Cloudchat: A Cloud-based Live Video Chat Service

George Lancaster (97288)
Dept. of Computer Science
University of Bristol
Bristol, United Kingdom
qv18258@bristol.ac.uk

***Abstract*—This paper introduces *Cloudchat*, a cloud-based live video chat service. *Cloudchat* is hosted by Amazon Web Services (AWS) and uses Web Real-time Communication (WebRTC) technology to live-stream video between users through a web interface. This document describes the application architecture, design rationale and the testing performed on *Cloudchat* as well as providing evidence of its robustness and scalability.**

***Cloudchat* can be found online at https://www.cloudchat.link.**

***Index Terms*—cloud, web-application, streaming, video chat.**

I. Introduction

A global increase in growth of consumer broadband means that more people than ever are using high-bandwidth internet applications [1] [2]. Multi-directional live video streaming is a high-bandwidth, CPU-intensive operation that is typically performed over a peer-to-peer network.

The motivation for developing *Cloudchat* was to demonstrate that the task of multi-directional video streaming can be sufficiently handled in a robust and secure fashion by cloud services.

A. WebRTC

WebRTC is a transport-agnostic open source project that provides web browsers and applications with the ability to stream data in real time. WebRTC is a similar technology to WebSockets in that they both maintain an open connection between nodes on a network. WebRTC is arguably better for streaming as it allows for transport using UDP. For a high-bandwidth operation, UDP is generally preferred over TCP [3]. WebRTC is traditionally performed in a peer-to-peer network of users, however this approach has two main drawbacks:

1) Peer-to-peer networks can be a security risk;
2) Group communication can be difficult to orchestrate with multiple peers.

Using a media server to process streams allows for the centralisation of control; all communication must be routed though a dedicated media server, which handles the sending and receipt of streams between users. This method is not only more secure, but it also allows for multi-directional video streaming.

Figures 1 and 2 illustrate the differences between WebRTC performed using a peer-to-peer network and a media server.

B. OpenVidu

The media server used by *Cloudchat* is OpenVidu. Openvidu is a wrapper for Kurento, a WebRTC media server that enables real-time group communication over the internet.

OpenVidu splits users into two roles: *subscribers* and *publishers*. Both roles have a uni-directional relationship with the media server. Data flows from a *publisher* to the server, and flows from the server to a *subscriber*. The OpenVidu server maintains a list of all *subscribers* to a *publishers* stream. To enable live video chat, all *publishers* are *subscribers*, and all *subscribers* are *publishers*.

OpenVidu was chosen to host the media server as it natively supports deployment on AWS. The OpenVidu team has supplied a Cloud Formation template to configure a basic media server on an Elastic Compute Cloud (EC2) instance [4]. Additionally, OpenVidu was chosen in place of Kurento as the Cloud Formation template made it easier to configure and deploy. This was important given the limited time to meet the project deadline.

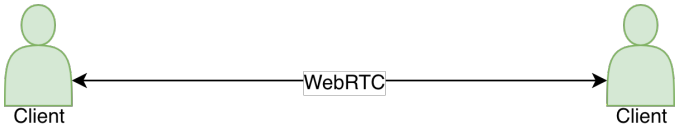

Fig. 1. Traditional peer-to-peer communication using WebRTC. Clients directly connect to one another.

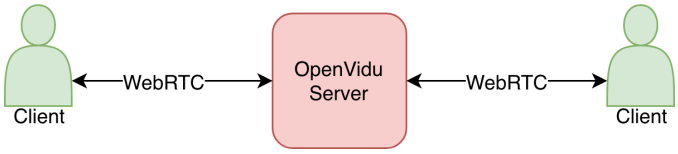

Fig. 2. WebRTC communication using an OpenVidu media server. All communication is routed through the media server.

C. Amazon Web Services

Cloudchat has been deployed on AWS. AWS was chosen as it is the oldest and most well-established cloud provider. AWS also provides services for sending emails (Amazon SES) and user authentication (Cognito), which is also an important part of this project.

