Scaling on AWS (Part 4) : > One Million Users

AWS Startups Follow
Jan 12, 2016 · 6 min read

In the fourth and final part of this series, I want to show you how to scale an architecture to serve millions of users. If you have not read the first, second or third part, I recommend doing so to see how I iteratively scaled a basic Amazon Web Services (AWS) architecture to serve an increasing number of users. Additionally, in the first three parts I discussed the following foundational AWS concepts:

- Decoupling application tiers to prepare for horizon scaling
- Choosing among the database options available on the AWS platform
- Shifting components of workload to appropriate AWS services for better performance and efficiency
- Using Auto Scaling to ensure that you have the correct number of EC2 instances available to handle the load for your application
- Automating your software development lifecycle with AWS services

To close out this scaling series, I want you to consider leveraging service-oriented architecture (SOA) and AWS application services.

Service-Oriented Architecture

Thus far, I've lead us on the path of decoupling your application into separate tiers and moving workload sub-components into the most fitting AWS services. How can we take this further? The answer is to apply service-oriented architecture (SOA) principles to your architecture. Instead of just splitting your application by server roles (for example, web vs. application), consider splitting resources further by the service they collectively deliver. The benefit of this is that you can scale each service independently. Your web and application tiers will have different resource requirements, and so will your different services.

Adopting SOA sounds great, but as a startup you might not want to spend the time or your limited resources to develop an enterprise-grade queuing or messaging system. But fear not, developers. The AWS platform makes it really easy for you to achieve a loosely coupled architecture by offering many plug and play generic services. I discuss some of these commonly used services in the following sections.

Don't Reinvent the Wheel

Sending emails, queueing, transcoding, monitoring, and logging are generic features that most software requires, but does not add differentiating value. If they don't add differentiating value, why should you spend time to develop them in-house? AWS provides a host of these generic services to help you build infrastructure quickly. The following illustration shows some application services that AWS offers.

Use Application Services to Facilitate Loose Coupling
(Don't reinvent the wheel)
Lambda
SES
SNS
CloudSearch
SWF
SQS
Elastic Transcoder

Amazon Simple Queue Service (SQS)
SQS acts as glue for a distributed and highly scalable system. When a system is stitched together via asynchronous messaging, different parts of the system can scale or fail independently. Imagine a job that needs to be processed in three steps. If you have independent fleets that are responsible for each step and step two fails, that job could remain in the queue until step two is fixed rather than be canceled completely.

SQS is also very valuable to startups because it removes the maintenance overhead from your software developers. As a managed service, SQS maintains availability of the queues for you. As a scalable service, SQS allows you to create an unlimited number of queues, which saves you from having to do capacity planning. Your software developers need to learn only a simple RESTful API to start using SQS.

Amazon Simple Notification Service (SNS)
SNS is a pub-sub service for sending messages to different AWS services. Subscribers of a topic receive messages published to that topic. For example, you can have CloudWatch fire messages to a SNS topic, and any AWS Lambda function that is subscribed to that topic will be triggered. While both SNS and SQS are messaging services on the AWS platform, SNS can push messages to multiple subscribers, eliminating the need to periodically check or "poll" for updates.

SNS also has a mobile push functionality that lets you deliver push notifications to Apple, Google, Fire OS, and Windows devices, as well as Android devices in China with Baidu Cloud Push. With push notifications, an installed mobile application can notify its users immediately by popping a notification about an event, without opening the application. For example, if you install a sports app and enable push notifications, the app can send you the latest score of your favorite team even if the app isn't running. The notification appears on your device, and when you acknowledge it, the app launches to display more information. Users' experiences are similar to receiving an SMS, but with enhanced functionality and at a fraction of the cost.

