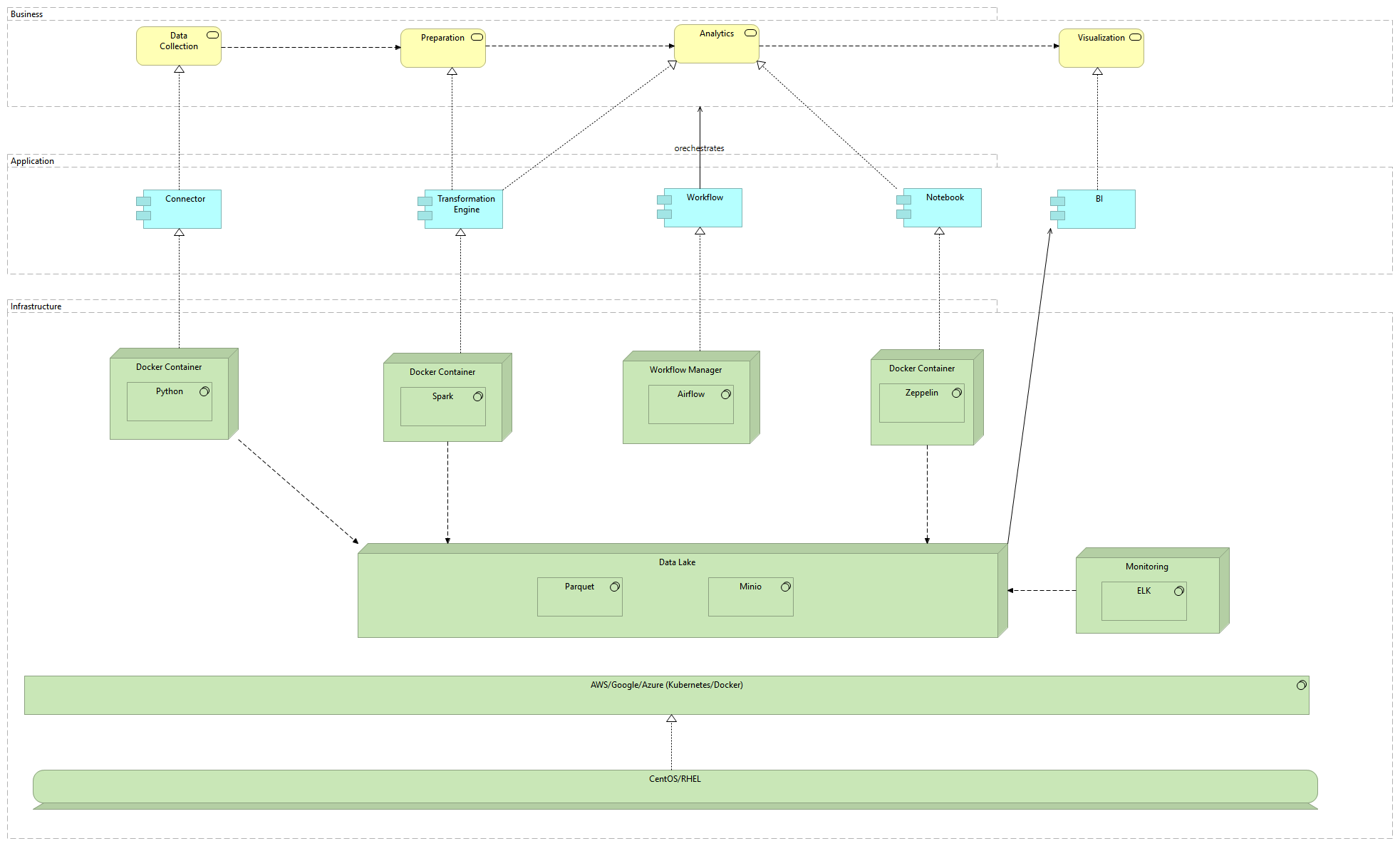
Global Data Platform Archiecture

Global Data Platform collects information from social and other providers, applies some transformation and makes the new data available for other consumers and services.

