Building Highly-available, Scalable and Reliable Architectures for Web-Applications in Cloud

Pavan Kumar Illa¹, Subrahmanyam Kodukula²
¹ PG Scholar, Department of CSE, K L University
² Professor, Department of CSE, K L University

ABSTRACT
Highly-available and scalable web hosting can be a difficult and costly proposition. Traditional scalable web architectures have not only needed to implement difficult solutions to ensure high levels of reliability, but have also needed an exact forecast of traffic to provide a high level of customer service. Dense peak traffic periods and wild swings in traffic patterns result in low utilization rates of expensive hardware, yielding high operating costs to maintain idle hardware, as well as an inefficient use of capital for underutilized hardware. Cloud computing provides the reliable, scalable, secure, and highly performing infrastructure required for the most demanding web applications, while enabling an elastic, scale-out and scale-down infrastructure model to match IT costs with real-time customer traffic patterns. It is critical to build a scalable architecture in order to take advantage of a scalable infrastructure. The cloud is designed to provide conceptually infinite scalability. However, we cannot leverage all that scalability in infrastructure if our architecture is not scalable. Both have to work together. In this paper, we illustrate the style of building scalable web application architectures using services available in the cloud. Next we design a cloud architecture for a sample java web application by using Amazon web services (AWS) cloud infrastructure by programming each scalable cloud component based on the nature of the web content of the application. Moreover we develop this application by following the best practices provided by AWS for cloud application developers and, we deploy this application into public internet through deployment service provided by AWS. Finally we explain how the effective load balancing and Auto scaling can be happened with the selected cloud components to attain the real time scalability that meets the customer demands i.e. how the designed application architecture can be able to leverage the scalable infrastructure provided by cloud.

Keywords: Scalability, High-availability, AWS cloud, Reliability

INTRODUCTION
It has become difficult for most of the IT industries and business organizations to buy computing infrastructure (e.g., computing servers, storage or network) due to the tremendous increase in the cost and maintaining such infrastructure, deploying and running business applications on their own becomes an overhead. In order to break all these overheads, organizations are migrating to cloud computing.

CLOUD COMPUTING
Cloud computing is a model that allows us to access software, servers and storage resources over the internet, in a self-service manner. Instead of having to buy, install, maintain and manage these resources on our own computer, we can access and use them
through a web browser. Instead laying out capital to buy hardware, software, we rent what we need, usually on subscription basis. The NIST (National Institute of Standards and Technology) Definition [1] of cloud computing is “Cloud computing is a model for enabling ubiquitous, convenient, on demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction”. It provides pay-per-use capabilities. As a user we see only the self-service interface to the computing resources we need. All these resources are maintained by cloud providers at some geographic location. Behind the scenes cloud computing providers have to do a lot of work to manage the entire infrastructure, technology, people that make this possible. To provide services easily, flexibly and profitably to thousands or even millions of user’s, they invest heavily in hardware, virtualization techniques, networking infrastructure and automation capabilities. There are thousands of cloud providers and solutions out. Among them Amazon web services (AWS) and Rackspace are the top most cloud computing service providers.

CLOUD ARCHITECTURES

In this section illustrates the style of building applications using services available in the Internet cloud. Cloud Architectures are designs of software applications that use Internet-accessible on-demand services. Applications built on Cloud Architectures are such that the underlying computing infrastructure is used only when it is needed (for example to process a user request), draw the necessary resources on-demand (like compute servers or storage), perform a specific job, then relinquish the unneeded resources and often dispose themselves after the job is done. While in operation the application scales up or down elastically based on resource needs.

Cloud Architectures address key difficulties surrounding large-scale data processing. In traditional data processing it is difficult to get as many machines as an application needs. Second, it is difficult to get the machines when one needs them. Third, it is difficult to distribute and coordinate a large-scale job on different machines, run processes on them, and provision another machine to recover if one machine fails. Fourth, it is difficult to auto scale up and down based on dynamic workloads. Fifth, it is difficult to get rid of all those machines when the job is done. Cloud Architectures solve such difficulties.

