AWS CUSTOMER PROPOSAL

This document presents a proposed architecture for migrating your **Web Application** to Amazon Web Services (AWS). Goals achieved in this proposal include *Scalability and Durability*, *Latency Reduction*, *Self-recovery*, *Security*, and integration with existing *On-Premise Components* all while keeping close to Infrastructure-as-a-Service (IaaS) options to simplify the transition to the cloud.

The next section of this document details the requirements and AWS services integrated to meet them (assumptions noted inline). Following that is a list of other/future considerations and opportunities. Finally, we provide an architecture diagram of the high-level design and flow.

SOLUTION DETAIL

The following sections break down the solution details by requirements and application components. For each part, the recommended AWS technologies are noted in ***Bold Italics***.

Scalability, Reliability, and Performance

To distribute load and allow the application to respond elastically to changing demand, we've front-ended multiple ***Amazon EC2*** instances with an ***Elastic Load Balancer***. We've chosen the ***Application Load Balancer*** (or ALB) flavor since it operates at OSI layer 7 and can do path-based routing as well as several other application-level features.

The application instances utilize ***Amazon EC2 Auto Scaling*** based on the ***Amazon CloudWatch*** CPU metric (typically the resource that constrains web app performance). At a configurable threshold, the autoscaling group launches additional load-balanced instances using a versioned ***Launch Template*** to support increased load. These instances launch in multiple ***Availability Zones*** to ensure that the application remains fault tolerant. CloudWatch instance recovery is also configured for them.

Database

For the database component, we've leveraged ***Amazon RDS*** in our design to provide ***Multi-AZ Cluster*** capabilities with minimal setup and management.[1] Non-master nodes can be used as ***Read Replicas*** in most cases for improved application performance.

The ***AWS Database Migration Service*** could potentially be used to reduce downtime required by syncing live data to AWS ahead of time.

Amazon Route 53 is also included in the design to provide a highly available DNS layer and advanced routing capabilities.

Content & Shared Storage

To maximize end-user performance (and reduce perceived latency), the ***Amazon CloudFront*** content delivery network (CDN) is

[1] Amazon RDS available for MySQL, PostgreSQL, Oracle, SQL Server, and MariaDB (other database engines may require an EC2-based approach)

configured to front the application requests. This component could be rolled out initially, even with an on-premise origin. Not only does CloudFront provide caching and ~150 global points of presence, it also includes a web application firewall (***AWS WAF***) and managed distributed denial of service protection (***AWS Shield***).

For the shared storage itself, ***Amazon S3*** object storage is likely the ideal choice due to its scalability, integration with CloudFront, and ease-of-use. Should the application require more traditional file-based storage, we can utilize ***Amazon Elastic File System*** (EFS) to provide a shared, NFS mountable, shared storage path on each instance.

Migration possibilities include ***AWS Snowball*** (physical data transport), or ***AWS DataSync*** (network-based transport) depending on data size and available cloud connectivity.

Security & Access

To provide end-to-end *in-transit* encryption, all services passing data are utilizing either their built-in encryption or HTTPS / IPSec / etc. secure tunnels. For *at-rest* encryption, all services utilize native capabilities. In many cases, the ***AWS Key Management Service*** can be used to support customer-provided encryption keys for an added layer of security.

The initial setup exposes only necessary components to the Internet. Application servers themselves sit in a private network, accessible only via a "bastion" or "jump box".

AWS Systems Manager could replace the jump box altogether, allowing access to run commands on instances without the additional attack surface of a jump box.

Of course, the account setup also follows a least-privilege approach, utilizing ***IAM Roles*** for system components, individual non-root ***IAM Users*** (with ***AWS Multi-Factor Authentication*** required), or single sign-on integration with an existing directory (via ***IAM SAML*** or ***IAM OIDC***) for personnel.

Connectivity

The design includes a secure ***AWS VPN*** (site-to-site) for connecting on-premise and cloud systems (and private access to AWS). Should the tunnel prove to be insufficient, a dedicated circuit and bandwidth are available via ***AWS Direct Connect***.[2]

Bulk Updates & Cost

For cost-effective bulk customer updates, we've created a batch update system utilizing ***Amazon EC2 Spot Instances*** via ***Amazon EC2 Fleet***. This approach allows simple management of reduced-cost EC2 compute while maintaining a baseline of capacity/ performance.

Overall cost savings come from leveraging ***Amazon EC2 Reserved Instances*** for long-running servers, and ***AWS Cost Explorer / Budgets*** for budget tracking, alerts, and detailed reporting.

Backup

While the architecture presented is designed for fault-tolerance, it's still essential to back up what's running *in* the cloud as well (e.g., in case of rollbacks, accidental deletion, etc.). ***AWS Backup*** centralizes the policies and management of snapshot-based backups for EBS, EFS, RDS, and other services.

