Beanworks Challenge

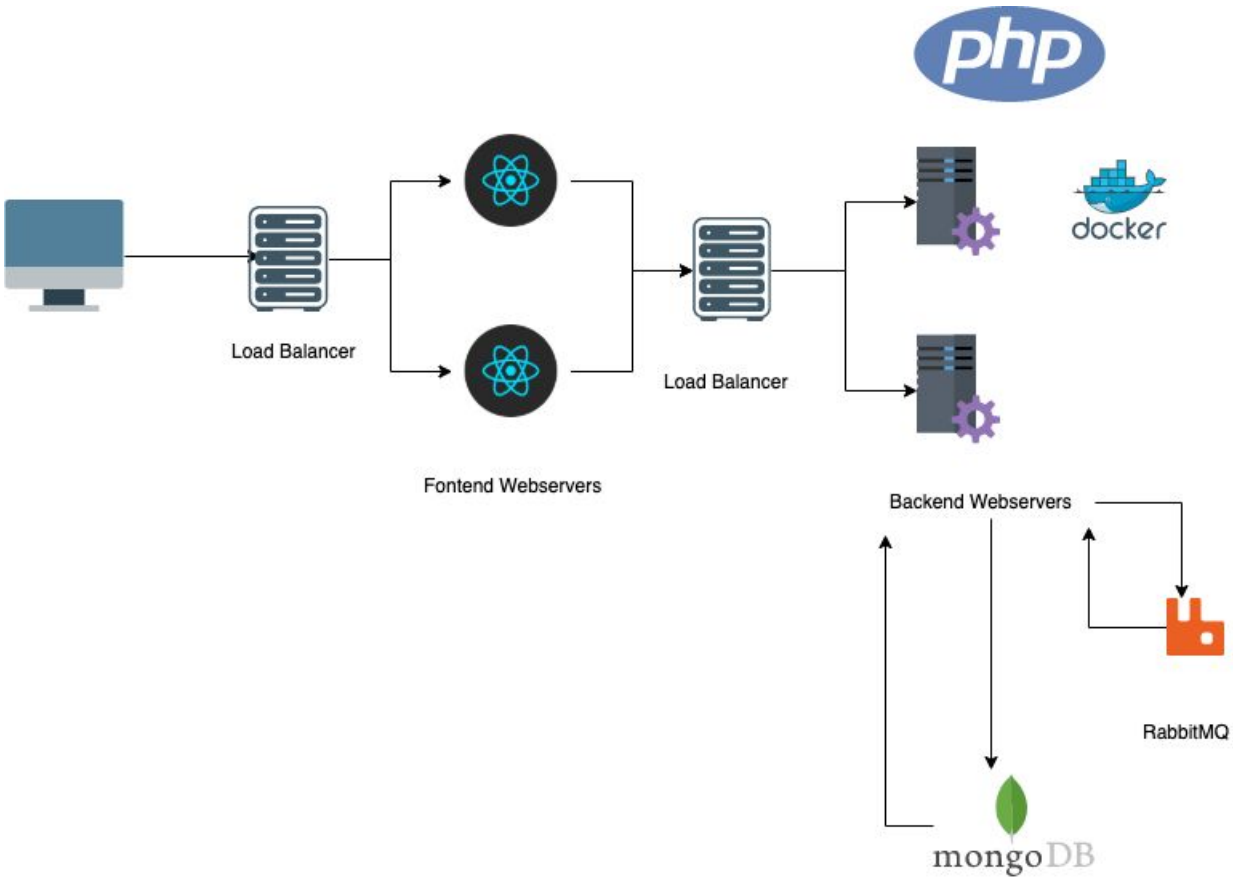
Synopsis

A sync solution that reflects Beanworks core product truths (user-friendly, easy on the eyes, informative).

Functional use case

https://drive.google.com/file/d/1nWXz_kZptxF9Ldm1uiJNTiiAZUkeP294/view

Suggested solution



Technical choices

React Based Front-End

Our front-end will be developed using react.

Symfony Framework based Back-End

https://symfony.com/

REST API:

We will develop various API related to data exports and other hackathon related challenges
We'll add Symfony/messenger to the dependencies to handle message queueing and consuming. The frontend will use its MessageBus to enqueue a new message after the challenge gets validated.
Symfony/messenger provides a command in charge of consuming messages. It can take a "transport name" as an argument to specify which queue to consume. Please note: a "transport" is a generic term used by Symfony/messenger synonym of "queue" in our solution.