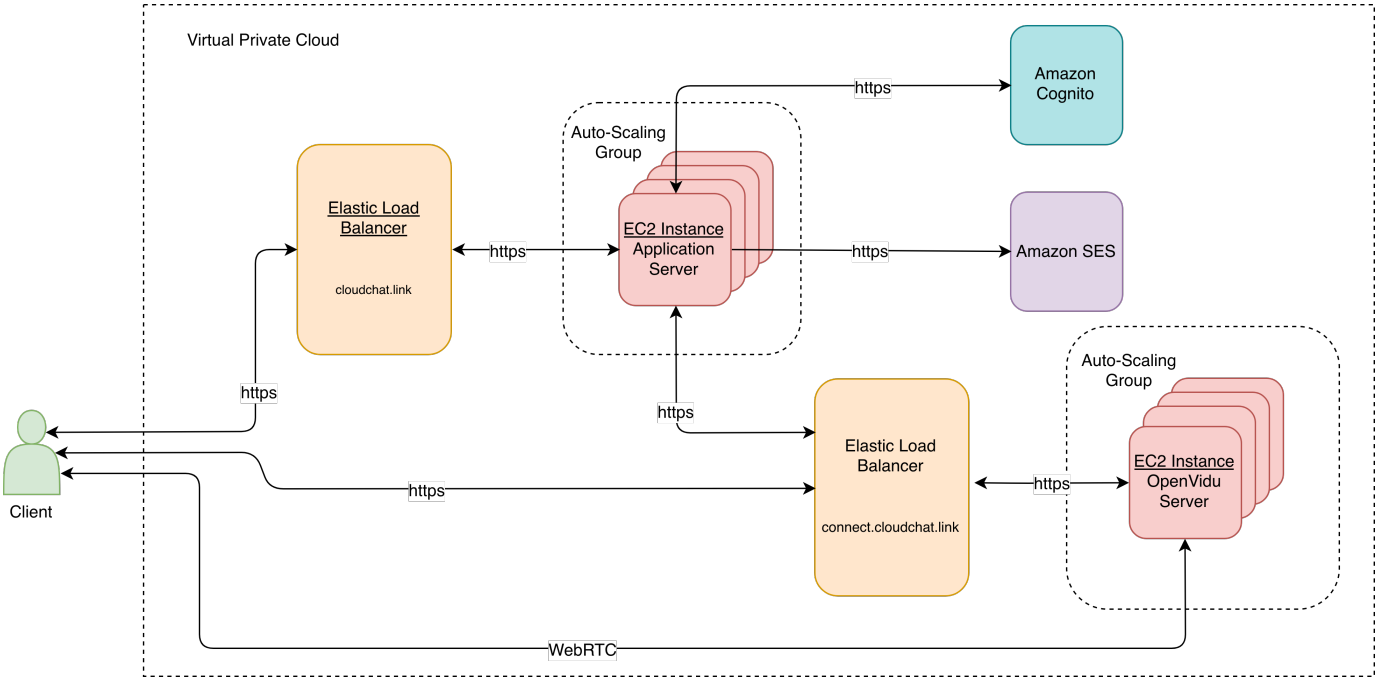

Fig. 3. The system architecture for Cloudchat.

D. Third Party Software and Code

Cloudchat builds upon an existing GitHub repository by the OpenVidu team [5]. The repository provides a simple Spring Boot web application server that connects clients to an OpenVidu Server.

This project has also used code from a GitHub repository that provides user authentication in a desktop application using Amazon Cognito [6]. Classes have been modified to enable user authentication in a web server environment.

II. Application Architecture

A. Application Server

The application server is running Java Spring Boot on an Nginx web server deployed using Elastic Beanstalk (EBS). EBS was used to create an auto-scaling group of t2.micro EC2 instances and an Elastic Load Balancer (ELB) to distribute requests to available instances. EBS was used due to its ease of implementation whilst allowing for full horizontal scalability of resources through server duplication. The application code was built and compiled using Maven. An alternative approach was made using Docker, however it was determined that there was no benefit to using it in this project.

