A Microservices Architecture for Collaborative Document Editing Enhanced with Face Recognition

Cristian Gadea, Mircea Trifan, Dan Ionescu
School of Electrical Engineering and Computer Science
University of Ottawa
Ottawa, Ontario, Canada
cgadea, mircea, dan@ncct.uottawa.ca

Marius Cordea, Bogdan Ionescu
Mgestyk Technologies Inc.
Ottawa, Ontario, Canada
marius, bogdan@mgestyk.com

***Abstract*—Modern web applications can now provide rich and dynamic user experiences, such as allowing multiple users to collaboratively edit rich-text documents in real-time from multiple devices. Application architectures are evolving to support the development and deployment of such interactive functionality by decoupling software components into microservices. This paper introduces the architecture and the implementation of a collaborative rich-text editor that makes use of microservices to enable and enhance its scalable co-editing functionality. This includes microservices for synchronizing unstructured text using operational transformations, for chat functionality, and for detecting and recognizing faces in images added to the editor. The architecture makes use of Docker to allow for the development and testing of individual services as separate containers enabling seamless deployment across the available network of computers and other computing devices. The system will be demonstrated by showing how microservices make it possible for multiple users to co-edit a document where images containing faces are added and recognized as part of the document content, thereby supporting the document creation process.**

***Index Terms*—collaborative editing, real-time web, microservices, docker, groupware, face detection, face recognition**

I. Introduction

With the rapid release cycles of today's modern web browsers, real-time protocols and standards such as WebSocket and WebRTC are now widely supported and have enabled the development of content driven online collaboration tools to facilitate information sharing, document management, audio and video stream, tasks coordination and communication among groups. Collaboration tools providing functionality such as chat, email sharing, web conferencing, whiteboard sketching, text editing, and resource sharing are now commonly available. Products such as Google Docs [1] have shown that web-based collaborative editing, which consists of multiple users changing a document at the same time from within their web-browser, is now possible in a reliable way, with benefits in user productivity and efficiency. The benefits in user productivity and efficiency of collaboration within products like Google Docs or Microsoft Sharepoint are discussed in [2] that addresses somewhat a larger scope: Enterprise Content Management solutions. With Google Docs, documents are limited to being stored within Google's data centers, and the proprietary synchronization algorithm and document format further hinders the work of developers. The stability of real-time browser-based communication has therefore allowed conflict-resolution techniques such as operational transformations (OT) [3] to be implemented to allow multiple users to work on the same rich-text document (containing advanced features such as tables) at the same time. The insertion of more interactive and complex data such as annotated pictures and video streams into synchronized documents continues to be an active area of research.

As an antidote to the increasing complexity of on-the-cloud services, the idea of breaking down sophisticated applications into granular services offered by various independent components of an application (and which are communicating with each other using language-agnostic APIs) looked promising to researchers in both academia and industry. A small fine-grained component which performs a single function, can be scaled elastically, is resilient, is composable through APIs, is minimal yet complete, and is easy to deploy and test has been called a "microservice". The intention was to create various patterns which can be assembled into solid and easy-to-manage services with reduced memory footprints.

In this paper, a basic architecture for a real-time web-based collaboration platform is introduced whose components are designed by following the principles of microservices. This paper builds on existing real-time web-based collaboration research previously described in [4][5].

Microservices, which were designed and developed as components of the proposed architecture, allow for flexible deployments independent of operating systems, databases and languages, as well as shorter development times. It gives smaller teams of designers and developers complete ownership of vertical slices of related developments down from the databases to the user interface. OT-based collaboration functionality is developed and deployed as a microservice that web clients can use to enable rich-text co-editing. Text-based chat functionality is a separate microservice which manages chat-related functionality such as keeping a history of previous messages.

In addition to demonstrating the separation of an application into the two microservices required for basic co-editing functionality, the editor is also enhanced through a novel component. A Face Recognition microservice that provides server-side image processing to enroll, detect and recognize faces from a separate Template Database has been added due to increasing importance of such components for the use in industries such as security. By dragging and dropping an image into the editor, users receive additional information about the image directly inside the editor, which is useful when composing a research document. It will be shown how the flexibility of the proposed microservices-based architecture makes it easy to add such powerful server-side features to a real-time collaboration scenario, thereby enabling rich multi-user collaboration for research and development projects.

