Cloud Computing Deployment of a Web Application Requiring Dynamic Infrastructure Scaling - A Case Study

Introduction

Our client is a leading test preparation institute targeting the multi-billion dollar training and test preparation market. Their training and test prep courses prepare students for the extremely competitive landscape of admissions into the top educational institutions worldwide. Traditionally, the client has used classroom based training curriculums for their courses, but recently decided to start a web-based offering to address the growing market for online test preparation. They planned to supplement the classroom based teaching by delivering their test preparation tools and training through a web and smart-client based application which would allow them to reach a much larger number of students.

Nagarro was contracted to build, host and maintain their online training and test preparation tools, and to establish overall technology strategies for their online presence. Nagarro built the initial online training and test preparation system with state-of-the-art test taking and reporting capabilities, and a student website containing collaborative features like blogs and forums. Very soon, we realized that the load on this kind of a web application would vary greatly over time. Just like news organizations have to worry about handling heavy loads in case of a big event, the number of concurrent users logged on to the website goes up by an order of magnitude near an examination or just after the results of one of the exams are announced. Because of the nature of the service provided, it is especially important for the web application to stay responsive at these times of very heavy load. Users (students) log on to the application at this time because they really need to!

Since this was the first online experience for our client, even the typical load and average number of expected users was not known. Additionally, our client planned to extend its online training and test preparation system by bringing in content from their partners, which would bring in their own training and test preparation content and content consumers (their students). These partnerships, while being good for business translated into tight deadlines for engineers, specially for IT and infrastructure engineers who had to quickly scale up (and down if possible) the IT infrastructure to meet changing business needs.

Due to vastly varying performance requirements and changing business needs, a traditional approach to provisioning and managing IT infrastructure would have ended up in us vastly over-provisioning hardware resources. We realized that we needed an elastic, utility computation model which could be dynamically scaled up or down to meet changing business realities.

In this document we discuss how Nagarro leveraged Amazon's Elastic Compute Cloud to deploy the clients web application, which could be dynamically scaled up within a matter of minutes and not days, weeks or months, to support a sudden (anticipated or unanticipated) increase in user community. The solution significantly reduces infrastructure cost by removing unused nodes (and hence the associated cost) by paying for computation only when required, while at the same time increasing application scalability, availability and security.

Section 2 of this document discusses the problem in greater detail followed by a discussion of possible solutions in section 3. Section 4 introduces EC2. Section 5 has a short description of what the application looked like before it was moved to EC2. Section 6 discusses different configuration and technology details which enabled implementation. This section has snippets of different configuration files used for the Apache HTTP server and Tomcat servers. Section 7 discusses the final process view of the web application after it was deployed on EC2. Section 8 discusses known limitations of our approach and suggests a few alternative approaches.

1. The problem

In the process of enabling the web presence of it's client through their consumer facing website, Nagarro identified following areas of concern:

Uneven load:

The load on a website such as this can vary greatly over time - it can

increase by an order of magnitude just before an important exam or just after the results of an exam are announced. Hence, the infrastructure needed to support sufficient computing power to handle the peak load. A traditional approach would have resulted in over-provisioning of hardware to meet peak demands; hardware which would have remained under-utilized most of the time. This over-provisioning would have resulted in increased equipment, energy, maintenance and management costs.

Emergent business:

Since this was the first online undertaking by this company, even the typical load and average number of users was not known. More importantly, it was difficult to predict the load a few days after a major advertisement campaign was launched, or a few months into the future as our client entered into partnership with other content providers to start serving their user-base. We needed to be able to setup an infrastructure which could quickly respond to changing business needs.

Management of IT infrastructure:

Managing hardware infrastructure involves managing servers, routers, switches, power supplies, cooling - pretty much everything. We needed the ability to quickly scale up our hardware as business grew and as our client created partnerships with other content providers. This involved buying servers, staffing more IT engineers to manage the infrastructure and many other activities not directly aligned to the core business of the organization. Moreover, buying hardware works for scaling up but it is not so easy to get rid of excess hardware capacity. While there are partial solutions, we needed to be able to buy just the right amount of hardware for the right amount of time and not have to pay for capacity that we did not need.