Applications built on Cloud Architectures run in-the-cloud where the physical location of the infrastructure is determined by the provider. They take advantage of simple APIs of Internet-accessible services that scale on demand, that are industrial-strength, where the complex reliability and scalability logic of the underlying services remains implemented and hidden inside-the-cloud. The usage of resources in Cloud Architectures is as needed, sometimes ephemeral or seasonal, thereby providing the highest utilization and optimum bang for the buck.

A) Benefits of Cloud Architectures

There are some clear business benefits to building applications using Cloud Architectures. A few of these are listed here:

- Almost zero upfront infrastructure investment: If we have to build a large-scale system it may cost a fortune to invest in real estate, hardware (racks, machines, routers, backup power supplies), hardware management (power management, cooling), and operations personnel. Because of the upfront costs, it would typically
need several rounds of management approvals before the project could even get started. Now, with utility-style computing, there is no fixed cost or startup cost.

- Just-in-time Infrastructure: In the past, if we got famous and our systems or our infrastructure did not scale, we became a victim of our own success. Conversely, if we invested heavily and did not get famous, we became a victim of our failure. By deploying applications in-the-cloud with dynamic capacity management software architects does not have to worry about pre-procuring capacity for large scale systems. The solutions are low risk because we scale only as we grow. Cloud Architectures can relinquish infrastructure as quickly as we got them in the first place (in minutes).

- More efficient resource utilization: System administrators usually worry about hardware procuring (when they run out of capacity) and better infrastructure utilization (when they have excess and idle capacity). With Cloud Architectures they can manage resources more effectively and efficiently by having the applications request and relinquish resources only what they need (on-demand).

- Usage-based costing: Utility-style pricing allows billing the customer only for the infrastructure that has been used. The customer is not liable for the entire infrastructure that may be in place. This is a subtle difference between desktop applications and web applications. A desktop application or a traditional client-server application runs on customer’s own infrastructure (PC or server), whereas in a Cloud Architectures application, the customer uses a third party infrastructure and gets billed only for the fraction of it that was used.

- Potential for shrinking the processing time: Parallelization is the one of the great ways to speed up processing. If one compute-intensive or data intensive job that can be run in parallel takes 500 hours to process on one machine, with Cloud Architectures, it would be possible to spawn and launch 500 instances and process the same job in 1 hour. Having available an elastic infrastructure provides the application with the ability to exploit parallelization in a cost-effective manner reducing the total processing time.

B) Examples of Cloud Architectures

There are plenty of examples of applications that could utilize the power of Cloud Architectures. These range from back-office bulk processing systems to web applications. Some are listed below:

a) Processing Pipelines

- Document processing pipelines – convert hundreds of thousands of documents from Microsoft Word to PDF, OCR millions of pages/images into raw searchable text
- Image processing pipelines – create thumbnails or low resolution variants of an image, resize millions of images
- Video transcoding pipelines – transcode AVI to MPEG movies
- Indexing – create an index of web crawl data
- Data mining – perform search over millions of records

b) Batch Processing Systems

- Back-office applications (in financial, insurance or retail sectors)
- Log analysis – analyze and generate daily/weekly reports
- Nightly builds – perform nightly automated builds of source code repository every night in parallel
Automated Unit Testing and Deployment Testing—Test and deploy and perform automated unit testing (functional, load, quality) on different deployment configurations every night

c) Websites

- Websites that “sleep” at night and auto-scale during the day
- Instant Websites – websites for conferences or events (Super Bowl, sports tournaments)
- Promotion websites
- “Seasonal Websites” - websites that only run during the tax season or the holiday season (“Black Friday” or Christmas)

The cloud reinforces some old concepts of building highly scalable Internet architectures [2] and introduces some new concepts that entirely change the way applications are built and deployed. Hence, when we progress from concept to implementation, we might get the feeling that “Everything’s changed, yet nothing’s different.” The cloud changes several processes, patterns, practices, philosophies and reinforces some traditional service-oriented architectural principles that we have learnt as they are even more important than before. In the next section, we will see some of those new cloud concepts and reiterated SOA concepts.

