Water -M Project

D3.2 Network Monitoring and Distributed

Operation System Implementation

**History**

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Version & Status | Author | Modifications |
| 21/11/2017 | V1.0 | Güven KÖSE (Mantis)  Mustafa KÖSE (Mantis)  Aydıncan POLATKAN (Mantis) | A first document |
| 23/11/2017 | V1.01 | Güven KÖSE (Mantis)  Mustafa KÖSE (Mantis)  Aydıncan POLATKAN (Mantis) | Structure of a document |
| 11/12/2017 | V1.02 | Güven KÖSE (Mantis)  Mustafa KÖSE (Mantis)  Aydıncan POLATKAN (Mantis) | Review |
|  |  |  |  |

**Table of Content**

[**1** **Introduction** 4](#_Toc500837323)

[**2** **Scope** 4](#_Toc500837324)

[**3** **What Is Water-M Analytics Platform?** 4](#_Toc500837325)

[**4** **What Are The Benefits Of The Platform?** 4](#_Toc500837326)

[**5** **System Architecture** 6](#_Toc500837327)

[**5.1** **Water-M Analytics Platform Connectors** 7](#_Toc500837328)

[**5.2** **Water-M Enterprise Service Bus (ESB) Module** 8](#_Toc500837329)

[**5.3** **Water-M Business Intelligence (BI) Module** 13](#_Toc500837330)

[**5.4** **Water-M Big Data Storage Module** 18](#_Toc500837331)

[**5.5** **Water-M Visualization Service** 21](#_Toc500837332)

[**6** **Screenshots From WaterM Analytics Patform** 24](#_Toc500837333)

[**7** **Conclusion** 29](#_Toc500837334)

# **Introduction**

This document describes the Smart Water Network Monitoring and Distributed Operation Systems about Task 3.2. of the Water-M Project. This task is focus on the definition of the operational data model and data analysis methods for Water-M. All the processes defined in this task have been considered at least three operational modes for the water grids: (i) normal operations focused on water supply assurance, (ii) small or controlled events and incidents, and (iii) emergency situations.

# **Scope**

Water-M big data analysis platform, architectural design and functional modules form the scope of this document.

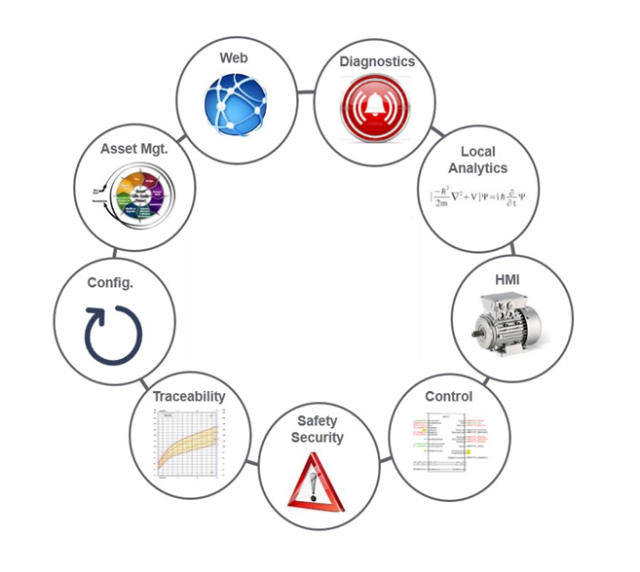
# **What Is Water-M Analytics Platform?**

Water-M Analytics Platform is a big data analysis platform that can analyse and give meaning the millions of data which is produced by Scada, IoT and Sensors that are using the management of water supply network. Also Water-M Analytics Platform can represent these meaningful data on interactive dashboards.



# **What Are The Benefits Of The Platform?**

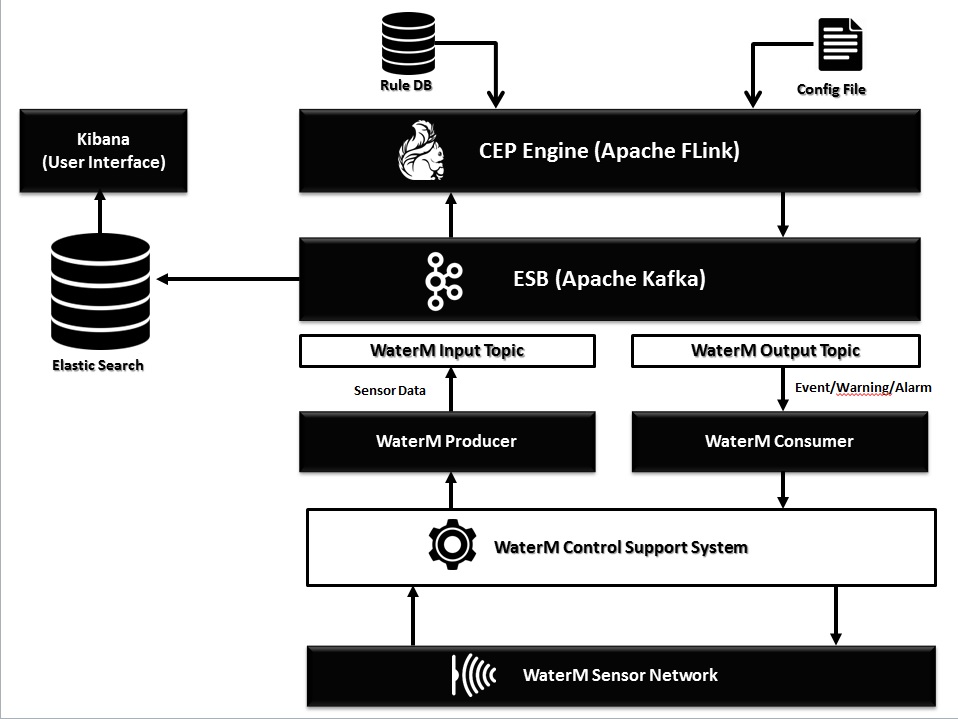
By means of Water-M Big Data Analysis Platform, data of Scada, IoT and Sensor that could be found in water supply network can be analyzed in real-time. While analyzing the data , error conditions can be monitored, alarm, report or any type of notification can be triggered. Even anomaly detection can be applied on the data.

Some of the Water-M Analytics Platform abilities that aim to improve the productivity on water supply network managements are listed below

* It provides real-time data flow and data analysis with high performance and low latency
* It has flexible and scalable architecture, with this feature, system can be easily adapted new requirement which could be change time to time.
* It provides easy and clean reports via its modifiable dashboard.
* It has feature to create new graphics, dashboards and reports with no extra code imple-mentation.
* Dynamic filters and quick analysis are applied from dashboards.
* It provides data migration for different departments.
* It provides long time access into produced big data
* It provides to apply any analysis on historical data
* It present the detailed analysis on data which can be either analog or digital
* It detect the anomaly on sensor data
* It provides to create alarm or notification mechanism for determined rules.
* It provides to detect critical errors in any device.
* It provides to find out the devices which are broken often and analyse the reasons.
* It calculates the shut down and run time duration for any device
* It provides to detect out of order devices
* It provides to detect user error via using detailed analysis.

# **System Architecture**

Water-M Analytics Platform is software which includes flexible, extensible, scalable, fast and high permanced, fault tolerance open source applications that has been accepted all around the world on big data processes.



Within the scope of project, Water-M Analytics Platform has been developed according to the functional architecture which is represented before.