Maintaining production like large environments:

There was an additional problem of maintaining large hardware infrastructure for simulating heavy loads. While most of the testing did not need to support very large number of concurrent users (typically a few testers manually tested the application), we did need to be able to simulate heavy load conditions. This implied that we needed to have extra servers available only for testing heavy loads. Again, the problem was of over-provisioning and since the client was not a software development company, they did not have any other use for the unused infrastructure.

2. Possible solutions

One of the options was to have redundant hardware nodes (web servers, database servers, routers) and application instances to support the large peak loads experienced by the application for very short times. Provisioning dedicated hardware to serve peak loads meant a large amount of the computing capacity would have stayed underutilized for most of the time. We could have started deploying our own virtual servers to better utilize the idle computing resources but setting up a virtual data center comes with its own challenges and is not aligned with our core business.

What we really needed was to be able to buy computing resources in the amount we needed and for the time we needed. We needed a utility computing service with "pay as you go" and "pay for what you use" approach to computing.

Amazon, through its EC2 (Elastic Compute Cloud) offering, has brought together utility computing, computing in the cloud and virtualization to the masses. Amazon provides storage, raw computing, bandwidth and a programmatic way to control the provisioning of computing resources. The computing resources are virtual servers1 that you start (instantiate) using EC2 command line tools. Amazon charges by the hour for the use of its computing resources. So the meter stops running when you terminate a running Amazon machine2.If designed properly, it is relatively easy to implement horizontal scalability using cloud computing capabilities provided by Amazon EC2.

Amazon's S3 which stands for Simple Storage System provides storage in the cloud - essentially an infinite storage for objects of variable size. Amazon EC2 instances images are stored on S3. S3 is also the storage of choice for backing up data on the non-persistent file-systems associated with the instances3.

Using EC2 along with S3 could allow us to separate infrastructure maintenance duties from application development and give us the ability to extend assets to handle peak loads, within minutes, without having to purchase hardware for infrequent, very high loads.

3. Amazon Elastic Compute Cloud (EC2)

Note: This section describes EC2 at the time of completion of the project. With the advances in virtualization technologies, and increase in competition, the features and offerings may change significantly over time.

Amazon describes its Elastic Compute Cloud (EC2) as a web service that provides re-sizable compute capacity in the cloud. It is designed to make web-scale computing easier for developers.What this means is if you need a server up and running, you can provision a new server by simply executing a few EC2 commands instead of contacting you IT department which in turn will go looking for hardware and then provision it. If your IT department needs to order the hardware, it could take hours (if not days or weeks) to get that new server. And if your need is transient, you will need to figure out what to do with the hardware afterwards when it's not needed anymore. This model of managing computing infrastructure is especially troublesome for smaller companies with systems deployed in production which need to quickly respond to changing business needs.

Amazon EC2 was a good choice for hosting the application for following reasons:

Dynamic scalability:

This is the "elastic" aspect of EC2. Using just a few commands, you can add (or remove) virtual hardware nodes in

matter of minutes. These nodes are typically pre-configured images that have your application software installed. The cost of storing these images on S3 is small, so you can store several images, each one specialized to perform a specific function.

Pay-as-you-go:

You only pay for running virtual servers. Since provisioning is easy and dynamic, you can provision extra virtual servers only when needed and shut them down when not needed, hence paying only for the computing that you used.

Track record:

Amazon is a trusted name in the IT and computing world with a proven track record and EC2 beta was already used by many companies which have published case studies, white papers and success stories showcasing their use of Amazon EC2.

Secure:

Amazon EC2 virtual nodes can belong to separate security groups each with its own security settings completely isolated from the other. At the same time, EC2 provides fine grained control to relax security settings between groups. This means that even for the same application, we could have different security settings for our database servers, web servers and middleware servers.

Additionally, EC2 uses virtualization which has it own benefits such as hardware independence and easy support for multiple operating system configurations.

Once the decision was made to move our application to Amazon EC2, we came up with the following list of requirements

Scalability:

We wanted to be able to scale up or down by dynamically provisioning EC2 virtual nodes.

Load balancing:

We wanted to be able to share the load between multiple virtual nodes.