Traditional applications were built with some pre-conceived mindsets that made economic and architectural-sense at the time they were developed. The cloud brings some new philosophies that we need to understand and are discussed in the following section.

C) Building Scalable Architectures

It is critical to build a scalable architecture in order to take advantage of a scalable infrastructure. The cloud is designed to provide conceptually infinite scalability. However, we cannot leverage all that scalability in infrastructure if our architecture is not scalable. Both have to work together. We will have to identify the monolithic components and bottlenecks in our architecture, identify the areas where we cannot leverage the on-demand provisioning capabilities in our architecture and work to refactor our application in order to leverage the scalable infrastructure and take advantage of the cloud.

Characteristics of a truly scalable application:

- Increasing resources results in a proportional increase in performance
- A scalable service is capable of handling heterogeneity
- A scalable service is operationally efficient
- A scalable service is resilient
- A scalable service should become more cost effective when it grows (Cost per unit reduces as the number of units increases).

These are things that should become an inherent part of our application and if we design our architecture with the above characteristics in mind, then both our architecture and infrastructure will work together to give us the scalability we are looking for.

5 TOUR LOG –A SAMPLE JAVA WEB APPLICATION

In this section, we describe a java web application that we have designed by using the on-demand infrastructure provided by Amazon Web Services. This application allows a user to store to the photos of their tour to a particular place by creating a separate Journal for each place. The application brings up hundreds of virtual servers on-demand, runs a parallel computation on them and
then shuts down all the virtual servers releasing all its resources back to the cloud—all with low programming effort and at a very reasonable cost for the caller.

In the next section we discuss the technical aspects of the application in detail.

1) Application Aspects

The application aspects deals with usage of infrastructure provided by AWS like servers, storage etc.These aspects fall under computational aspects, Storage aspects, Notification aspects, Deployment and management aspects. In the following sub section we discuss each aspect in detail and how those aspects can achieve the required scalable architecture to accommodate scalable infrastructure provided by cloud.

A) Computational Aspects

For Computational Aspects, the AWS infrastructure that we consider is Amazon Elastic Compute Cloud (Amazon EC2), Auto scaling, Elastic Load Balancer (ELB).

a) Amazon EC2

Amazon Elastic Compute Cloud (Amazon EC2) [3] is a web service that provides resizable compute capacity in the cloud. It is designed to make web-scale computing easier for developers.

Amazon EC2’s simple web service interface allows us to obtain and configure capacity with minimal friction. It provides you with complete control of our computing resources and lets us run on Amazon’s proven computing environment. Amazon EC2 reduces the time required to obtain and boot new server instances to minutes, allowing us to quickly scale capacity, both up and down, as our computing requirements change. Amazon EC2 changes the economics of computing by allowing us to pay only for capacity that we actually use. Amazon EC2 provides developers the tools to build failure resilient applications and isolate themselves from common failure scenarios.

b) Auto Scaling

Auto Scaling [4] allows us to scale our Amazon EC2 capacity up or down automatically according to conditions we define. With Auto Scaling, we can ensure that the number of Amazon EC2 instances we’re using increases seamlessly during demand spikes to maintain performance, and decreases automatically during demand lulls to minimize costs. Auto Scaling is particularly well suited for applications that experience hourly, daily, or weekly variability in usage. Auto Scaling is enabled by Amazon Cloud Watch and available at no additional charge beyond Amazon Cloud Watch fees. The following are the common uses for Auto Scaling.

