Log Centralisation and Audit Log: Process to track changes in applications and process logs for real-time monitoring

Create a centralised log of changes happening in various applications across various platforms and store it to enable querying to generate detailed reports of historical changes done in every application.

Log Centralisation:

Objective:

1. A central repository to visualise and monitor various logs being generated by different applications across different modules.
2. Real-time monitoring of various API endpoints and alerting in case of anomalies.
3. Provide interface for easy debugging of APIs by providing detailed information in case of failure.

Proposed Architecture:

Proposed Tech Stack:

1. Elasticsearch: The logs will be stored in Elasticsearch after being parsed in JSON format. Each Log entry will be stored as a json document in an elasticsearch index. Indexes will be created on a daily basis depending on the date and time of the log entry.
2. Logstash/Filebeat: Filebeat is a log forwarder which can be used to combine log files from various servers and forward them for parsing to logstash. Logstash parses these logs and outputs them to elasticsearch for querying and visualisation.
3. Kibana: Kibana acts as a visualisation tool which can be used as a centralised dashboard for real-time monitoring of APIs based on the various logs aggregated through logstash and filebeat.

Ingestion Pipeline Architecture:

1. The logs will be inserted using filebeat which will be setup on various servers reading and forwarding log files to a logstash pipeline.

2. Filebeat will be setup on all application servers and LoadBalancer servers and will continuously stream log data to logstash for parsing.

MiddleWare Architecture:

1. Logstash will be the middleware and the key job of logstash will be to parse logs into json documents and output them to elasticsearch.

Data Model:

1. There will be three indexes per day for each module.
    a. Nginx Logs index: This index will be used to monitor a modules access and response times. Beyond a certain threshold of response time latency a mail alert will be generated for the concerned stakeholders.
    b. Tomcat Logs Index. This index will be used to monitor access logs of various microservices associated with various modules. This will focus on monitoring the response times of microservices and alert mails will be sent to the concerned stakeholders when latency is reached beyond a predefined threshold.
    c. Application logs Index: These logs will focus on error logging and providing an interface for easy debugging of APIs and microservices. Alert mails will be sent in case of frequent errors so that action can be taken immediately.

AuditLogs:

Objective:

1. Create a common repository of changes of all applications and provide an input source for capturing changes in different applications.
2. Create a query layer to query on that data and generate reports or a reporting dashboard based on the results.
3. The Audit Log project will be a generic repository of changes and will be independent of the application it is tracking.

Proposed Architecture:

Proposed Tech Stack:

1. MongoDB/Elasticsearch: It will act as the primary datastore to store the changes in form of JSON documents. Each transaction or a change will be stored as a json document in the datastore.

2. Apache Kafka: Apache Kafka will act as a middle layer between the input source and the primary datastore. The middle layer will be a publisher/Subscriber queue where data from various sources or applications will be added and then consumed and added to the primary datastore. The main function of the middle layer is to increase the durability of the data in case the primary database goes offline for some time.
3. Spring/Spring Boot: This will be used to create a group of microservices which will help register an Application in the Audit Log and start the tracking of applications in a uniform manner.

Ingestion Pipeline Architecture:

1. The ingestion pipeline will consist of two REST APIs: Application Registration API and Tracking API. The proposed structure of the same is given below:

   App Registration API: Used to register an application with the audit log process. There will be a list of "dynamic parameters" also provided to create a mapping of dynamic parameter aliases with actual parameter names.

Method Type	POST
URL	http://api.moglix.com/moglix/ingestion/registerapp
Request Params	{"app_id":"1234","owner_name":"Vaibhav","owner_id":"1234","app_name":"Sample Application","datetime":"2019-03-19T20:17:46.384Z","added_date":"2011-08-12T20:17:46.384Z","vertical":"online","app_version":"1.0"}
Response	{"status":200,"message":"App Registered Successfully."} in case of a successful insert

   App Tracking API: Used to capture application changes and produce them in the Kafka cluster.

Method Type	POST
URL	http://api.moglix.com/audit-log/ingestion/{app_name}
Request Params	{"user_id":"4567","owner_id":"1234","datetime":"2019-03-19T20:17:46.384Z","username":"Vaibhav","owner_name":"Vaibhav","app_version":"1.0","modified_at":"2019-03-19T20:17:46.384Z","app_id":"1234","app_name":"sample application","data":{"description":"","reason":""},"vertical":"online","modification_id":"c0a027fa067e48c2a9628155e48ba87f","dynamic_paramters":{"d1":"d1_value","d2":"d2_value","d3":"d3_value","d4":"d4_value","d5":"d5_value","d6":"d6_value","d7":"d7_value","d8":"d8_value","d9":"d9_value","d10":"d10_value"}}
Response	{"status":200,"message":"Records successfully stored."} in case of a successful insert

MiddleWare Architecture:

The middleware as mentioned above will consist of an Apache Kafka Cluster to insert the application changes in the primary datastore.

1. Each application will have its own topic which will be dynamically created whenever an application registers with the audit-log process.
2. The topic partitions and replicas will also be set dynamically for each topic created.
3. Each topic will have 5 partitions to start with but this number will change depending on the number of messages produced and throughput target.
4. A consumer will be written to read all topics created by the applications and will be configured to look for new topics added every two minute so that it could start consuming a new topic with minimum lag. We can even have different consumer groups read from different topics. The number of consumer threads will be equal to the number of topic partitions.
5. The topics created will have “audit-log” prefix in the topic name to separate them from other topics that might be created for other purposes.

Data Model:

We can have three collections/indexes to capture the data for the tracking of application changes.

- App Registration: This table will hold all the applications that have registered with the audit log process.
- App Tracking: There can be one common table to hold the changelog data of all applications or separate table to hold the changelog data of each application. This will depend on the data volume and the type of queries that are to be performed on the data.
- App parameter mapping: In this table the mapping for the field dynamic parameters will be stored per application so that the parameters are mapped with the applications actual parameters.

Data Ingestion Scheme:

1. App Registration: Whenever an application registers with audit-log using the app registration API:
    a. an entry will be made in the app registration table/Index. The insert will be initiated through the App registration API and the app_id will be the unique identifier or the row_id of the record.

    b. An entry will be created against the application name in the App parameter mapping table. This will have app_id as row_id and dynamic parameters will be stored with corresponding actual parameters in form of json.
    c. A new application tracking index will be created and mapping of that index will correspond to the json provided in the App registration request and all the dynamic parameters will have “string” datatype.
2. App Tracking: Whenever an application start publishing the data in the audit log using the ingestion API:
    a. The consumer will start processing the data and start to insert data in the tracking index.
    b. The mapping of dynamic parameters will be loaded from the App parameter mapping index once and stored in a Cache Structure and it the mapping will be read from here after this.