Failover:

We wanted to be able to gracefully handle shutting down of one or more virtual nodes. This shutdown could be a result of planned scaling down (because of low load) or could be the result of a service crash. We wanted to make sure that in the event of a shutdown or crash, all user sessions were transparently moved to the remaining servers.

4. About the web application

Before we discuss our solution, it will be useful to discuss the web application in question. Without going into details, we would like to focus on the aspects important for this discussion. For most parts, this is your typical JEE web application - it has a Struts and Spring based web application which uses Hibernate for data access and MySQL for persistent storage. It uses the Apache HTTP server for serving static content as well as content from Tomcat (with which it communicates using mod_jk.

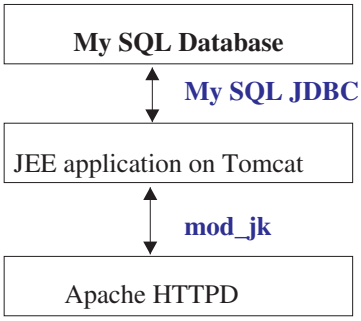

Figure 1: Process view of the application before moving to EC2

5. Scaling JEE application servers on EC2

The critical part of the system was a JEE web application with MySQL database in the backend. After running performance tests, looking at historical data and analyzing the results, we realized that the MySQL database did not have any problem handling even the highest loads that were expected for this

application. It was the JEE web application running on Tomcat which generated complicated processor and memory intensive reports that needed to be scaled up to support the peak loads. This web application also served our client's .NET based desktop client. Hence we needed to be able to scale up (and down) the JEE application server nodes on demand and distribute the load between servers.

We decided to use multiple Tomcat servers running on different nodes each connected to a single Apache HTTP server through the mod_jk plugin. We used mod_jk for both load balancing and failover.

We used MySQL master-slave replication for failover as well as for ensuring that we do not loose user sessions in case of an EC2 node failure (it is very rare, we have never experienced it).

Multiple Tomcat servers shared session by storing them in the MySQL database using the same master-slave setup which was used for the application data persistence.

Storing session in the database allowed other Tomcat servers to pick up and handle sessions of a crashed Tomcat instance.

mod_jk - The apache plugin for tomcat

mod_jk is an Apache plugin which handles communication between an Apache HTTP server and Tomcat server. In addition to handling http requests, mod_jk supports load balancing between multiple tomcat servers, failover and sticky sessions. Sticky sessions are used to ensure that once a client is connected to a particular backend Tomcat server, all subsequent requests

from that client are sent to the same Tomcat server. In case the original Tomcat server becomes unavailable, mod_jk starts sending requests to one of the servers still responding. A simplified worker.properties file used to configure two Tomcat server instances for load balancing, failover and sticky sessions is shown below4:

```
1  # This is the list of workers. There are
2  # 3 workers defined here, the first two are
3  # regular workers, the third is the load
4  # balancer which uses the other two.
5  # ----------------------------
6  worker.list=100p_w1, 100p_w2, 100p_lb
7  # ----------------------------
8  # First worker, pointing to tomcat installed
9  # on 100p.prod01.nagarro.net
10 # ----------------------------
11 worker.100p_w1.port=8009
12 worker.100p_w1.host=100p.prod01.nagarro.net
13 worker.100p_w1.type=ajp13
14 # Load balance factor
15 worker.100p_w1.lbfactor=1
16 # ----------------------------
17 # Second worker, pointing to tomcat installed
18 # on 100p.prod02.nagarro.net
19 # ----------------------------
20 worker.100p_w2.port=8009
21 worker.100p_w2.host=100p.prod02.nagarro.net
22 worker.100p_w2.type=ajp13
23 # Load balance factor
24 worker.100p_w2.lbfactor=1
25 # ----------------------------
26 # Load balancer
27 # ----------------------------
28 worker.100p_lb.type=lb
29 worker.100p_lb.balanced_workers=100p_w1, 100p_w2
30 worker.100p_lb.sticky_session=1
```

In the listing shown above, mod_jk is configured with two Tomcat servers (also referred to as workers at some places), one running on host 100p.prod01.nagarro.net and the other running on host 100p.prod02.nagarro.net. There is a third worker configured as a load balancer (see line number 28). In this scenario, both Tomcat workers will share the load equally (see line numbers 14 and 24). Property "sticky_session" on line number 30 is set to 1 which tells mod_jk load balancer worker that once a user session is associated with a particular backend Tomcat server, all subsequent requests from that user should be directed to that same Tomcat server5.