All communication to the application server is routed through an ELB using https, allowing for a secure login and signup to *Cloudchat*. To enable https, a domain name needs to be registered and given an SSL certificate. Domain name registration was provided by Amazon Route 53, and the certificate was supplied and verified by Amazon Certificate Manager. Using Amazon for these services allows for greater integration into the AWS ecosystem.

B. OpenVidu & Kurento Media Server

OpenVidu acts as a wrapper for a Kurento Media Server. It has been deployed on a t2.large EC2 instance using a modified Cloud Formation template supplied by the OpenVidu team [4]. Internally, the server is comprised of a pre-configured Kurento Media Server linked to a OpenVidu server via a WebSocket.

Communication to the media server is routed through an ELB. To join a video chat session, clients must request a token from the media server through the application server. If the request is successful, the token can be used to connect directly to the media server from the JavaScript client using WebRTC. Sticky sessions have been enabled to ensure that users are consistently routed to the same media server in one session.

C. Application & Media Server Load Balancers

Two load balancers have been used to automatically distribute incoming application traffic across the multiple EC2 instances. The decision to separate the application and media server was a deliberate one, which allows for each auto-scaling group to scale independently of each other. This design decision allows for greater flexibility in the number of users on each part of the system. The application load balancer is publicly available at https://www.cloudchat.link and the media server load balancer is available at https://connect.cloudchat.link.

D. Amazon Cognito

Amazon Cognito is an auto-scaling service that controls pools of users for web and mobile applications. *Cloudchat* uses Cognito for user registration and authentication. To combat spam account creation, users must verify the email

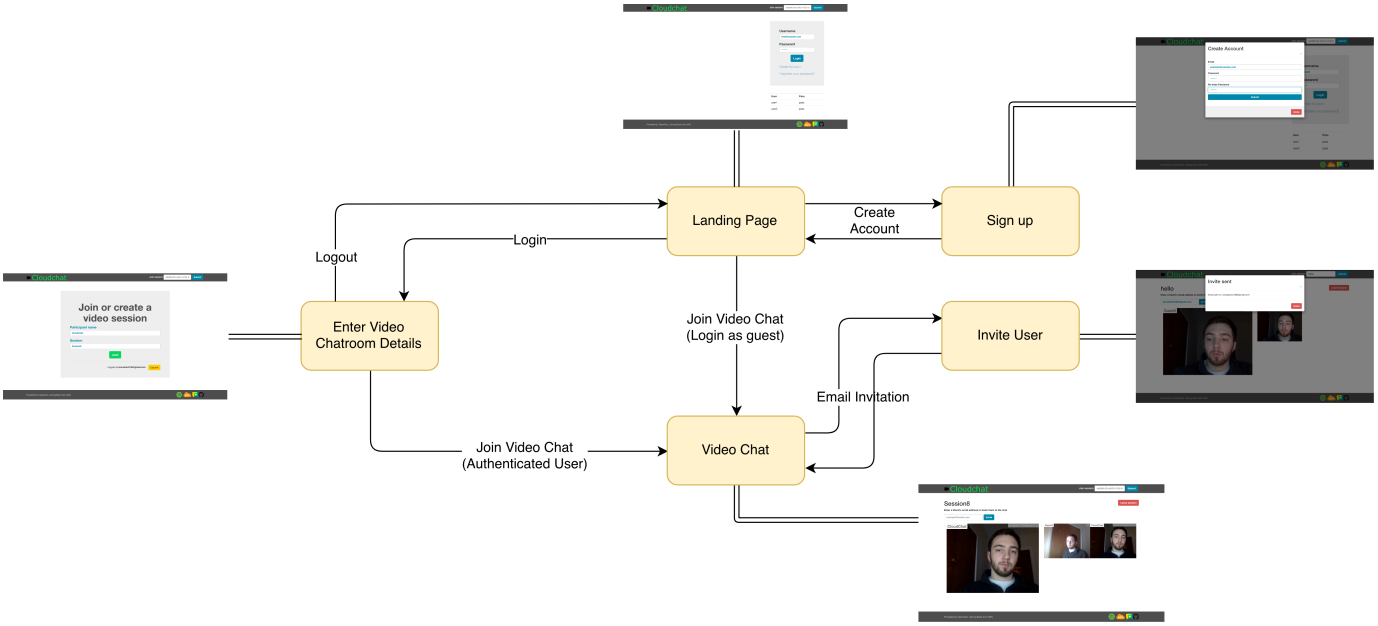

Fig. 4. The sitemap of *Cloudchat*.

address used to create their account. Cognito handles email authentication by automatically sending a verification email to all users upon registration. Clicking a link in the email confirms that the user owns the email account registered to their *Cloudchat* account.

