White Paper on

w2bill™ PAYMENTS

CONTEXT
ARCHITECTURE
CAPABILITIES
ROADMAP

INDEX

INTRODUCTION

As new advancements arise at a rapid rate, led by the transformation triggered by smartphones and tablets, new technologies including mobile wallets and on-demand apps, a new era of digital assistants, and enhanced connectivity technologies - such as NFC and Bluetooth – are becoming an intrinsic part of the way consumers interact with the products and services they rely on every day.

Mobile payments, on-demand consumption, the rise of 'near-field' technologies and the sharing economy, are currently helping to shape the way we use and pay in our ordinary lives. And with this status quo, customers expect a wide variety of options when they're about to pay, with little or no friction at all.

Prominent brands and e-commerce platforms have evangelized customers to expect a no-friction payment experience, storing their consumer information, such as credit card details and other similar ways of collecting money, as a means of speeding up the whole purchase experience and guaranteeing continuity upon each visit. With Stored Credentials as the preferred payment method, this results in fierce competition between fintech startups, credit card providers and other payment handlers to target every piece of the demographic tissue – especially the younger, lucrative and more digitally-apt crowd – and grab their payment method preference for continuous use.

Almost every player in every market feels the need to track their payment channels to the consumer's needs and habits, to ensure the kind of friction-free and real-time payment experience the market has been growing accustomed to. In the 'big data' era, success in this area is widely measured by the ability to capture quick counter-consumption payments, best suited to the customer's location – physical store, mobile, desktop, tablet or wearable gadget. The compelling experience that, for instance, mobile wallets are able to provide in terms of speed and reward – the gamification of buying – seem to be an important key to drive bigger consumption, speed up collection, and generate a more well-rounded satisfaction, as the payment act – or the 'checking-out' – is starting to be a seamless part of the shopping and consumption experience.

CONTEXT

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.

Businesses now support their flexible, adaptive, innovative and modern business models, especially on the pressure they put into operating costs. From the start-up 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.

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 very little time to do it.

ARCHITECTURE

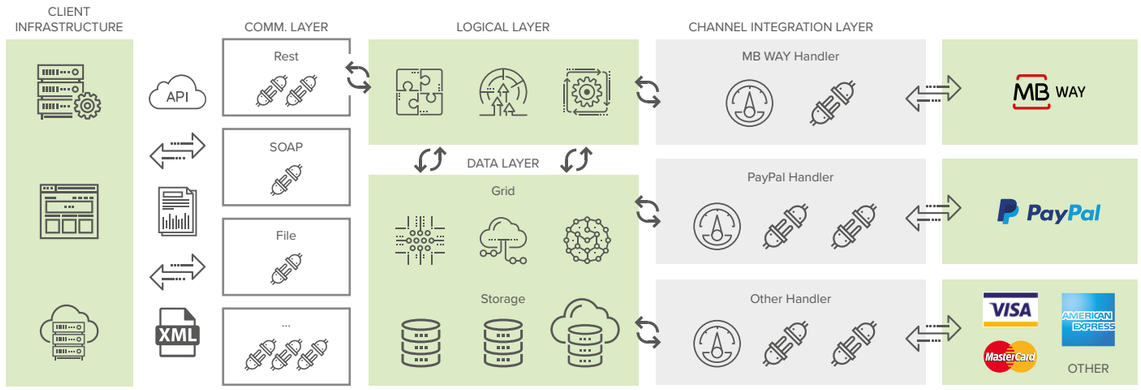

Figure 1. w2bill™ Architecture Overview

w2bill™ was designed, from its early stages, with the future in mind. Following old paths made little sense in a world marked by constant evolution, and with the growing certainty that today's solution will most likely be outdated soon. With such concerns in focus, different ways of achieving the set objectives were considered, where scalability, fault-tolerance, reliability, resilience and configurability were key. What is true today will soon be outdated and outpaced. The current acceptable 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.

From a functional perspective, w2bill™ incorporates the notion of Payment Orders, which enable a client to simplify the management of financial transactions. For instance, when requesting for a payment to be done, the client creates a Payment Order within the platform, indicating the amount but also the channels on which it can be paid (along with any relevant additional information).

