# Neato SmartApps Backend Deployment for Scalability

Raja Software

Last updated: 04/02/2014

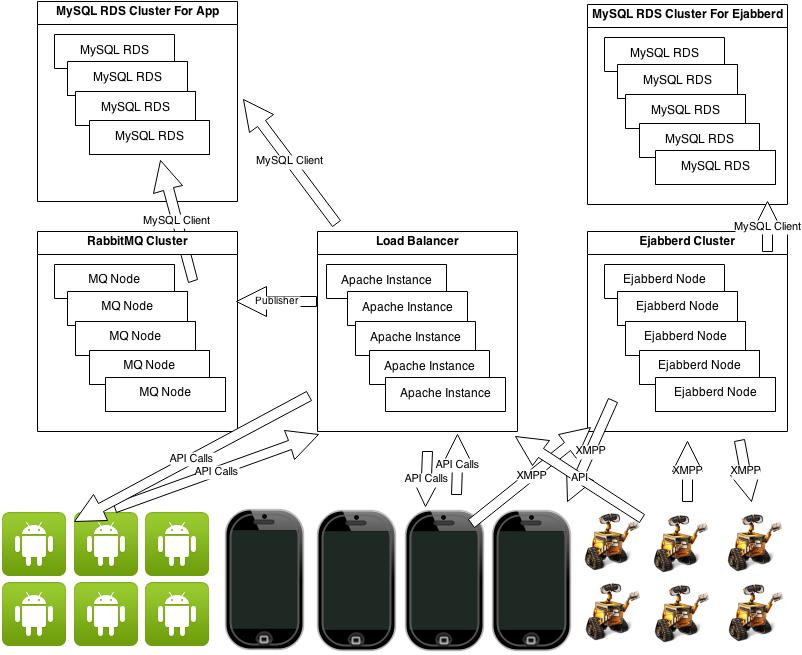
|  |  |  |
| --- | --- | --- |
| Date | Version | Author/Comments |
| 04/01/2014 | 0.1 | Initial draft |
| 04/02/2014 | 0.2 | Adding topology diagram and more information in the performance testing section. |

# Summary

This document summarizes the proposed Neato SmartApps backend deployment topology to ensure scalability. It also briefly includes the performance testing that was done Aug/Sep 2013.

# Proposed Topology

This section goes over all the components in the system and the tunings that we would be doing at each level to ensure that the system is scalable. We would be deploying the application on the AWS EC2 cloud. The diagram below shows the proposed topology and section below explains the components involved.



# Apache

There would be multiple EC2 instances that would be running PHP, MySQL Client, Apache, Ejabberdctl and AMQP publisher. Each of these instances would be running CentOS operating system. Each of these instances would be medium instance (<http://aws.amazon.com/ec2/instance-types/>).

MySQL client would be required to connect to the MySQL RDS.

Ejabberdctl is command line admin tool that is used by the backend to create Ejabberd accounts and sending messages. Ejabberdctl accepts node name as parameter and it would be executing exact same commands with node name, on an Ejabberd cluster that would be located on another machine.

Similarly, AMQP publisher that is used to publish the notifications on RabbitMQ would be connecting to RabbitMQ, which is running on separate instance.

Few other tweaks that we would be doing on each of these instances are:

**Apache Modules**

We would be enabling GZip compression modules so that the returned responses are compressed.

We would also be enabling header and expires module that would ensure that the JS/CSS files that are used by the support backend are cached by the browsers and the Apache calls would cut down.

**APC Cache**

APC (Alternative PHP Cache) is an opcode cache that significantly speeds up PHP applications, by caching both PHP code and user variables. We would be enabling the APC cache on each instance.

**Yii**

We would be running the Yii framework in the YiiLite configuration which supports the APC cache integration out of the box. Also we would be setting the logging levels in the Yii to only log severe errors and set environment to production to reduce logging.

# Load Balancer

AWS gives an easy way to configure load balancer where in we can add multiple nodes that are running Apache. We would be adding multiple Apache nodes depending upon the traffic. An AMI would be kept so that we can bring more instances up as and when the traffic goes up. All the Apache instances would be then added in this load balancer.

