AWS Architecture Design - Start-up

Solution Objective

A manageable, secure, scalable, high performance, efficient, elastic, highly available, fault tolerant and recoverable architecture that allows the startup to organically grow.

This **Solution Document** elucidates the measurable requirements hereby with the solution sketched below addressing few key service mappings of AWS to cater the requirements of **Start-up** enabling their customers to interact real time.

Solution Plan *(Proposed)*

In order to substantiate the rapid and significant growth of the mobile application, the proposed solution should enable few key metrices, namely

- Manage User Identities along with User Data Sync
- Distribution of Load (effectively)
- Rapid & on-demand Scaling
- Security of Data (at rest, in transit), Encryption Process
- High Performance, throughput of the database
- A self-healing infrastructure, DR Mechanisms
- Environment Tiering Strategy
- Proper communication, notification mapping(s) of outcomes
- Data Storage, Archival Strategy
- Analytical Capability from the data-sets

To start with, for managing the user authentication and authorization – we map **AWS Cognito** service which lets you add user sign-up, access control of the mobile app easily. It felicitates scaling up to millions of users and supports sign-in with enterprise identity providers via SAML 2.0. In addition, it also integrates with social identity providers such as Facebook, Google and Amazon for secure and scalable user directory using User Pools as a fully managed service. The user pool is a standards-based identity provider supporting identity & access management standards such as OpenID Connect, SAML 2.0 and OAuth 2.0. It supports multi-factor authentication and encryption of data at rest, in transit in parallel of being complaint with multiple industry standard compliance protocols. The role definition maps users to roles and enables authorization accordingly. It has an easy integration with the app-launch and the UI is to be customized with the company branding front hereby enabling center of all user loads & interactions.

The flow post authentication and authorization move ahead to **AWS API Gateway** which enables the integration of REST & WebSocket APIs acting as a front door for the mobile application to access business logic, data, the functionalities from backend services i.e. the workloads running on EC2s, lambda.

The **load balancer (LB)** automatically distributes the incoming traffic across the insourced targets. It handles the varying load on the applications efficiently making the system highly available & fault tolerant. The (Layer-7) LB identifies the app route requirements & routes the requests accordingly to the EC2 Instances or the Lambda.

The workload is distributed within **2 Availability Zones (AZ)**. The AZ-1 has the public subnet catering to the web-sourced traffic workloads through the App-Servers, Auto-Scaling enabled based on the requirement of the application getting processed through EC2 instances. The AZ-2 hosts the standby replica of the relational database instance, just in case of a failover. *(ensuring High Availability of the database)*

The data processed from AZ-1 EC2s moves into the **AWS managed Relational DB Service – RDS** for data storage *through* the **AWS Kinesis** which felicitates the easy collection, processing of data streams in "*real time*" at any scale.

Kinesis also enables ingesting real time data such as video, audio, application logs, website clickstreams, IoT telemetry data for machine learning, analytics by analyzing the data as soon as it arrives rather that making the data to go through storage & collection as a requirement.

AWS Lambda, a serverless compute works on auto-triggers based on the API request type invokes, thus processes responses to triggers such as change in data, request of repetitive information (through Elastic-Cache), shift in system states. Integrates well with S3, Kinesis and CloudWatch.

AWS Elastic-Cache (Redis) offers in-memory data stores. Enables the data intensive applications by improving the performance of the app by retrieving data from high throughput and low latency in memory data stores. It stores ephemeral data in-memory for sub-millisecond response.

AWS S3 is used for the object storage capability and integrated along with **AWS Glacier** through its "Lifecycle Policies" for archival of data.

