Product Data Lake

Architecture, version 1.7, May 2018

Introduction

As a participating party, you see the Product Data Lake as a cloud service where the technology behind is already in place (unless you choose to host a private instance of Product Data Lake yourself).

The public version of Product Data Lake is hosted on *Amazon Elastic Computing Cloud* (AWS) using the *Amazon Linux servers* in Frankfurt, Germany.

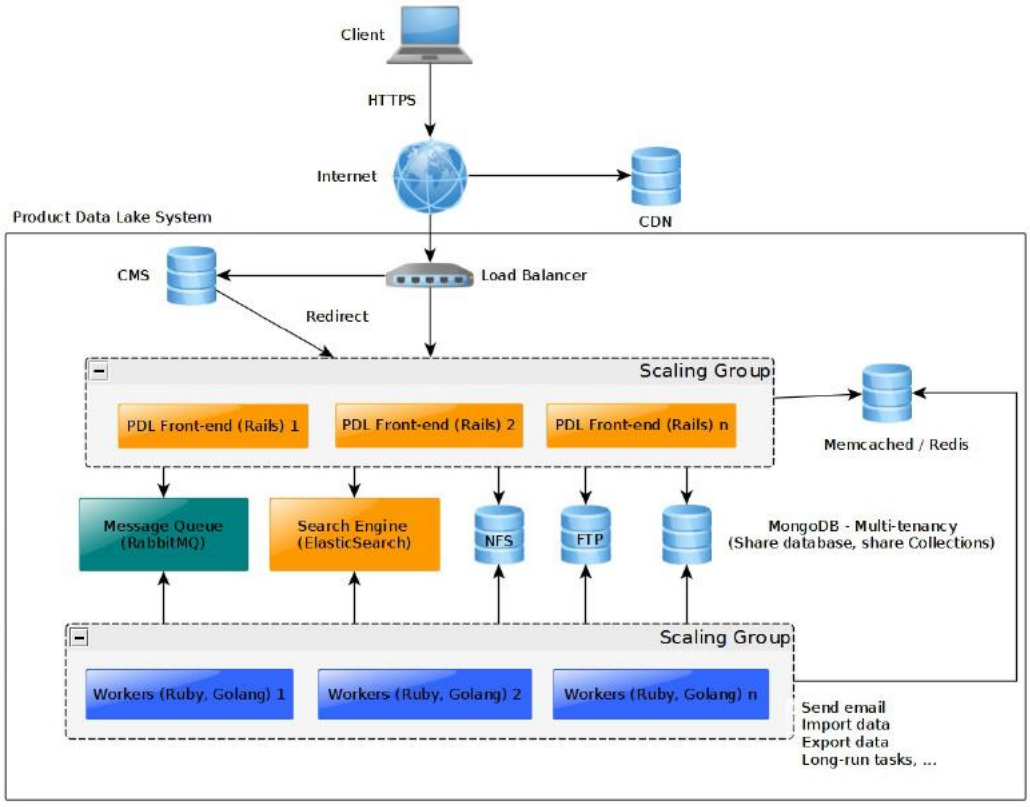
Components

Below are the logical components of the Product Data Lake (PDL) solution:

- Rails WebApp for PDL administrators.
- Rails WebApp for end users.
- Database System: Storing user information as well as transaction data. PDL uses MongoDB as the primary DBMS.
- Search Engine: ElasticSearch is used as search engine for PDL, which helps improve user search experience.
- Queue: RabbitMQ/Memcached and Redis are used as queue/broker systems for organized long-running tasks dispatched by the web application.
- Worker: Written on Golang language, which is suitable for memory/data heavy tasks, running in the background and communicate with the main WebApp through queue.
- NFS storage: Storing user input files.

Deployment View

The illustration below describes the physical software and hardware components that comprise the PDL ecosystem:



From the outside, end users can access PDL via Internet and be secured with HTTPS protocol (forced) to ensure user privacy. The administrator can use CMS to edit the landing page of PDL, where the public version of PDL is build using LocomotiveCMS. The end users will be redirected to PDL after clicking on Sign in / Sign up. Sign in / sign up can also be called individually in a given language from other websites.

PDL is a web application mainly based on Ruby on Rails for front-end part where Content Delivery Network (CDN) are used to enhance user experience while loading web pages. CDN is implemented by using Amazon CloudFront / S3.

The Front-end will also be scaled out automatically by using Amazon Elastic Load Balancer and auto scaling group.

The Message Queue works based on a Producer / Consumer model. Front-end produces jobs and Workers will consume them. Workers are implemented in both Ruby and Golang as for example for jobs such as email sending, data importing/exporting, long-run tasks. These workers will also place in auto scaling group and scale out automatically based on workload.

Search is also an important feature of the PDL, so we have chosen ElasticSearch because it is distributed, scalable, and highly available. It also provides real-time search and analytics capabilities.

To reduce accessing to Database, Memcached / Redis are used.

NFS is used to store user upload files, temporary files for both front-ends and workers. PDL fetches files on FTP domains and create results from pull requests on FTP domains. Alternatively, data can be put and fetched by APIs.

PDL uses MongoDB as database system implemented as a Multi-Tenant Architecture on it. Share database, share collections are chosen for the database architecture. Party_ID is used in collections to determine which data belongs to specific tenants.

Additional documentation available

For an introduction to the Product Data Lake, please refer to the Product Data Lake Overview.

For how to get a party account on the Product Data Lake, please refer to Product Data Lake Getting Started.

For how to interface with Product Data Lake, please refer to the Product Data Lake Data Exchange Guide and the Product Data Lake Put and Get API Guides.

For how to operate further interactive functionality on the Product Data Lake, please refer to the other Product Data Lake User Guides.

Product Data Lake contact information:

By phone: Denmark: +45 40410528. By eMail: support@productdatalake.com