Scale Your Web Application — One Step at a Time

Scale Your Web Application section consists of - a) Easing your server load, b) Reduce write requests c) Reduce read load by adding more read replicas and a general overview of Building Scalable Cloud Architectures.

Scale Your Web Application — One Step at a Time

Often teams experiencing frustration as they attempt to scale their application particularly around the cost and complexity related to scaling. Customers think scaling, it often means horizontal scaling and microservices, but usually, people aren't sure about how to dive in and effectively scale their sites.

Now let's talk about different scaling options. For instance, if your current workload is in a traditional data center, you can leverage the cloud for your on-premises solution. This way you can scale to achieve greater efficiency with less cost. It's not necessary to set up a whole powerhouse to light a few bulbs. If your workload is already in the cloud, you can use one of the available out-of-the-box options. Designing your API in microservices and adding horizontal scaling might seem like the best choice unless your web application is already running in an on-premises environment and you'll need to quickly scale it because of unexpected large spikes in web traffic.

So how to handle this situation? Take things one step at a time when scaling and you may find horizontal scaling isn't the right choice, after all. For example, assume you have a tech news website where you did an early look review of an upcoming and highly anticipated smartphone launch, which went viral. The review, a blog post on your website, includes both video and pictures. Comments are enabled for the post and readers can also rate it. For example, if your website is hosted on a traditional Linux with a LAMP stack, you may find yourself with immediate scaling problems.

Let's get more details on the current scenario and dig out more:

Where are images and videos stored? How many read/write requests are received per second? Per minute? What is the level of security required? Are these synchronous or asynchronous requests? We'll also want to consider the these synchronous or asynchronous requests? We'll also want to consider the following if your website has a transactional load like e-commerce or banking:

How is the website handling sessions? Do you have any compliance requests—like the Payment Card Industry Data Security Standard (PCI DSS compliance)—if your website is using its own payment gateway? How are you recording customer behavior data and fulfilling your analytics needs? What are your loading balancing considerations (scaling, caching, session maintenance, etc.)?

So, if we take this one step at a time:

Step 1: Ease server load

We need to quickly handle spikes in traffic, generated by activity on the blog post, so let's reduce server load by moving image and video to some third party content delivery network (CDN). A CDN solution, which is highly scalable with built-in security to verify origin access identity and handle any DDoS attacks. CloudFront can direct traffic to your on-premises or cloud-hosted server with its 113 Points of Presence (102 Edge Locations and 11 Regional Edge Caches) in 56 cities across 24 countries, which provides efficient caching.

Step 2: Reduce read load by adding more read replicas

MySQL provides a nice mirror replication for databases. Oracle has its own Oracle plug for replication and AWS RDS provide up to five read replicas, which can span across the region and even the Amazon database Amazon Aurora can have 15 read replicas with Amazon Aurora auto-scaling support.

If a workload is highly variable, you should consider Amazon Aurora Serverless database along with amazon Lamda to achieve high efficiency and reduced cost. While most mirror technologies do asynchronous replication, AWS RDS can provide synchronous multi-AZ replication, which is good for disaster recovery but not for scalability. Asynchronous replication to mirror instance means replication data can sometimes be stale if network bandwidth is low, so you need to plan and design your application accordingly.

It is recommended that you always use a read replica for any reporting needs and try to move non-critical GET services to read replica and reduce the load on the master database. In this case, loading comments associated with a blog can be fetched from a read replica as it can handle some delay in case there is an issue with asynchronous reflection.

Step 3: Reduce write requests

This can be achieved by introducing queues to process the asynchronous message. Amazon Simple Queue Service (Amazon SQS) is a highly scalable queue, which can handle any kind of work message load. You can process data, like rating and review; or calculate Deal Quality Score (DQS) using batch processing via an SQS queue. If your workload is in AWS, It is recommended using a job observer pattern by setting up Auto Scaling to automatically increase or decrease the number of batch servers, using the number of SQS messages, with Amazon CloudWatch, as the trigger. For on-premises workloads, you can use SQS SDK to create an Amazon SQS queue that holds messages until they're processed by your stack. Or you can use Amazon SNS to fan out your message processing in parallel for different purposes like adding a watermark in an image, generating a thumbnail, etc.

Step 4: Introduce a more robust caching engine

You can use something like Amazon Elasticache for Memcache or Redis to reduce write requests. Memcached and Redis have different use cases so if you can afford to lose and recover your cache from your database, use Memcache. If you are looking for more robust data persistence and complex data structure, use Redis. In AWS, these are managed services, which means AWS takes care of the workload for you and you can also deploy them in your on-premises instances or use a hybrid approach.