Although users have the ability to sign into *Cloudchat*, it is not a requirement of joining or creating a video chat room.

E. Amazon Simple Email Service

Amazon Simple Email Service (SES) provides an API for sending emails to an application's users. *Cloudchat* uses SES to enable chat participants to invite users to their current chat session. The invitation email contains a link to the chat session the email was sent from, for example:

https://www.cloudchat.link/?data=chatsessionname.

When a user clicks the link, the client uses the data parameter in the url to redirect the user to the chat session. All email invitations are sent from *master.cloudchat@gmail.com*. Unfortunately, repeated invitations cause some email clients to label the emails as spam.

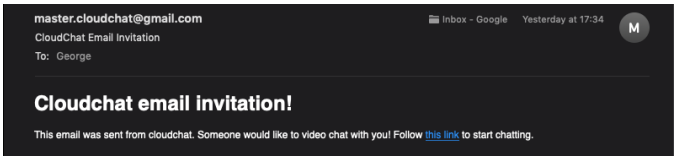

Fig. 5. Email invitation sent from a video chat session.

F. Client

The client is comprised of html, JavaScript and CSS. jQuery and Bootstrap have also been used. The client has been built upon an existing GitHub repository [5] and is contained within a single webpage. It provides navigation by showing and hiding html elements.

When logging in or signing up, the client communicates with the application server and provides feedback when there has been a problem. Examples of this include: trying to create an account that already exists, and trying to log in without email authentication.

III. Scaling & Availability

A. Application Sever

The application server is contained within an auto-scaling group, which maintains between 1 and 4 servers. The auto-scaling group has a threshold of 70% CPU utilisation before instantiating another EC2 instance. It is also based in a single region and split across three availability zones, meaning that if AWS has an outage in one zone, the application server should still be available in another zone.

The application server is running on a t2.micro instance, which allows for vertical scaling through CPU-bursting above 100% CPU utilisation. Because the application has been deployed using EBS, it would be trivial to re-deploy on a larger instance size and therefore raise the maximum CPU utilisation in a production environment.

Using a distributed file system is not an issue as Cognito handles user authentication.

B. Media Server

Although it is important for the application server to scale, the focus of this project was on providing a scalable media server. OpenVidu does not yet officially support automated scaling using AWS, therefore the media servers were wrapped in an auto-scaling group behind an application load balancer.

This maintains between 1 and 4 servers depending on CPU utilisation. The media servers run on t2.large EC2 instances.

A Medium post from the OpenVidu team claims that a c5.large instance can maintain 28 *publisher* streams and 168 *subscriber* streams, for a total of 196 streams per server [7]. Another study found the upper limit of concurrent streams on a c5.large instance to be 140 [8], although this study was conducted on an unsupported version of OpenVidu deployed using Docker. The auto-scaling group can therefore be expected to host 112 publisher streams and 672 subscriber streams, for a total of 784 concurrent streams. Although this number is relatively small, this solution scales horizontally and the number of concurrent streams can be increased by raising the limit on the maximum number of instances in the auto-scaling group.

The auto-scaling group is also split across three availability zones, giving it the same availability as the application server.

IV. Load Testing

A. Application Server

The application server has been tested using Locust, a free open source load testing framework for Python. The test opens 10 new connections to the index page per second up to a maximum number of 250 concurrent connections, and a peak request rate of 23.1 requests per second.

Running for one minute, a total of 1,531 requests were successfully made, with only 58 failures. The average response time was 63ms. Figures 6, 7 and 8 show the results from the Locust test.