Adding a Tomcat server node to scale up the JEE application serving capability is pretty simple - launch the appropriate Amazon machine instance, add a worker to the list of workers, configure other worker properties like the load factor, tell the load balancer worker about the new worker and then tell Apache to reload the configuration changes.

Additionally, if one of the Tomcat servers were to crash, the load balancer would automatically start sending requests to one of the available Tomcat servers and since the session is replicated using the MySQL database, this migration is transparent to the end users. For more information on these configurations, refer to the mod_jk documentation.

MySQL replication and failover

MySQL replication enables statements and data from a (master) MySQL server to be replicated to one or more MySQL (slave) servers. This is often referred to as master-slave replication or simply as replication. Few important characteristics of MySQL replication are -

- Data and statement replication is performed asynchronously.
- Typical master-slave setups allow data to be written only to the master node.
- There is usually a small delay associated with replication. This means that if the master node crashed, you could lose a small amount of data. For typical master-slave replication where slaves are connected to the master over a high speed (100Mbps or faster) network, the delays are of the order of few milliseconds to a couple of minutes.

We used MySQL master-slave replication for failover. The MySQL JDBC driver has built-in support for failover. We configured the data source in all Tomcat instances to use the failover property of the MySQL JDBC driver's along with instructions to use the failover node for writes as well as reads.6.

In the event of a MySQL master node failure, the MySQL JDBC driver starts using the slave for all reads and writes and another MySQL slave is launched which starts using the original slave as master,

in effect promoting the original slave to master node. For more detail on these configurations, see the relevant MySQL server and MySQL JDBC driver manuals.

Configuring Tomcat for failover

Configuring Tomcat for failover and load balancing was rather straight forward. We first configured a datasource to work with MySQL failover. This ensured that Tomcat automatically started using the slave database for all reads and writes in case of a master failure. We then separately configured the server to use JDBC and one of the tables in the MySQL database for stroing the session data. A typical configuration section which allows Tomcat to use a database for session persistence is show below:

```
1  <Manager
2  className="org.apache.catalina.session.Persistent
   Manager"
3  saveOnRestart="true" maxActiveSessions="-1"
   minIdleSwap="2"
4  maxIdleSwap="2" maxIdleBackup="2">
5  <Store className="org.apache.catalina.session.
   JDBCStore"
6  connectionURL=
7  "jdbc:mysql://100p.prod.mysql.01.nagarro.net,100p.
8  prod.mysql.02.nagarro.net/100P_prod?
9  user=XXX&amp;password=XXXXXX&amp;failOverRead
   Only=false"
10 driverName="com.mysql.jdbc.Driver"
11 sessionDataCol="session_data sessionIdCol="session_id"
12 sessionLastAccessedCol="last_access"
13 sessionMaxInactiveCol="max_inactive"
14 sessionTable="tomcat_sessions"
15 sessionValidCol="valid_session"
16 sessionAppCol="app_name">
17 </Store>
18 </Manager>
```

Notice that the configuration snippet listed above directly specifies the JDBC connection URL (including the failover information) and does not use the data source used by the JEE application.

6. Other supporting components, setups and configurations

In addition to the tools and techniques discussed earlier, there were several other components, tools and techniques used to support the deployment on EC2. Some of those are discussed below:

Server and service monitoring using NAGIOS:

We used NAGIOS for monitoring servers, services and resources. NAGIOS is an open source host and service monitoring tool with ability to send out alerts based on pre-set configuration and thresholds. This included monitoring parameters like number of users logged on to each web application node, status of master-slave replication, hard disk and memory usage on each server. NAGIOS can warn us of an impending node failure (or of a significant load increase) well in advance and if needed, additional virtual servers can be provisioned to share the load and avoid a service outage.

Source code control repository mirroring:

We created a subversion read only mirror on an EC2 instance for creating builds on EC2. This mirror used a secure tunnel to synchronize itself from the main repository located inside our network. This was used for automated upgrades and patch installation.

