.2 ORCHESTRATOR

For successfully accepted orders, the Orchestrator component oversees the control of the flow of tasks to be executed, from start to end – successful or in error. It tracks the evolution of each task by scheduling it's work within the available Executor components, based on their capabilities and the required work to be done. The approach selected was of a stateless, decentralized service, being able to split the management of each flow among all running instances. All interactions between components occur via messaging. The supporting broker technology ensures persistence of messages, high availability, confirmation upon delivery (at least once delivery), fault tolerance, message batching among many others. This approach allows the engine to scale vertically, by making use of multiple cores per machine, and scale horizontally, launching multiple instances in a machine cluster.

1.3 EXECUTOR

The Executor component is responsible for the actual running of the work tasks within each workflow. This component is designed for self-scaling within its available resources, as well as deployable in multiple instances for load management and balancing. Each component is dedicated to a work domain, such as RDBMS interactions, file system integrations, or other more specialized features dependent on customer requirements and constraints.

2. DATA ACCESSIBILITY

Given data is stored, it needs to be retrieved for processing, and then stored again when changed (either updated or removed altogether). Knowing beforehand that a given customer may select several approaches **w2bill**™ has implemented a Data Grid solution to handle all aspects of data manipulation. This layer enables all interacting Services to completely abstract on how and where the data is handled, they just access it in a performant and reliable way.

3. PERSISTENCY

It is immediately recognized that data must be kept safely, consistently and securely, to be used. It must also scale easily, and have as little constraints in it's management as possible. Because of this, there is no direct solution or clear approach on which technology is best. Standard RDBMS's are tried-end-true platforms, that provide a multitude of capabilities, most people take for granted, but have shortcomings in either cost, scalability or performance. More recent NoSQL solutions have less wide-spread adoption in large enterprises, but commonly are inherently scalable and performant, at a fraction of the cost. And then there are ways closer to actual physical storage, like HDFS, ZFS, and others. To handle persistency **w2bill**™ has employed a "choose any" approach. In other words, whichever is best suited for a customer, or a specific domain inside a customer, the solution will support it.

Given data is stored, it needs to be retrieved for processing, and then stored again when changed (either updated or removed altogether). Since the approach with persistency has been a "choose any", and knowing beforehand that a given customer may select several approaches, **w2bill**™ has implemented a Data Grid solution to handle all aspects of data manipulation. This layer enables all interacting Services to completely abstract on how and where the data is handled, they just access it in a performant and reliable way.

LOGGING/METRICS PLATFORM

METRICS

All components are implemented to produce execution metrics and store them in a centralized repository where the system performance can be measured and monitored. With these metrics, it is possible to know how long an order takes to be processed, as well as how long each component takes to execute the required operation. With these capabilities, it is possible to detect anomalies in components' instances by monitoring the execution times of each system, along with other relevant information necessary for scheduling of performance enhancing actions, bottleneck detection or scaling decisions. Metrics are enriched with contextual data. It is possible to know the host where the component is running, the number of available cores and memory, which real-time group it belongs, along with additional relevant information.

LOGGING

In distributed deployments, component instances will be spread across several machines of a cluster. Each instance will produce logs that must be stored and grouped with the logs of all system components to allow analysis of the system behavior. All components of **w2bill**™ Framework are prepared to write application logs to a file – in the local machine –, and to send a logging message with additional data, allowing users to correlate all logs for a specific order or workflow execution. This enables a clearer big picture of the system. These logs are centralized in an Elastic search index with a pre-defined schema. It is possible to search logs by an order id, a correlation id or by a log message. Logs are published asynchronously to the broker improving system performance.

DISCOVERY SERVICE & CONFIGURATION SERVER

Components instances can be started in any available host of the cluster, manually or automatically based on workload. To be able to manage and control all instances, a Service Registry and a Configuration Server was implemented. All components retrieve their configuration from the central configuration server, and must create a record – with their IP address and HTTP port – in the Service Registry. Each service instance is responsible for registering itself with the Service Registry during start-up and unregistering on shutdown.

MONITORING

Components' instances are continuously monitored using health check endpoints provided by each of them, and by metrics' analysis. With it, it is possible to act, in order to start and stop instances, maintaining the desired Service Level and the effective use of resources of the running environment.

