



Version 1.1

Abstract

[Abstract summary of the document]

**Application Tracking System**

High Level Design Document

Contents

[1 Introduction 3](#_Toc127173242)

[2 System Overview 4](#_Toc127173243)

[What is application tracking system solution? 4](#_Toc127173244)

[3 Solution Architecture 5](#_Toc127173245)

[3.1 Solution Diagram 5](#_Toc127173246)

[ATS Microservices: 7](#_Toc127173247)

[Each service typically implements a set of distinct features or functionality and communicates across each other through a well-defined, lightweight mechanism to serve a business goal. 7](#_Toc127173248)

[4 Design Considerations 8](#_Toc127173249)

[4.1 System architecture guidelines 8](#_Toc127173250)

[4.2 Assumptions and Dependencies 9](#_Toc127173251)

[5 NFR Considerations 10](#_Toc127173252)

[5.1 Scalability 10](#_Toc127173253)

[5.2 Usability 10](#_Toc127173254)

[5.3 Reliability 10](#_Toc127173255)

[5.4 Performance 10](#_Toc127173256)

[5.5 Security 11](#_Toc127173257)

[5.5.1 User Authentication 11](#_Toc127173258)

[5.5.2 User Authorization 12](#_Toc127173259)

[5.5.3 Service Level Authentication 12](#_Toc127173260)

[5.6 Supportability 12](#_Toc127173261)

[5.7 Infrastructure Requirements 12](#_Toc127173262)

[5.8 Components Description 14](#_Toc127173263)

[5.8.1 API Gateway 14](#_Toc127173264)

[5.8.2 Central Configuration 15](#_Toc127173265)

[5.8.3 Circuit breakers 15](#_Toc127173266)

[5.8.7 Design Pattern Considerations 16](#_Toc127173267)

[6 Technology stack / Framework/Infrastructural Components 17](#_Toc127173268)

[6.1 Logging 17](#_Toc127173270)

[6.2 Exception Handling 17](#_Toc127173271)

[6.3 Security 17](#_Toc127173272)

[6.4 Data Access 18](#_Toc127173273)

[6.5 Object Creation 18](#_Toc127173274)

[6.6 Data Transfer between Layers 18](#_Toc127173275)

[7 Integration (External Systems) Architecture 20](#_Toc127173276)

[7.1 Integration 20](#_Toc127173277)

# Introduction

The purpose of this High Level Design (HLD) Document is to present the structure of the application tracking system solution that enables organization to provide full set of services to digitize hiring and improve the efficiency of recruitment teams. This documentcan be used as a reference manual for how the modules interact at a high level.

The HLD will:

• present all of the design aspects and define them in detail

• describe the user interface being implemented

• describe the hardware and software interfaces

• describe the performance requirements

• include design features and the architecture of the project

• list and describe the non-functional attributes

# System Overview

### What is application tracking system solution?

Application tracking system (ATS) is cloud based platform. It has several components which have been designed to meet the present and future demands of the hiring process. ATS Solution is supported by advanced technology infrastructure. It has high standards of business functionality.

The ATS is capable of being implemented in stages.

# Solution Architecture

## Solution Diagram

##### The following Fig. provides ATS Microservice based application architecture on AWS cloud infrastructure.

Diagram

Description automatically generated

Fig:ATS Application Arcitecture

## ATS Microservices:

The functionalities of **ATS** will be catered by set of **Microservices** relevant to specific business functions.(For Ex- functions like new opening , candidate registration, schedule interview etc.)

## Each service typically implements a set of distinct features or functionality and communicates across each other through a well-defined, lightweight mechanism to serve a business goal.