As responses are processed, the status of the Payment Order evolves. Each response is treated individually, and does not necessarily imply its amount matches the original one, meaning partial payments for any given Payment Order are automatically handled, along with partial reversals triggered by the end client.

Additionally, as the financial transactions can be handled by Payment Orders, there is embedded control to make sure the same transaction isn't requested twice while in progress, further enforcing validations to avoid erroneous scenarios.

Associated with Payment Order is the notion of validity, which further enables the customer to manage how it desires its financial transactions to be handled. As an example, it is possible to define a given Payment Order must be paid within the first 5 attempts of payment, otherwise, it is automatically cancelled.

The three core concepts behind w2bill™ PAYMENTS

1. PROCESSING ENGINE

w2bill™ PAYMENTS incorporates a Processing Engine, that further enables complex task management. The Processing Engine 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.

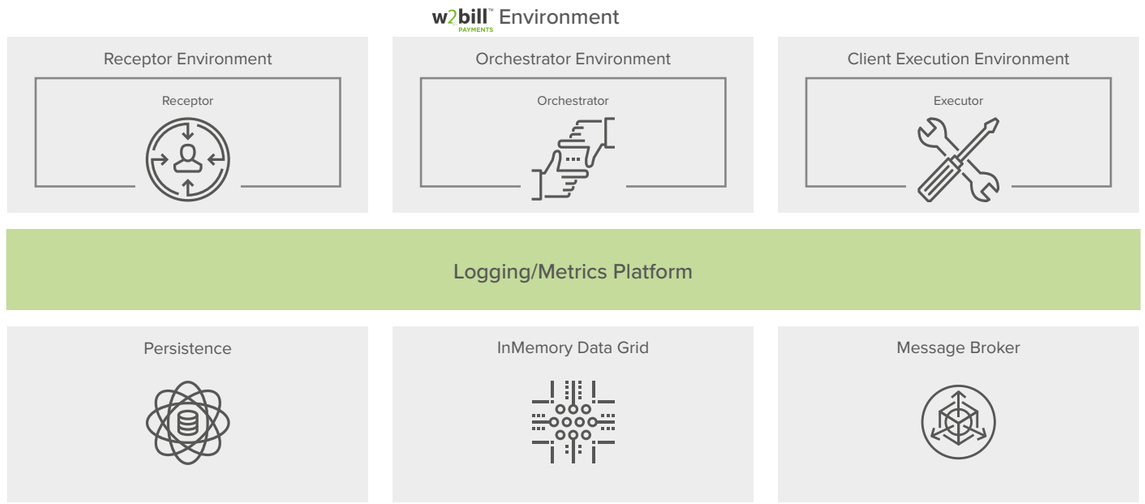

Figure 2. w2bill™ PAYMENTS Processing Engine Overview

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 its 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 persistency 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 work flow. 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.

Imagining a scenario where a customer is already operating with a limited set of batch operations, where the financial transactions are carried out by daily sending and receiving files to and from the banks (or other financial institutions). The customer identifies as crucial to implement a real-time channel, such as PayPal. The new channel is deployed through a dedicated set of components, which don't affect the existing operation. New flows are configured, new functionality is enabled, but the platform continues executing.

2. PLATFORM

2.1 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 in 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.

As an example, a customer starts noticing the processing of real-time requests is often hitting timeouts, causing requests to have to be reprocessed or even end-client dissatisfaction. By analysing the metrics, it is possible to identify a machine has been modified and is now running at over-capacity instead of the planned 30%. Actions can then be carried out to perform a different approach of scaling, enabling the issues to be solved.

2.2 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 behaviour. All components of w2bill PAYMENTS are prepared to write application logs to a file – on 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 ElasticSearch index with a predefined schema. It is possible to search logs by an order id, a correlation id or by log message. Logs are published asynchronously to the broker improving system performance.

2.3 DISCOVERY SERVICE & CONFIGURATION SERVER

Components' instances can be started on 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.

2.4 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, to start and stop instances, maintaining the desired Service Level and the effective use of resources of the running environment.

This capability is tied with the Metrics functionality to continuously monitor and react to changes in the infra-structure, and plan actions to mitigate the impacts.

