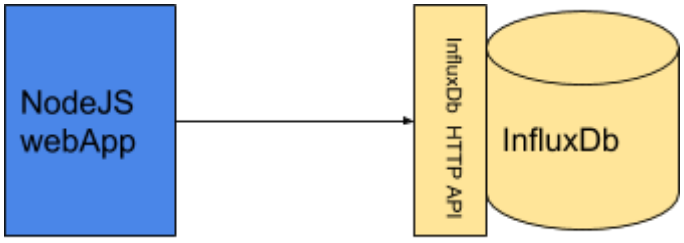
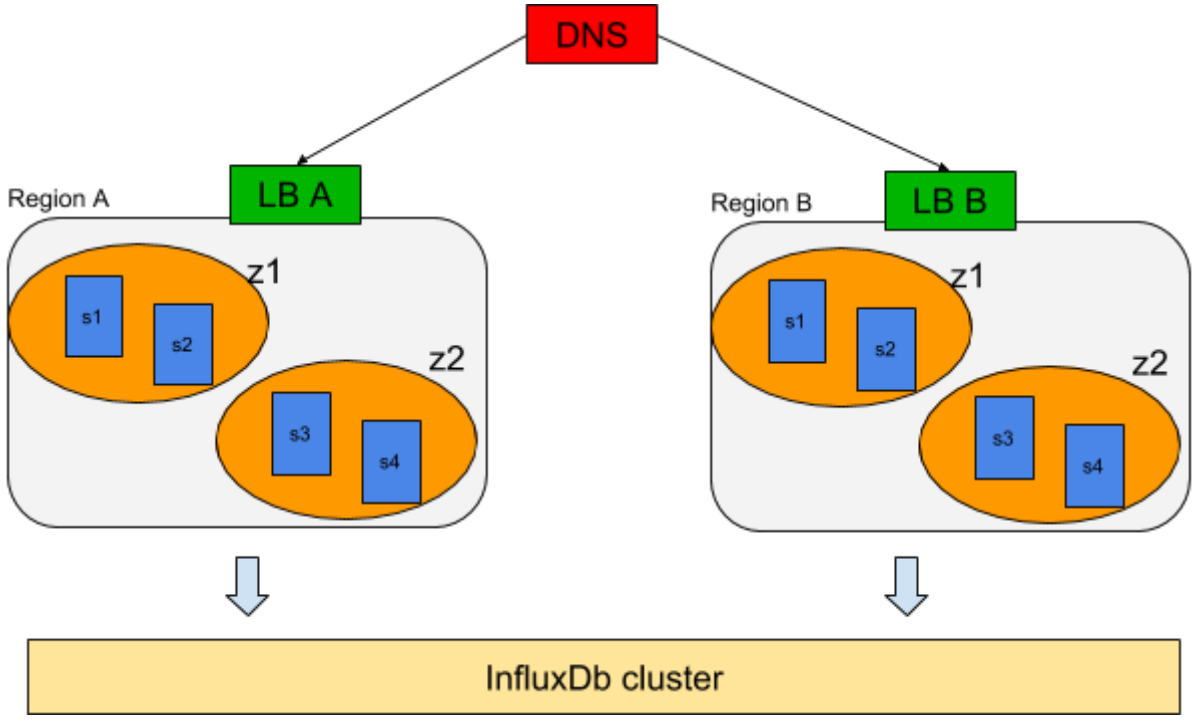
Design of the proposed solution

The design of the web application is made by two main components: the application server and the database layer. Since the application goal is to store the timestamp of each POST request into a database, I decided to use “InfluxDB” as the database technology. It is able to store whatever kind of data, associated to the timestamp in which that data is added to the db. In this way I avoided the need to synchronize the application servers on the timestamp (for example using NTPD with server pools) and I received for free the timestamp storing of the event. Indeed each POST request made to the application server (a nodeJS web application) calls a write API on the InfluxDB node that write a “POST request event” into the database within the related timestamp.

The architecture of the proof of concept is the following:



Since the real scenario involve a geographical distribution of the web app, using multiple datacenters and zones, the target architecture must take into considerations other components, like DNS and load balancer. The following is the final architecture:



With such an architecture we support mutiDC, because the DNS takes care of the user IP address and resolve the app hostname with the nearest region entry point. The entry point is the Load Balancer of a region. It is configured to route the received request to one of the available node in the zones of the region and its selection policy can be customized.
The zones are isolated set of nodes that can be configured to respect some conditions. For example a zone can be configured to have at least 30% free memory of total memory. If it's not the case a scale up or scale out of some nodes can be accomplished. It can also be set a scale down trigger if the free memory is too much and we want to save resources and energy from our cluster (i.e. money). All these configurations can be set on the components that will monitor and manage the nodes in the cluster. In that way the architecture will be elastic on demand.
The scalability involve an automation in preparing and deploying the nodes. Moreover it is important to establish a level of "indirection" between the different nodes and components. This way there's no need to have a "a priori" knowledge of the architecture, so that it can be dynamic to respond to the workload's changes.
Each node of the web app cluster must be launched with an autoconfiguration task that install the required packages, load the application and the configuration of the environment (in the proof of concept it has been used a mixture of shell script and Ansible playbooks together with ansible-pull, to create a pull model update automatically run by the node at the startup instead of loading a new configuration manually on each node).
The networking also must be automated, creating link between components.
Finally a monitoring service must be set up to check the availability and reachability of the nodes to ensure the user requests can be correctly forwarded to the destination. In case it's not possible, a system of alert must be set to trigger a restore process an reestablish a working architecture.
The database layer, in this architecture, has been designed as a single box because InfluxDb offer a cluster runtime able to support a distributed deploy. Since the webapp can be accessed from all over the world the data written in it must be replicated on all the instances. The assumption I made on the task is that the application is mainly used to write data than to read them, so, in the case of InfluxDb, the replication can be accomplished using a quorum of write nodes equal to 1, obtaining a lower response time on write operations and improving the overall performances.

The depicted scenario could be sometime affected by burst of web requests. If it's the case, an event based message queue (like Apache Kafka) could be used to collect the events. In this way no user would have his request being refused by the application server, and the availability would be granted, resulting in a good solution to the ingesting problem of large data.