EasyEvent: An Environment Monitoring System

Requirements Analysis

The architecture and system design depends on some factors that I consider critical. I consider this factors as questions:

1) Does the small servers will poll for data or data will come asynchronously from sensors, or sensors will sent data periodically to small servers? This fact is important because of the nature of the events.

2) What kind of events does the monitoring service will receive? Are there critical events that require an urgent processing or all events will have the same priority?

3) How much events can be generated per second by each sensor?

4) How much events can be generated by each device?

5) What kind of information does each event contains?

6) How much time does the event have to be persisted? For example, after receiving an event, how much time in should be in database: one month, one year, one day?

7) What time intervals are critical for collecting data? It means, if the systems misses data for an hour, what happens?

8) What kind of application can be run on sensors? Does sensors has an operating system that can run high level languages? Does sensors have a constrained software environment, it means, they don't have an operating system available and just have a basic embedded firmware?

9) What connectivity does sensors have with small servers? Does communication can be bi-directional?

There are different kind of systems and events associated to them, it means, different kind of information that can be monitored. Some system examples could be:

System Description	Requires real time?
Light intensity and temperature monitoring on a natural forest reserve.	In case of a fire, an alarm should be sent almost in real time. Most of information doesn't have to be registered online.
Earthquake detection system.	In case of alarm of possible earthquake, information must be available as soon as possible. Periodic recordings can be persisted on batch on a periodic basis, for example, every 2 hours and so on.

Signal quality on a zone.	Not necessary to send real time information. Batch information representing a period of time can be fetched.
Intruder detection system.	In case of an intrusion situation, real time alarms are necessary. This system is alarm oriented, only intrusion events are of interest.
Tsunami detection system.	In case of Tsunami alarm, real time alarm must be generated.
Monitoring of water consumption.	Not necessary to send real time information. Batch information representing a period of time can be fetched.
Cosmic radiation variation.	Not necessary to send real time information. Batch information representing a period of time can be fetched.
Vehicle Fleet monitoring system	Monitors vehicles GPS position. Position should be reported online. Some events like panic button pressed need real time.
Number of sales done per salesman.	Important to send online information, not critical to send real time. Batch reports could be sent on time intervals, for example, 1 hour and so on.

The system that I will design here, **EasyEvent**, should be enough flexible to support the types of systems listed previously, because those systems could be inside the category “Environment Monitoring Service”.

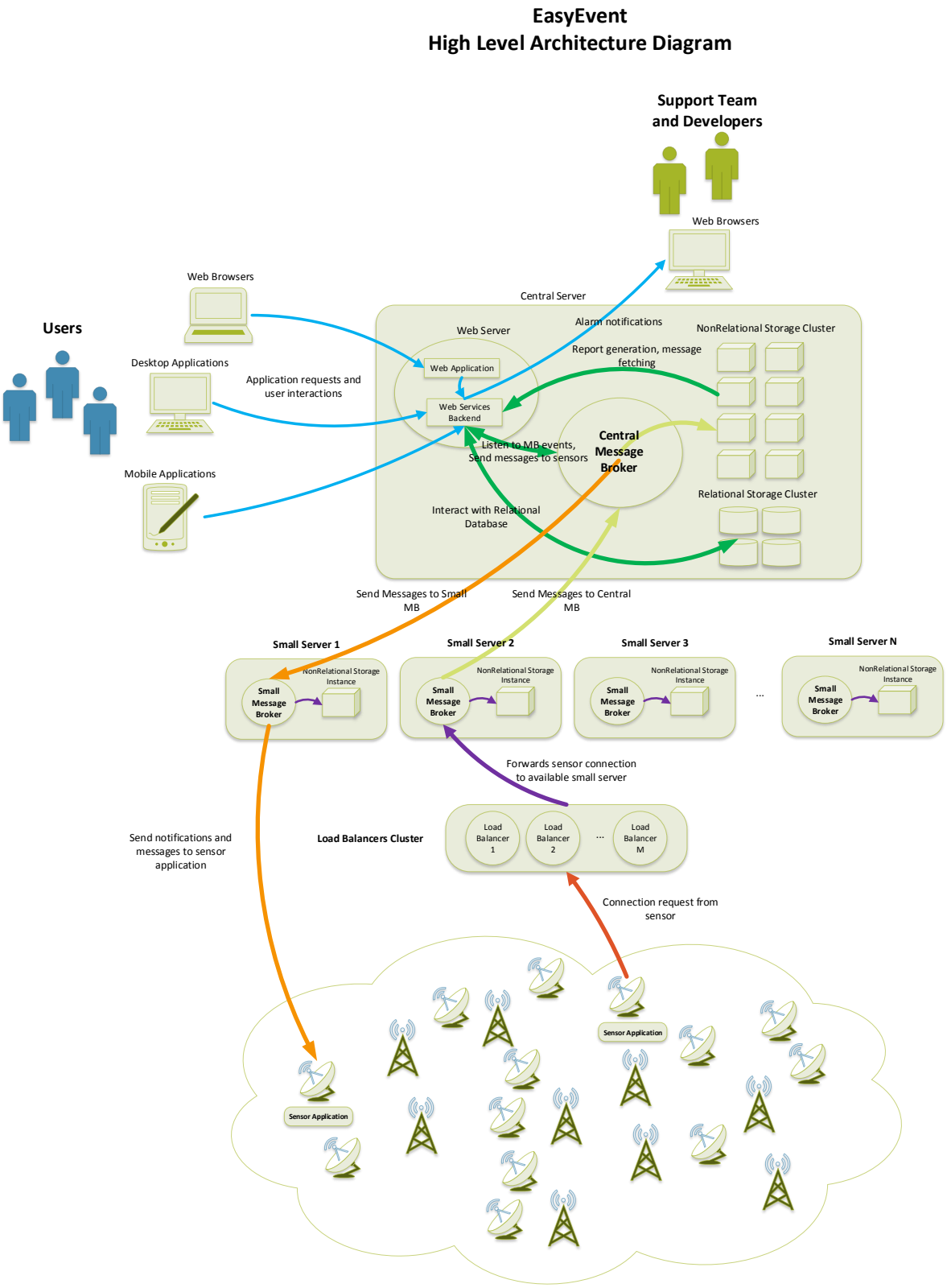
I will use a multi message broker architecture based on asynchronous hi-priority messages, using topic/subscribe model for notifications and a master-slave polling for fetching low-priority messages.