2.5 FRONTEND

w2bill PAYMENTS presents a unified GUI that enables common operations to be performed over the Platform, such as diagnosis, tracing, log validation and status checks.

The approach chosen allows each customer to have dedicated screens or views, representing relevant information to be displayed, instead of a static screen showing predefined data. It also incorporates configurable profiles and roles, associated with each user, further defining what each user can view, create, modify or remove, based on organizational requirements.

The Frontend is based on Angular JS technology, enabling an up to date look & feel, as well as device-responsive interactivity. Both the information displayed and the look & feel of the Frontend is specific, on a customer by customer basis. The rationale is that it makes the most sense to adapt the Frontend to the corporate image and way of working, not the other way around.

This way, it'll be possible to, for instance, monitor the volume of processed transactions daily, per channel, and display that information on the Frontend for the users with appropriate privileges to view.

2.6 AUTHENTICATION SERVICE

To comply with the need of security and validated service access, w2bill PAYMENTS 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 PAYMENTS –, 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 a LDAP server, depending on the customer needs.

This capability greatly empowers the organization in safeguarding the access to its financial transactions, ensuring only valid and authenticated users can interact with it.

2.7 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 integration aspects are met. As an example, it is possible to deploy one Executor component implemented over 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.

This capability is particularly important due to the variety of channels existing. It is possible to expect certain providers to prefer technologies with stricter protocols, focused on bit-level performance, as well as others preferring light-weight approaches completely differing from one another. Legacy solutions are not impaired nor do they impair newer ones, when integration with w2bill PAYMENTS is concerned.

As referred previously, this capability can be used for the client to use its own implementation of channels that have proprietary logic, which is not to be incorporated directly into the platform but benefits from using it.

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 customization. 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.

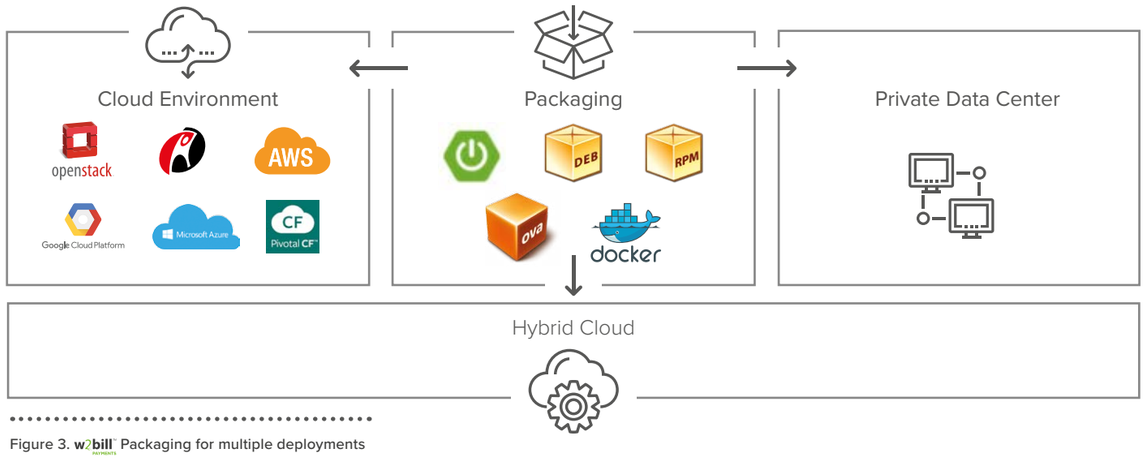

Figure 3. w2bill PAYMENTS Packaging for multiple deployments

3. DATA PERSISTENCY

It is immediately recognized that data must be kept safely, consistently and securely, to be used. Given the scope of the solution involves handling financial transactions and interacting with internal platforms for reconciliations of capital, it is paramount that no information is ever lost. Data must be tracked and safe from unwanted manipulation. It must also scale easily, and have as little constraints in its management as possible. Because of the diversity in concerns, there is no direct solution or clear approach on which technology is best, to ensure persistency. Standard RDBMS's are tried-and-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, including mixed strategies with segregated data between platforms.

To understand better the following concepts, it is relevant to understand how w2bill™ is designed, particularly on how its components interact. w2bill™ is packaged as a set of micro-services, each responsible for a certain domain of actions. With this approach, it becomes possible for new components to be added or old ones removed without jeopardizing the consistency of the whole solution.