The remaining sections of this paper are as follows: Section II covers other relevant work in this domain. Section III then presents the microservices architecture. Section IV provides the implementation details for this architecture and Section V discusses the resulting system. Finally, Section VI provides concluding remarks and avenues for future work.

II. Related Work

Microservices, as described extensively in [6], are a new type of software architecture that composes an application as a collection of independent individual services. Each service is dedicated to a single business capability. Services can be implemented in many programming languages and can be tested and deployed automatically.

The increasing demand for a more manageable virtualization layer has spurred the creation of open-source containerization solutions such as Docker (described in [7]), which was used to enable the microservices architecture proposed in this paper. Docker provides an additional layer of abstraction and automation of virtualization by making use of resource isolation features of the Linux operating system. Docker "containers" consist of deployed applications. This is referred to as "container based virtualization" or "operating system level virtualization" (as opposed to hardware virtualization provided by VMWare or VirtualBox). The advantages of using Docker include rapid application deployment, portability across machines, component reuse and version control, minimal overhead, a lightweight footprint, sharing, and simplified maintenance. While they did not focus on real-time collaboration, architectures that make use of Docker-based microservices have been previously explored in [8] and [9].

Collaboration tools have been defined as an integrated combination of collaboration technologies. A collaboration tool has to offer features such as connectedness, awareness, sharing, and communication in order to facilitate collaborative work [10]. Communication features refer to pushing or pulling information into or out of an organization, collaboration refers to sharing the information and building a shared understanding, and coordination refers to the delegation of tasks, including via session coordination protocols like Session Initiation Protocol (SIP).

In 2009, Google started beta-testing Google Wave, a real-time collaboration environment which Google hoped would eventually displace email and instant messaging. Acquired by Google for possible inclusion in Google Wave, EtherPad was another web-based editors to allow for smooth, character-by-character real-time text collaboration using operational transformations. Both Google Wave and EtherPad are now open source projects [11][12].

Heinrich et. al. [13] present a Generic Collaboration Infrastructure for the implementation of collaborative web applications and successfully applies it to existing rich-text editors. By converting changes to the browser's Document Object Model (DOM) into OT operations, web-based editor content remains synchronized among multiple users in real-time. The use of additional microservices to enhance the editing experience was not explored, however.

A classification of existing libraries for web based shared editing including Google Docs Real-Time API is presented in [14].

The inclusion of microservices to assist the document editing process was recently explored by the New York Times [15]. The NY Times Editor allows journalists to tap into the power of machine learning algorithms by applying tagging and annotations as the users type new words. The additional information that is brought in from the microservice augments the text and assists the author in real-time. They found the microservices-based editor to be a welcome change from their previous monolithic content management system (CMS). Their prototype used a simple web-based text editor and did not have any collaborative features.

This paper will present a microservices-based architecture that uses Docker to allow augmenting a collaborative rich-text editor with features essential for collaboration such as chat, as well as more complex features such as face recognition of inserted images. To the knowledge of the authors, this is the first time that face recognition functionality has been integrated into a web-based editor.

III. Microservice-Based Architecture for a Collaborative Platform

The architecture proposed in this paper uses separately deployable service components to enable high scalability and ease of deployment. Unlike in monolithic systems which can quickly grow in complexity and become unmanageable, microservices are decoupled, distributed, and independent of each other. The system to be implemented consists of a web-based user interface containing a rich-text editor that provides collaboration, chat and face recognition functionality, all in one.

The following functional requirements have therefore been identified for the system:

1) A web-based user interface must provide access to rich-text editing functionality (styles, tables, pictures, etc.) within a real-time collaborative editor.
2) All document changes are synchronized in real-time among all users within a specific session by using XML-based operational transformations (OT) to synchronize the editor's DOM contents.
3) Documents must be persisted in a database.

4) Users must be able to chat with other users in the same session.
5) A history of chat messages that pertain to a specific document must be retrieved the next time the document is accessed.
6) Users must be able to insert images into the editor via a drag-and-drop action.
7) Any image within the document can be submitted for processing by a face detection algorithm.
8) Any image in which at least one face has been detected can be submitted for processing by a face recognition algorithm.
9) Any image in which one face has been detected can be enrolled into a database by providing at least a name for the individual in the image.

The proposed system has the following non-functional requirements:

1) Users must be able to access the user interface via a modern web browser.
2) The system must be capable of being deployed to major cloud providers such as Amazon AWS.
3) The system must scale by supporting the deployment of new collaboration servers, chat servers, and face recognition servers, as well as their corresponding databases.