Off-site replication of MySQL database:

We used secure connections for off-site (out of EC2) replication of the master and slave databases. This was done using both MySQL master-slave replication as well as rsync.

Backing up MySQL database on S3:

We save MySQL database dumps on a daily basis both on an EC2 server node as well as in S3. We keep weekly backups in S3 going back 53 weeks (about a year), daily backups going back 30 days and hourly backups going back several hours.

Set of base server images:

We have a set of base server images with intelligence to morph into a required target server. One of the several base server images is selected based on requirements and then asked to morph into one of the supported target server types by down-loading contents from S3 to populate its filesystem and by communicating with other nodes. This is important for supporting quick and automated scaling up by provisioning extra servers as well for handling failovers by quickly starting a new node to replace a failed one. These server images also have intelligence to persist their image and the contents of their filesystem to S3. This is useful for shutting down (terminating) an instance when not needed and later starting the instance.

Using Apache to serve static content:

Since Apache can serve static content much faster than Tomcat and Tomcat servers were the ones running CPU and memory intensive jobs, we moved all static content to Apache and served it in compressed form. This also decreased the content size and improved the client response.

7. Deploying the web application on EC2

The final web application deployment is shown in figure 2. As you can see, this process view is significantly different from the original process view but it is the result of the introduction of several very important architectural qualities and infrastructural capabilities into the system. An interesting point to note is that while almost all of these architectural qualities and infrastructural capabilities would have been introduced into the system eventually, the unique nature of EC2 accelerated this. For example, while the ability to dynamically scale up and down is always needed, the lag time associated with provisioning hardware (in absence of a virtual data center with excess capacity) makes this ability a low priority for most systems deployed on traditional infrastructure. Similarly, absence of persistent storage on EC2 instances 7 forced engineers to design, implement, test and deploy very comprehensive data and disk backup strategies right from day one.

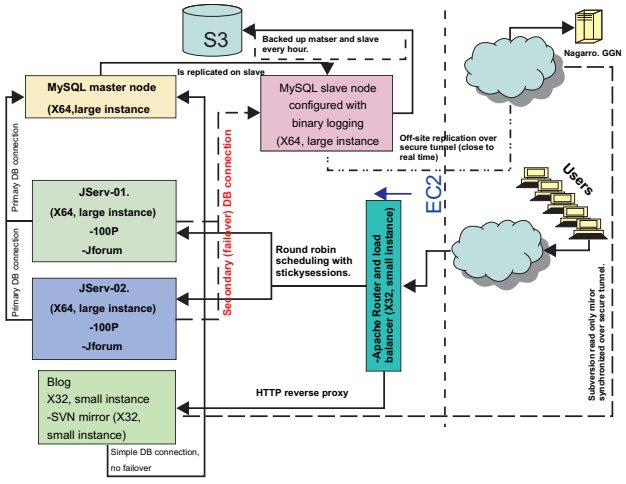

Figure 2: Process view of the application after deployment on EC2

The process view shown in figure 2 shows two Tomcat servers, marked "JServ-01" and "JServ-02" which serve the same set of JEE web applications. The front-end Apache server ,marked "Apache Router", serves static content and uses mod_jk for load balancing the two Tomcat servers and for supporting failover.

You will also notice that the process view has several components that have not been discussed in this document. In particular, we did not discuss the Wordpress blog used by the client's website which was not deployed for scaling up because it was not the critical part of the website. The ability to take tests, view results and reports and discuss course material was far more critical as well as resource intensive hence these components were deployed to be dynamically scalable and to support failover.

8. Conclusion

From a programming point of view, following JEE specifications, guidelines and best practices ensured seamless move to a scalable deployment with failover support requiring only few configuration changes and no code changes at all.

It is important to point out that the current deployment is not infinitely scalable. At a certain point, the approach of saving sessions in the database for persistence and sharing across instances will make database access the performance bottleneck. Our measurements for the configuration shown earlier indicate that this should not be a problem if we increased the number of Tomcat servers by a factor of 3 to 4. There are other ways to cluster JEE web application containers. Some of them are:

Use Terracota to store and replicate session data:

Terracota is a Java based open source network attached storage used for storing and replicating non-critical data8in a cluster. It can be used to store and replicate user sessions.

