Faculty of Mathematics and Information Science
Warsaw University of Technology

Cloud Computing

The Sentinel project

Marcin Dardziński, Paweł Rzepiński, Ryszard Szymański,
Sebastian Sowik

Contents

1 Introduction

1.1 Project description

The Sentinel project provides a dashboard presenting insights into the volume and quality of the conversation towards followed brand, products or services in a time-aware manner. The most important features include:

1. Tracking keywords appearances – real-time monitoring of occurrences of selected keywords or their variations on the Internet. Keywords can include, but are not limited to: brand name, key products or events, most important staff members, company-owned domains. User can manage followed keywords using the website.
2. Evaluating sentiment expressed in the text – numerical summary of emotions taking into account keywords and evaluated automatically.
3. Listing of mentions – direct links and content of all of the mentions; to allow for quick response from the PR department.

The processed data is gathered from the live feeds of Twitter, Hacker News, Reddit and Google News aggregator. It consists of text (of comment, tweet, news article excerpt) and metadata (author, url, timestamps).

1.2 Functional requirements

Use case diagram presented below captures Sentinel system functional requirements.

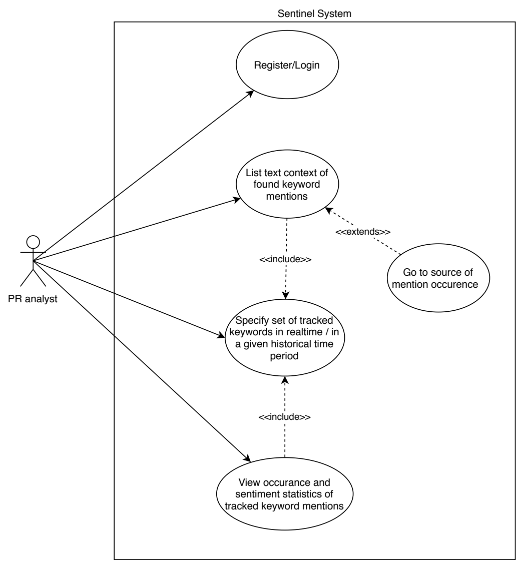
Sentinel System
Register/Login
List text context of found keyword mentions
<<extends>>
<<include>>
Go to source of mention occurence
PR analyst
Specify set of tracked keywords in realtime / in a given historical time period
<<include>>
View occurance and sentiment statistics of tracked keyword mentions

Figure 1: Use case diagram

1.3 Non-functional requirements

Non-functional requirements according to FURPS classification:

Type of requirement	No.	Description
Usability	1	Visualization of sentiment scores over time associated with the tracked keyword
	2	There should be a web browser installed on the user's computer in order to be able to use all of the apps functionalities.
Reliability	3	The system should be secure, scalable and highly available by using cloud architecture best practices.
Performance	4	Live sentiment score values should arrive with a delay not longer than 10 seconds.
Supportability	5	User manual delivered along with the system.

2 Data

The data used in the project is retrieved from multiple sources: Google News, Hacker News, Reddit and Twitter. The vast majority of the processed data is text-based and features natural language – posts, comments, tweets, news headlines and articles. Therefore, in order to facilitate the development of the remaining components of the system, all of the downloaded mentions had to be converted to a unified format presented in Table 1.

Attribute	Description
Text	Content of the comments, tweet or article.
Url	Url of the specific mention.
Origin Date	The data at which the mention was submitted to the specific source.
Download Date	The date at which the mention was downloaded by the system.
Source	The source of the downloaded mention.
Metadata	Mention metadata dependent on the specific source.

Table 1: Unified format model

This process required a detailed analysis of the used APIs as different issues occurred among different data sources.

Hacker News is a social news website that focuses on topics related to computer science and entrepreneurship. The unofficial streaming API available at http://hnstream.com/ was used in the project. The comments delivered by the service contained html elements which had to be removed before the conversion to the unified format. Those were removed using the BeautifulSoup package[1]. Moreover, the amount of the streamed data is very fluctuating dependant on the activity of the community.

The Google News API aggregates news articles from more than 30 thousand publishers. It allows to get top headlines from a particular source or country – usually around a 100 articles per minute. As the API covers news from different articles written in other languages than English had to be filtered out. Moreover, the dates of the released article were not converted to the UTC timezone, but corresponded to the countries in which they were published. The data available in Google News comes with a 15 minute delay, hence the API had to be queried repeatedly with an interval of 5 minutes.

The Reddit API exposes a live stream of user comments – about 1000 comments per minute. Each comment contains rich meta data information, including it's score (number of upvotes) and context (e.g. whether the comment was a reply to another comment).