The key benefits are,

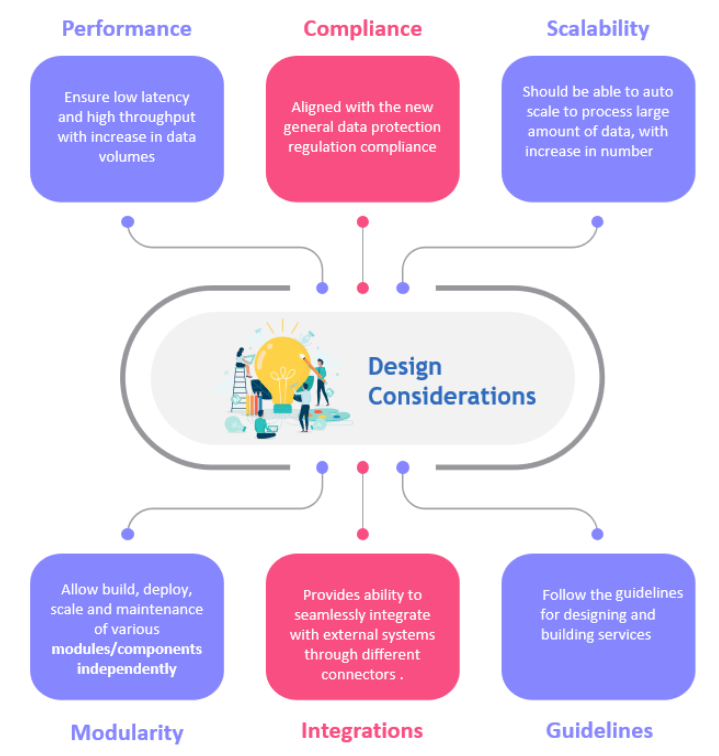
* Individual services are much faster to develop, and much easier to understand and maintain.
* ·        Each service can be independently deployed and scaled.
* ·        Each service has its own database schema and ensures loose coupling.

Common Services:

* Opening Management Service
* Candidate Management Service
* Interview Scheduler and Feedback Service
* User management Service

# Design Considerations

The below diagram depicts the key design considerations, namely Performance, Compliance, Scalability, Modularity, Integrations and Guidelines.



## System architecture guidelines

Proposed solution should be based on the following architectural principles

* Follow twelve-factor app methodology as described in <https://12factor.net/>
* Ensure consistency in architecture design options across all solutions.
* Build to change  instead  of building to last. Consider how the application may need to change over time to address new requirements and challenges, and build in the flexibility to support this.
* The system architecture should support "modularity". The resulting software comprises of well defined, independent components. The components could be then implemented and tested in isolation before being integrated to form a desired software system.
* Make use of re-usable components as much as possible. The system architecture to be defined will have to adapt to the boundaries, interfaces, functions, effectiveness, and behavior of the re-used system component.
* When making choice between different third party components, go with open source, community supported and highly adapted option.
* Use of microservices architecture is recommended.
* The system architecture should support SSO used by Enterprise application suite.
* The system architecture should be scalable.
* The system architecture should be extensible. New capabilities can be added to the software without major changes to the underlying architecture.
* The system architecture should be platform independent.
* The system architecture needs to have adequate security controls.

## Assumptions and Dependencies

* Configurable role-based user authentication and authorization component would be required.
* Design and implement an encryption strategy that meets the needs of the applications.
* Provide monitoring and reporting functionalities to support service level management.
* Document contains high level design data. Further discussions are required to cater requirements in detail.
* Distributed applications also need some kind of central logging framework so that all data can be centralized to generate the log data. It would be developed if not already in use for ATS applications landscape.
* Cross-cutting concerns or gateway offloading**.** Depending on the features offered by each API Gateway product, you can offload functionality from individual microservices to the gateway, which simplifies the implementation of each microservice by consolidating cross-cutting concerns into one tier. This is especially convenient for specialized features that can be complex to implement properly in every internal microservice, such as the following functionality:
* Authentication and authorization
* Service discovery integration
* Response caching
* Retry policies, circuit breaker
* Load balancing
* Logging, tracing, correlation
* Headers, query strings, and claims transformation
* IP whitelisting

# NFR Considerations

## Scalability

An application typically uses three types of scaling. X-axis scaling is horizontally cloning the application, Y-axis scaling is splitting the various application functionalities, and Z-axis scaling is partitioning or sharding the data. When Y-axis scaling is applied to monolithic applications, the application is broken into many smaller units aligned with business functions that are in line with microservice characteristics.