Automatically Scaling Your Amazon EC2 Fleet: Auto Scaling enables us to closely follow the demand curve for our applications, reducing the need to provision Amazon EC2 capacity in advance. For example, we can set a condition to add new Amazon EC2 instances in increments of 3 instances to the Auto Scaling Group when the average CPU utilization of your Amazon EC2 fleet goes above 70 percent; and similarly, we can set a condition to remove Amazon EC2 instances in the same increments when CPU Utilization falls below 10 percent. Often, we may want more time to allow our fleet to stabilize before Auto Scaling adds or removes more Amazon EC2 instances. We can configure a cool-down period for our Auto Scaling Group, which tells Auto Scaling to wait for some time after taking an action before it evaluates the conditions again. Auto Scaling enables us to run our Amazon EC2 fleet at optimal utilization.

c) Elastic Load Balancing (ELB)

Elastic Load Balancing [5] automatically distributes incoming application traffic across multiple Amazon EC2 instances. It enable us to achieve even greater fault tolerance in our applications, seamlessly providing the amount of load balancing capacity needed in response to incoming application traffic. Elastic Load Balancing
detects unhealthy instances within a pool and automatically reroutes traffic to healthy instances until the unhealthy instances have been restored. Customers can enable Elastic Load Balancing within a single Availability Zone or across multiple zones for even more consistent application performance. The following are the common uses of the ELB.

Auto Scaling with Elastic Load Balancing: Let’s say that we want to make sure that the number of healthy Amazon EC2 instances behind an Elastic Load Balancer is never fewer than two. We can use Auto Scaling to set these conditions, and when Auto Scaling detects that a condition has been met, it automatically adds the requisite amount of Amazon EC2 instances to our Auto Scaling Group. Or, if we want to make sure that we add Amazon EC2 instances when latency of any one of our Amazon EC2 instances exceeds 4 seconds over any 15 minute period, we can set that condition, and Auto Scaling will take the appropriate action on our Amazon EC2 instances — even when running behind an Elastic Load Balancer. Auto Scaling works equally well for scaling Amazon EC2 instances whether we’re using Elastic Load Balancing or not.

B) Storage Aspects

We used Amazon Simple Storage Service (Amazon S3), Amazon SimpleDB for storing input and output data of our applications. We made use of the Amazon S3 API’s to for programming according to our needs.

a) Amazon S3 Amazon S3 [6] is storage for the Internet. It is designed to make web-scale computing easier for developers. Amazon S3 provides a simple web services interface that can be used to store and retrieve any amount of data, at any time, from anywhere on the web. It gives any developer access to the same highly scalable, reliable, secure, fast, inexpensive infrastructure that Amazon uses to run its own global network of web sites. The service aims to maximize benefits of scale and to pass those benefits on to developers. Following are the common use cases for Amazon S3.

Content Storage and Distribution: Amazon S3 provides a highly durable and available store for a variety of content, ranging from web applications to media files. It allows us to offload our entire storage infrastructure onto the cloud, where we can take advantage of Amazon S3’s scalability and pay-as-you-go pricing to handle our growing storage needs. We can distribute our content directly from Amazon S3 or use Amazon S3 as an origin store for pushing content to our Amazon Cloud Front edge locations. For sharing content that is either easily reproduced or where we’re storing an original copy elsewhere, Amazon S3’s Reduced Redundancy Storage (RRS) feature provides a compelling solution. For example, if we’re storing media content in-house but we need to provide accessibility to our customers, channel partners, or employees, RRS is a low-cost solution for storing and sharing this content.

b) Amazon Simple Amazon SimpleDB [7] is a highly available and flexible non-relational data store that offloads the work of database administration. Developers simply store and query data items via web services requests and Amazon SimpleDB does the rest. Unbound by the strict requirements of a relational database, Amazon SimpleDB is optimized to provide high availability and flexibility, with little or no administrative burden. Behind the scenes, Amazon SimpleDB creates and manages multiple geographically distributed replicas of our data automatically to enable high availability and data durability. The service charges only for the resources actually consumed in storing our data and serving our requests. We can change our data model on the fly, and data is automatically indexed for us. With Amazon SimpleDB, we can focus on application development without worrying about infrastructure provisioning, high availability, software maintenance, schema and index management, or performance tuning.
C) Notification Aspects  Amazon Simple Notification Service (Amazon SNS) is used for the notification of updates about other aspects.

a) Amazon SNS  Amazon Simple Notification Service (Amazon SNS) [8] is a web service that makes it easy to set up, operate, and send notifications from the cloud. It provides developers with a highly scalable, flexible, and cost-effective capability to publish messages from an application and immediately deliver them to subscribers or other applications. It is designed to make web-scale computing easier for developers.

