System Design Information

Functional Requirement

- Client (Employer) registration and platform onbording
- Employee registration and platform onbording
- Client monthly invoice generation (Payroll Invoice)
- Client invoice payment collection and releasing employee payout
- Robust emails delivery

Non function Requirement

- **Platform availability and crash recovery**: We are using database replicas to recovery any node failure automatically
- **Platform horizontal scalabilty**: To support scaling, we use **Kubernetes Horizontal Pod Autoscaler (HPA)** to automatically scale the number of pod replicas up or down based on real-time traffic patterns and resource utilization.
- **Data consistency**: To support data consistency we are using Polyglot databases where PostgreSQL is well know CP system in distributed applications. While MongoDB is AP system but we can trune read & write concerns to achieve desired data consitency.
- **Performance**: This architecture can handle thousands of request per seconds without performance bottleneck

System Constraints

Core Functionalities

Due to time constraints, in this design I only focus on the core part of the system related to the EOR (Employer of Record) model, specifically for employees hiring.

While we can also support the AOR (Agent of Record) model for contractor and freelancer hiring but that part of the system is outside the scope of this design and is not covered here.

Use Case of AI Capability

I am using the AI service in this design to support two main features that is generating contract document and generating multiple email templates. There might be many other use cases as well for AI service but in this flow I am focusing only two use cases. In this design AI service is an aggregator multiple AI plaform APIs (OpenAI, Claude, Gemini)

I can generate client risk profile via AI as well but the data reliabity is not as must as 3rd part service provider provides us that is why this use case is excluded from AI scope.

I am assuming the client hires the talent and onboard them on platform but if we do talatent discovery at out end then there may be many more use cases of AI like

- Applicants Resume parsing
- Client job description matching

Notification Service

The design scope of the notification service is to support only one communication channel that is email but it can easily extendable to support other channels like push notification, WhatsApp Notifications, Slack Notifications as well

Proposed Communication Protocols

- **HTTP/2.0**: To support GRPC's inter service communications
- **HTTP/1.1** For REST APIs expose to clients
- **AMQP**: To support reliable message queuing & delivery

Proposed Message Brokers

- **Apache Kafka**: Supports high-throughput event streams, which can later serve as audit trails or activity logs.
-
- **RabbitMQ**: Ensure reliable and guaranteed message delivery, which supports acknowledgments, retries, and persistent queues to achieve 100% delivery assurance even in case of failures… It can also use to create dead later queue later on which is not in this design scope.

Proposed Security Measures

- Authentication: JWT with RS256 (Asymmetric) encription
- Authorization: RBAC + ABAC
- Security: HTTPS with HA proxy for TLS termination
- MTLS (Mutual TLS): Secure inter-service communications
- CORS: Prevent Cross Origin Requests

Proposed APIs (REST + GRPC)

REST

- Register User [POST /api/v1/users]
- Login User [POST /api/v1/users/login]
- Create Legal Entity [POST /api/v1/legal-entities]
- Create KYB Profile for Business [POST /api/v1/legal-entities/{legalEntityId}/kyb-profile]
- Create Risk Profile for Legal Entity [POST /api/v1/legal-entities/{legalEntityId}/risk-profile
- Activate Legal Entity [PUT /api/v1/legal-entities/{legalEntityId}/activate]
- Create Contract [POST /api/v1/contracts]
- Fetch Legal Entity Risk Profile [GET /api/v1/legal-entities/{legalEntityId}/risk-profile]
- Create Payroll [POST /api/v1/payrolls]
- Create Salary Structure [POST /api/v1/contracts/{contractId}/salary-structure]
- Create Benefits [POST /api/v1/contracts/{contractId}/benefits]
- Sign Contract by Legal Entity
  [PUT /api/v1/contracts/{contractId}/signBy/{signerLegalEntityId}]
- Create KYC Profile for Employees or Contractors
  [POST /api/v1/legal-entities/{legalEntityId}/kyc-profile]
- Activate Contract [PUT /api/v1/contracts/{contractId}/activate]
- Tag Contract to Payroll [PUT /api/v1/payrolls/{payrollId}/tag-contract/{contractId}]
- Fetch Forex for currenciey pairs [GET /api/v1/forexs]
- Create Beneficiary [POST /api/v1/beneficiaries]
- Trigger Payrun [POST /api/v1/payrolls/{payrollId}/payrun]
- Update Invoice [PUT /api/v1/invoices/{invoiceId}]

GRPC APIs

- Fetch forex by currency pairs
  - Method: fetch_forex_by_currency_pairs
  - Input Arguments: currency_pairs {"USE-EUR", "USD-GBP", "USD-AUD"}
- Activate Legal Entity
  - method: activate_legal_entity
  - arguments: legal_entity_id
- Activate Contract
  - method: activate_contract
  - aruments: contract_id
- Sign Contract by legal entity
  - method: sign_contract_by_legal_entity
  - arguments: legal_entity_id, contract_id
- Create risk profile for legal entity
  - method: create_legal_entity_risk_profile
  - arguments: legal_entity_id
- Create contract
  - method: create_contracts
  - arguments: {"first_party_legal_entity_id","second_party_legal_entity_id", "contract_type": {"EOR","OFFER_LETTER"}}
- Tag contract to payroll
  - method: tag_contract_to_payroll
  - arguments: contract_id, payroll_id
- Generate Email Template
  - method: generate_email_template
  - aruments:{ template_type: registration_email, username: "employee_name" }
- Assign Funding Account
  - method: assign_funding_account
  - arguments: legal_entity_id
- Lock FX Rates to Payroll
  - method: lock_fx_rate_to_payroll
  - arguments: payroll_id + fx_rate_data

Proposed Deployment Approach (Kubernates + Terraform + Service Mesh)

We can create this Infrastructure in AWS region: us-east-1 via **Terraform** (Infrastructure as Code) tools. So when we want to expand our business in multiple regions then to replicate entire Infrastructure can be as quick as possible without manual interversions, which ensures the identical Infrastructure creation in other regions as well. This approach speed up multi region launch.

The block digram is self explanatory. I prefer to use kubenates cluster for deployment as it offers robus montoring and scaling out of the box. I am using 3 Pods per server in this design. I am also prefer to use service mash here which abstracted out application level boilerplates setup for Observibility (logs, metrics, tracing), Traffic Control and Faliure Recovery (backoff retry and circuit breaker). Also it secures service to service communication as well (using MTLS). For the sake of simplicity I am not showing serive mash in the architecture design.