FRONTENDS

w2bill™ presents a unified GUI that enables common operations to be performed over the Platform, such as diagnosis, tracing, log validation and status checks. The chosen approach allows each customer to have dedicated screens or views, representing relevant information to be displayed, instead of a static screen showing pre-defined data.

AUTHENTICATION SERVICE

To comply with the need of security and validated service access, **w2bill**™ embeds an authentication service enabling the services to be used only by properly authenticated users. With this feature, all requests made to the exposed services must be authenticated through a central authentication server – as provided by **w2bill**™ –, ensuring only the services each user has granted policies are accessed. This server implements the OAuth 2.0 protocol and can authenticate users in a relational database or an LDAP server.

TECHNOLOGIES

The entire solution is based on the technology stack presented below in more detail. In any case, given the architectural decisions made earlier on, it becomes fully possible to opt for specific solutions using different technologies, if certain key integrations aspects are met. As an example, it is possible to deploy one Executor component implemented in C/C++, or Microsoft C#, or even Python. The choice in the persistency stack can be replaced by other NoSQL solutions such as MongoDB, or even more traditional RDBMS's like MySQL, Postgres or Oracle. Across the entire solution, the choice of technology has been focused on its adequacy (for its given set of responsibilities), as well as its proven track-record, industry wide-spread adoption, reliable enterprise-grade support and evolution roadmap. With the growing adoption of cloud-based approaches instead of on premise, certain facets of architecture, design and underlying implementation were considered to assure the solution was prepared without the need of specific customisation. By addressing these concerns early, the selection of technologies was further refined, where only the ones with appropriate levels of compliance were left as eligible.