Twitter offers an API for real-time filtering of tweets, by following specified keywords, users and combining criteria using special filter syntax. This source offered the highest amount of data as on average 350,000 tweets per minute are being published.

3 Architecture

The architecture of the system can be presented from multiple perspectives: data flow between components, services used, typical user usage path.

3.1 System architecture outline

The website presents a live dashboard to the user and allows him to manage tracked keywords. The login process is handled by Cognito.

The data is collected by multiple connector processes running on EC2 instances. The followed keywords are dynamically retrieved from the database and searched in the incoming streams. Successful matches are sent to the Kinesis stream which then get's

[1] https://www.crummy.com/software/BeautifulSoup/bs4/doc/

processed and forwarded to DynamoDB using Lambda function. The transformation process is composed of sending texts to the Comprehend service for sentiment analysis and matching against all keywords observed by the users.

The CloudWatch service is used to monitor the health status of the whole pipeline. The unusual structure of data and exceptions are logged.

3.2 Data flow

The complete flow of the data in the system is presented in Figure 2. Bidirectional connections are related to keywords. Mention objects passed throughout components are uniquely identified by their UUIDs. The Transformer function may create and save to the database multiple copies of such an object if there are several keywords detected in the text of the mention.

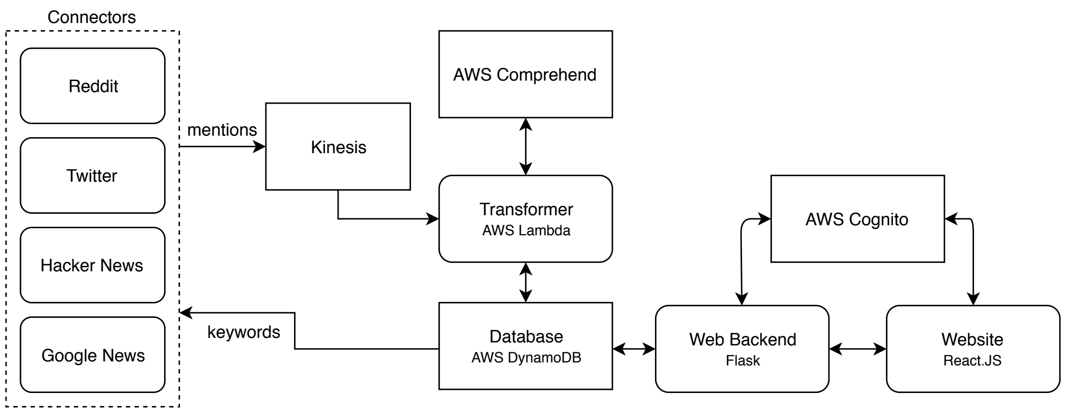

Figure 2: Data flow throughout the system's components.

3.3 Cloud services and technologies

The system utilizes multiple services to gather and process data. The most important ones are depicted in Figure 3. Only manual integrations are being taken into regard – services indirectly (e.g. AutoScaling and CloudWatch for AWS Beanstalk) are omitted for brevity.

3.4 Scalability

In order to assure the scalability of the whole system every part of it has to be scalable (preferably horizontally). Connectors can be placed on separate, high-end machines (vertical scaling) and instances can be partitioned based on subset of particular source's

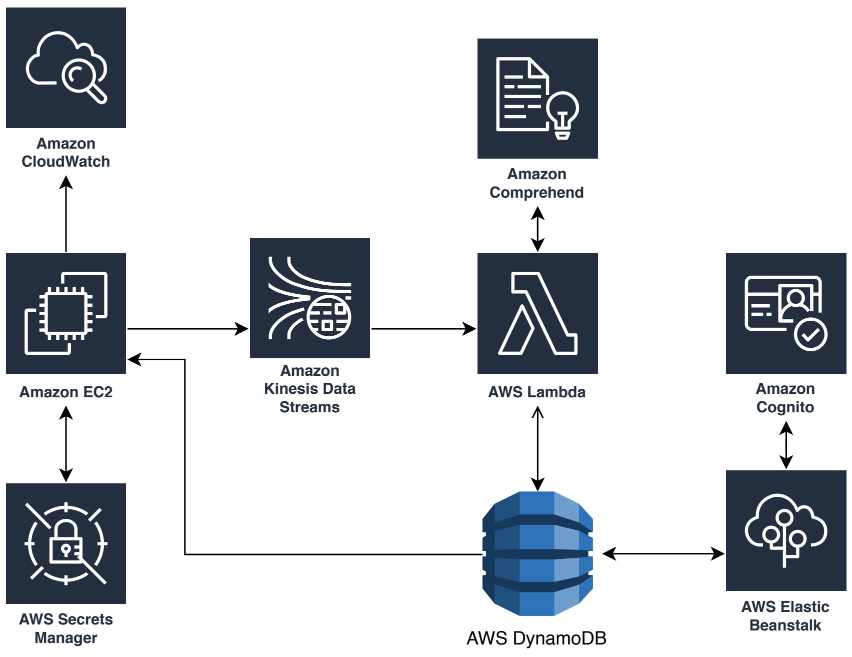

Figure 3: Services used in each of the system's components.