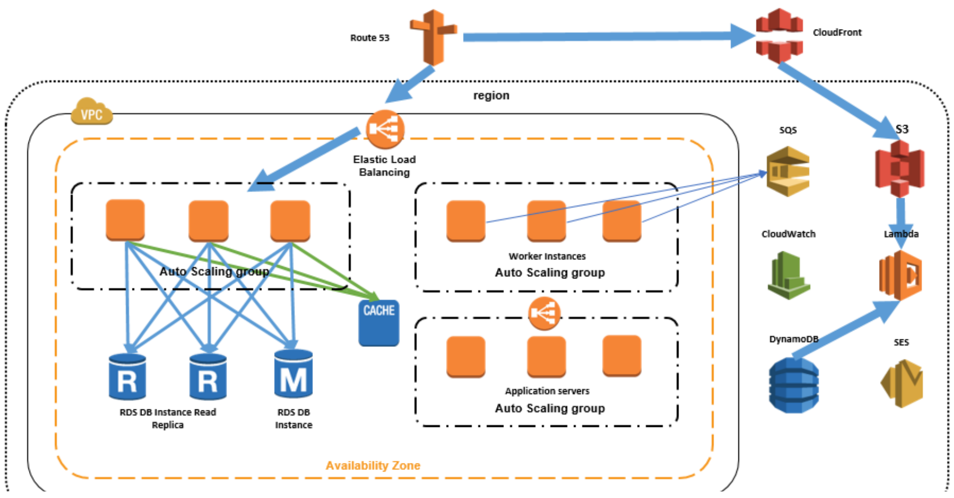
AWS Lambda
AWS Lambda runs your code written in Java, Node.js, and Python without requiring you to provision or manage servers. Lambda will run and scale your code with high availability, and you pay only for the compute time you consume in increments of 100 milliseconds. The code you upload will become Lambda functions, which can be triggered by various events on the AWS platform. Lambda functions can be thought of as first-class compute resources that you can insert into your AWS architecture. A common use case is to have a Lambda function listening to S3 events and performing custom processing logic when objects are uploaded into S3. AWS Lambda is a fairly new but powerful service on the AWS platform.

Users > One Million

Let's assume my application has become hugely popular, and it's now getting close to a million visitors. Serving a million users and more is going to require that we bring together all the approaches I've discussed so far:

- Deploying across multiple Availability Zones
- Using Elastic Load Balancing (ELB) between tiers
- Using Auto Scaling
- Using service-oriented architecture
- Serving content using appropriate AWS services (for example, local vs. Elastic Block Store vs. S3)
- Caching off DB using ElastiCache
- Moving state off tiers so you can horizontally scale

The following diagram illustrates usage of aforementioned approaches.



Users @ Five to Ten Million

Between five to ten million users, you might run into issues with your database due to contention on the write master. There are several methods of mitigation, but they essentially require you to further decouple in the database layer. These methods include database federation, sharding, and using NoSql.

Database Federation

Database federation is about separating databases by function. For example, you might choose to store data for your forum, users, and products in three distinct databases. Again, this allows your components to scale independently. This database strategy can complement the SOA strategy. For example, you might decide you want to keep data belonging to separate services in separate databases. On the flip side, this architecture is more complex and requires you to write more sophisticated queries to fetch data.

Sharding

If your data is still too large to be managed under a federation schema, you can consider sharding. This is horizontal scaling at the database tier. With sharding, you store data across multiple databases and spread the records evenly. You can have users with last names in the A through M range in one database and the rest in another. Sharding allows you to scale larger than federation, but it requires more logic in your application to dynamically change the target database depending on the data you need.

NoSql

Lastly, if you have independent tables in relational database, you could consider moving that data to a NoSql database such as Amazon DynamoDB. Relational database has overhead associated with maintaining relationships between data. So why take on that overhead when you don't need to extract relationships from your data? DynamoDB is a managed, scalable, and high performance key-value store. Like RDS, DynamoDB takes care of administrative tasks for you so you can focus on high-value tasks. Data is stored across multiple facilities across a region to achieve high availability. DynamoDB also scales automatically with the size of data.

Next Steps

I hope this blog series has given you some food for thought regarding scaling on the AWS platform. For additional resources, please refer to our reference architecture section to see commonly used architecture, read our documentation to stay informed, and go to the startup blog section to see how your peers are using AWS services.