On the functional architecture, All real time data has been collected by “ WaterMPrducer” which is defined on Apache Kafka that has been using as ESB ( Enterprise Service Bus ). and then the real time data is send to “WaterM Input Topic” which has all sensor data. This “WaterM Input Topic” provides concurrent multi access platform to share the data for any other services. All these data flow processes happen on “WaterM Sensor Network” layer.

Also these collected real time data is transmitted to CEP (Complex Event Processing) Module which are developed with Apache Flink. CEP Module processes the coming data packages according to rules which are determined by system user and produces necessary actions.

The actions outputs that are produced by CEP Module are converted the data and send to the other topic services with name “WaterM Output Topic” that is also created on Apache Kafka in ESB. “WaterM Output Topic” can be accessed simultaneously by many software services. On the WaterM Architecture, the data which comes to “WaterM Output Topic” is dispatched to the related sensor process services by “WaterM Consumer”.

To save and visualize the all data either coming or produced, in “WaterM Sensor Network” Layer, ElasticSearch cluster which has inverted-index structure to store data is set up. With ability of near-real time indexing process, ElasticSearch can easily handle all kind of data coming from whole WaterM system.

We are using Kibana to visualize the data in ElasticSearch. With easy connection between ElasticSearch and Kibana, Kibana creates dashboards, charts, tables etc. while using the data stored in ElasticSearch. Also this provides the near-real time monitoring for all data in WaterM system. Advantages of the Water-M Analytcis Platform System Architecture;

* Scalability : All system components of Mind4.0 can be run on any amount of distributed system.
* Low Latency : Mind4.0 has a capacity to process real-time very high volume of data with minimal delay.
* Fault Tolerant: With strong and intelligent architecture, Mind4.0 can provide minimum interruption on any system component failure.
* High Availability: Mind4.0 uses durable hardwares and produce confidential software for all projects.

## **Water-M Analytics Platform Connectors**

Water-M Connectors component is a software that collects different formatted data from different applications and converts these different types into to one common format to be processed by ESB.

Water-M Analytics Platform has multiple connectors. Also it provides to integrate new connectors which is used for different softwares. These are the some of important data types which are supported by Water-M Analytics Platform;

File System

S3

HDFS

FTP

HBASE

IBM MQ

LOG Files

JDBC

Azure IoTHub

Couchbase

Kinetica

Oracle GoldenGate

SAP HANA

Vertica

MS SQLServer

Oracle

MySQL

DB2

IMS

PostgreSQL

VSAM

CICS

Apache Ignite

Blockchain

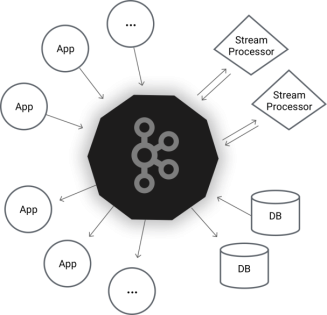
Cassandra

CoAP

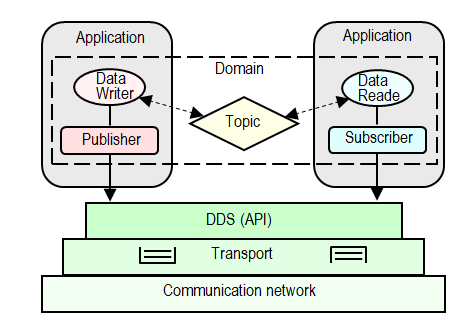
MongoDB

## **Water-M Enterprise Service Bus (ESB) Module**

This component is a messaging application sending big data collected from different sources by Water-M Connectors to different software components in the platform. Water-M ESB module has high availability, low latency, scalability and distributed processing capabilities.



The ESB or DDS (Data Distribution Service) is based on a communication paradigm called publisher-subscriber centric data (DCPS: Data-Centric Publish-Subscribe) in which the middleware operates on the nature of data exchanged between the elements of the application, leaving in the background who is the producer and the consumer them. The DDS middleware creates a data space classified according to previously declared types (topics). Applications that follow this model form a distributed system in which nodes communicate with each other by sending (the publishers) and receiving (subscribers) data to the space defined by the middleware anonymously.



In particular, this model defines a small number of classes of elements:

* Domain: It represents the logical association between participants in a distributed system. Outside of a common domain, entities cannot communicate with them.
* Participant: These are containers the rest of entities that enable communication within the same domain.
* Data Writer (DW): An entity responsible for generating and publishing a specific type of data.
* Publisher: It is a DW container that shares a quality of service (QoS), that is, a set of features which control some aspect of the distribution.
* Data Reader (DR): An entity that provides access and consumes a specific type of data.
* Subscriber: It is DR containers that share a quality of service (QoS) common.
* Topic: It represents the association of a name with a data type within a given domain.

Communication between entities is carried out based on that reference the same data type (topic): by the publishers that are the elements that create the new data for that topic, and by subscribers that are elements in the system register interest in receiving the data related to certain topics. The middleware itself is responsible for enabling communication between them automatically and transparently.

The model is characterized by being weakly coupled, since both publishers and subscribers are decoupled with respect to time (the data can be produced by the middleware stored for later use, for example in the case of new subscribers that require knowing the previous state system) and the space (publishers and subscribers do not know each other and can be installed in different processing nodes).

In the case of publisher, the mechanism of data recording is direct: once the application has generated data to transmit, it performs one of the operations associated with the DW for publishing data (e.g. Write or dispose). Once executed, the data is transmitted in a unidirectional form (the origin is an DW and destiny of one or more DRs) and asynchronous, although the specification provides operations to block the application until the data are delivered to your / s destination / s. However, data reception in the subscriber side can be done on either a synchronous or an asynchronous. This interaction model is valid not only for receiving data but also for the notification of any event occurring during communication (e.g. the failure of the specified QoS). Concretely, the application can be notified in two paradigms:

* Notifier/Listeners, where subscribers register a callback function for asynchronous access to the event occurred while the application continues to run.
* Conditions and Wait-sets, these are synchronous mechanisms of access to event. The Wait-sets can block the application until they meet one or more conditions. These can be of three types:
  + GuardConditions: These are created and managed conditions completely by the application.
  + ReadConditions These are conditions that indicate the characteristics of the desired data are received and therefore are created by the application but internally managed by the middleware.
  + StatusConditions: These are conditions created by default to monitor the status of certain internal entities (Topic, Subscriber, DR and DW). Consequently, the creation and management of these conditions depends solely and exclusively of the middleware.

Finally, it is noteworthy that the DDS specification defines only the interface (API) and therefore does not include a concurrency model that defines how to process the data and events. The concurrency model is a characteristic aspect is defined in each implementation of DDS.

DCPS middleware layer is responsible for encoding / decoding and send / receive data within the global space using standard transport mechanisms, such as UDP, TCP or other protocols such as shared memory. Applications simply use the elements defined by DDS to read / write data to create virtual global space middleware without considering implementation details of lower level and without knowing who or how many items are on the other side communication.

Although the paradigm publisher / subscriber is not new in the development of distributed applications, DDS emphasizes flexibility in system configuration: each entity has a number of individually associated QoS parameters, some of which may be modified at run time, and thus dynamic reconfiguration is performed in the system.