Figure 1 illustrates the microservices architecture that allows the three previously-described services to be developed by separate teams, as well as tested and deployed individually. The web client is denoted as *Editor UI* and contains the user interface (UI) for the editor. It can make use of MVC Frameworks such as Angular [16] or Bootstrap [17] to create a pleasing user interface for additional elements such as the chat.

In order to provide the collaborative editing functionality, the contents of the JavaScript-based editor must be synchronized to the *Document Database* via a *Collaboration Server* that makes use of operational transformation technology to allow multiple (possibly conflicting) real-time changes to result with a consistent state of the document. The OT-based *Collaboration Server* is therefore the main component which supports the collaborative editing functionality of the architecture introduced in this paper. It enables functionality such as group undo, locking, conflict resolution, operation notification and compression, group awareness, and HTML/XML tree-structured document editing, among other features.

The *Chat Server* provides text-based messaging ability for all users in the collaborative session, and persists their messages in the *Chat History Database* for future retrieval.

The *Face Recognition Server* performs the enrolling, detection and recognition functionality by using the *Template Database* to store enrolled faces and comparing them with faces on user-provided images.

IV. Implementation

This section describes the decisions that were made when implementing the microservices-based architecture described in Section III.

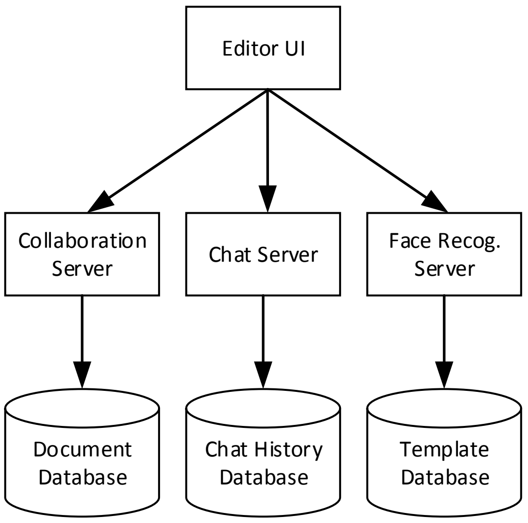


Fig. 1. Basic microservices architecture for collaborative editor enhanced with face recognition.

A. Database Microservices

The following data repositories were created as microservices:

1) MongoDB for the *Document Database*
2) MySQL for the *Chat History Database*
3) MySQL for the *Template Database*

The implementation used Docker to create and manage all microservices. Docker provides a standardized set of commands to interact with images and containers. Docker images are created from scripts saved as a `Dockerfile`. The image built from the `Dockerfile` is then executed to produce a running microservice container. An example of a `Dockerfile` used to create a custom MySQL database image is shown in Listing 1 below. The image uses the official MySQL version 5.5 image from the public Docker Hub repository [18] and modifies the `my.cnf` configuration file to allow for larger file upload sizes.

Listing 1
Sample Dockerfile for modified MySQL image.

```
FROM mysql:5.5
MAINTAINER testuser1@ncct.uottawa.ca

ENV MYSQL_ROOT_PASSWORD ****
VOLUME /var/lib/mysql

RUN echo "max_allowed_packet = 16M" /etc/mysql
    ↪ /my.cnf

EXPOSE 3306
```

The Docker `build` command is used to construct an image and the `run` command is used to start a container based on the created image. Listing 2 first shows the command for building the MySQL image based on the `Dockerfile` in Listing 1. The `run` command then creates the corresponding container by specifying the local path to mount as a volume into the

TABLE I
FACE DETECTION AND RECOGNITION REST API

Function	Description
/init	Initializes the system.
/enroll	Creates a model of faces and stores them in the database.
/match	Compares facial images and gives the euclidean distance.
/getEnrolled	Returns the IDs of faces existing in the database.
/detect	Detects faces in the image.

image where the data will be persisted, thereby surviving container restarts. Finally, the `docker run` command is shown that allows starting an interactive shell into the previously-launched mysql container, which is useful for running manual commands and queries against the database.