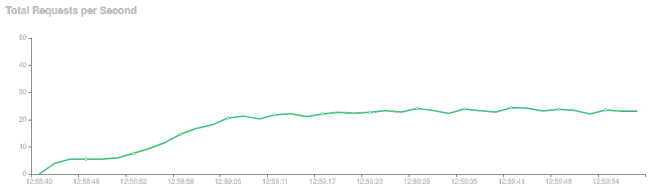

Fig. 6. Total requests per second over the test time.

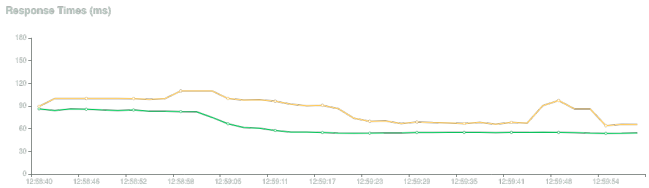

Fig. 7. Response times over the test time. The green line shows the median response time, and the yellow line shows the 95% percentile.

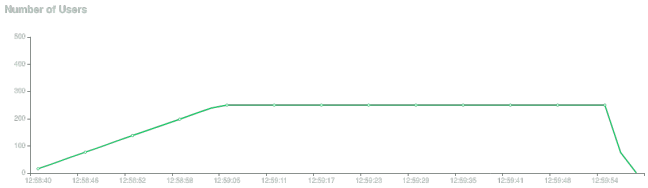

Fig. 8. The number of users in the test.

V. Discussion and Further Work

A. Development Costs

The total amount of money spent during development was $12. The majority of this cost came from registering the domain name with Route 53, which cost $10 + VAT. Based on usage during development, Amazon estimates a total running cost of $35.28 for January 2019.

B. Further Work

Cloudchat has generally been a success, meeting the requirements of providing a multi-directional video chat service using the cloud. Additional work could be done on testing the scalability of the media server. Currently, it can only theoretically handle more users than reported in [7] and [8]. Testing would be done in the same way as the aforementioned studies, a GitHub repository is available to test a single OpenVidu media server [9]. This could be adapted to test the load-balanced server.

Currently, logging into *Cloudchat* gives no additional benefits to the user. One extension could add a social aspect to the application, allowing users to maintain a list of friends.

Cloudchat does not support persistent sessions. If a user reloads the page whilst logged in, they will be redirected to the login page. Persistent sessions could be implemented by using the Amazon Cognito Identity JavaScript library, which can store Cognito cookies in the browser.

C. Additional Information

Cloudchat only functions when used on **two** devices as the JavaScript webcam API can only receive data from one source at a time.

References

[1] John B Horrigan. Broadband adoption and use in america. *Broadband National Government Plan*, 2010.

[2] Peter Adams, Mark Farrell, Barney Dalgarno, and Edward Oczkowski. Household adoption of technology: the case of high-speed broadband adoption in australia. *Technology in Society*, 49:37–47, 2017.

[3] Xiaotang Zhang and Henning G Schulzrinne. Voice over tcp and udp. *Columbia University Computer Science Technical Reports, CUCS-033-04*, 2004.

[4] OpenVidu. cloud formation template. https://s3-eu-west-1.amazonaws.com/aws.openvidu.io/CF-OpenVidu-latest.json, 2018.

[5] OpenVidu. openvidu-js-java. https://github.com/OpenVidu/openvidu-tutorials/tree/master/openvidu-js-java, 2018.

[6] Amazon. aws-cognito-java-desktop-app. https://github.com/aws-samples/aws-cognito-java-desktop-app, 2018.

[7] OpenVidu. Openvidu load testing: a systematic study of openvidu platform performance. https://medium.com/@openvidu/openvidu-load-testing-a-systematic-study-of-openvidu-platform-performance-b1aa3c475ba9, Dec 2018.
[8] Emmanuel André, Nicolas Le Breton, Augustin Lemesle, Ludovic Roux, and Alexandre Gouaillard. Comparative study of webrtc open source sfus for video conferencing. In *2018 Principles, Systems and Applications of IP Telecommunications (IPTComm)*, pages 1–8. IEEE, 2018.
[9] OpenVidu. openvidu-loadtest. https://github.com/OpenVidu/openvidu-loadtest, 2018.