D) Deployment and Management Aspects

AWS Elastic Beanstalk has been used for the deployment and management purposes.

a) AWS Elastic Beanstalk  AWS Elastic Beanstalk [9] is an even easier way for us to quickly deploy and manage applications in the AWS cloud. We simply upload our application, and Elastic Beanstalk automatically handles the deployment details of capacity provisioning, load balancing, auto-scaling, and application health monitoring. At the same time, with Elastic Beanstalk, we retain full control over the AWS resources powering our application and can access the underlying resources at any time. Elastic Beanstalk leverages AWS services such as Amazon Elastic Cloud Compute (Amazon EC2), Amazon Simple Storage Service (Amazon S3), Amazon Simple Notification Service (Amazon SNS), Elastic Load Balancing, and Auto Scaling to deliver the same highly reliable, scalable, and cost-effective infrastructure that hundreds of thousands of businesses depend on today. AWS Elastic Beanstalk is easy to begin and impossible to outgrow.

b) AWS Elastic Beanstalk Functionality for JAVA developers

To deploy Java applications using AWS Elastic Beanstalk, we simply:

- Create our application as we normally would use any editor or IDE (e.g. Eclipse).
- Package our deployable code into a standard Java Web Application Archive (WAR file).
- Upload our WAR files to Elastic Beanstalk using the AWS Management Console, the AWS Toolkit for Eclipse, and the web service APIs, or the command line interface.
- Deploy our application. Behind the scenes, Elastic Beanstalk handles the provisioning of a load balancer and the deployment of our WAR file to one or more Amazon EC2 instances running the Apache Tomcat application server.
- Within a few minutes we will be able to access our application at a customized URL (e.g. http://myapp.elasticbeanstalk.com/).

Following are the service highlights of AWS Elastic Beanstalk.

2) Tour Log Cloud Application Architecture

Tour Log Cloud Application Architecture is designs of software application that use Internet-accessible on-demand services. Application was built on Cloud Architecture is such that the underlying computing infrastructure is used only when it is needed (for example to process a user request like uploading photo), draw the necessary resources on-demand (like compute servers or storage), perform a specific job, then relinquish the unneeded resources and often dispose themselves after the job is done. While in operation the application scales up or down elastically based on resource needs.

The approach was to build an application that not only scales with demand, but also without a heavy upfront investment and without the cost of maintaining idle machines (“down bottom”). To get a response in a reasonable amount of time, it was important to distribute the job into multiple tasks and to perform a Distributed Tour Log operation that runs those tasks on multiple nodes in parallel.
Figure 1 depicts the high-level architecture of Tour Log.

Figure 1 shows a high-level depiction of the architecture. The output of the user request Service, which is a sorted list of links and in a single file, is given to Tour Log as input. It takes journal entries as a second input. It then returns updated journal and stored into a single file. Since the overall process is asynchronous, developers can get the status of their jobs by calling Get Status () to see whether the execution is completed.

![Figure 1: Level 1 Tour Log Application Architecture](image1)

Figure 2 shows the level 2 architecture of Tour Log Architecture, which uses Amazon EC2, Amazon S3, and Amazon SimpleDB. The detail discussion on these components has already been given in previous section.

A) Workflow

When a user requests for a process to be done, consider uploading an image and writing some comments about the image. When we upload the image, it will be stored in Amazon S3. The metadata will be stored in Amazon SimpleDB. When an image is to be modified the computational task will be done by Amazon EC2. The notification of update or comments regarding the image will be send to the user by using Amazon SNS.