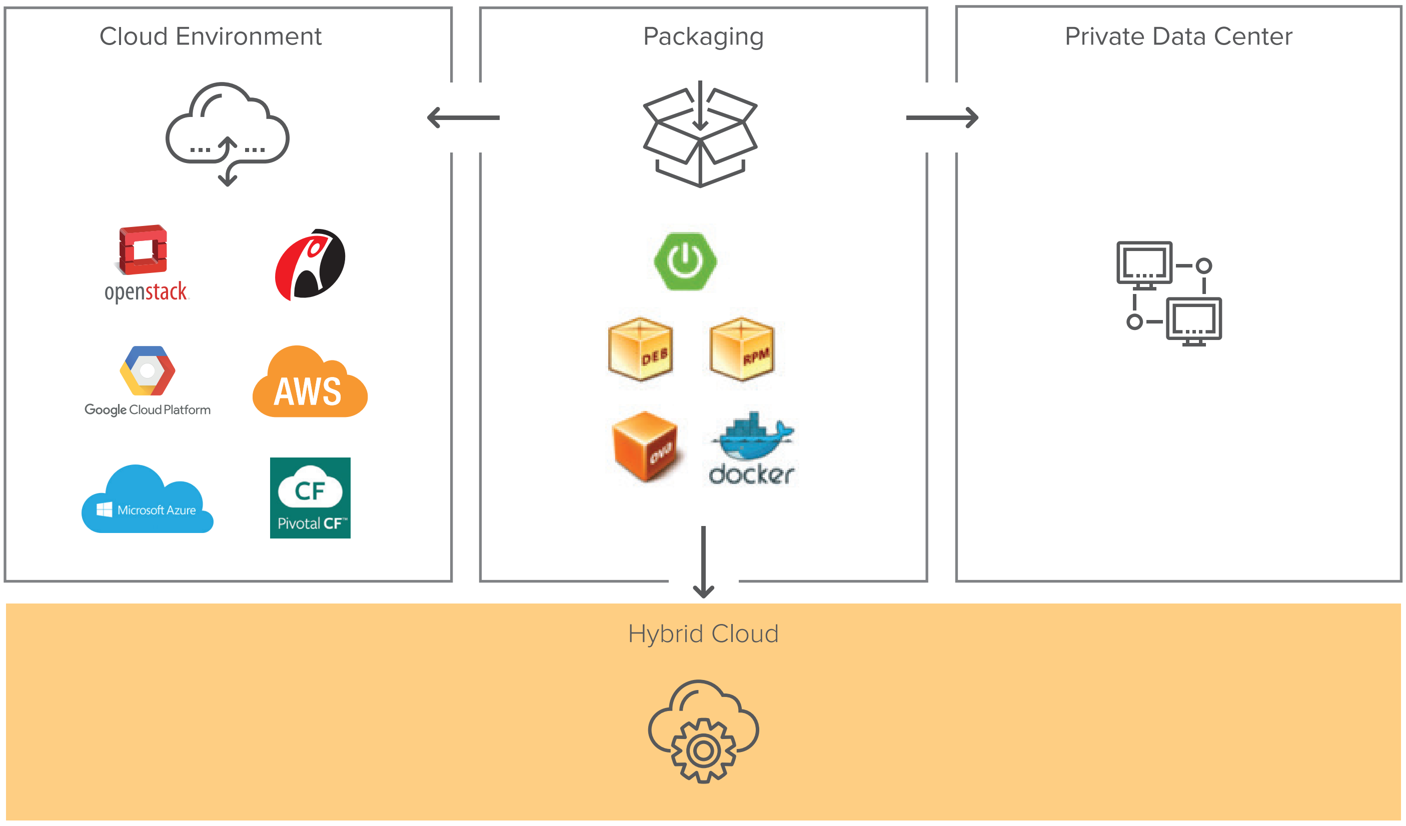

Figure 3. **w2bill**™ Packaging for multiple deployments

CAPABILITIES

A summarized list of What does **w2bill**™ Real-time Task Management Framework provide:

- Concept of order, tracking each request from start to end
- Concept of duplicated order through configurable key fields
- Concept of configurable and versioned services
- Concept of service grouping for logical aggregation
- Concept of service users and respective authentication
- Cross-platform self, up and out scaling
- Request validation rules, from syntax to business criteria
- Task flows managed by orchestrators and implement through executors
- Order lifecycle with differentiated Order status
- Multiple integration languages: SOAP, REST, etc.
- Fault Tolerant design through component clustering
- No central point of failure
- Logging with performance metrics
- Component health checks and monitoring
- Data grid for high performance, ACID data access and manipulation
- Clustered, resilient and distributed data persistency
- Multiple data persistency solutions support (RDBMS, NoSQL)
- Distributed orchestrator approach
- Standard Executors for common operations, with capability for client-specific enhancements
- Flow Pausing and Resume
- Alternative flow execution for standard flow failure scenarios
- Inter-component communication through messaging for at least once delivery assurance

ROADMAP

Fully aware the product of today isn't the product of tomorrow, since the needs and technology of today will surely be replaced in the future, the **w2bill**™ solution has a constantly evolving roadmap of features. These new capabilities will come from the new experience gathered from our customers, as well as their insight, advice and suggestions. There is no aim in having customer-specific branches stopped in time over each deployment, but assure a recurring evolution targeted at providing new and better ways to support technology and business.

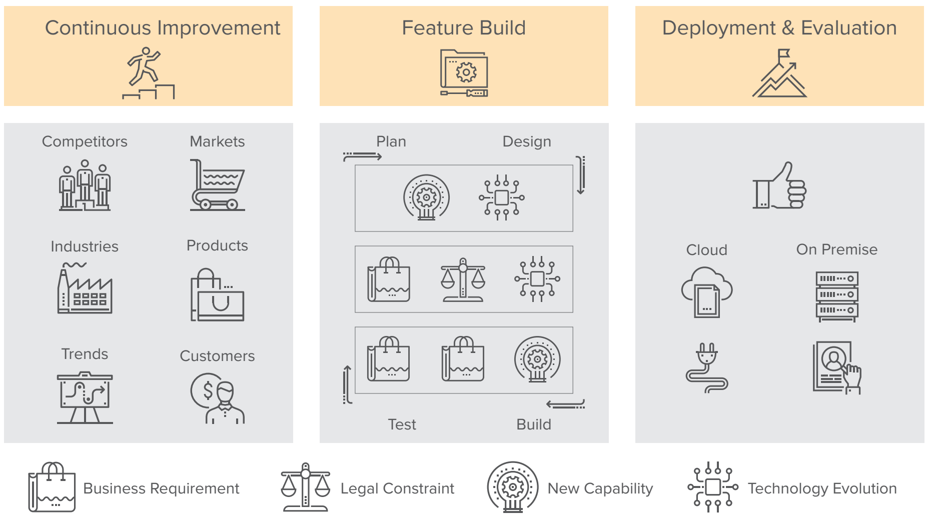

Figure 4. **w2bill**™ Architecture Overview

By constantly investigating on what is happening and about to happen, as well as converging experiences across markets, customers and mindsets, we envision the path of improvement to be made available. Partnership with our clients is key in the forming of this vision and the definition of the roadmap.