data stream (subreddits in case of Reddit, geographical regions in Twitter). Kinesis Data Streams are able to handle loads of terabytes per hour depending on the number of allocated shards. Data processing in Lambda functions is performed in batches of configurable size and is vertically scalable. DynamoDb is a NoSQL database designed to handle high loads, thus read and write throughput can be adjusted depending on the usage.

4 Components

The system is divided into several components, which are logically and technically separated from each other.

4.1 Connectors

Connectors are the modules responsible for collecting the data from the sources. They are placed on EC2 instances. Each of the connectors is listening to the stream of messages – text data feed from its source. Received texts are compared against the set of observed keywords and detected mentions are converted from JSON responses to the unified format. The common object representation guarantees proper validation. Later, mentions are forwarded to the AWS Kinesis data stream.

For performance reasons each listener for every source runs in another process and there is no further preprocessing of mentions in the connectors. Moreover, the keyword finder detects only the first match of any keyword to speed up the processing. Updates to the keywords set are handled by a listener running on a separate thread.

The acora package was used for efficient keyword detection – it provides an implementation of the Aho–Corasick algorithm. This algorithm is capable of detecting multiple keywords at once and processing text characters only once without backtracking. The algorithm works by building minimal finite-state machine for a given keywords set. The build automaton highly resembles a triee with additional connections. Use of acora functionality that allows to break algorithm operation on the first occurance of keyword provided efficient way of detecting interesting texts.

Robust error handling was introduced to ensure continuance of data streams and resilience to anomalies in the incoming data. Connectors are gathering logs in daily-rotating file sinks stored locally on the machine. Entries with higher severity levels (warn and above) are also published to AWS CloudWatch logs. Custom metrics describing incoming data – items and hits counts – are periodically gathered and forwarded to CloudWatch by service running on another thread to avoid additional load on connectors thread and prevent time drift. Such data is later displayed on the dashboard and used as trigger for alarms regarding stream health.

The secrets required to access external API are downloaded and stored using AWS SecretsManager.

4.2 Data stream

Streaming mentions data from connectors was implemented with the AWS Kinesis Data Streams. Data Streams is a real-time data streaming platform providing scalability and durability of consumed data with 24 hours retention. This service is able to process huge amounts of real time data, but scaling shards capacity is not managed by AWS and can only be configured from the management console or the AWS API.

AWS Kinesis Firehose was considered for this use case. Unlike Kinesis Data Streams, Kinesis Firehose scalability is managed by AWS, but restriction on the type of consumers to S3 and Redshift did not allow its use.

Connectors integration with data streams is provided by the kinesis-python library which wrapps all functionality in two classes for consumer and producer.

Attribute name	Data type	Additonal info
Author	string	partition key
OriginDate	datetime	range key
Keyword	string	index partition key
SentimentScore	float	
Id	UUID	
DownloadDate	datetime	
Url	string	
Text	string	
Source	string	
Metadata	JSON	

Table 2: Mention model schema

4.3 Lambdas

Mentions data transferred by AWS Kinesis Data Streams is consumed by Lambda function which calls AWS Comprehend, retrieves text sentiment, creates a model for every keyword occurrence independently and saves the enriched data into Dynamodb.

Keyword occurrence detection was implemented with the nltk package providing support for multi-word expression tokenizer. This solution allows to leverage handling of white characters provided by the tokenizer and find all keywords in given text. Tokenizer allows for finding all singe and multi-word keywords in efficient manner. Efficiency is provided by building trie of tokens in initial step of algorithm and then doing fast lookup for keywords.

Serverless framework was used in order to improve productivity of development Lambda functions and deploying changes to AWS. Serverless provides unified interface for developing serverless solutions for different clouds. Deploying function to AWS requires specifying configuration file specifying necessary permissions to resources and events triggering function execution. In case of AWS cloud as a target, Serverless configuration file other than shared parameters for all targets, has fragments of AWS Cloudformation scripts.