Listing 2
SAMPLE DOCKER COMMANDS.

```
docker build -t="mysqldb" .

docker run -v //c/Users/user/mysql_data:/var/
    ↪ lib/mysql --name mysqldbn -e
    ↪ MYSQL_ROOT_PASSWORD=**** mysqldb

docker run -it --link mysqldbn:mysql --rm
    ↪ mysql sh -c 'exec mysql -h"172.17.0.3" -
    ↪ P"3306" -uroot -p"****"'
```

B. Main Microservices

The following additional components were implemented as microservices to provide the main functionality required from the system:

1) *Webserver* - A simple Node.js-based (Express) web server was created to host the HTML, JavaScript and CSS files developed by the UI team.
2) *Collaboration Server* - A Node.js server was written by a team of developers to support Operational Transformations and to communicate with the web client via a WebSocket connection.
3) *Chat Server* - A JEE (Java Enterprise Edition) server was developed by a different team of developers by using Java Spring and WebSockets to enable chat and chat history functionality.
4) *Face Recognition Server* - A JEE server was developed by a team to provide a REST API to image processing functions accessed using Java Native Interface (JNI) from libraries implemented in C.

While the *Collaboration Server* microservice and the *Chat Server* microservice communicate using custom JSON-based messages using the WebSocket protocol, messages are exchanged with the `Face Recognition Server` microservice by using the REST API shown in Table I.

C. Managing Multiple Microservices

Docker Compose is a tool from the Docker ecosystem allowing the definition and execution of a multi-container application [19]. It uses a configuration file written in the YAML ("YAML Ain't Markup Language") format, a simplified variant of JSON that uses indentations.

Listing 3 shows an example of the `docker-compose` file for the setup of the data volume of MongoDB, the MongoDB daemon, the *Collaboration Server* and the web server for the web client. The *Chat Server* and *Face Recognition Server* were included in a similar fashion.

Listing 3
PARTIAL CONTENTS OF "DOCKER-COMPOSE" FILE.

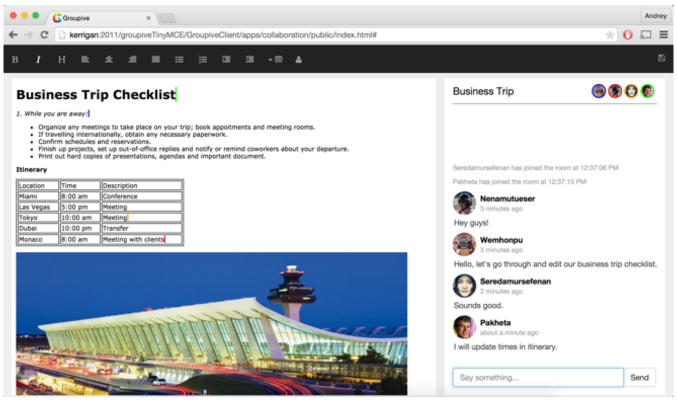
```
mongodata:
  image: mongo:latest
  volumes:
    - /data/db
  command: --break-mongo

mongo:
  image: mongo:latest
  volumes_from:
    - mongodata
  ports:
    - "27017:27017"
  command: --smallfiles

collaborationserver:
  build: CollaborationServer
  ports:
    - "8080:8080"

webserver:
  build: CollaborationClient
  ports:
    - "8090:8090"
```

To help manage and organize Docker images, a Docker Registry [20] is also available for installation and was used throughout the implementation and deployment stages of this system. It is itself implemented as a Docker container that holds images of the various microservices used. The stored microservices are annotated by using JSON-formatted metadata, allowing images to be queried by their metadata. Once an image is built and stored in the registry, it can be deployed and a container from any machine that has access to the registry. By deploying multiple docker containers, the computational load (such as the OT algorithms of the Collaboration Server and the image processing of the Face Recognition Server) can be distributed across different physical machines.

D. Building and Testing

Due to the fact that the implementation consists of a suite of smaller services, additional testing opportunities are present. Microservices allow more testing options such as unit, integration, components, contracts and end-to-end tests. When compared to the monolith approach, microservices make it is easier to define the tests for the isolated component behaviour where clear boundaries are present.

The unit, integration and component tests can be lumped together to make sure the microservice business logic is implemented correctly. At a different granularity, tools such as Selenium [21] are used for the end-to-end user tests involving



Groupive

kerrigan:2011/groupiveTinyMCE/GroupiveClient/apps/collaboration/public/index.html#

Andrey

Business Trip Checklist

1. While you are away:

- Organize any meetings to take place on your trip; book appoitments and meeting rooms.
- If travelling internationally, obtain any necessary paperwork.
- Confirm schedules and reservations.
- Finish up projects, set up out-of-office replies and notify or remind coworkers about your departure.
- Print out hard copies of presentations, agendas and important document.