QoS parameters included in the DDS standard, can be divided into five categories according allowing variations in aspects of:

* Availability: This encompasses all those parameters control policies for queuing and storing of data in the system.
  + History: It indicates the number of samples to be stored until the data delivery is effective.
  + Durability: It specifies how certain samples should be stored by History.
  + Lifespan: It indicates the maximum validity of the stored samples.
  + Lifecycle: It specifies the maximum time that should be kept stored samples once there are no entities to update those instances.
* Reception and format: They specify how they should be delivered and presented the data at the destination.
  + Presentation: It selects different delivery policies and grouping of samples received.
  + Reliability: It sets system reliability in data reception.
  + Partition: It creates a namespace within a particular domain, logically separating different topics common to a particular domain.
  + Destination\_Order: It sets how to sort the samples received, either by one or more DW, for the same instance.
  + Ownership: It indicates whether an instance can be modified by one or more DW. It is the main mechanism of redundancy within the standard.
* Temporal: Its objective is to control the latency in the data distribution. It will be discussed more specifically in the standard analysis for real-time systems.
* Use of resources: Limits the number of resources consumed in the system.
  + Resource\_Limits: It limits the amount of memory to use by setting the maximum number of samples per topic by topic and request samples per instance.
  + Time\_Based\_Filter: It limits the bandwidth and processing data by setting a minimum time between data collection.
* User Settings: Parameters for adding user information in the application level to the various entities. It included in this category user\_data parameters, and Group\_Data Topic\_Data.

QoS policies can be set at different levels in DDS (topics, publishers, DW, subscribers and DR), which allows designers to build applications specific contracts based on QoS requirements that require and offer different entities. As the identity of publishers and subscribers are unknown to each other, the DDS middleware is responsible for determining whether the policies of quality of service (QoS) offered by publishers are compatible with those required by the subscribers, and allowing the transfer of data between them only if the requirements are compatible.

Some benefits of the Water-M Analytics Platform ESB component are;

* Read and write streams of data like a messaging system.
* Write scalable stream processing applications that react to events in real-time.
* Store streams of data safely in a distributed, replicated, fault-tolerant cluster.
* Run as a cluster on one or more servers.
* A topic can have zero, one, or many consumers that subscribe to the data written to it.
* Each partition is replicated across a configurable number of servers for fault tolerance.
* Each server acts as a leader for some of its partitions and a follower for others so load is well balanced within the cluster.
* Producers publish data to the topics of their choice.
* The producer is responsible for choosing which record to assign to which partition within the topic.
* Consumer instances can be in separate processes or on separate machines.
* If all the consumer instances have the same consumer group, then the records will effectively be load balanced over the consumer instances.
* If all the consumer instances have different consumer groups, then each record will be broadcast to all the consumer processes.
* Messages sent by a producer to a particular topic partition will be appended in the order they are sent.
* A consumer instance sees records in the order they are stored in the log.
* A distributed file system like HDFS allows storing static files for batch processing. Effectively a system like this allows storing and processing historical data from the past.
* By combining storage and low-latency subscriptions, streaming applications can treat both past and future data the same way.

## **Water-M Business Intelligence (BI) Module**

This component is a business intelligence application with scalable, distributed, workable and high usability capabilities that can process data delivered to it by the Water-M ESB in near real-time, allowing it to execute complex rules on that data. With the Water-M BI component, complex events and rules can be manipulated on large data sets, and this component works as a CEP Engine.

The CEP engine is event processing that combines data from multiple sources to infer events or patterns that suggest more complicated circumstances. The inputs of this element can be an independent resource or the results of another CEP rule. This architectures are very natural for modeling systems needing to react to streams of data produced by the external world, such as the data produced by sensors. Therefore, this component of the architecture is really important due to the ability to identify meaningful events (such as opportunities or threats) and respond to them as quickly as possible. The main items used in the Water-M complex event processing are shown below.

Events : An event is an immutable record of a past occurrence of an action or a state change in the system, for example: a new reading of measured on a grid, a new position of a vehicle on a battlefield.

The events are processed by IT systems through their representations as objects that include properties that describe the action that caused the event.

Complex Events: Events that representing the abstraction of other events, these latter are known as members of the complex event. For example: The 2004 tsunami represents an abstraction of a set of natural events.

Derivative Events: Events created from the application of a method or process for one or more events. For example, the analysis of stock prices over a period of time can cause a new event indicating whether shares have risen or fallen over the period.

Event Producer: System or application that generates events, for example, electric meter, clock, sensor.

Event Sink or Event Consumer: System or application, event consumer once they have been processed. For example: Dashboard, Database.

Event Stream: Ordered linear sequence of events, often time-ordered, bounded or unbounded and can contain different types of events. The event streams are so dimensioned from the application of different criteria (time, space, origin).

Window: A bounded portion of a stream of events.

Cloud Events: Partially ordered set of events, bounded or not, where the partial order is imposed by causality, time or any other relationship between the events of the cloud. Usually the cloud events from different event producers and contain different types and event streams.

The difference between a flow and a cloud of events is that there is no relationship between the events that totally order events in a cloud. A flow is a cloud while a cloud cannot be flow because its events are not completely sorted.

Event Processing Agent: Software module responsible for processing event clouds.

Event Processing Network: Set of event processing agents (EPAs) and set channels and producers of events connected to them.

Event Processing Language: High-level language used to establish the behavior of event processing agents (EPAs).

Rule: Prescribed methods used in event processing. In the rules are defined the sentences that establish processing windows, patterns and conditional clauses event processing.

Event Pattern: Template composed of a set of other templates, relational operators and variables that purports to represent a sequence known or supposed of events that completed by the abstraction of a complex event. For example, the pattern of loss event of speed with the heating of engine above the threshold maximum recommended if both occur in a window of 5 seconds may involve a complex event of serious breakdown to a vehicle.

Complex Event Processing: Event processing - simple or complex-including reading, creation, transformation and abstraction.

Event Stream Processing: Event stream processing, for example applications that process event streams share price at the stock exchange as they arrive in time to calculate aggregate sums, averages, volume, etc..

Apache Flink (https://flink.apache.org/ ) is used as a CEP component in WaterM Project. In the scope of the project, some determined rules for waste water plant and sample codes which are implementation of these rules in Flink are listed below

Sample rules for waste water processes;

If (Phase1WasteWaterLevelSensorState == On)

SecondGridState = ON

Else

SecondGridState = OFF

If (Phase3WasteWaterLevel > 10m)

SecondPumpState = ON

Else

SecondPumpState = OFF

If (Phase3WasteWaterLevel > 15m)

ThirdPumpState = ON

Else

ThirdPumpState = OFF

If (Phase3OxygenLevel < 2 mg/lt)

Blower1State = ON

Else

Blower1State = OFF

If (Phase4FlowRate > 700 lt/m)

PromotionPump = ON

Else

PromotionPump = OFF

If (Phase5PhValue > 9)

LimeTankPump = ON

Else

LimeTankPump = OFF

If (Phase6OxygenLevel < 1.5)

NitrogenPump = ON

PhosphorusPump = ON

Else

NitrogenPump = OFF

PhosphorusPump = OFF



Third party integration points of the Water-M BI component;

Boundary

Cloudwatch

Csv

Datadog

Elasticsearch

Email

Exec

File

Gelf

Google\_bigquery

Google\_cloud\_

Graphite

Graphtastic

Gzip

Http

Irc

Jason

Jira

Juggernaut

Kafka

Librato

Loggly

Lumberjack

Metriccatcher

Mongodb

Nagios

Newrelic

Opentsdb

Pagerduty

Pipe

Rabbitmq

Rackspace

Redis

Redmine

Riak

Riemann

S3

Sns

Solr\_http

Sqs

Statsd

Stdout

Stomp

Syslog

Tcp

Udp

Webhdfs

Websocket

Xmpp

Zabbix

Zeromq

Some benefits of the Water-M Analytics Platform BI component are;