Pattern: each microservice has its own instance or container that is isolated. Service-level load balancing can be accomplished across same services hosted in multiple instances.

## Usability

As Application tracking system is carried by various types of clients i.e. whether they have knowledge of computers or not so the application designed for hiring process must be easy to use. The application must have graphical user interface and it must have the ability to provide informative error messages. The qualities of the ease of use which can be measured are learning time it points out the time required to learn the application, number of errors while working with the normal speed furthermore the likeness of the client to measure the system i.e. the client fulfilment in utilizing the framework. The interfaces of the system ought to be clear, easy and simple to use and understand.

## Reliability

* ATS environment should be up and running 24 x 7. In case of any crashes system should again restart on its own.
* The solution should maintain its performance over the time. It implies how well the system performs in peak hours. The software should be a robust system which has the capacity to handle the bugs without failure i.e. how effortlessly it handles the bugs because of data or handling, surprising conditions while working conditions furthermore the software imperfections, and if the system is robust it is reliable also.
* The application must self-contained, consistent and complete within itself. The failure rate in the application tracking system should be least or negligible as the system is supposed to be reliable. Reliability of the system depends on the failure free transactions and how fast the system is able to recover from the failure.

## Performance

1. The offered solution must complete 99% of provided services in less than 5000 milliseconds, over both the peak and non-peak hours
2. The system or software should process up to 4,000 transactions every second as submitted to it without failure.
3. Performance tuning of each services will be done. It will ensure guaranteeing the up-time, optimal utilization of resources, and keeping services running without glitches.

## Security

* The APIs will be secured using the standard protocols like OAuth 2 and allowing only the authorized system/app to access these APIs on the basis of valid access token.
* Proposed solution must provide guard against various security threats (OWASP Top 10 etc.). System components should utilize industry-proven security standards and protocols.
* Data Encryption - both for data in transit and at rest.

### User Authentication

* Amazon Cognito, which enables you to add code to your application to authenticate users either directly through a user pool, through a social identity provider, or OpenID Connect (OIDC) identity provider..
* Amazon Cognito provides web identity federation and allows user to authenticate with a Web identity provider (Google, Amazon, Facebook)

Graphical user interface, application, Teams

Description automatically generated

Figure - Access token security for microservice APIs on Amazon EKS

The application client shown in Figure represents a service-to-service workflow on Amazon EKS, and shows the following three steps:

1. The application client requests an access token from the Amazon Cognito user pool token endpoint.
2. The access token is forwarded to the ALB endpoint over HTTPS when requesting the microservice API, in the bearer token authorization header. The ALB is configured to use IP Classless Inter-Domain Routing (CIDR) range filtering.
3. The microservice deployed to Amazon EKS validates the access token using [JSON Web Key Sets (JWKS)](https://docs.aws.amazon.com/cognito/latest/developerguide/amazon-cognito-user-pools-using-tokens-verifying-a-jwt.html), and enforces the authorization claims.

### User Authorization

* We expect Individual Application to expose endpoint to provide support for the user provisioning
* We need the common framework (UI, Workflow/etc) to accomplish the provisioning.
* Concept behind this we will have Authorization service in platform  allows to integrate with internal services and exchanges java web tokens (JWT) as a security object and individual application will authenticate and authorize the user for the allowed access.