Itinerary

Location	Time	Description
Miami	8:00 am	Conference
Las Vegas	5:00 pm	Meeting
Tokyo	10:00 am	Meeting
Dubai	10:00 pm	Transfer
Monaco	8:00 am	Meeting with clients

Business Trip

Seredamursefenan has joined the room at 12:37:06 PM

Pakheta has joined the room at 12:37:15 PM

Nenamutueser 3 minutes ago
Hey guys!

Wemhonpu 3 minutes ago
Hello, let's go through and edit our business trip checklist.

Seredamursefenan 2 minutes ago
Sounds good.

Pakheta about a minute ago
I will update times in itinerary.

Fig. 2. A four-user collaborative document editing session with a table and other rich-text elements.

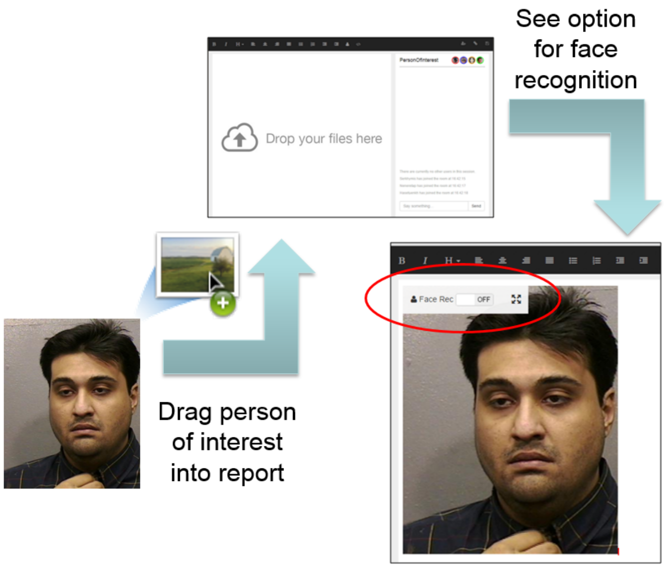
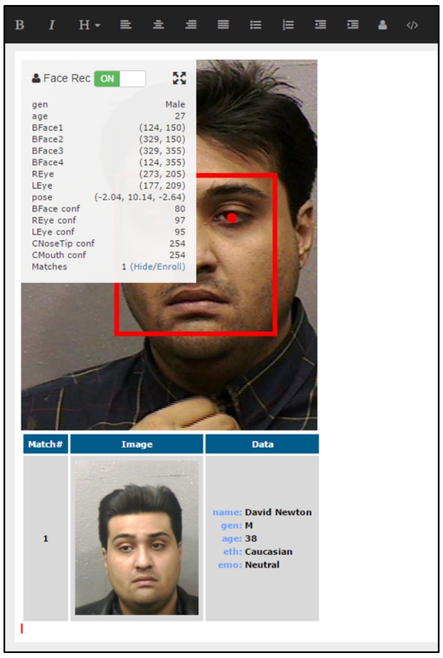

See option for face recognition
Drop your files here
Drag person of interest into report
Face Rec
OFF

Fig. 3. Drag-and-drop operation reveals on/off button for face recognition on top of any image (button is circled in red).



Face Rec ON

```
gen                         Male
age                           27
BFace1                (124, 150)
BFace2                (329, 150)
BFace3                (329, 355)
BFace4                (124, 355)
REye                  (273, 205)
LEye                  (177, 209)
pose     (-2.04, 10.14, -2.64)
BFace conf                    80
REye conf                     97
LEye conf                     95
CNoseTip conf                254
CMouth conf                  254
Matches          1 (Hide/Enroll)
```

Match#	Image	Data
1		name: David Newton gen: M age: 38 eth: Caucasian emo: Neutral

Fig. 4. Applying face detection to the image reveals key landmarks and detected attributes, as well as database matches.

verifying GUI behaviour. Selenium is an open source web application testing framework. It automates tests with a record and play mechanism. Multiple web browsers and operating systems are supported.