Use JBoss clustering:

JBoss application server uses JGroups to provide clustering support out of the box.

Use Memcached for replicating session data:

Memcached is another distributed object caching system which can be used for session replication and sharing. You do need to change a few Tomcat classes to use this. These changed classes are available on the net with open source license.

All of the options discussed above can also be used to reduce load on the master database by first looking up the distributed cache for data. For scaling database transactions, MySQL clustering (along with appropriate partitioning) is a very good option. The MySQL clustering engine with filesystem based storage, however, was relatively new at this time and depending on the database schema, using a clustered engine might also require changing the schema9.

In addition to mod_jk based load balancing, we could also use round robin DNS which relatively easy to implement. There are several commercial solutions like Websphere Virtual Enterprise and Oracle Grid which can also be used for scalability and failover but we restricted ourselves to freely available open source components.

It is important to realize that EC2 can only provide scalable computing resources. Your computation itself still has to be able to take advantage of this scalability.

If you application is not designed to support scaling out (horizontal scaling), your only option is to use a larger machine instance (scale up).

One of the biggest problems with older web based clients is use of state-full request-response cycles. Making use of today's rich internet clients (AJAX, Flex etc.) to design applications that use stateless request-response cycles will also make systems far more scalable.

Overall, Amazon's EC2, although currently in beta (at the time of writing this document), is a mature and viable cloud based utility computing solution for meeting computing demands of web based applications with varying load and changing business needs. In the beginning we had identified three main concerns with using EC2 - lack of paid support, no support for static IP addresses for instances and no persistent storage for machine instances. Amazon has already addressed all three of these issues and has since added several other features to EC2 like high CPU instances for CPU intensive computing, user selectable kernels and ability to place instances in different locations and geographic regions which makes it even more attractive for deployments like this one.

Footnotes

...servers1
These servers are also referred to as Amazon machine instances or simply instances.

...machine2
An Amazon machine instance is a virtual server that you start using EC2 command line tools. This is the on-demand computing resource provided by Amazon

...instances3
Since EC2 now provides persistent storage, backing up file-systems on S3 is not as useful anymore

...below4
This listing is not complete; its included for illustrative purposes only. Please refer to the Apache httpd load balancer and mod_jk documentation for details.

...server5
There is an additional small change needed in Tomcat configuration for mod_jk to identify which requests have corresponding session in which Tomcat.

...reads.6
The default failover behavior of the driver is read only. In our case, a MySQL master node failover is considered catastrophic and we do not expect the master to recover.

...instances7
Amazon has since introduced persistent storage for EC2 which, at the time of writing this document, is still under limited trial.

... data8

By non-critical, all we mean here is data which if lost will cause minimal user inconvenience.

nagarro

Silicon Valley : 226 Airport Parkway, Suite 440, San Jose, CA 95110, USA; Ph : +1 (408) 436 6170; Fax : +1 (408) 436 7508
Chicago : 300 S. Wacker Drive, Suite 2335, Chicago, IL 60606; Ph : +1 (312) 235 3250; Fax : +1 (312) 873 4745
New York : 2 Penn Plaza, Suite 1500, New York, NY 10121, USA.; Ph : +1 (212) 799-2899; Fax : +1 (646) 424 - 1140
Atlanta: 400 Galleria Parkway, Suite 1500, Atlanta, GA 30339; Ph : +1 (678) 401-3131; Fax : +1 (678) 826-0688
Frankfurt : Mainzer Landstrasse 27-31, 60329 Frankfurt a. M.; Ph : +49 69 2740 150; Fax : +49 69 2740 15 111; Mobile : +49 160 82 772 75
Stockholm : Torshamnsgatan 39B, Box 13,164 93 Stockholm, Sweden; Ph : +46 (0) 8-751 35 46; Fax : +46 (0) 8 - 457 88 61
India : 15, Electronic City, Sector - 18, Gurgaon - 122015, Haryana, India; Ph :+91(124) 3048646-47, 4016775; Fax : +91 (124) 2455304