This command offers the ability to specify a limit to avoid memory leaks. It can be after a certain amount of consumed messages or a time limit.

RabbitMQ

https://www.rabbitmq.com/
RabbitMQ will be the backend in charge of queueing messages. The Frontend application(react app) will call API (Symfony(PHP) based Backend APP) and Backend App will send new messages to it, then PHP consumers will be in charge of consuming them and take action accordingly.

MongoDB

We will use MongoDB as a database.

Docker and Kubernetes

The command is executed through a dedicated container taking the “transport name” as a parameter. We will run as many instances of the container as we have consumers, Kubernetes being in charge of restarting the instance once the command has exited (after hitting a limit as specified above).

Supervisor

Supervisor will be used to run sync consumers

Detailed solution

Sync Pipeline

Beanworks Sync App	←---------	Xero
Mongo DB. accounts	←----------	Account
vendors	←----------	Contact (Supplier)
Sync		

Xero: Data Attributes

https://developer.xero.com/documentation/api/types
Mongo DB Data schema

Database: beanworks
Collections:

User: User detail
Accounts: Account record imported from Zero
Vendors: Vendor record imported from Xero
Pipelines: Sync pipeline document
Pipeline_logs: Logs related to a particular pipeline

User -> Clicks sync -> backend API (/api/sync/<uid>) is invoked

- Check if another sync pipeline is in progress -> Display details of current in progress pipeline
- Return 200 and pipeline object

    - Initialize new sync pipeline
        - insert in sync document in mongo
        - dispatch sync message (push message to RabbbitMQ)

Pipeline processing

- Sync message will be handled by API consumer
- Update sync document
- Dispatch Account and Vendor sync message
- Account and Vendor sync message handler will process the message

Account Sync Message Processing

- Export accounts from Xero
- Check is account id exists in MongoDB
    - If yes update the document
    - Else insert a new document
- Update the sync document with progress metadata

Vendor Sync Message Processing

- Export Vendors (contacts type Supplier) from Xero
- Check is account id exists in MongoDB
    - If yes update the document
    - Else insert a new document
- Update the sync document with progress metadata

API

Detailed Interactive API Documentation:

https://documenter.getpostman.com/view/2272502/SW7Z599h?version=latest

Login API

- POST /api/<version>/login
  Post Param Json: { “email”: “admin@beanworks.com”, “password”:”admin”}
  Response: json
  {
  "status": "OK",
  "data": {
  "token": "5dd2dee59486240068292872"
  }
  }

Sync APIs

- Sync data
  POST : /api/<version>/pipeline/
  Header: {“api-token”: “toke”}
  Response: json
  {
  “message”: “OK”,
  “Data”: {
  “_id”: ObjectId(“xxxxxx”);
  “status”: active
  “Created_on: : ISODate(“xxx”)
  }
  }

- Sync details
  GET : /api/<version>/sync/<id>

Account APIs

- List of accounts
  GET: /api/<version>/accounts/

- Account Details

GET: /api/<version>/accounts/<id>

- Search
  GET: /api/<version>/accounts/search

Vendor APIs

- List of vendors
  GET: /api/<version>/vendors/

- Vendor Details
  GET: /api/<version>/accounts/<id>

- Search
  GET: /api/<version>/vendors/search

Deployment

Travis For building Image

Jenkins for deploying a tag-based release

Monitoring and Alerts

Sentry Integration

Data Dog Dashboard

MongoDB connections and usage
Api status
RabittMQ status

Datadog Alerts

Development Environment

Prerequisite

Docker
GIT
Make command

GIT Clone application

Git clone https://github.com/ash-singh/beanworks-sync-app
beanworks-sync-app

Move to the beanworks-sync-app folder

Run **make init**

Beanworks app will be accessible at **http://localhost:3000**

API project will be accessible at **http://localhost:8888**

Unit Test API:

make test-api

Unit Test API:

make test-react

PHP coding standard

make lint

PHP Static code analysis

make static-analysis

Next step If I had time

Outh2 and 2-factor authentication

Elastic search integration

- Account and Vendor free text search
- Pushing application logs (Haproxy logs, api logs etc)
- Kibana Dashboard for reporting

MongoDB cluster

Use Content Delivery Network(CDN) like cloud flare for Beanworks react App

Possibly move to AWS managed services

Configure Pagerduty Alerts