In the next pages, I will describe **EasyEvent** architecture by describing the following topics:

1) Brief system modeling.
2) Describe system architecture.
3) List software modules that compose system architecture. How and in which order would you implement software modules?
4) Highlight critical points.

1. BRIEF SYSTEM MODELING

The following picture shows the **EasyEvent** high level architecture design:



Document written by Juan Sebastian Lopez Villa, Software Developer.
Exclusive design for EasySolutions recruitment process. Jun/22/2016

2. DESCRIBE SYSTEM ARCHITECTURE

We can visualize the system on several layers. Each layer has associated software components:

Sensors Layer

- **Sensors application.** The hardware is ready but software modules may be pending. First check what kind of communication can be performed by sensors such as TCP/UDP sockets, subscribe to a topic using Message Queue library, Web Service (HTTP) request, etc.

Load Balancers Layer

- **Deploy configuration.** It's not necessary a software module, but a configuration and deployment module.

Small Servers Layer

- **Small Message Broker application.** The message broker should contain a handler for each message received from sensors and from Central Message Broker. Also, it should implement a driver or connector for persisting messages on Non-Relational Storage Instance.

Central Server Layer

- **Central Message Broker.** Contains handlers for messages received from Small Message Brokers. Contains automatic process for fetching data. Contains drivers and connectors for persisting messages on Non-Relational Storage Cluster.
- **Non-Relational Storage Cluster.** More than software, contains deployment configurations. The cluster can be on front a distributed file system like HDFS of Apache Hadoop. The No-SQL could be MongoDB or Cassandra.
- **Web Services Backend.** Contains web services interfaces for interaction with client applications like Web Application, Mobile Application and Desktop Application. Contains interfaces for report generation.
- **Relational Storage Cluster.** More than software, contains deployment configurations. Several database servers can be configured in a cluster for scalability, performance and fault tolerance.

User Application Layer

- **Web Application.** Contains HTML/CSS view and Javascript client code for browser application. Client code interacts with web services backend. Deployed on web server.
- **Desktop Application.** Contains binary desktop application for interaction with web services backend.
- **Mobile Application.** Contains mobile application for interaction with web services backend.

3. LIST SOFTWARE MODULES THAT COMPOSE SYSTEM ARCHITECTURE. HOW AND IN WHICH ORDER WOULD YOU IMPLEMENT SOFTWARE MODULES?

The following are the software modules that I can identify on **EasyEvent** system design. The order in which they are listed is the order in which I would implement them.

1. **Non-Relational and Relational Entities (Database Design):** Create model design. Define which entities will be persisted on Non-Relational Storage and which ones in Relational Storage. At a glance, messages will be persisted on Non-Relational storage like MongoDB or Cassandra. Users, sensor information like coordinates, firmware version, calibration parameters, small server information, would be persisted on Relational Storage like MySQL/MariaDB or PostgreSQL.