[2] Specifics depend on geographic location and carrier availability

These backups initially sit in a *warm* storage tier backed by S3, then, via a ***Lifecycle Policy***, are transitioned to ***Amazon S3 Glacier*** for *cold* (long-term) storage.[3]

Other Considerations & Opportunities

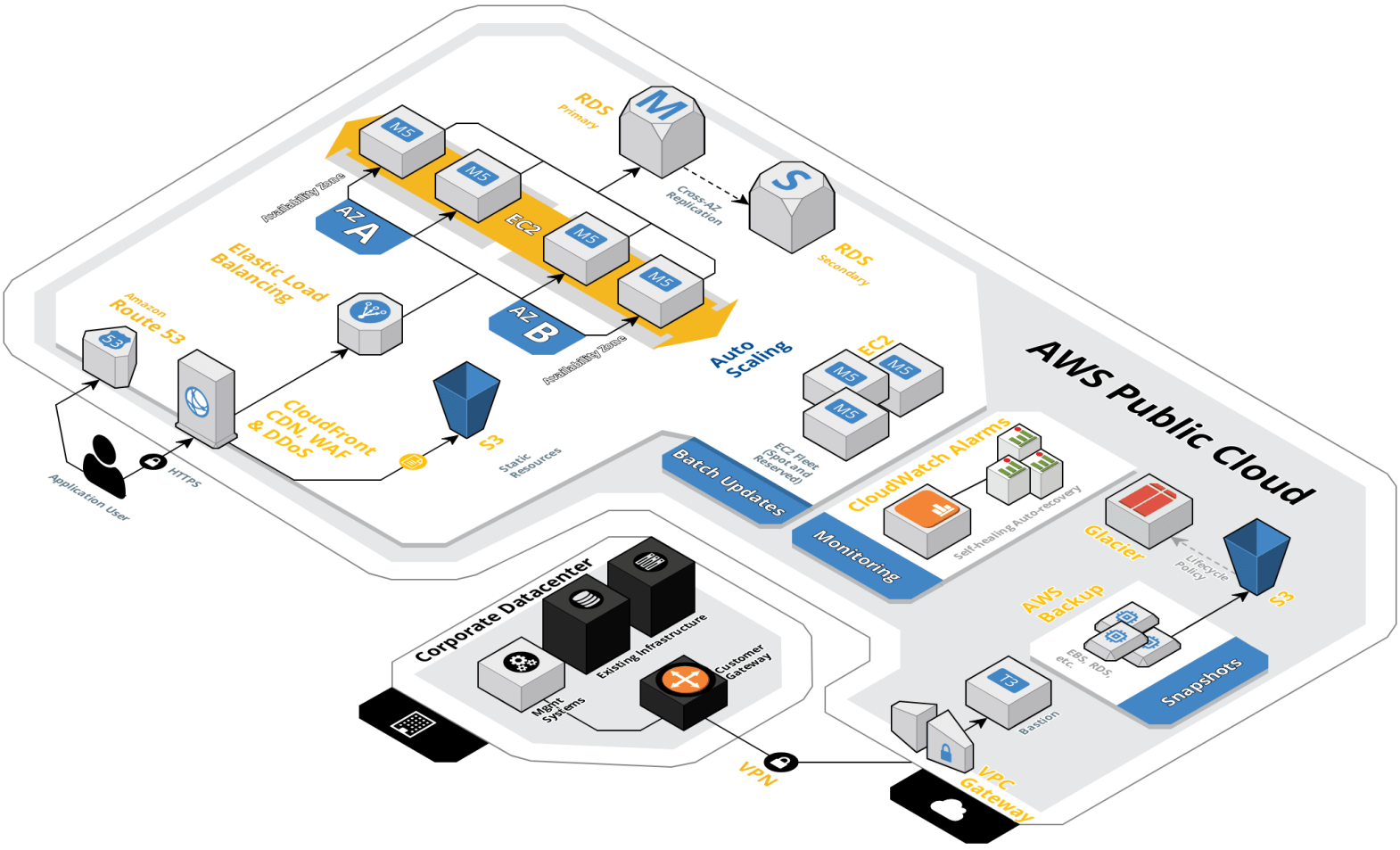
To ensure that optimal processes and supports are in place for this migration, we recommend starting with an ***AWS Well-Architected Review***. The review results are captured in the console and used to measure future improvement.

This review, combined with ***AWS Trusted Advisor*** guidance, helps ensure that this workload continues to follow evolving best-practices over time.

Future iterations (or stretch goals) could incorporate a serverless or container approach (***AWS Lambda***, ***Amazon Elastic Container Service*** / ***AWS Fargate***). Configuration management via ***AWS OpsWorks*** or infrastructure-as-code automation using ***AWS CloudFormation*** could provide improved repeatability and self-documenting systems.

ARCHITECTURE DIAGRAM

The diagram below illustrates the major components, connectivity, and high-level flow.



[3] Lifecycle policy can include deletion after retention period expiration