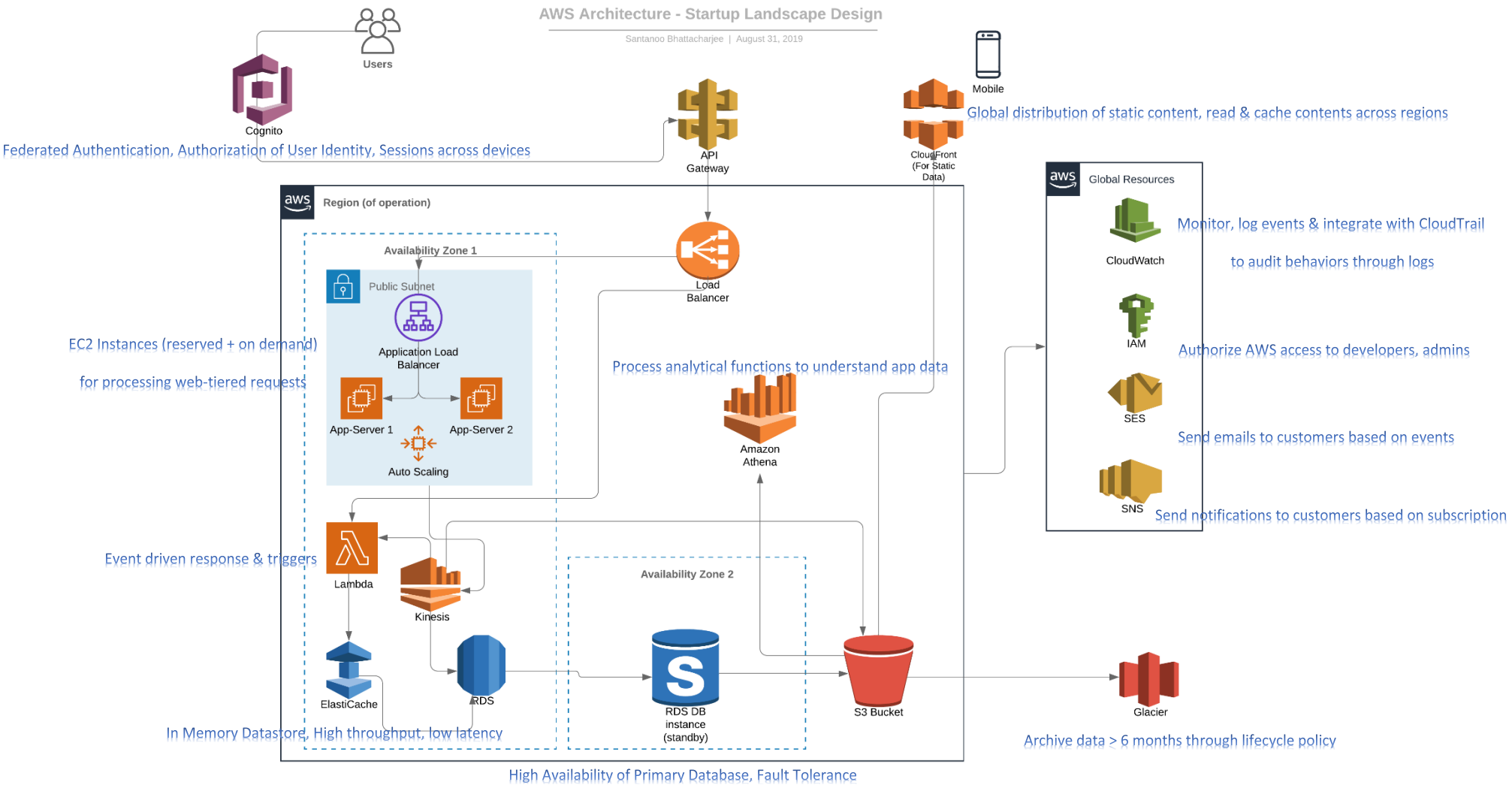
AWS Athena is integrated with the S3 service for interactive queries to analyze the data-sets. The collected data after being streamed, refined and stored on the S3 buckets over the time & Athena enables us to run analytics on it hereby to be able to visualize and understand app-data usage etc.

The Static content through S3 would be resonated to **AWS CloudFront** for distribution (Content Delivery) to customers globally with low latency, high transfer speeds.

The AWS Global Services are used along with all the specified services, deeply integrated as in when needed. **CloudWatch** enables the detailed monitoring, periodic & measured invokes as in when needed. **IAM** felicitates security of the environment by maintaining AWS User Groups, User & Service Policies for the developers, administrators as the delivery team expands. **AWS SNS** and **AWS SES** are notification & email services mapped to the triggers for the AWS services to send notifications to consumers.

In addition to this, **AWS CloudFormation** templates would be used for managing the environments, replicating it across multiple environment tiers based out of the standards we have as the blueprint for the environments.

