AWS Startups Blog

Scaling on AWS (Part 2): > 10K Users

by AWS Admin | on 17 DEC 2015 | in Startup Spotlight | Permalink | Share

By David Kuo, Solutions Architect, AWS

Welcome to the second post in a series of primers designed to help you scale your startup on Amazon Web Services (AWS). In the first post, I showed how to deploy a basic architecture to get started on AWS quickly. I also showed how to decouple the architecture into separate tiers to enable horizontal scaling, which allows businesses to easily handle increased traffic.

In this post, I continue the discussion by walking you through an example of scaling on AWS to support a growing website. My goal is to achieve an architecture that will serve over 10,000 users. I start with the database options for creating an independent data tier, and I enlist the help of several AWS services along the way.

Database Options on AWS

When it comes to deploying databases on AWS, there are two general directions:
1. Use a managed database service such as Amazon Relational Database Service (Amazon RDS) or Amazon DynamoDB
2. Host your own database software on Amazon EC2

In the following sections, I briefly describe each of the services to help you choose the option that's right for your business.

Amazon RDS
Amazon RDS allows you to offload generic database administrative tasks to AWS. RDS can automatically perform nightly backup, upgrade and patch database software, and create a redundant database instance in a different AZ for seamless failover. For startups with limited resources, RDS frees your database developers from administrative tasks, allowing them to focus on value-creating tasks. RDS gives you the option to choose from a variety of engines that include Microsoft SQL Server, Oracle, MySQL, PostgreSQL, and Aurora. RDS (SQL Server and Oracle) also has a flexible licensing model that allows you to bring your own license (BYOL).

Amazon DynamoDB
Some applications benefit from a non-relational database like DynamoDB. DynamoDB is a fast and flexible NoSQL database service for all applications that need consistent, single-digit millisecond latency at any scale. It is a managed cloud database, and supports both document and key-value store models.

Amazon EC2
Lastly, if you need a database engine not offered by AWS or need full control of the operating system on the database server, you can host your custom database software on an EC2 instance. I recommend using managed database services when possible because they are designed to help your database administrators focus on high-value tasks.

SQL vs. NoSQL

A common discussion on the database topic is the choice between SQL and NoSQL. The right answer depends on what you're trying to achieve. SQL is often a good starting point. Here are some basic guidelines:

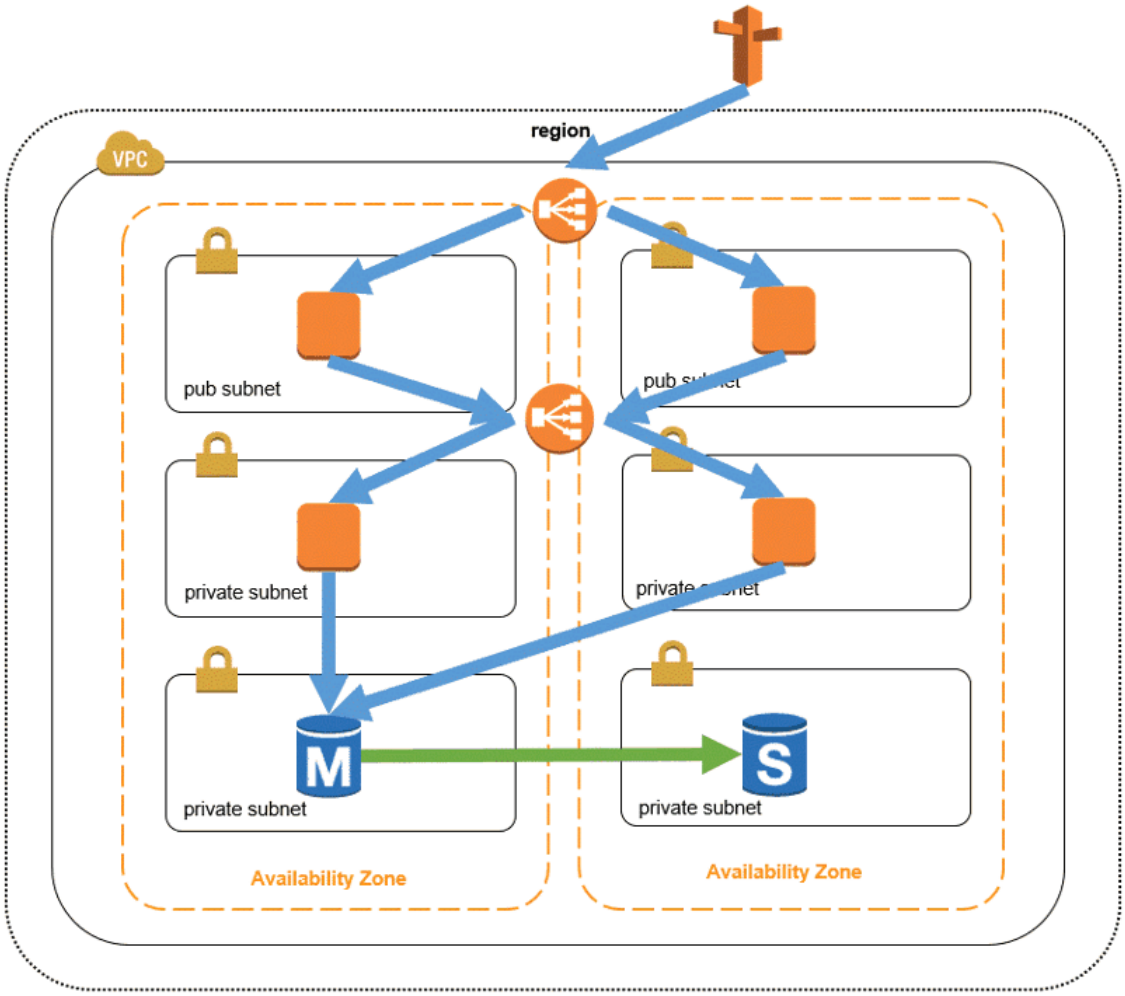
Choose SQL if:
• You want to take advantage of a well-established technology.
• You want to leverage the wealth of existing code, communities, examples, and clear patterns to scalability.
• You do not have massive amounts of data. When I say massive, I'm talking about generating terabytes of data within the first year.

Choose NoSQL if:
• You have highly non-relational data
If you do not need to store relationships between data objects to extract information, storing data in NoSQL using a key-value format provides you with input and output data quickly.
• You have massive amounts of data (in the TB range)
NoSQL is a distributed database that scales horizontally. The key-value format lends itself to be sharded easily. NoSQL data architecture can maintain high throughput even for large data sets.
• You need rapid ingest of data (thousands of records/sec)
Due to the simple key-value structure, NoSQL data can be inserted and removed quickly. This makes NoSQL a good candidate for dealing with high-volume data.

For the example that I walk you through in the following sections, I use RDS. I start with the assumption that I will need to extract relationships from the data to

search for things such as the top five searched products or the number of users who visited last Tuesday and looked at a specific product category.

The following illustration shows an updated architecture. The green arrow between the master and standby databases represents the synchronous data replication when the Multi-AZ feature is enabled.