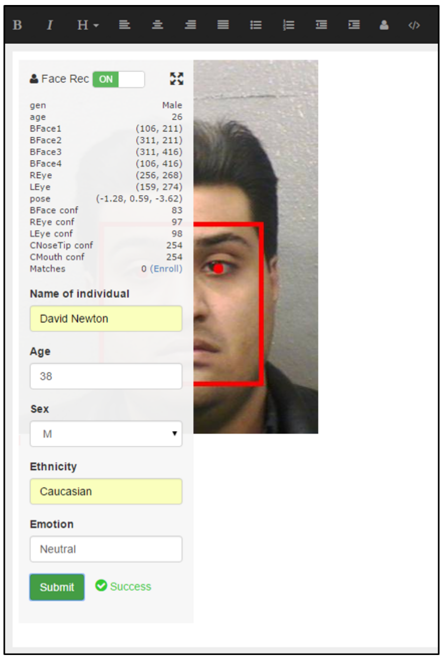
Jenkins [22], another free Java based tool for continuous integration, was used to build the various components and deploy them as Docker containers. The Selenium tests, which were configured to perform actions such as open multiple browsers and simulate multiple users typing in the editor at the same time, were then executed by Jenkins and the results monitored through the Jenkins user interface. If all tests pass, it is a good sign that the docker container is ready for deployment into production.

V. Results

The resulting collaboration system was first evaluated by connecting multiple users to the same document and performing typical rich-text document editing actions. Figure 2 shows a four-user collaboration session (online users are denoted by the coloured images in the top-right corner of the interface). The document along the left side of the screen remains synchronized, even with complex rich-text editing operations such as creating bulleted lists, manipulating tables (inserting/removing rows/columns), and inserting images. Chat messages are displayed on the right side of the screen. Both the document and past chat messages are restored from the corresponding databases in the case that the user refreshes the entire webpage. The system was tested with more than 80 users in the same collaborative session and all users remained synchronized.

Figure 3 shows the action of dragging and dropping an image containing a face into the editor. Once the image is added to the document, the user can move the mouse over the image to reveal an on/off button, which submits the image for processing to the Face Recognition Server deployed as a Docker-based microservice. Once analyzed, Figure 4 shows how the obtained landmark data is applied on top of the original image. Additional attributes such as gender and age are also listed on top of the image. Finally, any matching entities that were previously enrolled in the current database are inserted as a table after the image, along with the existing information about the match. Figure 5 shows how new faces can be enrolled into the database directly from the document. Information such as name, age, sex, ethnicity and emotion are stored along with the template (generated from the image) in the *Template Database* for future retrieval.

The current microservices-based implementation requires the web client to open multiple WebSocket connections to the server (one for the Collaboration Server and one for the Chat Server). While this approach was found to perform well on



Face Rec ON

gen	Male
age	26
BFace1	(106, 211)
BFace2	(311, 211)
BFace3	(311, 416)
BFace4	(106, 416)
REye	(256, 268)
LEye	(159, 274)
pose	(-1.28, 0.59, -3.62)
BFace conf	83
REye conf	97
LEye conf	98
CNoseTip conf	254
CMouth conf	254
Matches	0 (Enroll)

Name of individual: David Newton

Age: 38

Sex: M

Ethnicity: Caucasian

Emotion: Neutral

Fig. 5. Menu for enrolling new images to the database.

desktop PCs, having multiple active WebSocket connections is not ideal for mobile devices where battery life and limited processing power must be carefully considered. An architecture that is able to deliver data from multiple microservice through just one WebSocket connection would therefore be more appropriate for such scenarios.

There are additional downsides of microservices. For example, challenges with SQL JOINs across databases are still easier and faster for monoliths since they don't need to worry about distributed systems issues such as intermittent network connectivity. Microservices assume that the data is split into bounded containers, while in the monolith case, the cross-service JOIN operations can be performed more efficiently. Also, as was shown in Section IV, microservices imply significant operations overhead, duplication of development efforts, issues related to chaining interfaces, as well as increased asynchronicity complexities. Overall, however, the benefits of a microservice-based architecture were found to outweigh the downsides for the system described in this paper.

VI. Conclusion

A new collaborative web based rich text editor with integrated tools for face detection, recognition that provides automatically metadata annotation was described and implemented using microservices. The Docker containers allows for integrating software written in different languages, databases on a variety of operating systems by relinquishing functionality control to smaller cohesive teams of developers. The microservice containers can be easily deployed on available machines, evolved in a continuous integration manner and elastically scaled with the load.