The general view of the architecture is:



# Connector

Connectors are used to collect data from different data sources and store them in the data lake. A simple flow overview is the following:

## Requirements

### Configuration

1. Users should be able to configure the connector from the portal application.
2. Configurations should be stored in a way they are accessible to connectors.

### Security

1. Data at transit should be secured and encrypted (SSL)
2. Any credentials and authentication information should be encrypted before stored.

### Throughput

1. Connectors should be designed to handle high volume data and spikes. The connector should specify high throughput thresholds.

### Fault Tolerant

1. Data loss is not acceptable, therefore connector architecture should be fault tolerant.

## Twitter Connector

# Analytics App

Analytics application component is composed and uses other components which perform different roles. Compute engine performs transformations and aggregations over the datasets and can be used to create batch views and machine learning models. MapReduce programming model is the base of its implementation.

## Requirements

# Development

## Code Style

Same code style should applied across DEV/QA. A template should be created and applied for all. It should include things like:

* Line length
* Class file template (automated class doc like name, date, version, etc.)

## Best Practices

Best practice deal with compliance and consistency for all the development activities. E.g. when someone wants to fix or update a connector like Tweeter connector it should be easy to find the right repo, module, class. If one connector is named ConnectorTwitter and other connector is named FacebookConnector this might create confusion. Another example is related to method naming. E.g. TwitterConnector has a method named connectTwitter and FacebookConnector has a method named connect. The second convention is better over the first one since the class/module name already suggests what it is doing. Furthermore, the first naming practice does not facilitate design patterns like STRATEGY.

## Dependency management

For dependency management pip is going to be used.

## Unit Testing

Unit tests are written using pytest.

# Testing

GDP should be thoroughly tested in order to ensure high quality of the platform.

There are two main qualities of the tests:

1. Automated. In order to achieve CI/CD and eventually CDD. Test automation is a key factor for speed-to-market and reduced release cycles. Once the system complexity increases, low automation increases the release cycle time and introduces risks for the quality of the system, eventually resulting in low customer satisfaction and mistrust.
2. Scalability. Dealing with big data requires scalability for every component in the architecture. Test automation framework and components should have the same requirement. Imagine that the test suites increase from 100 to 1000s, but they are tightly coupled and can be executed on a single machine. They used to take less than an hour, but now they take 6h or more and the time increases as more tests are added. This impacts the speed-to-market and CI/CD. In order to avoid this, the tests should be designed and implemented with scalability requirement in mind so that when needed, adding more hardware and compute power scales the system and the total time remains the same (< 1h).
3. Reporting. Testing framework should output reports which could be later analyzed to determine the root cause and fix the issue. Also, trends can be observed and further analysis can be performed.

## Unit Tests

Unit tests are the fastest tests executed on every CI build. They provide fast feedback and the CI pipeline fails if they fail.

## Functional Tests

Functional tests should validate the platform performs its functional requirements.

## Stress Tests

Using stress tests one can determine what is the breaking point.

## Performance Tests

Dealing with big data with high velocity and high volume make performance tests first class citizens.

# Logging

# Monitoring

Monitoring is essential when dealing with multiple components. Actually, monitoring implements some of the platform itself.

Monitoring has the following requirements:

1. Automated.
2. Alerts. Whenever a component fails or the platform under performs, an alert should be raised and notifications should be sent or actions performed to fix the issue.

## System Metrics

At minimum, every component should monitor the following metrics:

1. Memory
2. CPU
3. Disk
4. Network

# Security

Security is one of the qualities which should not be overseen. Important aspects include:

1. Data in transit: usually achieved using encryption (i.e. SSL). A domain certificate should be issued and used across the platform.
2. Data at REST. Data at REST must be secured and protected from unauthorized access.
3. User Management. Users, roles and groups management service should be provided. Most likely this is not going to be a part of the platform, but offered by cloud providers using directory services (e.g. AD)
4. Access Management. Authentication and authorization ensure data is protected against unauthorized use. Identity federation allows users to authenticate against an external system (Identity Provider, IDP). The application itself acts as a Service Provider (SP).