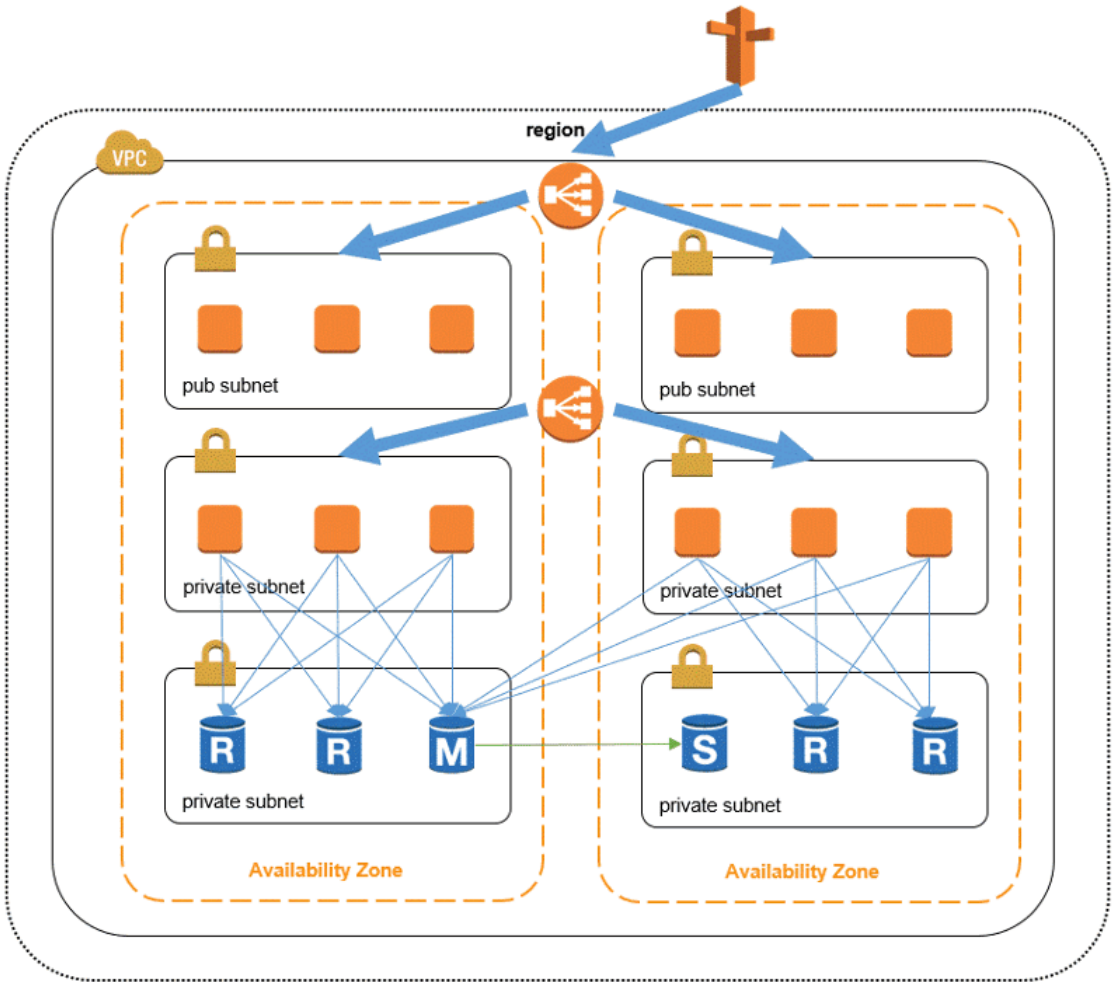
Users > 10K

After decoupling my application tiers, I'm now ready to horizontally scale. Remember from my previous post that my original architecture had some issues. First, it lacked redundancy. Second, my application sat in a single AZ. On AWS, I can address these issues by deploying across multiple AZs. For my application tier, I can stand up servers in multiple AZs and load balance incoming traffic across EC2 instances using Elastic Load Balancing (ELB). ELB is a redundant, horizontally scaled service that requires zero administration from my end and supports SSL termination and sticky sessions. ELB also performs health checks on instances so that only healthy instances receive traffic. With this setup, I can handle more traffic by adding more servers in each AZ.

In the database tier, I can add redundancy by using the Multi-AZ feature of RDS. To enable this feature, I simply select a check box in the RDS console during the RDS launch process.