Plans for future developments include the addition of new microservices to further enhance the editing experience, including a named entity recognition extension operating on data sources such as news-wire documents or tweets. In addition, methods of scaling automatically will be investigated.

References

[1] (2009) Google Docs. Google Inc. [Accessed: October 2015]. [Online]. Available: http://docs.google.com

[2] S. Hullavarad, R. OHare, and A. K. Roy, "Enterprise Content Management solutionsRoadmap strategy and implementation challenges," *International Journal of Information Management*, vol. 35, no. 2, pp. 260–265, 2015.

[3] A. H. Davis, C. Sun, and J. Lu, "Generalizing Operational Transformation to the Standard General Markup Language," in *Proceedings of the 2002 ACM conference on Computer supported cooperative work*. ACM, 2002, pp. 58–67.

[4] R. Dagher, C. Gadea, B. Ionescu, D. Ionescu, and R. Tropper, "A SIP Based P2P Architecture for Social Networking Multimedia," in *DS-RT 2008: 12th IEEE/ACM Int. Symp. on Distributed Simulation and Real-Time Applications*. IEEE Computer Society, October 2008, pp. 187–193.

[5] C. Gadea, B. Solomon, B. Ionescu, and D. Ionescu, "A Collaborative Cloud-Based Multimedia Sharing Platform for Social Networking Environments," in *ICCCN 2011: Proc. of 20th IEEE Int. Conf. on Computer Communication Networks*. IEEE Computer Society, August 2011, pp. 1–6.

[6] S. Newman, *Building Microservices*. O'Reilly Media, Inc., 2015.

[7] K. Matthias and S. P. Kane, *Docker: Up & Running*. O'Reilly Media, Inc., 2015.

[8] G. Toffetti, S. Brunner, M. Blöchlinger, F. Dudouet, and A. Edmonds, "An Architecture for Self-Managing Microservices," in *Proceedings of the 1st International Workshop on Automated Incident Management in Cloud*. ACM, 2015, pp. 19–24.

[9] J. Stubbs, W. Moreira, and R. Dooley, "Distributed Systems of Microservices Using Docker and Serfnode," in *Science Gateways (IWSG), 2015 7th International Workshop on*. IEEE, 2015, pp. 34–39.

[10] T. Mei, Y. Rui, S. Li, and Q. Tian, "Multimedia Search Reranking: A Literature Survey," *ACM Computing Surveys (CSUR)*, vol. 46, no. 3, p. 38, 2014.

[11] Apache Wave (Incubating). Apache Software Foundation. [Accessed: February 2016]. [Online]. Available: http://incubator.apache.org/wave/

[12] Etherpad. The Etherpad Foundation. [Accessed: February 2016]. [Online]. Available: http://etherpad.org/

[13] M. Heinrich, F. Lehmann, T. Springer, and M. Gaedke, "Exploiting Single-User Web Applications for Shared Editing: A Generic Transformation Approach," in *Proceedings of the 21st international conference on World Wide Web*. ACM, 2012, pp. 1057–1066.

[14] I. Koren, A. Guth, and R. Klamma, "Shared editing on the web: A classification of developer support libraries," in *Collaborative Computing: Networking, Applications and Worksharing (Collaboratecom), 2013 9th International Conference Conference on*. IEEE, 2013, pp. 468–477.

[15] NYTimes R&D Labs Editor. [Accessed: February 2016]. [Online]. Available: http://nytlabs.com/projects/editor.html

[16] AngularJS - Superheroic JavaScript MVW Framework. Google Inc. [Accessed: February 2016]. [Online]. Available: https://angularjs.org/

[17] Bootstrap The World's Most Popular Mobile-First and Responsive Front-End Framework. Twitter Inc. [Accessed: February 2016]. [Online]. Available: http://getbootstrap.com/

[18] Docker Hub. Docker Inc. [Accessed: February 2016]. [Online]. Available: https://hub.docker.com/

[19] Docker Compose. Docker Inc. [Accessed: February 2016]. [Online]. Available: https://docs.docker.com/compose/

[20] Docker Registry. Docker Inc. [Accessed: February 2016]. [Online]. Available: https://docs.docker.com/registry/

[21] D. Burns, *Selenium 2 Testing Tools: Beginner's Guide*. Packt Publishing Ltd, 2012.

[22] A. M. Berg, *Jenkins Continuous Integration Cookbook*. Packt Publishing Ltd, 2015.