### Service Level Authentication

* Each micro service will have a service account to run

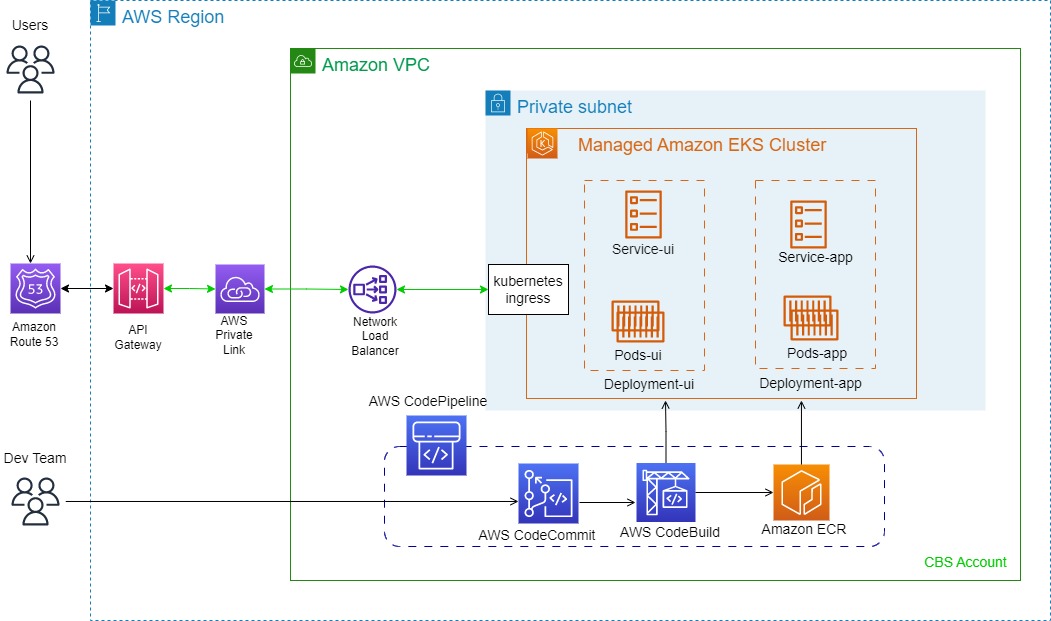
## Supportability

* Actions which the different user roles can perform within the system must be configurable without code changes.
* System must provide performance data on request (# of concurrent users, #of service calls / # of transactions, average and peak service response time). It should be highly scalable to accommodate massive user load that is expected in festive/offer sales.
* Proposed solution has the ability to monitor and maintain the system, and include testability, configurability and upgradeability.
* System must provide internal and external messaging in synchronous and asynchronous mode, if any required.

## Infrastructure Requirements

* System should implement N- tier architecture or Micro-Services architecture (MSA) wherever appropriate
* It should support both on premise and cloud-based deployment with auto-scaling to match demand.
* System must be able to consume external service (web services).
* System must provide integration capabilities in order to ex-change information with various internal/external applications. It should be open, flexible, and scalable and easily integrate with other systems.

**Deployment:**



*Figure* Deployment diagram

## Components Description

### API Gateway

* Allows to connect from Internal or external requests
* Should leverage the service discovery for the service instance identification using AWS cloud Map
* API gateway has a caching capability to increase performance.
* We can throttle API gateway to prevent attacks.
* It uses Amazon Cognito user pool as authorizer for a Rest api.
* Service level account are maintained within the central configuration repositories for Dev, QA, Stage and Prod)

### Central Configuration

* All application specific configurations are maintained in the central configuration repositories.
* We will keep all the configurations out of the application to a centralized access store, such a GitHub repositories.
* Microservice applications will retrieve the configurations acting as config client on the service startup.
* Microservices will be notified when changes occurred on the updates to repositories

### Circuit breakers

* We need circuit breakers the help the micro services function even when one of the services in the ecosystem is down or not performing well.
* Allows user to continue even when there are failures to certain part of the solutions.