2. **Object Oriented (Class and Interfaces) Design:** Define classes and interfaces. Some possible objects would be Message, Sensor, SensorConfig, User. Some possible interfaces would be MessageSender, MessageReceiver, MessageFetcher.

3. **Message Protocol:** Design message protocol (if the hardware is provided but no message or data protocol is provided, message protocol is critical)**.** Communication protocol specification and implementation.

4. **Sensor Message Sender and Receiver:** (if the hardware is provided but no application for sending messages, this would be fundamental). Code for sending information encapsulated in message protocol.

5. **Small Message Broker In/Out coming Message Handler:** Send messages to sensor applications. Receive messages from sensor application. Send messages to central Message Broker. Receive messages from Central Message Broker. All message sends and receives would be using Publish and Subscribe model.

6. **Small Message Broker Connector to Non-Relational Storage:** Driver for Non-Relational storage. CRUD operations on storage would be implemented.

7. **Small Message Broker State Monitor:** Automatic process that would check connection status with Central Server, number of connected sensors, storage usage.

8. **Central Message Broker In/Out Message Handler:** Receive messages from Small Message Brokers. Send Messages to Small Message Brokers.

9. **Central Message Broker Connector to Non-Relation Storage:** Driver for Non-Relational storage. CRUD operations implementation.

10. **Central Message Broker Message Fetcher:** Automatic process for retrieving batch information on a periodic basis. Implementation would be a scheduler with different tasks programmed.

11. **Central Message Broker State Monitor:** Check storage capacity. Check connections status of Small Server Brokers.

12. **Web Services Backend HTTP interfaces:** CRUD operations implemented as RESTful web services using GET, POST, PUT, DELETE HTTP verbs.

13. **Web Services Backend Listener to Central Message Broker Events:** Interface for receiving notifications from Message Broker. Listener for events generated by message of high priority such as alarms. Could be implemented as an event aggregator pattern.

14. **Web Services Backend Connector to Central Message Broker:** Interface for sending messages or notifications to sensor applications.

15. **Web Services Backend Connector to Non-Relational Storage:** Driver for CRUD operations on Non-Relational storage.

16. **Web Services Backend Connector to Relational Storage:** Driver for CRUD operations on Relational storage.

17. **Web Services Backend Report Generator:** Contains business logic for report generation and integrates all services for storage operations.

18. **Desktop Client Application:** RESTful client for web service backend. User interface for interaction.

19. **Mobile Client Application:** RESTful client for web service backend. User interface for interaction.

20. **Web Client Application:** RESTful client for web service backend. User interface for interaction.

4. HIGHLIGHT CRITICAL POINTS.

What happens if a small server goes down?

All messages stored in small database won't be lost, but sensors just will connect again to load balancer cluster and will be forwarded to other available small server. Central server will notice that the small server went down and will schedule a special fetch procedure after small server goes up. If small server doesn't goes app, will generate an alarm to support team for checking small server.

What happens if central server goes down?

Downtime should be minimal or zero, but can happen. In that case, if it's a scheduled maintenance, a message can be send to queue notifying small server subscribers. This way, small servers will wait for the end of maintenance message to be able to send alarms. There should be an emergency queue for critical systems in case of central server goes down.

What happens if a small server reaches its maximum capacity of storage?

Small server will stop receiving more messages and connection for sensors will be closed, this way, sensor will connect again to load balancer cluster and will be forwarded to an available small server.

What happens if central server reaches its maximum capacity of storage?

An alarm will be generated to support team when the capacity be near the limit. If the storage reaches the limit, all data will be rolled to make new space. On critical systems, a backup warehouse should exist for avoiding data loss.

What happens if central server reaches its maximum capacity of processing?

An alarm will be generated to support team so that more small servers get activated. Ideally, a maximum number of load should be estimated, nevertheless, sometimes load could exceed expectations causing problems. Load testing should perform in order to measure the real processing limits of central server.

What happens if a small server reaches its maximum capacity of processing?

Small server will stop receiving more additional sensors connections. Load balancers cluster should know how much is the load of each small server and balance it.

How does this system can scale?

If more sensors are added, just add more small servers. Each small server should subscribe its message broker to central server message broker and should be added to load balancer cluster's list of nodes. Central storage should be deployed on a distributed file system like Apache Hadoop (HDFS) and NoSQL databases like a MongoDB cluster could be used for persisting messages. A relational database like MySQL/MariaDB could be used to store reports and data mining results, user information, small servers and sensors profiles and configurations. Having a distributed file system, scalability of storage can be executed easier just adding more storage nodes.