This componentization is achieved, as referred, by the modelling of the components as stand-alone micro-services – or application –, capable of interaction with each other, with several integration services providing functionality for this end. The Configuration Service and the Discovery Service allows the components to register and configure themselves, enabling their interaction with the full platform when they become 'plugged in'.

4. CHANNEL INTEGRATION

The previous strategy is an important step for the quick and easy integration of multiple channels. Following these guidelines of platform integration, new micro-services targeted for specific channels' logic and management, can be incorporated without disrupting the existing platform or requiring major changes to it.

It is also a key enabler for standard business and technical scenarios, that are more and more frequent. A customer can have its own tailored channels or want to perform a quick incursion into new markets with different regulations, or even embrace changes in its own operating markets – new technologies, new laws, new business opportunities. For any of these scenarios, the micro-service building process can range from a simple configuration of existing micro-services that feature a wide array of capabilities suitable for the new specific purpose, to enhancements of already existing services, or ultimately custom implementation of dedicated logic with new micro-services.

The platform copes with any of these approaches with limited disturbance. Customers can grow and adapt without fearing loss of service. It is also assured that client-specific integrations, for private or proprietary protocols, can be interfaced with the whole solution without leaving the client's control. It is possible, as an example, for a client to have its own implementation of a channel integrated with the w2bill™ solution. It leverages the capabilities of the platform, without publishing or relinquishing control of its private implementation.

CAPABILITIES

A summarized list of what does w2bill™ provide:

- Process files for offline batch processing, received from financial institutions
- Generates files for offline batch processing, sent to financial institutions
- Support real-time payment requesting and respective cancellation
- Enable real-time payment confirmation and rejection
- Support synchronous and asynchronous communication models, for real-time
- Generate reporting based on financial transactions processed
- Integrate new payment channels (different protocols)
- Incorporate proprietary or private transactional channels of the customer
- Support General Ledger reporting
- Support insert-on-missing or report-on-missing reconciliation tactics (real-time with batch processing)
- Configurable tasks and flows
- Cross-platform self, up and out scaling
- 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 solution support (RDBMS, NoSQL)
- 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.
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.

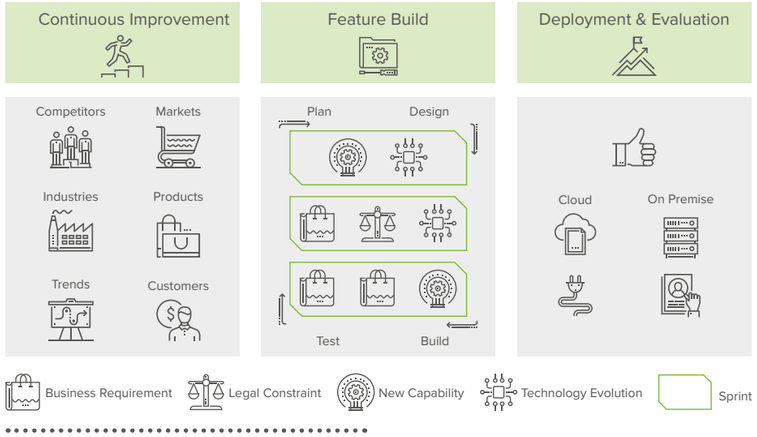

Figure 4. w2bill™ Roadmap approach

APPENDIX

Some technical insights on w2bill™

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 centres, 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.
Regardless of this choice, as stated, the solution is designed to interact with multiple Data Persistency solutions, based on specific customer requirements.

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.

PROCESSING ENGINE

JAVA 8

w2bill™ 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 in multiple platforms, its 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 its 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.

RESOURCE MANAGEMENT

APACHE MESOS

w2bill™ is a naturally distributed system. Any of its components and respective instances can be executed on multiple machines, thus contributing to 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™ components in any sort of installation, from on-premise to cloud or a mix of both.

CMAS
SYSTEMS CONSULTANTS
Av. Dom João II, nº 44 C, 2.2 • 1990–095 Lisboa, Portugal
T: +351 919531710 • mail@cmas-systems.com • www.cmas-systems.com

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