The designed Cloud application utilizes all the scalable components of AWS, hence it leads to overall scalability to leverage the scalable infrastructure provide by cloud with minimal cost effort.

![Figure 2: Storage, Compute level Architecture of Tour Log](image2)
Figure 3 shows the overall scalable architecture of Tour Log Application.

![Figure 3: Scalable Cloud Architecture for Tour Log](image)

**Running Amazon EC2 instances**

<table>
<thead>
<tr>
<th>Instance ID</th>
<th>AMI ID</th>
<th>Root Device</th>
<th>Type</th>
<th>State</th>
<th>Status Checks</th>
<th>Alarm Status</th>
<th>Monitoring</th>
<th>Security Groups</th>
<th>Key Pair</th>
</tr>
</thead>
<tbody>
<tr>
<td>i-56565656</td>
<td>ami-00000000</td>
<td>obs</td>
<td>t1.micro</td>
<td>running</td>
<td>2/2 checks</td>
<td>none</td>
<td>basic</td>
<td>elasticbeanstalk-default</td>
<td>tour</td>
</tr>
<tr>
<td>i-56565656</td>
<td>ami-00000000</td>
<td>obs</td>
<td>t1.micro</td>
<td>running</td>
<td>2/2 checks</td>
<td>none</td>
<td>basic</td>
<td>quicklaunch-1</td>
<td>tour</td>
</tr>
<tr>
<td>i-56565656</td>
<td>ami-00000000</td>
<td>obs</td>
<td>t1.micro</td>
<td>running</td>
<td>Loading...</td>
<td>none</td>
<td>basic</td>
<td>quicklaunch-1</td>
<td>tour</td>
</tr>
</tbody>
</table>

**Sample EC2 configuration**

- **EC2 Instance Type**: t1.micro
- **EC2 Security Groups**: elasticbeanstalk-default
- **Existing Key Pair**: tour
Auto scaling configuration

**Auto Scaling**

Auto-scaling automatically launches or terminates EC2 instances based on defined metrics and thresholds called triggers. Auto-scaling will also launch a new EC2 instance in the event of a failure. These settings allow you to control auto-scaling behavior.

- **Minimum Instance Count**: 1
- **Maximum Instance Count**: 4
- **Availability Zones**: Any
- **Scaling Cooldown Time (seconds)**: 360
- **Scaling Trigger**
  - **Trigger Measurement**: NetworkOut
  - **Trigger Statistic**: Average
  - **Unit of Measurement**: Bytes
  - **Measurement Period (seconds)**: 5

Load Balancer Configuration

**Load Balancing**

These settings allow you to control the behavior of your environment’s load balancer.

- **HTTP Port**: 80
- **HTTPS Port**: OFF
- **SSL Certificate Id**: 

**EC2 Instance Health Check**

These settings allow you to configure how AWS Elastic Beanstalk determines whether an EC2 instance is healthy or not.

- **Application Health Check URL**: /
- **Health Check Interval (seconds)**: 30
- **Health Check Timeout (seconds)**: 5
- **Healthy Check Count Threshold**: 3
- **Unhealthy Check Count Threshold**: 5

**CONCLUSION**

Instead of building your applications on fixed and rigid infrastructures, Cloud Architectures provide a new way to build applications on on-demand infrastructures. Tour Log demonstrates how such applications can be built. Without having any upfront investment, we were able to run a job massively distributed on multiple nodes in parallel and
scale incrementally based on the demand (users, size of the input dataset). With no idle time, the application infrastructure was never underutilized.

ACKNOWLEDGEMENTS

We like to express our gratitude to all those who gave us the possibility to carry out the paper. We would like to thank Mr.K.Satyanarayana, chancellor of K.L.University, Dr.K.Raja Sekhara Rao, Dean, and K.L.University for stimulating suggestions and encouragement. We have further more to thank Prof.S.Venkateswarlu, Dr.K.Subrahmanyam, who encouraged us to go ahead with this paper.

REFERENCES