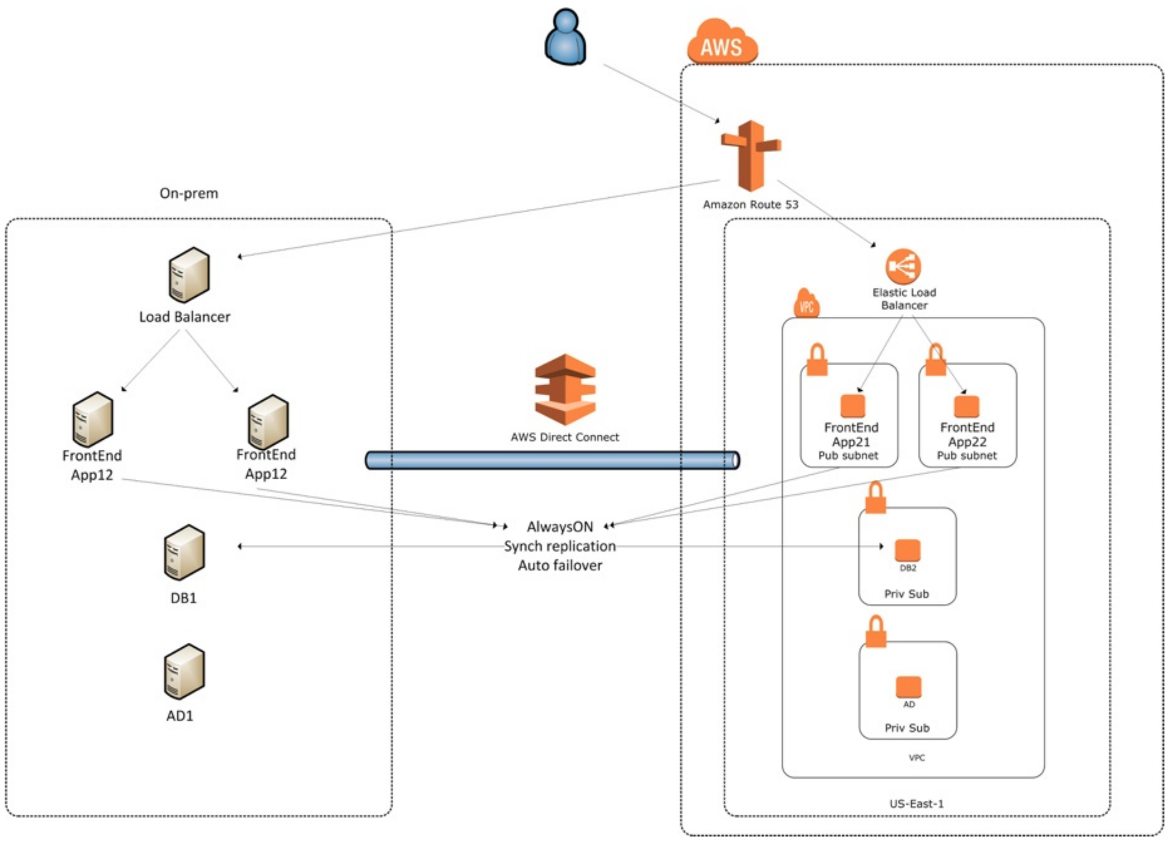
Step 5: Scale your server

If there are still issues, it's time to scale your server. For the greatest cost-effectiveness and unlimited scalability, It is always suggested using horizontal scaling.