* Creating workflows to analyze big data.
* Running complex event rules on the real time big data.
* Is stateful and fault-tolerant and can seamlessly recover from failures while maintaining exactly-once application state.
* Distributed architecute.
* Scalable to 1000s of nodes and beyond.
* Streaming: Processing that executes continuously as long as data is being produced.
* Performs at large scale, running on thousands of nodes with very good throughput and latency characteristics.
* Processing that is executed and runs to completeness in a finite amount of time, releasing computing resources when finished.
* Provides results that are accurate, even in the case of out-of-order or late-arriving data.
* Guarantees exactly-once semantics for stateful computations.
* Supports stream processing and windowing with event time semantics.
* Supports flexible windowing based on time, count, or sessions in addition to data-driven windows.
* Fault tolerance is lightweight and allows the system to maintain high throughput rates and provide exactly-once consistency guarantees at the same time.
* Designed to run on large-scale clusters with many thousands of nodes, and in addition to a standalone cluster mode.

## **Water-M Big Data Storage Module**

It is the component for data storage which is scalable to store the whole big data collected by Water-M Analytics platform and to make it available for analyses, executable as distributed, holds the data in reverse index files and based on nonrelational search engine. ElasticSearch is being used for retention of the big data produced in Water-M Analytics Platform,

Elasticsearch is an open-source, broadly-distributable, readily-scalable, enterprise-grade search engine. Accessible through an extensive and elaborate API, Elasticsearch can power extremely fast searches that support your data discovery applications. Basic concepts of the elasticsearch environment are describes below;

Near Realtime (NRT): Elasticsearch is a near real time search platform. What this means is there is a slight latency (normally one second) from the time you index a document until the time it becomes searchable.

Cluster: A cluster is a collection of one or more nodes (servers) that together holds your entire data and provides federated indexing and search capabilities across all nodes. A cluster is identified by a unique name.

Node: A node is a single server that is part of the cluster, stores data, and participates in the cluster’s indexing and search capabilities. A node can be configured to join a specific cluster by the cluster name.

Index: An index is a collection of documents that have somewhat similar characteristics. For example, we can have an index for raw water sensor data, another index for a prosssed data, and yet another index for analyzed data. An index is identified by a name (that must be all lowercase) and this name is used to refer to the index when performing indexing, search, update, and delete operations against the documents in it.

Type: A type used to be a logical category/partition of your index to allow you to store different types of documents in the same index, eg one type for sensors, another type for warnings.

Document: A document is a basic unit of information that can be indexed. For example, you can have a document for a single sensor data, another document for a single warning, and yet another for a single alarm. This document is expressed in JSON (JavaScript Object Notation) which is a ubiquitous internet data interchange format.

Shards & Replicas: An index can potentially store a large amount of data that can exceed the hardware limits of a single node. For example, a single index of a billion documents taking up 1TB of disk space may not fit on the disk of a single node or may be too slow to serve search requests from a single node alone. To solve this problem, Elasticsearch provides the ability to subdivide the index into multiple pieces called shards. Sharding is important for two primary reasons:

* It allows to horizontally split/scale the content volume
* It allows to distribute and parallelize operations across shards (potentially on multiple nodes) thus increasing performance/throughput

In a network/cloud environment where failures can be expected anytime, it is very useful and highly recommended to have a failover mechanism in case a shard/node somehow goes offline or disappears for whatever reason. To this end, Elasticsearch allows to make one or more copies of the index’s shards into what are called replica shards, or replicas for short. Replication is important for two primary reasons:

* It provides high availability in case a shard/node fails. For this reason, it is important to note that a replica shard is never allocated on the same node as the original/primary shard that it was copied from.
* It allows to scale out your search volume/throughput since searches can be executed on all replicas in parallel.

To summarize, each index can be split into multiple shards. An index can also be replicated zero (meaning no replicas) or more times. Once replicated, each index will have primary shards (the original shards that were replicated from) and replica shards (the copies of the primary shards). The number of shards and replicas can be defined per index at the time the index is created. By default, each index in Elasticsearch is allocated 5 primary shards and 1 replica which means that if you have at least two nodes in your cluster, your index will have 5 primary shards and another 5 replica shards (1 complete replica) for a total of 10 shards per index.

On the other hand, conventional SQL database managements systems aren't really designed for full-text searches, and they certainly don't perform well against loosely structured raw data that resides outside the database. On the same hardware, queries that would take more than 10 seconds using SQL will return results in under 10 milliseconds in Elasticsearch.

Elasticsearch provides Fast, Incisive Search against Large Volumes of data. A user expresses an ES query with a simple language, Query DSL. A query examines one or many target values, and scores each of the elements in the results according to how close they match the focus of the query. The query operators enable you to optimize simple or complex queries that often return results from large datasets in just a few milliseconds. The Elasticsearch design is much simpler and much leaner than a database constrained by schemas, tables, fields, rows, and columns.

During an indexing operation, Elasticsearch converts raw data such as log files or message files into internal documents and stores them in a basic data structure similar to a JSON object. Each document is a simple set of correlating keys and values: the keys are strings, and the values are one of numerous data types—strings, numbers, dates, or lists.

Adding documents to Elasticsearch is easy and it's easy to automate. Simply do an HTTP POST that transmits your document as a simple JSON object. Searches are also done with JSON: send your query in an HTTP GET with a JSON body. The RESTful API makes it easy to retrieve, submit, and verify data directly from a command line. Even if they are developing with a client such as Python or Ruby, many developers use the cURL tool for debugging and developing with Elasticsearch.

It's important to remember that Elasticsearch isn't a relational database, so DBMS concepts usually won't apply. The most important concept that you must set aside when coming over from conventional databases is normalization. Native Elasticsearch doesn’t permit joins or subqueries, so denormalizing your data is a essential.

ES will typically store a document once for each repository in which it resides. Although this is counterintuitive from the perspective of a conventional DBMS, it is optimal for Elasticsearch. Full text searches will be extremely fast because the documents are stored in close proximity to the corresponding metadata in the index. This design greatly reduces the number of data reads, and ES limits the index growth rate by keeping it compressed.

Elasticsearch can scale up to thousands of servers and accommodate petabytes of data. Its enormous capacity results directly from its elaborate, distributed architecture. And yet the ES user can be thankfully unaware of nearly all of the automation and complexity that supports this distributed design.

In Elasticsearch, these intensive operations occur automatically and imperceptibly:

* Partitioning documents across an arrangement of distinct shards (containers)
* In a multi-node cluster, distributing the documents to shards that resides across all of the nodes
* Balancing shards across all nodes in a cluster to evenly manage the indexing and search load
* With replication, duplicating each shard to provide data redundancy and failover
* Routing requests from any node in the cluster to specific nodes containing the specific data that you need
* Seamlessly adding and integrating new nodes as you find the need to increase the size of the cluster
* Redistributing shards to automatically recover from the loss of a node

## **Water-M Visualization Service**

It is the visualization and analysis component which provides to search on the data stored in Water-M Analytics platform, to determine the results of analysis, to determine the data with the charts, to create dashboards and reports and these whole actions to be processed dinamically by users. Apache Kibana (<https://www.elastic.co/products/kibana>) has been used for visualization layer on the Water-M project.

