# **AWS solution architecture for Startup**

* 1. **Purpose**

Provide a high-level vision of how the business solutions for the startup currently using the LAMP stack can be delivered in easily manageable, secure, high performance, scalable, efficient, fault tolerant and recoverable infrastructure by leveraging existing AWS services at a lower cost, quickly with flexibility to scale based on-demand.

* 1. **Scope**

This document describes the conceptual solution architecture for the startup environment using AWS services based on the analysis of the requirements. This conceptual architecture may evolve or refine further to incorporate architecture changes for business and AWS services

* 1. **Requirement Analysis**

1. Business needs the infrastructure solution to be cost effective while, able to meet future demands by scaling-in and out dynamically according to needs and fault tolerant.
2. The solution should cater for disaster recovery or business continuity with reasonable RPO and RTO to prevent business loss and reputation
3. Better user experience with low response time via browser with load balancing irrespective of the hosted location
4. Security for the data at rest and in transit as part of the solution proposed
5. Security for user and privilege/access management in place with clear roles defined for seamless supports during team expansion
6. Data lifecycle management process as part of the incorporated/proposed solution
7. Quick, automated and easily provision able infrastructure environment based on a blueprint architecture
   1. **Overall Assumptions and considerations for the various solutions proposed**

The solution proposed is based on simplified yet holistic approach towards fulfilling the requirement analysis and follows key guiding principles below

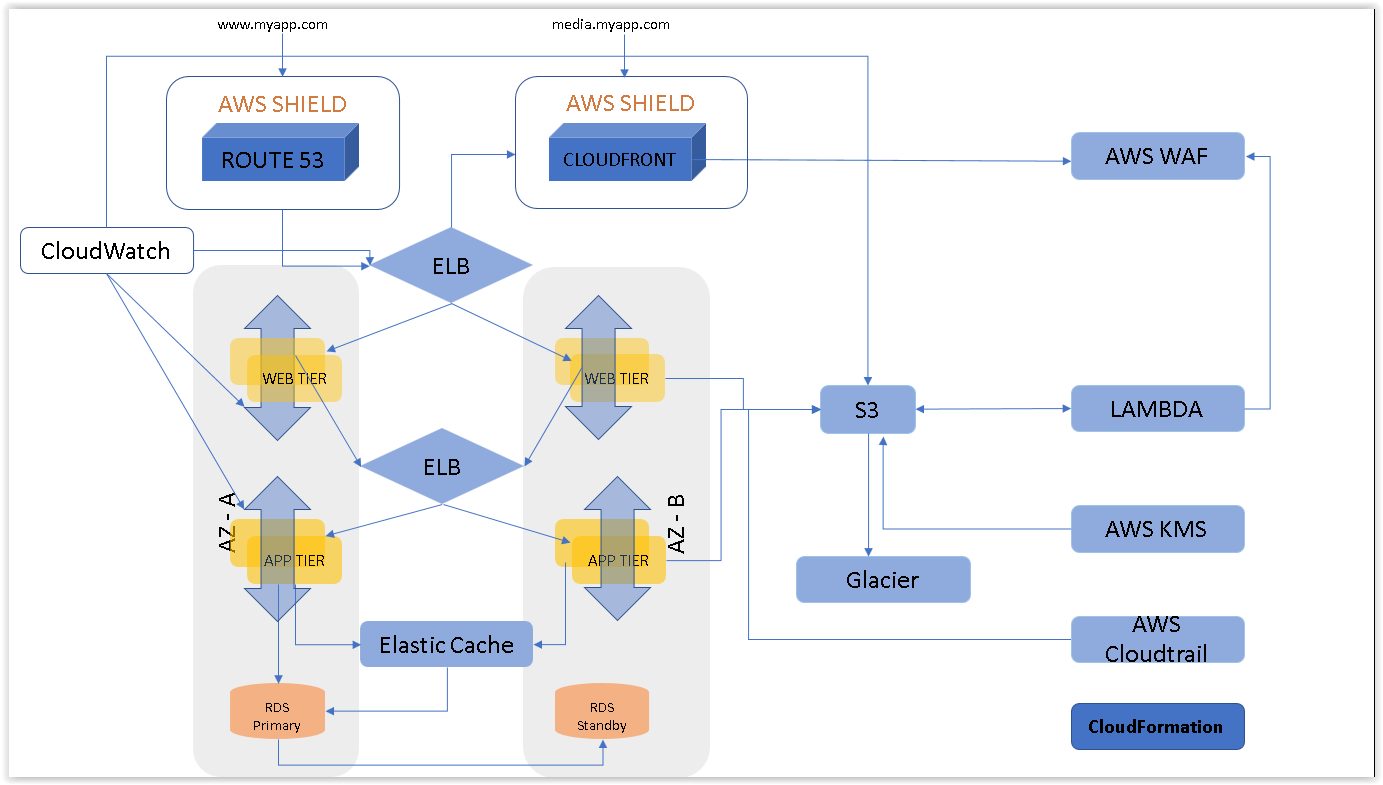
* Promote automation and self service via the AWS service offerings with ability to maintain infrastructure as a code for quick environment readiness
* Redundancy in-place for each of the components as part of the solution architecture to avoid single point of failure at every level
* Auto scaleup and scale down of the resources on all the 3 tiers based on real time load with effective load balancing services at different tiers
* Allow flexibility and scalability towards future changes on both the side (i.e. AWS as well as from startup business architecture perspective)
* Incorporated solution enables security and compliance on user’s access and privileges, making it compliant for review and audits at any point of time and real time
* Data lifecycle management incorporated for data based on business requirements to achieve better storage tiering, data retention and archival at optimal cost and SLA
* Data security incorporated for data at rest and data in transit achieved using AWS services
* Content delivery network as part of solution architecture for business customers/users to have a better response times to requests from any part of the world