When you use Multi-AZ, RDS will create a standby database instance in a different AZ and replicate data to it synchronously. This synchronous data replication makes your failover experience seamless. The connection string does not change because the database DNS name does not change. You will have to include retry logic into the database connection code, but you can see that RDS has taken on the heavy lifting. Outside of failover events, this standby database instance also enhances availability because it minimizes downtime during planned maintenance. RDS knows to perform updates first on the standby prior to an automatic failover.

When your applications become popular and you need to serve a high number of read requests, you can scale the database tier with Amazon RDS Read Replicas. Read Replicas are available if you are using MySQL, PostgresSQL, or Amazon Aurora. RDS MySQL and RDS PostgresSQL allow up to five Read Replicas and leverage native replication capability of MySQL and PostgresSQL that are subject to replication lag. Amazon Aurora allows up to 15 Replicas and experiences minimal replication lag because Aurora Replicas use the same underlying storage. The following illustration shows an updated architecture with Amazon RDS Read Replicas.



Horizontal scaling at the application layer, RDS Multi-AZ, and RDS Read Replicas will allow your system to scale pretty far. But for serving over ten thousand users, I suggest we decouple the architecture even further.

Shifting Loads to Increase Performance and Efficiency

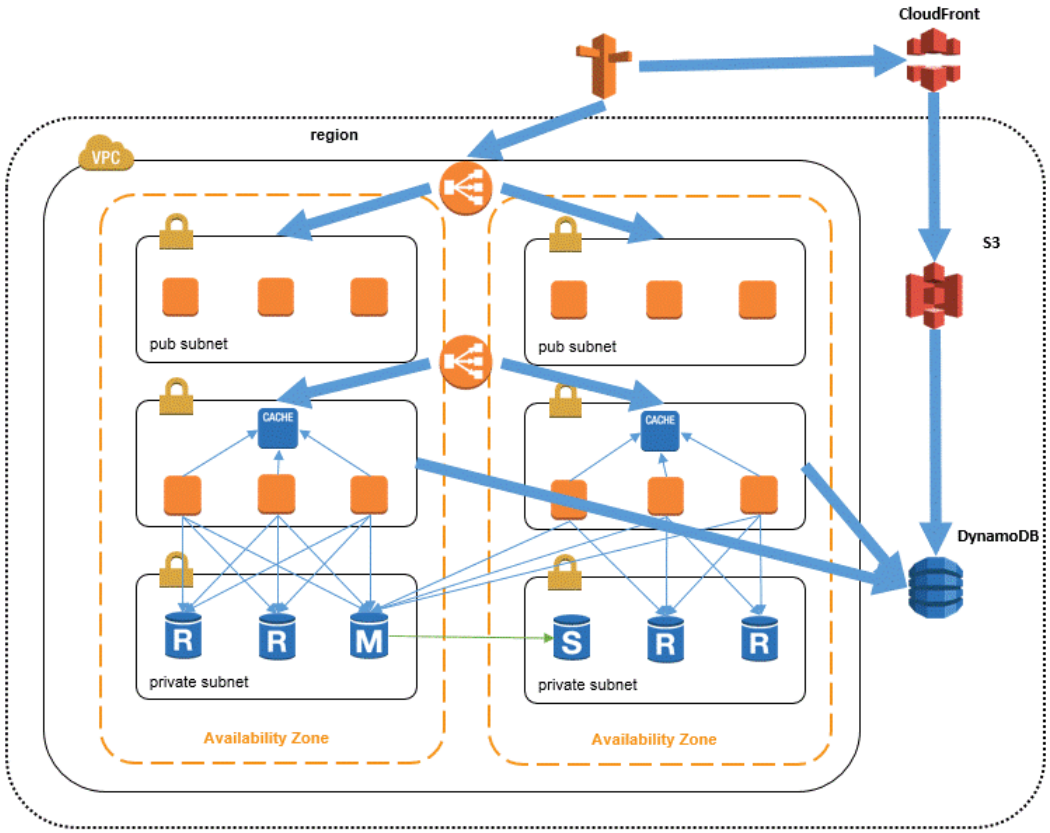
The AWS platform offers services in many flavors so that you can provision just enough performance, availability, and durability depending on your requirements. Your ability to be aware and leverage the entire palette of AWS services will be key to adding scalability to your application.