The other cross cutting concerns of the architecture would be addressed as mentioned below:

**Logging**

Centralized Logging on AWS will be implemented in order to collect, analyze, and display Amazon CloudWatch Logs in a single dashboard. This solution uses Amazon OpenSearch Service and Kibana, an analytics and visualization platform that is integrated with Amazon OpenSearch Service, that results in a unified view of all the log events.

**Monitoring**

Also, every service will be responsible for providing their health metrics data using MSSL (Micro Service Status Library) which can be later integrated with any monitoring tool like **Prometheus** to monitor the service health and generate alarms in case of something fails. This can further be integrated to monitoring dashboards like **Grafana** for data visualization.

**Integration**

The microservices will communicate with REST over http for synchronous messaging and interface over either JMS/MQTT/AMQP for asynchronous messaging.

### Design Pattern Considerations

• Follow twelve-factor app methodology as described in <https://12factor.net/>

•      Saga distributed transactions pattern

•      Database per Service ensures loose coupling

•      Server Side Discovery of Micro Services with the help of API Gateway that uses API cloud map

•      Messaging and Remote Procedure Invocation Patterns to support Asynchronous and Synchronous Inter Process Communication

•      Support for combination of Single Service Per Host & Multi Services per Host pattern for effective resource utilization

# Technology stack / Framework/Infrastructural Components

|  |  |
| --- | --- |
| Language | Java 17 |
| Frameworks | Sprint boot 2.6, Spring (Core, AOP, Security, Data), [Spring Cloud Sleuth](https://cloud.spring.io/spring-cloud-sleuth/),  JPA, Hibernate |
| Cloud | AWS |
| DB | AWS RDS- Postgres, NoSql - Dynamo DB |
| Object Store | Amazon S3, EFS |
| Messaging | AWS SQS, SNS |
| Monitoring | Cloud watch, Cloud Trail, Prometheus |
| Devops | CI/CD, AWS pipeline, AWS code build, Docker, ECR, EKS |
| Others | Git, Maven, Liquibase, IAC tool -Terraform , AWS Cognito, AWS amplify |



## Logging

* Centralized Logging on AWS will be implemented in order to collect, analyze, and display Amazon CloudWatch Logs in a single dashboard.
* This solution uses Amazon OpenSearch Service and Kibana, an analytics and visualization platform that is integrated with Amazon OpenSearch Service, that results in a unified view of all the log events.
* Technical logging will be there to support all technical errors.
* Business or journal logging can be provided to log the flow. Only limited users will have access to business logs.
* Slf4j or log4j can be used for the logging.

## Exception Handling

* Custom exception handler will be written as a cross cutting concern.
* Various custom exceptions can be written to handle business and technical errors.

## Security

* OAuth, Spring security
* Secured access to servers (https)
* Role based authorization will be provided.
* Encryption/decryption algorithms will be used for web service and other accesses.
* Role based access can be provided using IAM roles.

## Data Access

* JPA with hibernate as JPA provider will be used as persistence framework.
* Secured data access
* Transaction management

## Object Creation

* Object are instantiated, assembled, and managed by a Spring IoC container.

## Data Transfer between Layers

* Business logic will be implemented using various interfaces, DTO objects will be there to transfer data between various layers.
* Loose coupling will be there between various layers.

*Figure* Class diagram

# Integration (External Systems) Architecture

## Integration

* The microservices will communicate with REST over http for synchronous messaging and interface over AMQP for asynchronous messaging.



**COPYRIGHT NOTICE**

Copyright © 2013-15 Nagarro Software Pvt. Ltd. **All rights reserved.**

This document is confidential and proprietary to Nagarro Software Pvt. Ltd. and no part of it should be reproduced, published, transmitted or distributed in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, or stored in any information storage or retrieval system of any nature nor should it be disclosed to third parties without the prior express written authorization of Nagarro Software Pvt. Ltd.

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision History** | | |  |
| Version | Date | Author/Contributor | Comments |
| 0.1 | 12-02-2023 | Sourabh Yadav | [comments] |
|  |  |  |  |