**Solution 1: Normal 3 tier system with web-tier and application-tier in separate stateless processes with** **EC2**

**Characteristics and Assumptions:**

1. The solution will be hosted on EC2 instances with autoscaling at web and application tier based on cloudwatch metric and autoscaling policies defined for scaling-up and scaling-down
2. Object storage will be done at S3 and archived to Glacier based on Data lifecycle policies defined. Data will be encrypted at rest and in-transit using the encryption methods available at S3
3. The solution will be available across availability zones to provide HA and fault tolerance
4. Content delivery network will be taken can by Cloudfront and edge locations and security for the web application will be offer by AWS WAF
5. Solution will not have large batch jobs and will not have significant upload of contents to S3
6. Elastic load balancer will be available at web and application tier to better load balancing and overall performance
7. Cloudformation templates will be used to build parallel environments for other purposes like development, testing

**Flow Diagram:**

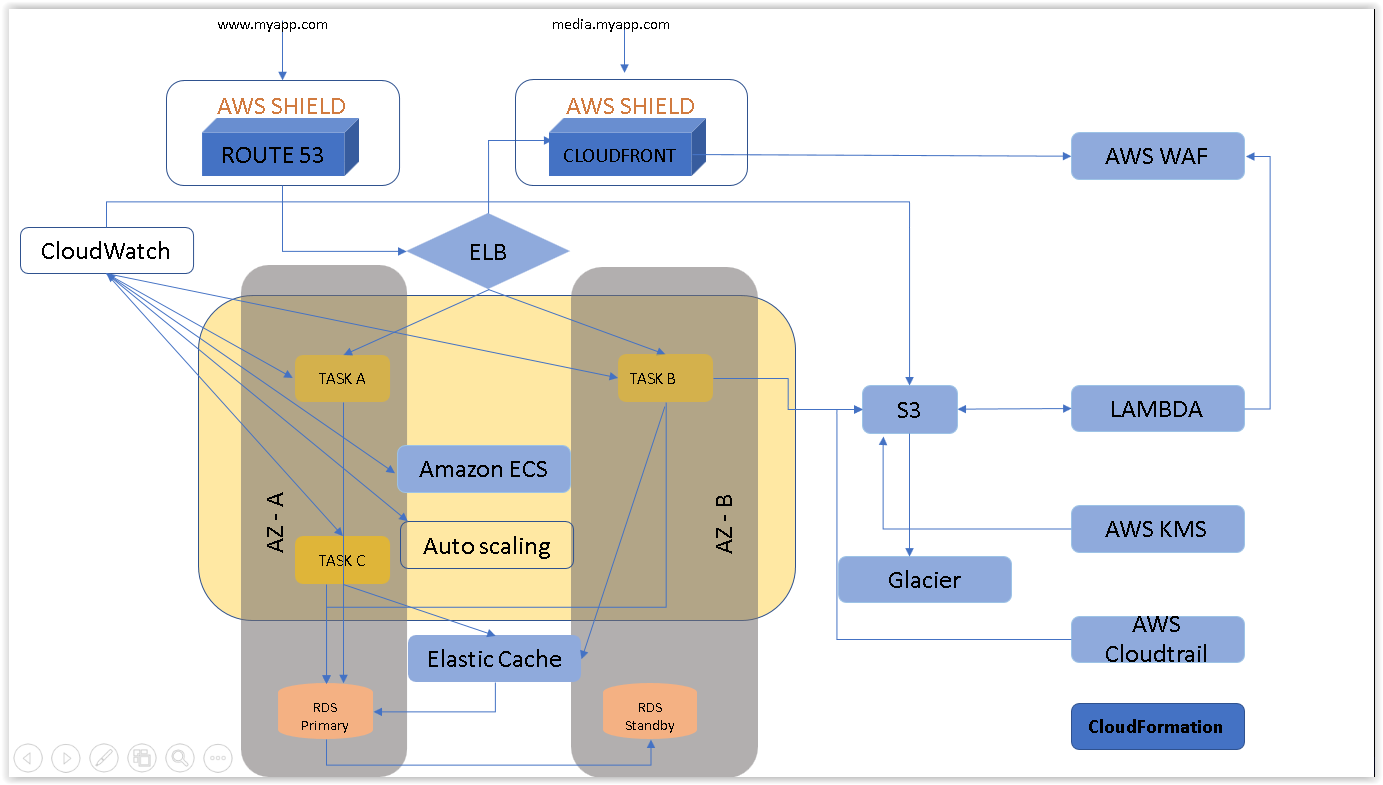


**Solution 2: Using Containers to Build a Microservices Architecture using EC2 Container Services (ECS)**

**Characteristics and Assumptions:**

1. Business components of the startup supports microservices architecture
2. The container instances comprising the cluster live in the customer’s VPC and are controlled and secured entirely by the customer.
3. Container instances communicate with the ECS service through an “agent,” which is a Docker container running on the instance. This is automatically included in the Amazon ECS-Optimized Amazon Linux AMI.
4. ECS includes shared state, optimistic schedulers for short running tasks and long running services.
5. Task and service state management is provided by a fully ACID compliant, distributed datastore.