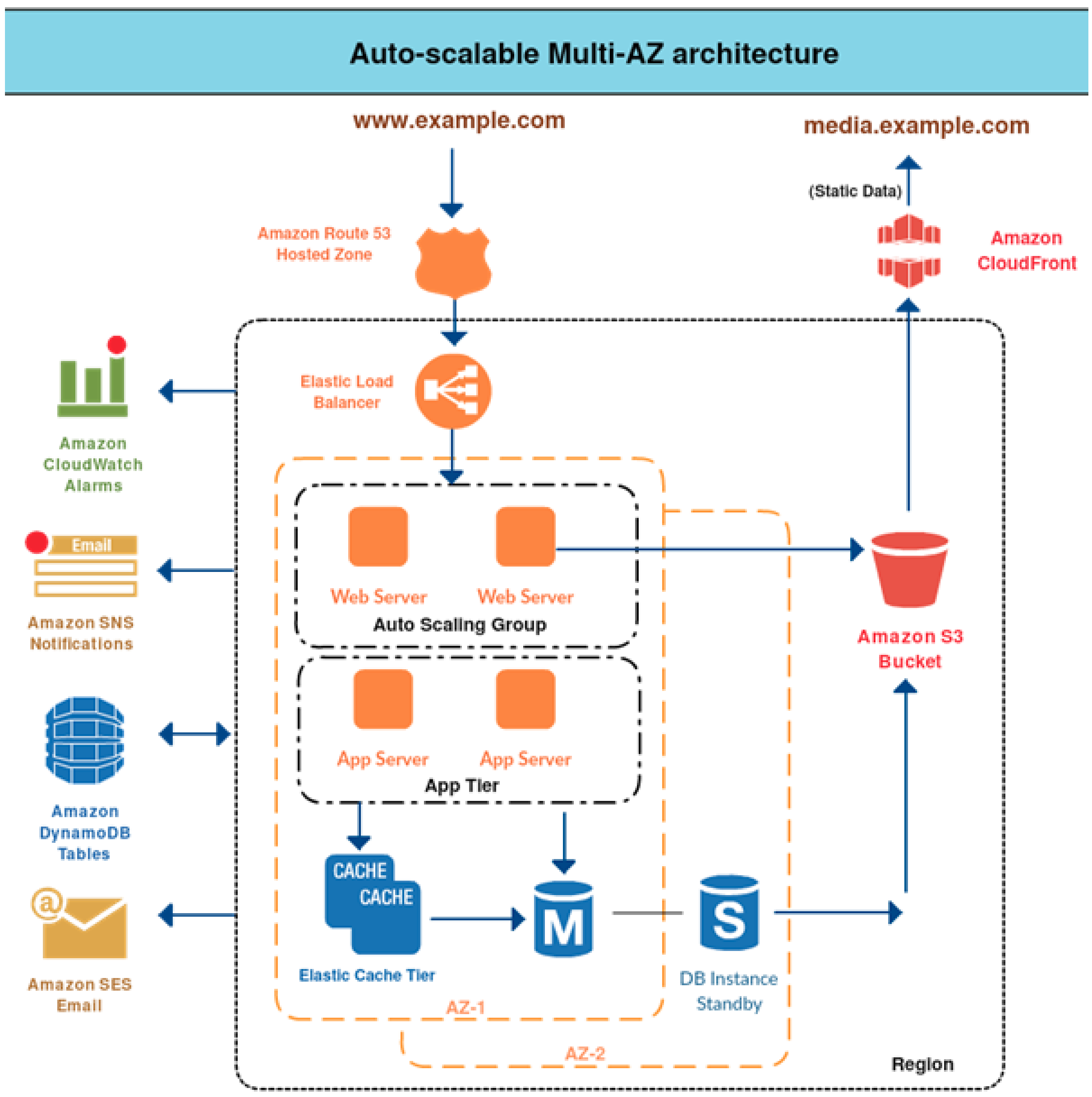
However, use cases like database vertical scaling may be a better choice until you are good with sharding; or use Amazon Aurora Serverless for variable workloads. It will be wise to use Auto Scaling to manage your workload effectively for horizontal scaling. Also, to achieve that, you need to persist the session. Amazon DynamoDB can handle session persistence across instances.

If your server is on premises, consider creating a multisite architecture, which will help you achieve quick scalability as required and provide a good disaster recovery solution. You can pick and choose individual services like Amazon Route 53, AWS CloudFormation, Amazon SQS, Amazon SNS, Amazon RDS, etc.

Route 53, AWS CloudFormation, Amazon SQS, Amazon SNS, Amazon RDS, etc. depending on your needs.



In this architecture, you can run your regular workload on premises, and use your AWS workload as required for scalability and disaster recovery. Using Route 53, you can direct a precise percentage of users to an AWS workload. If you decide to move all of your workloads to AWS, the recommended multi-AZ architecture would look like the following:



In this architecture, you are using a multi-AZ distributed workload for high availability. You can have a multi-region setup and use Route53 to distribute your workload between AWS Regions.

CloudFront helps you to scale and distribute static content via an S3 bucket and DynamoDB, maintaining your application state so that Auto Scaling can apply horizontal scaling without loss of session data. At the database layer, RDS with multi-AZ standby provides high availability and read replica helps achieve scalability.

This is a high-level strategy to help you think through the scalability of your workload by using AWS even if your workload in on premises and not in the cloud yet.

It is highly recommended creating a hybrid, multisite model by placing your

It is highly recommended creating a hybrid, multisite model by placing your on-premises environment replica in the public cloud like AWS Cloud, and

using Amazon Route53 DNS Service and Elastic Load Balancing to route traffic between on-premises and cloud environments. AWS now supports load balancing between AWS and on-premises environments to help you scale your cloud environment quickly, whenever required, and reduce it further by applying Amazon auto-scaling and placing a threshold on your on-premises traffic using Route 53.

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, you cannot leverage all that scalability in infrastructure if your architecture is not scalable. Both have to work together.

You will have to identify the monolithic components and bottlenecks in your architecture, identify the areas where you cannot leverage the on-demand provisioning capabilities in your architecture and work to refactor your application in order to leverage the scalable infrastructure and take advantage of the cloud.

Characteristics of a truly scalable application:

1. Increasing resources results in a proportional increase in performance
2. A scalable service is capable of handling heterogeneity
3. A scalable service is operationally efficient
4. A scalable service is resilient
5. 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 your application and if you design your architecture with the above characteristics in mind, then both your architecture and infrastructure will work together to give you the scalability you are looking for.