Amazon Simple Storage Service (Amazon S3)
I could store assets on web servers (EC2 instances), but that solution is harder to scale in the long term and not very cost-effective. For instance, as traffic grows, requests for static assets will be competing with requests for web pages. I will have to monitor the bandwidth utilization on those EC2 instances and potentially upgrade to instance types with more bandwidth. Another challenge is synchronizing assets across web servers. If I keep all assets on one server, I will have to spin up a large instance just to store and serve the assets. If I spread the assets across all web servers, I will need to implement application logic to intelligently fetch assets from different servers.

All these challenges can be overcome with Amazon Simple Storage Service (Amazon S3). S3 is a highly durable and available object storage solution that is great for storing videos, log files, and image files. S3 achieves 11 9s of durability by making multiple copies of data across a region. Objects stored in S3 are widely accessible through RESTful APIs. S3 can serve as a central static asset repository that knows to partition storage automatically to increase performance as the number of requests increases over time. S3 objects are globally accessible through URLs, and saves you from having to pay for high bandwidth EC2 instances.
Amazon CloudFront
When you use S3 as a central repository for assets, you can leverage a content distribution network (CDN) service on AWS to scale further by caching and serving your frequently requested content from 50+ edge locations around the globe. Amazon CloudFront is capable of serving both static and dynamic content, supports SSL certificates, and costs you zero dollars for data transferred from an origin (EC2 or S3) to CloudFront.

Amazon DynamoDB
It's worth pointing out DynamoDB again because it can help you scale by storing session or state data. You can store relational data in RDS and offload session data to DynamoDB. It's a managed NoSQL database that lets you fine tune performance with provisioned throughput. It is fast, predictable, and highly secure. If you have key-value format session data, using DynamoDB can let you offload work from a primary database and increase overall system performance.

Amazon ElastiCache
Lastly, I recommend Amazon ElastiCache to hold frequently accessed data and further decrease load from a primary database. ElastiCache is protocol compliant with Redis and Memcached. This means if you are using Redis or Memcached today, you don't need to refactor code to start using ElastiCache. A benefit of using ElastiCache is that it is a managed service. This means AWS will detect and replace unhealthy nodes for you. If you use ElastiCache Redis, you also have the option to stand up replicas in a different AZ to increase availability. The following illustration shows another updated architecture with Amazon ElastiCache.



In the next post, I will continue to iterate this architecture with the intention of serving 500,000 users. Auto Scaling will finally come into the picture. You will learn to use automation and monitor services to help you optimize the AWS infrastructure.

Summary

- Understand that SQL and NoSQL are different tools; each one is a great choice for a specific purpose.
  • Use managed services whenever you can on AWS. Managed services handle undifferentiated heavy lifting so you can focus on tasks that add differentiating value to your organization.
  • Design a horizontally scalable architecture across Availability Zones to increase application availability. AWS offers Elastic Load Balancing (ELB) to distribute traffic across EC2 instances.
  • Leverage RDS features to achieve redundancy and increase availability with ease. Multi-AZ creates a standby database to increase availability. Read Replicas can be used to scale and increase performance for read-heavy workloads.
  • Think about decoupling and spreading workload over appropriate AWS services to scale and to improve performance and efficiency. S3 is great for storing static assets; CloudFront helps you deliver content with lower latency; DynamoDB and ElastiCache allow you to offload session and cache data from a main architecture.

Continue reading: Scaling on AWS (Part 3): >500K Users

References
Scaling Up to Your First 10 Million Users

TAGS: Guides & Best Practices, Startup Spotlight

Resources

Top Posts
Getting Started
How to Guides
Founder Stories
Tips and Tools
AWS Startup Events