**Flow Diagram:**

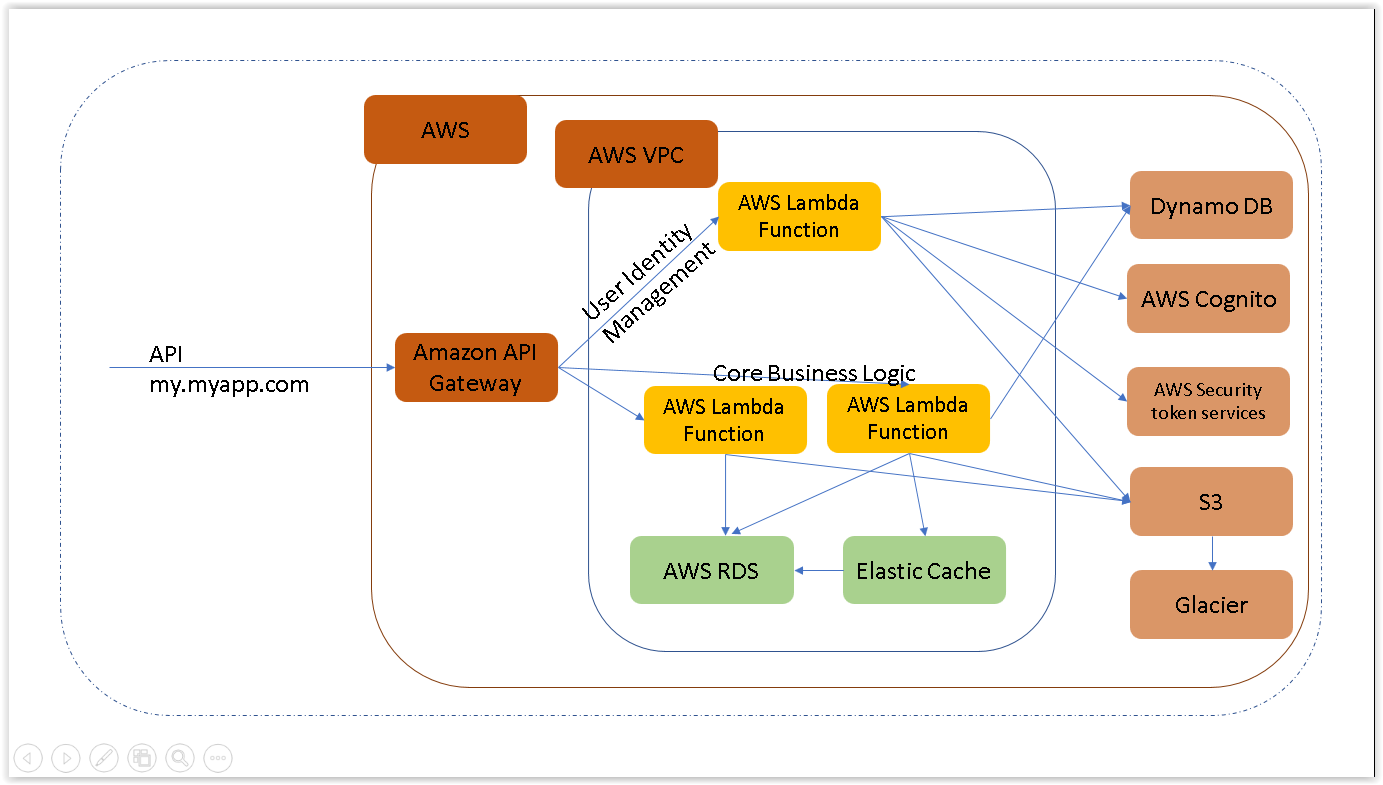


**Solution 3: AWS Serverless Multi-Tier Architectures (Mobile Back End)**

**Considerations and Assumptions:**

1. Business components of the startup can run in serverless microservices architecture and in future they will still be using the programming languages supported by AWS API gateway
2. Presentation layer will be a mobile application running on each user’s smartphone.
3. Amazon API Gateway and AWS Lambda will be the logic tier which is globally distributed by the Amazon CloudFront distribution created as part of each Amazon API Gateway API. A set of Lambda functions can be specific to user/device identity management and authentication, and managed by Amazon Cognito, which provides integration with IAM for temporary user access credentials as well as with popular third-party identity providers. Other Lambda functions can define the core business logic for mobile back end.
4. The various data storage services can be leveraged as needed with versioning and data lifecycle management based on archival policy defined based on business requirements