Kibana is an open source analytics and visualization platform designed to work with Elasticsearch. Kibana can be used to search, view, and interact with data stored in Elasticsearch indices. It can be easily performed advanced data analysing and visualizing data in a variety of charts, tables, and maps.

Kibana makes it easy to understand large volumes of data. Its simple, browser-based interface enables users to quickly create and share dynamic dashboards that display changes to Elasticsearch queries in real time.

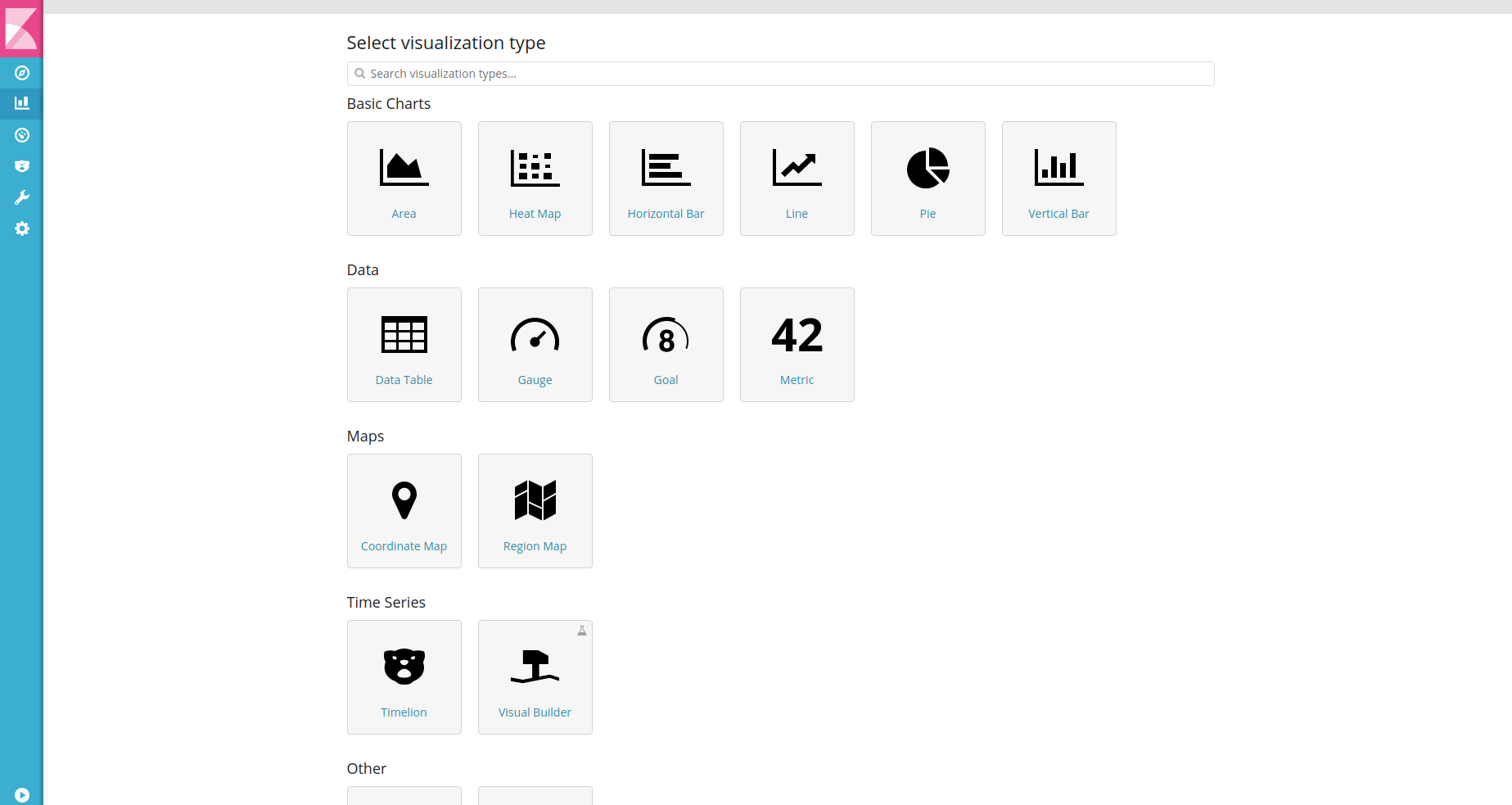
Water-M Big Data Analysis Platform's Visualization Service component provides the abilities below:

* It is possible to search, to display and to interact with the stored data in Water-M Big Data Storage via this component,
* It includes the enhanced data analysis instruments to visualise the data in various graphics, charts and maps.
* It enables the dinamic indicator boards which are able to display the revisions in index queries; to be formed and shared summarily via its interface on simple browser basis.
* It displays the search results' numbers and records detailedly.
* The temporal distribution of the records provided from the search results is able to be filtered in time by being displayed in a histogram bar.
* Relative time intervals are able to be descripted such as certain range between now and 30 min. 25 sec. before.
* Renewal intervals are able to be chosen to update the data on the visualisation interface. For example; updating the data in every 5 seconds.
* Search queries can be saved, then be reopened and also the searchings saved in the visualisation process can be used.
* Dinamic filters, such as is, is not, is one of, is not one of, is between, is not between, exsist, does not exsist can be used on search screen.
* It provides visualisation instruments in which the data can be analyzed by creating dinamic grap-hics on the selected data.
* It provides various graphics such as area, line, bar, pie, heat map, data board, data table, word cloud, time series etc. can be formed dinamically.
* It can create marks on the map bases for the areas which includes spatial information and also can form heat maps.
* Visualization can be used on bar and pie charts with the subrefractions. For example; first five message types produced in a management and first three devices producing these messages can appear on the same chart.
* Time sequence analyses and charts can be formed by using the data of distinct data resources.
* All of the images can be saved to be used for creating the dashboard later on.
* Dinamic dashboards can be formed by using the saved charts.
* The location and size of the dashboards formed before, can be personalized with drag-drop.
* Dinamic filtrations can be formed on dashboards by clicking on the related parts. For example in an ABC Business, the types of messages can be displayed immediately by clicking on the device which produce most number of messages.
* All graphics are automatically updated when any filter is applied from Dashboard.
* It provides to make analysis that based on Machine Learning techniques.
* It detect the abnormal data changes via applying the anomaly detection methods. With this way, system can warn the user for sensors which produce wrong data.
* It provides the user management and security services
* It provides the automated report system for all created analysis and gives the ability to export these report in many different format.