4.4 Database

The results of data processing described in previous section are stored in Amazon DynamoDb. Amazon DynamoDb is a fully managed, the no-sql document database with high scaling capabilities across mulitple nodes in many regions.

Attribute name	Data type	Additional info
User	string	partition key
Keyword	string	range key
CreationDate	datetime	

Table 3: Keyword model schema

The choice of the database had a huge impact on the models schema. DynamoDb has a concept of *partition key* and *sort key*. The first one is used to determine the node in which the document will be stored, the second one defines the order in which the documents will be stored within a partition. Also, to perform queries other than full table scan, it is necessary to provide a partition key to query against (this limitation can be mitigated with global secondary indexes).

The schema for the Mention model is described in table 2. It was decided that the author name will serve as the partition key and creation date as a sort key. Such pair will provide the most optimal distribution of entries across database nodes. It is worth mentioning that other variants were considered, mainly keyword as partition and date as sort key. However, this variant did not provide good distribution across nodes — the number of keywords is expected to be small in comparison to the amount of data. If one keyword would happen to dominate over the others it might lead to one database node being overloaded while other observe low traffic.

The database was carefully indexed to achieve best performance for the desired access paths. The most frequent query will be finding mentions of specific keyword. To perform the query efficiently the global secondary index was created, with keyword as partition key and date as sort key.

4.5 Web application

To present the results of data collection, a dedicated web application was created. It consists of two components.The first one is a backend app written using the Flask framework. It's responsibility is to fetch data from the DynamoDb database and perform necessary aggregations and present results to the frontend application. It also enables users to manage their keyword sets. The app also utilizes PynamoDb — a library for interactions with DynamoDb database and pandas for data manipulation.

The second component is frontend React.js app written in Typescript. It primarily consists of a dashboard written using the plotly.js and material-ui libraries. It allows the user to query for mentions for their set of keywords over a selected period of time.

The dashboard displays plots of sentiment over time and total number of mentions over time. It also allows to dig into details of a specific mention. Moreover, the user can also observe a specific keyword in live mode – the dashboard get's updated with new incoming data.

User management and authentication is handled by the Amazon Cognito service. It acts as an identity provider for both components using OAuth2.0 standard.

Both application were built as Docker images and then deployed to Amazon Elastic Beanstalk service.

5 Development process

The whole development process was based on the GitHub platform. As our solution consisted of multiple projects we used a monorepo to host all of them together. The common code – data models – was placed in a shared Python package.

To ensure high development quality several measures were undertaken including code quality assurance (linting, formatting and type checking, code reviews), containerization, continuous integration and continuous deployment.

5.1 Code quality

In order to efficiently work on a big, long running project it is very important to maintain high code quality. Code review was invaluable in enforcing consistent naming conventions, maintaining common style and catching code logic bugs. Each person was the owner of some modules of the system, thus was responsible for theirs development and adherence with overall architecture assumptions.

Code analysis tools like black formatter and flake linter where used to enforce consistent style conventions in Python code. Usage of type hints along with the type checking tool mypy helped to avoid many errors resulting from the dynamic nature of Python language. Another package – Pydantic provided field validation at runtime.

To minimize the number of regression errors and assuring correctness we maintained high unit test coverage. For the most part, tests checked correctness of communication with external APIs including response parsing, validation checking and conversions.

Test were written using pytest, pymock, jsonpickle among others. Pytest provides comfortable methods for writing parameterized tests and test fixtures which make writing tests faster and eliminate code duplication. Pymock was helpful in cutting off dependencies in tests, in accordance with good unit tests practises. For tests of webserver backed mock of DynamoDb was created to provide stable and reproducible integration tests. Jsonpickle was used to save responses in readable file format and used in tests fixtures to check correctness of parsing methods.

5.2 Continuous integration and deployment

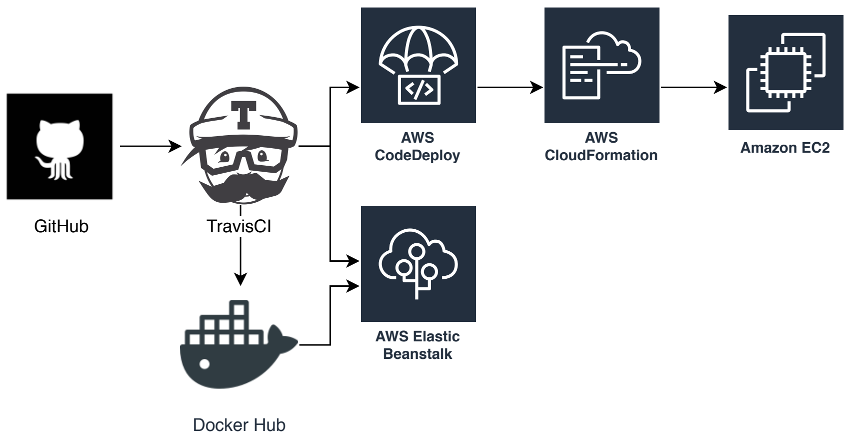

Figure 4: Services used in the development process of the system.