**Flow Diagram:**

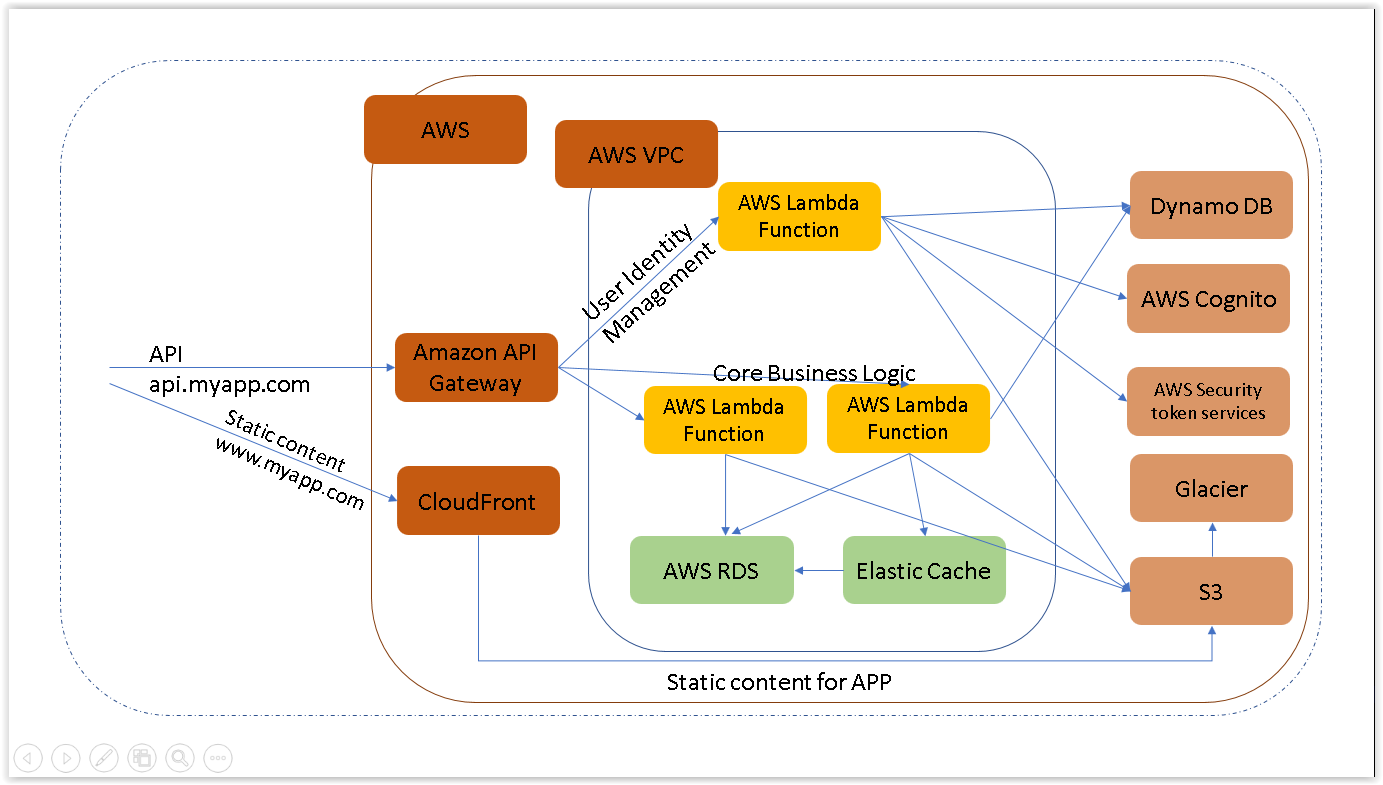


**Solution 4: AWS Serverless Multi-Tier Architectures (S3 Hosted Website)**

**Considerations and Assumptions:**

1. Major part of the business application is static content with minimal business logic
2. Business components of the startup can run in serverless microservices architecture and in future they will still be using the programming languages supported by AWS API gateway
3. Static website content hosted in Amazon S3, distributed by Amazon CloudFront.
4. Amazon API Gateway and AWS Lambda will be used to serve static web content hosted in Amazon S3 can directly integrate with Amazon API Gateway, to handle the backend logic which can be CORS compliant.
5. The various data storage services can be leveraged as needed with versioning and data lifecycle management based on archival policy defined based on business requirements

**Flow Diagram:**



**Solution 5: AWS Serverless Multi-Tier Architectures (Serverless Microservices Environment)**

**Considerations and Assumptions:**

* Repeated overhead for creating each new microservice, for core business logic
* Issues with optimizing server density/utilization,
* Complexity of running multiple versions of multiple microservices simultaneously, and
* Proliferation of client-side code requirements to integrate with many separate services.

(Amazon API Gateway even allows for the cloning of existing APIs).

**Flow Diagram:**