# MySQL

Instead of installing the MySQL on an instance, we would be using AWS MySQL RDS (<http://aws.amazon.com/rds/>). Note that bringing in RDS would NOT require any change in the MySQL interaction. RDS gives easy way to enable MySQL clustering that also automatically shards (partitions) tables across the nodes, enabling databases to scale horizontally. With this clustering, we can scale exponentially with each RDS node in the cluster.

# Ejabberd

We would be moving the Ejabberd to its own instance. This would be again a medium instance with CentOS as operating system. It would have Erlang, Ejabberd and MySQL client running. We would be doing following changes in the existing implementation:

Ejabberd stores the data in the Mnesia DB but has easy way to configure it with other DB. We would be using MySQL as Ejabberd database by setting following parameters in the ejabberd.cfg file. The MySQL that would be used by the Ejabberd instance would be again a separate RDS instance.

{odbc\_server, {mysql, "server", "database", "username", "password"}}.

Additionally, we would be enabling the Ejabberd cluster.

* All the nodes in the cluster should have the ability to connect to port 4369 of all another nodes. We would have to set the security group settings on these EC2 instance to support it.
* All the nodes must use same magic cookie (~ejabberd/.erlang.cookie must be the same on all nodes). This is needed because all nodes exchange information about connected users.
* In the current implementation, other than the default Ejabberd module only 1 additional module is added which is mod\_eventful. In order to make it work in cluster, we would need to have Router, Local router, Session manager and S2S manager modules installed on each of these nodes.

# Application Tuning

We would need to change following parameters in the application to enable this topology.

* DB connection strings are configurable from the config file and we would need to point it to the RDS instance.
* Ejabberd host again is configurable but there is no parameter available for nodename, We would require this parameter while interacting with the Ejabberd cluster. This parameter needs to be added in the configuration file and used in the application.
* Similarly for RabbitMQ publisher, we would need to start accepting node name for the RabbitMQ cluster and the Queue to which we are publishing the data.
* We have successfully tried the Load balancer, Ejabberd clustering and Ejabberd with MySQL as backend database.
* We have also tuned the memory allocation for Apache and PHP.
* We also ran the Yii in the YiiLite configuration.
* We have NOT tried RabbitMQ clustering yet.

# Performance Testing

We have used JMeter to test various scenarios and garnered numbers.

Few things to note while reading these numbers are,

* These tests were performed in the month of August 2013 and since then there have been significant changes in the application structure. For example, the XMPP messages were not routed through RabbitMQ that time and SetRobotProfileData3() method was not sending XMPP messages.
* All these tests were performed with Timed Mode on.
* While running these test cases, we were hitting the test cases from 3 different machines.
* We were interested to know if the servers crashed and did not measure the latency because of the increase in the traffic.
* We ran these test cases on the AWS EC2 instances with RDS backend as explained above.

The most common scenario with the timed mode on, we tested this specific scenario. This JMeter test, does following 3 things:

Ping call to server

Ejabberd login

Ejabberd log out

15K calls were sent from 3 machines simultaneously using JMeter.

Please note that in this case, following 3 calls are going to the Apache server:

* Ping call
* Because of the Login, another call goes to Apache from the Ejabberd call back function.
* And when the ejabberd log off happens, another call goes to Apache from the Ejabberd call back function.

Apache instance never goes beyond 40% usage.

Logins are happening directly on the Ejabberd instance that already has the logins created. It is not failing but the memory consumption reaches 45%.

Overall, we were able to sustain 50K simultaneous HTTP calls, with:

* Load balancer with 2 medium instances in it
* Each of these instances were interacting with 1 RDS instance
* 1 EC2 medium instance was used as Ejabberd node. Ejabberd accounts were created beforehand, hence we were only connecting to Ejabberd server but were NOT actually creating the Ejabberd accounts.

Similarly, on Ejabberd, we could see the 30K simultaneous connections and the memory usage was not going above 45%.

End.