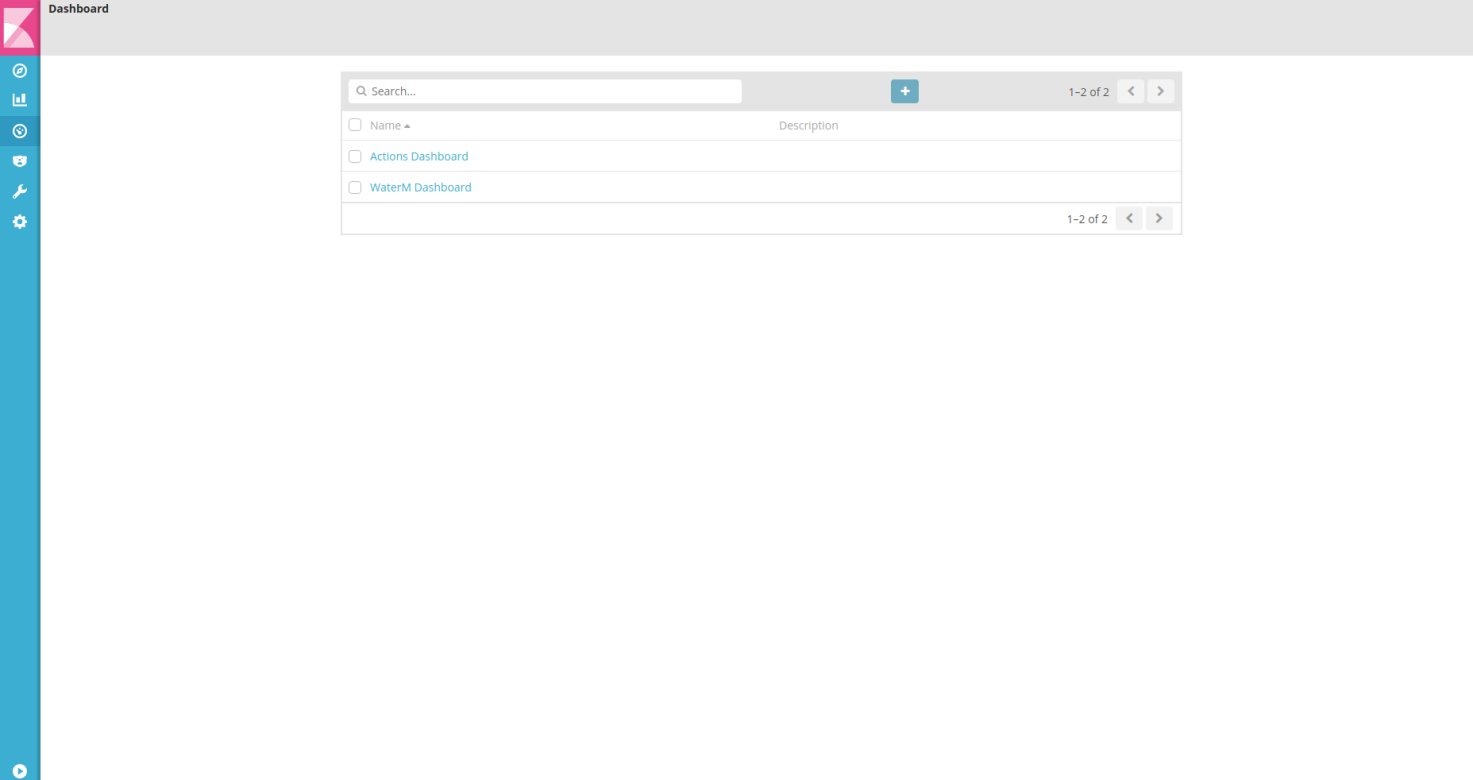
# **Screenshots From WaterM Analytics Patform**