Proposed* Architecture: AWS *Landscape Design



Solution Flow

1. Through AWS Cognito “App” users would be authenticated via user pools created or through social logins, federated authentications using SAML. The process validates the users securely and forwards authorized content request to the API Gateway.
2. The API Gateway handles all the tasks involved in accepting and processing thousands of concurrent API calls hereby acting as a front door for applications - which runs the business logic and workloads. The requests are forwarded to the Elastic Load Balancer.
3. The Elastic Load Balancer (*internet facing*) would use a round-robin load distribution mechanism (based on Layer-7 protocol) & send the request either to the App-Servers (*if it needs to be freshly processed*) OR to the lambda function (*if it already has the response function defined)*
4. The Public Subnet hosting the App-Servers would internally balance the incoming traffic on it through internal application load balancer. The EC2 instances would be processing the request(s) & move the same to the RDS DB Instance through Kinesis.
5. Kinesis would enable the data stream to be available “real-time” for further processing of it (e.g. if that is integrated with the logic of user driven behavioural patterns, suggestion mechanism etc.) and then move the data forward to RDS for cataloguing it accordingly.
6. The RDS Standby instance hosted through Multi-AZ mechanism enables AWS Primary RDS (hosted in AZ-1) to automatically replicate data onto a secondary instance (hosted in the AZ-2). In the event of a failure, automatic failover to the standby database instance takes place.
7. The Elastic Cache allows the quick setup of caching the app usage, patterns of processing thus handling incoming “repetative” calculation intensive requests on its In-Memory datastore hereby increasing throughput & reducing the latency.
8. The objectstore S3 stores the data-set and scales up the data availability, durability & replicaion forward to CloudFront for static contents distribution across multiple geographical regions.
9. The Athena service consumes standard file-data stored onto the S3 and runs interactive query service through SQL on it to analyze the data for user-behavioural patterns, app-data metrices felicitating analytical driven decisions.
10. Periodically, the data from S3 moves to Glacier (over 6 months) through the lifecycle policy associated for archival of data.
11. The Global Resource services CloudWatch, SNS, SES, IAM works for monitoring the functionalities - overall service usage, health & availability, notification to subscribers, email deliveries of events & authentication to access the AWS environment(s) & services by developers respectively.

Assumptions, Addressing the Concerns

- Scaling to meet the demand, but with uncertainty around when and how much this demand will be – they are very concerned about buying too much infrastructure too soon or not enough too late!
  Usage of Compute Tiers: Reserved & OnDemand + Spot Instances – Positioning them appropriately on placement groups & defining Launch Templates
  Usage of Storage: S3 being highly scalable, durable & reliable storage option – maintaining availability & usability of data, managing data growth, retention through lifecycle policies.
  Usage of DB Tier: using managed RDS DB instance & designing Fault Tolerance, HA Mechanism for the database across zones.
- Disaster Recovery planning, Ability to easily manage and replicate multiple environments based on their blueprint architecture
  Defining DR Template with decision points of RTO & RPO. Ensuring Data residency, replication process with Multi-AZ availability. The single point of failure(s) is made highly available by cross replication & proactive monitoring thresholds along with CloudFormation Templates standardized & archived for instant recovery in case of a catastrophic situation. CloudFormation templates also strengthens the blueprint for replication as multiple environments.
- Manage user identities & sync user specific data across multiple devices
  Cognito by built-in functionality enables the use of authentication, authorization & sync along devices appropriately.
- Ability for Service Providers to send notifications to consumer
  Through SNS, SES, CloudWatch & Lambda – users are appropriately notified instantly/periodically based on the events they subscribe for.
- Ability to run analytics on top of collected data, with analytics they should be able to visualize & understand app data usage
  By using Kinesis, Athena – The data-sets are analyzed, metrices are refined appropriately thus could be visualized to study the flow & usage of the App.
- Their ability to configure their database and data access layer for high performance and throughput
  Elastic Cache enables In-Memory datastore hereby felicitating transactional, repetitive queries etc. to be cached & retrieved appropriately thus increasing the performance & throughput in addition to reducing the overall latency, which could arise with repetitive DB operational calls if done on R-DBMS always.
- Effective distribution of load
  Reliable & efficient distribution of traffic to healthy app servers, lambda functions. It scales itself up to meet the demand of application traffic. In addition, it also integrates with autoscaling to ensure that more instances are available to cater the demand, whenever needed.
- A self-healing infrastructure that recovers from failed service instances
  Auto-Scaling & load balancing for mixed distribution with appropriate scaling policies, launch template invokes through CloudWatch monitoring metrices, Availability Zone distribution for automatic DB Failover, Self-healing Cloud Formation template stacks from blueprints across AZs during catastrophic situations.
- Security of data at rest and in transit
  Data at rest: Using Industry standard AES-256 encryption algorithm; Data in transit: using Transport Layer Security (TLS-1.2). Audit Data which isn't encrypted periodically through CloudTrail logs over CloudWatch.
- Securing access to the environment as the delivery team expands
  Using IAM Groups, Defined Access Policies & proper user management for Developers & Administrators.
- An archival strategy for inactive objects greater than 6 months
  Using AWS Glacier through lifecycle policy of S3 buckets, periodically (6 months, data is archived for retention).