To ensure good standards of code and speed up development and deployment process, Continuous Integration and Deployment service – Travis – was used. Each push to the project repository on GitHub would trigger a build process, during which the apps would be built and linter checks and tests would be run. Additionally, if changes were applied to the main development branch, they would be automatically deployed to the AWS infrastructure. Analyzers running on Amazon Lambda would be deployed directly. In case of web applications (both frontend and backend), first docker images would be built and pushed to DockerHub. Then, updated apps configuration would be deployed to Amazon Elastic Beanstalk. In case of connectors, deployment relies on AWS CodeDeploy service.

5.2.1 Infrastructure

Deployment of the connectors required CodeDeploy and appropriate AppSpec files to prepare environment and start all of the processes. Additionally, appropriate KeyPairs has to be created in order to access EC2 instances using ssh.

To automate management of all resources needed in the system we utilized Cloud Formation templates. Separate files were created for each of the solution's modules.

A Libraries and licences

Name	Version	License	URL	Description
acora	2.2	BSD	website	fast multi-keyword text search engine
awscli	1.16.155	Apache 2.0	website	Command Line Environment for AWS.
beautifulsoup4	4.7.1	MIT	website	Screen-scraping library
black	19.3b0	MIT	website	The uncompromising code formatter.
boto3	1.9.148	Apache 2.0	website	The AWS SDK for Python
botocore	1.12.159	Apache 2.0	website	Low-level, data-driven core of boto 3.
Click	7.0	BSD	website	Composable CLIs toolkit
cloudwatch-fluent-metrics	0.3.0	BSD	website	AWS CloudWatch Fluent Metrics
flake8	3.7.7	MIT	website	the modular source code checker
Flask	1.0.2	BSD	website	A simple web framework.
Flask-Cognito	1.13	ABRMS	website	Authenticate users to Cognito user pool via JWT.
jsonpickle	1.1	BSD	website	Python serialization library
kinesis-python	0.1.8	Apache 2.0	website	AWS Kinesis producer and consumer library
mypy	0.701	MIT License	website	Optional static typing for Python
newsapi-python	0.2.3	MIT	website	An unofficial Python client for the News API
nltk	3.4.2	Apache 2.0	website	The Natural Language Toolkit
numpy	1.16.3	BSD	website	Array computing with Python.
pandas	0.24.2	BSD	website	Powerful data structures
praw	6.1.1	FreeBSD	website	'Python Reddit API Wrapper'
prawcore	1.0.1	FreeBSD	website	Core layer for PRAW 4+.
psaw	0.0.7	FreeBSD	website	Pushshift.io API Wrapper for reddit.com public comment/submission search
pydantic	0.27	MIT	website	Data validation library
pynamodb	3.3.3	MIT	website	A Pythonic Interface to DynamoDB
pytest	4.5.0	MIT license	website	simple powerful testing with Python
python-hn	0.0.3	MIT	website	library for Hacker News Search API
python-twitter	3.5	Apache 2.0	website	A Python wrapper for the Twitter API
requests	2.21.0	Apache 2.0	website	Python HTTP for Humans.
rope	0.14.0	GNU GPL	website	a python refactoring library

Table 4: Listings of main packages used in the system.

B Manual

In order to log in to the web application perform following steps:

1. Go to `http://sentinel.eu-central-1.elasticbeanstalk.com/`.
2. Log in as `sentinel-test@mini.pw.edu.pl` using `Iamminitester2019` as password.
3. In `Menu > Keywords` you can manage the set of observed keywords.
4. In `Menu > Home` you can plot the sentiment value and number of mentions regarding keywords and date range specified in dropdown controls. You can use `Live Mode` checkbox to enable auto-refresh of data using previously specified query.

In order to set up whole infrastructure:

1. Create KeyPair named `sentinel-connectors-key_pair`.
2. Use CloudFormation stacks in `cloud_formation` folder of the code repository to set up individual components.
3. In SecretsManager console substitute temporary key values for real secrets for external APIs.
4. Deploy connectors, lambda functions and web application using TravisCI based on configuration file `.travis.yml`. Appropriate credentials to AWS have to be set in Travis before.