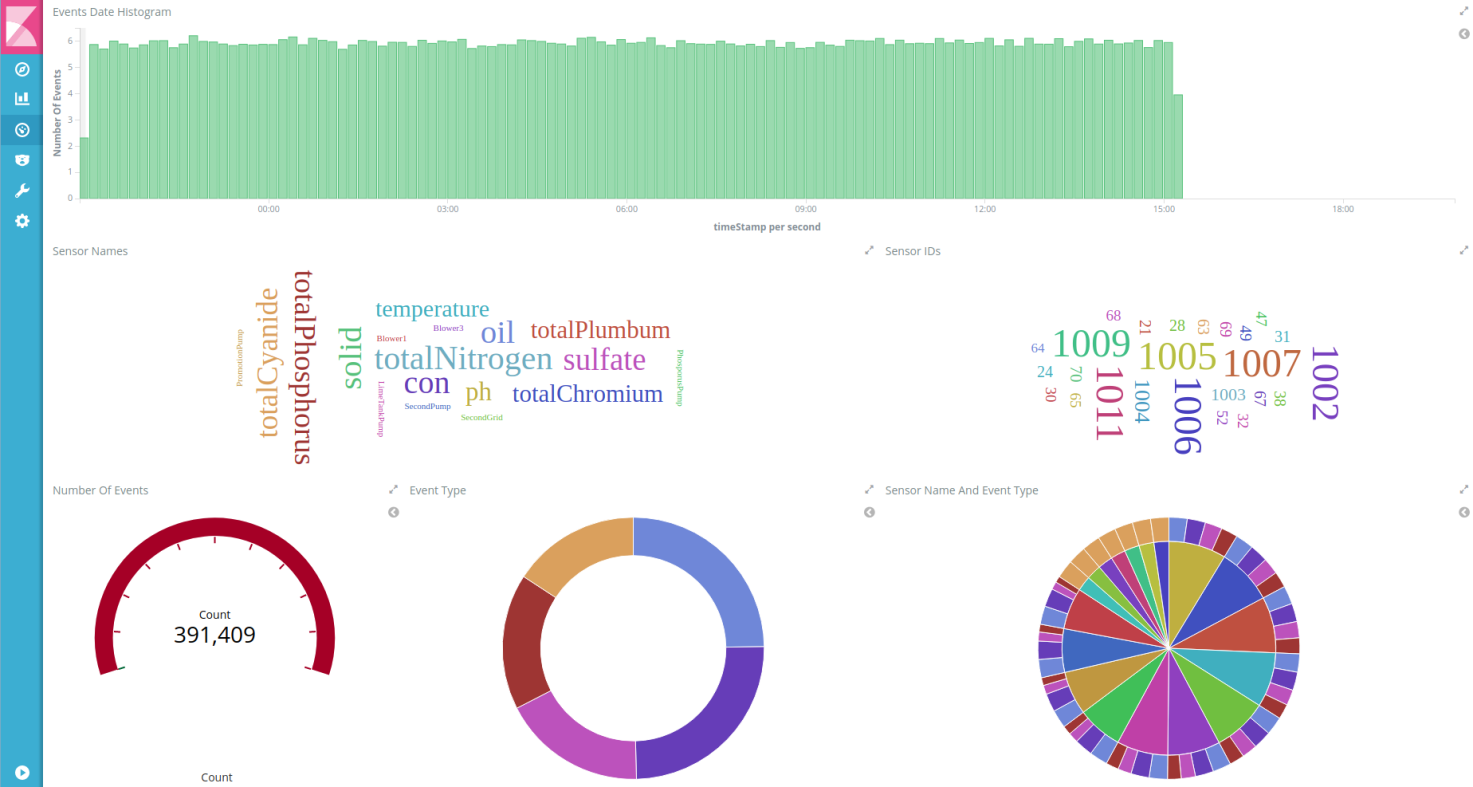
* Search Results For Sensor Events



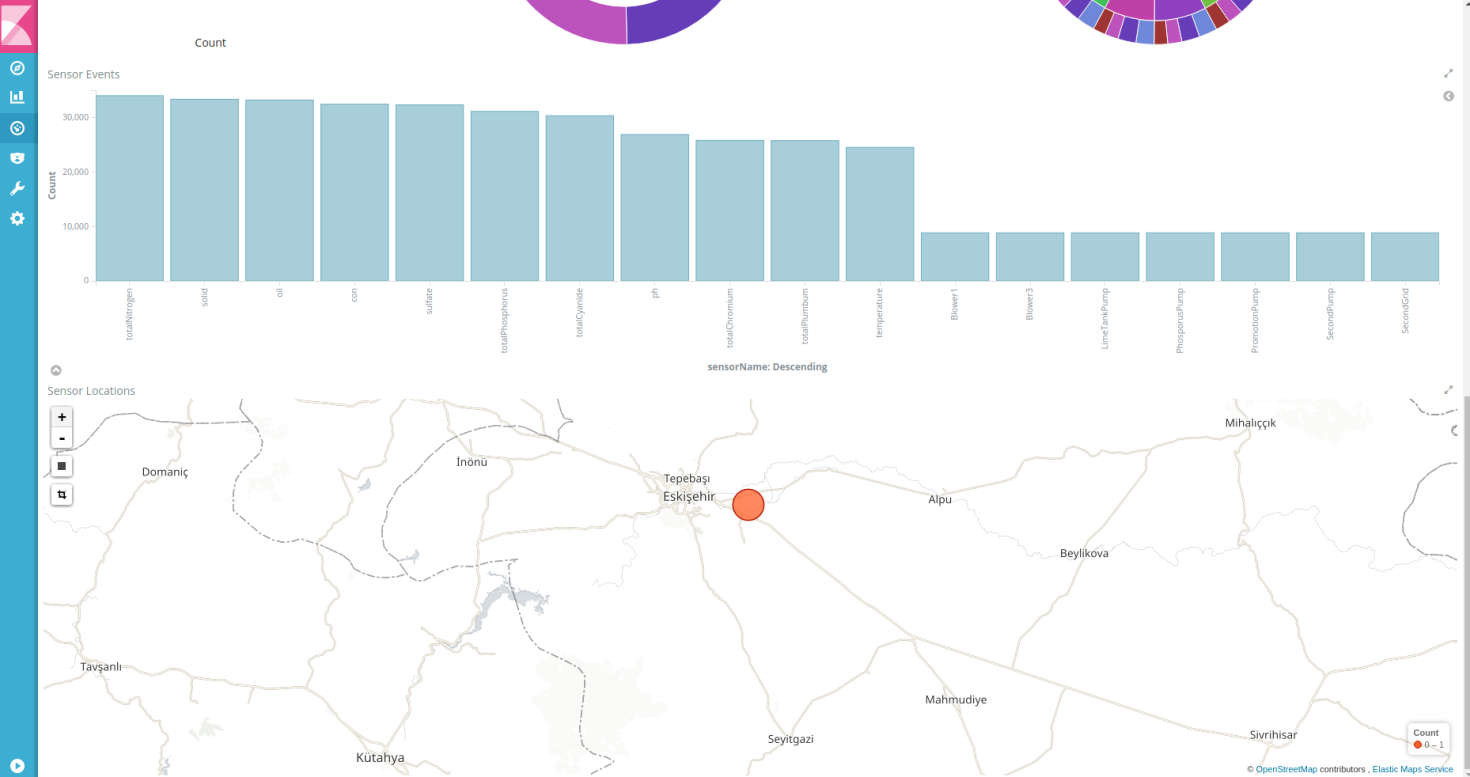
* Creating Visualization From Sensor Data



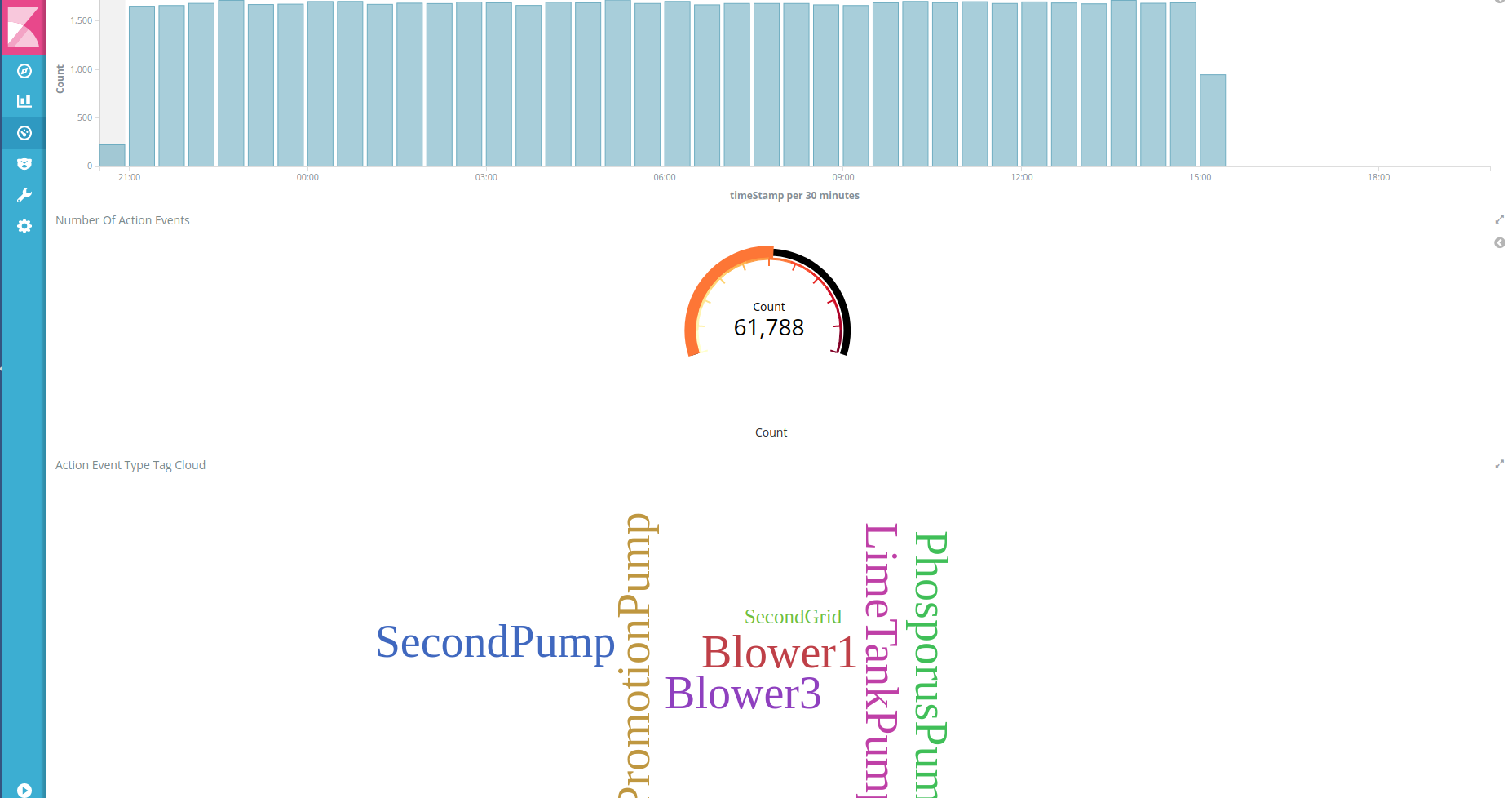
* Stored Dashboards For Water-M Analytics