APPENDIX

Some technical insights on w2bill™ Realtime

PROCESSING ENGINE

JAVA 8

w2bill™ Realtime is built using the latest version of Java language. It enables to take advantage of all new features and performance improvements introduced with version 8. Java has been selected as the reference language given its wide industry adoption, the simplicity to deploy on multiple platforms, it's code portability. Java has a widespread number of open source plugins and frameworks, an extremely active developer community, and an extensive evolution roadmap as well as a support group.

PIVOTAL SPRING FRAMEWORK

Pivotal Spring Framework is an open source framework that supports the development of Java applications, by providing help with infrastructure needs and supplying a consistent programming model over different technologies.
It has been widely used throughout the Java development industry, as an alternative to the Enterprise Java Beans model.

Pivotal, and particularly it's Spring team, are always planning the future and driving the framework to respond to new business requirements. Relevant examples are the Cloud Stream project and the introduction of the reactive programming in next release 5.0. w2bill™ will follow these evolutions closely to extract from them any relevant improvements.

PIVOTAL RABBITMQ

RabbitMQ is an open source message broker managed by Pivotal and is well supported by the Spring Framework. It features reliability, flexible routing, clustering and highly available queues. It integrates with multiple programming languages, making it easy to integrate the messaging pattern with different application, thus enabling w2bill™ to quickly and seamlessly incorporate specific client modules that are developed in languages other than the core w2bill™ implementation language.
RabbitMQ is a broker capable of supporting multiple protocols, which further enables the integration capabilities of w2bill™, not restricting its messaging approach to one specific protocol.

RESOURCE MANAGEMENT

APACHE MESOS

w2bill™ is a naturally distributed system. Any of its components and respective instances can be executed in multiple machines, thus contributing for better performance through horizontal scalability, as well as avoiding central points of failure, by relying on clustering and fault-tolerance techniques.
The implementation is designed to use not only common virtualization of machines, but also containerization approaches, such as Docker or Kubernetes.
The distribution and resource management are thus a concern that has been properly addressed using Apache Mesos, which features centralized handling for deployment and scaling of **w2bill**™'s components in any sort of installation, from on-premise to cloud or a mix of both.

DATA ACCESSIBILITY

APACHE IGNITE

Apache Ignite is an In-memory Data Fabric that provides high-performance data, compute and service grids. It supports fully ACID-compliant distributed transactions, ensuring consistency across all data and supporting standard SQL syntax to query the objects stored in the data grid. Accessing this data grid is possible through multiple programming languages.

Another relevant feature is the advanced clustering capabilities enabling scalability, fault-tolerance and high performance requirements.
Currently, **w2bill**™ uses the data grid as a layer to interact with the persistency solution, with read-through or write-through approaches. The effective writing is executed in an asynchronous manner, to expedite performance.

PERSISTENCY

APACHE CASSANDRA

The choice of Apache Cassandra for storage was based on its multitude of capabilities, particularly:
• Decentralized nature, avoiding single points of failure or network-based bottlenecks
• Fault Tolerance, through its data replication across nodes and data centers, together with the capability of failed node replacement without downtime
• Scalability, by use of "nearly infinite" multiple nodes to support processing and storage growth
• Elasticity, by relying on its nodes increase for read/write throughput
• Professional support and proven use on global enterprises (CERN, eBay, GitHub, Apple, Netflix to name a few)

The **w2bill**™ Framework relies on Apache Cassandra solution to store its information across its components. The use is indirect, however, as the layer of accessibility is performed via Data Fabric Apache Ignite. It is the responsibility of this layer to effectively communicate with Apache Cassandra, for effective storage (write) and reading of data.

CMAS – Systems Consultants, Lda
Av. Dom João II, nº 44 C, 2.2
1990-095 Lisboa
T: +351 919531710
mail@cmas-systems.com
www.cmas-systems.com