* Dashboard Graphs For Analysed Results



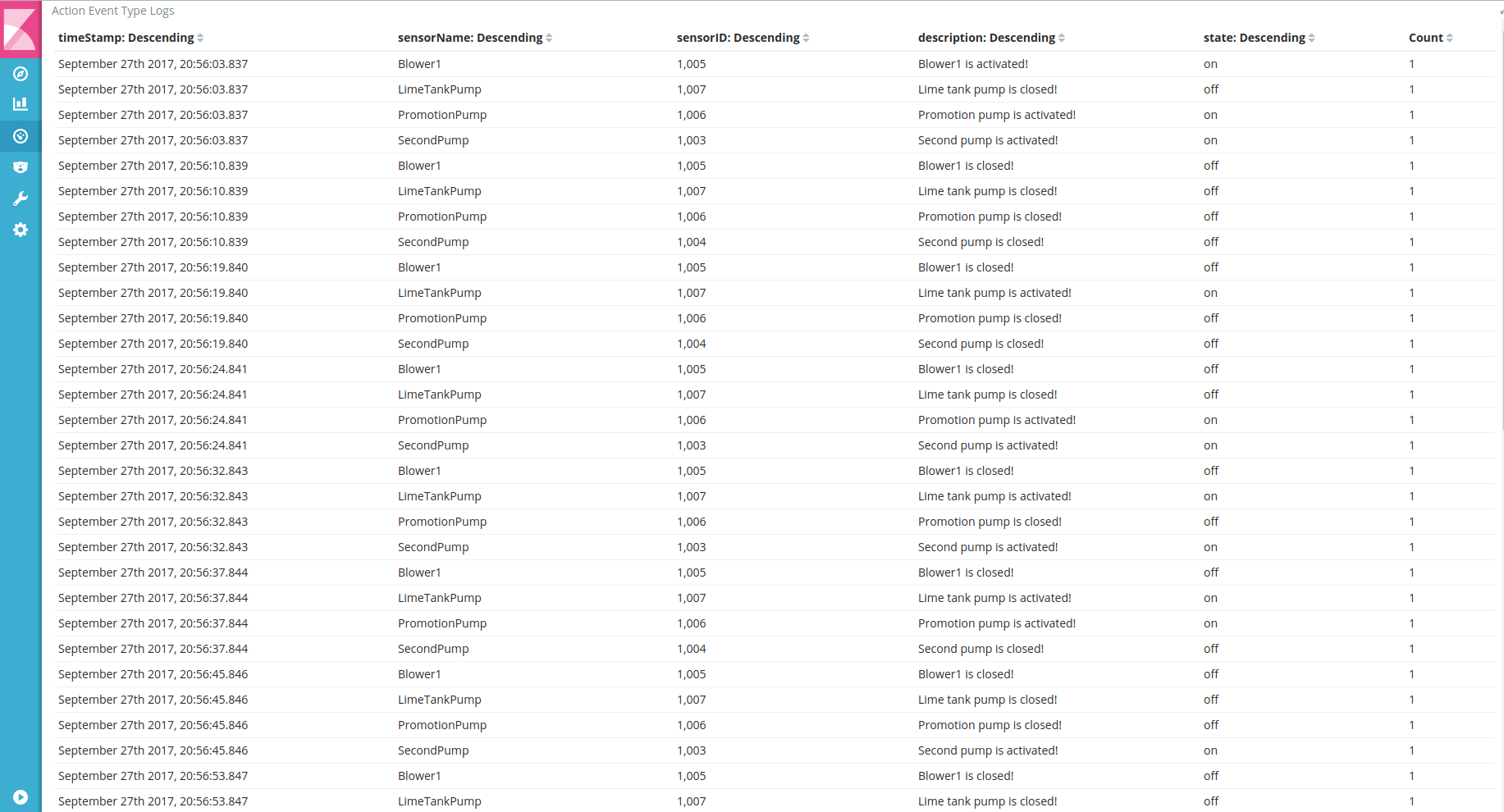
* Dashboard Graphs For Analysed Results



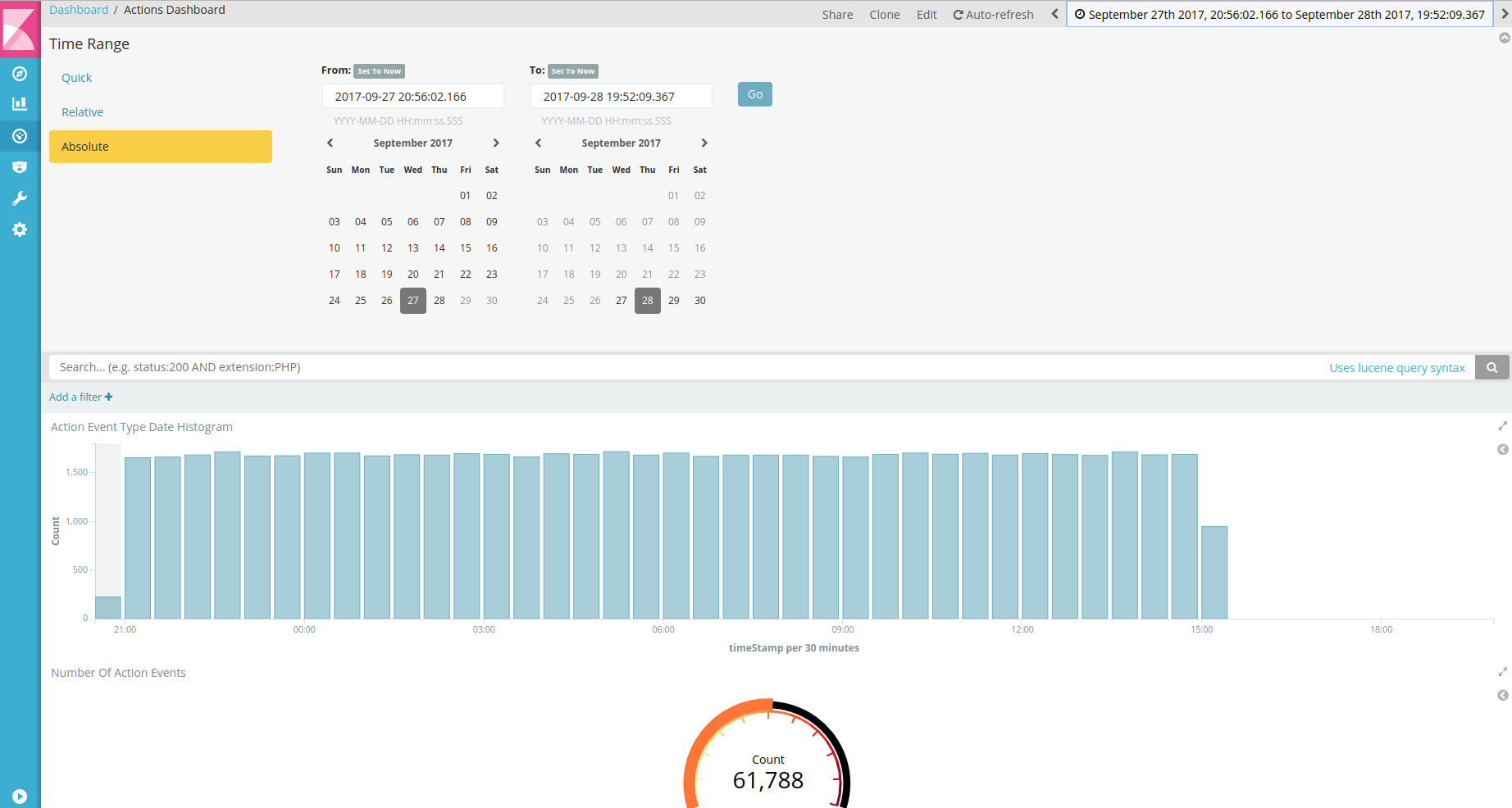
* Dashboard Graphs For Analysed Results



* Dashboard Graphs For Analysed Results



* Data Table For Analysed Results



* Select Date/Time Range Screen

# **Conclusion**

In this report Water-M Big Data Analytics Platform functional architecture and its software components have been explained.