Fachhochschule Nordwestschweiz
Hochschule für Technik

Jitsi Videoconferencing on AWS

PCLS Case Study HS23

Verfasser/in

Julie Engel, Luca Plozner

Ort, Datum

Windisch 5.1.2024

Inhaltsverzeichnis

1 Beschreibung des Use Cases

Dieses Projekt wurde während des Herbstsemester 2023 im Rahmen des Public Cloud Moduls durchgeführt. Ziel des Projektes ist es, die «Jitsi Meet» Videoconferencing Software auf AWS bereitzustellen.

Jitsi ist eine Sammlung von kostenlosen VoIP-, Videoconferencing- und Instant Messaging-Anwendungen für das Web, Windows, Linux, macOS, iOS und Android. Es zeichnet sich durch seine Flexibilität und Anpassungsfähigkeit aus, was es zu einer attraktiven Lösung für Video-Konferenzanforderungen macht. Des Weiteren ist Jitsi Open-Source unter der Apache 2.0 Lizenz.

Jitsi-Meet ist die Software für das Videoconferencing des Jitsi-Softwarestacks. Sie basiert auf WebRTC und wird über den XMPP-Server «Prosody» verwaltet. «Jitsi-Videobridge» ist die Serversoftware, welche die aktiven Videostreams an die Teilnehmer verteilt.

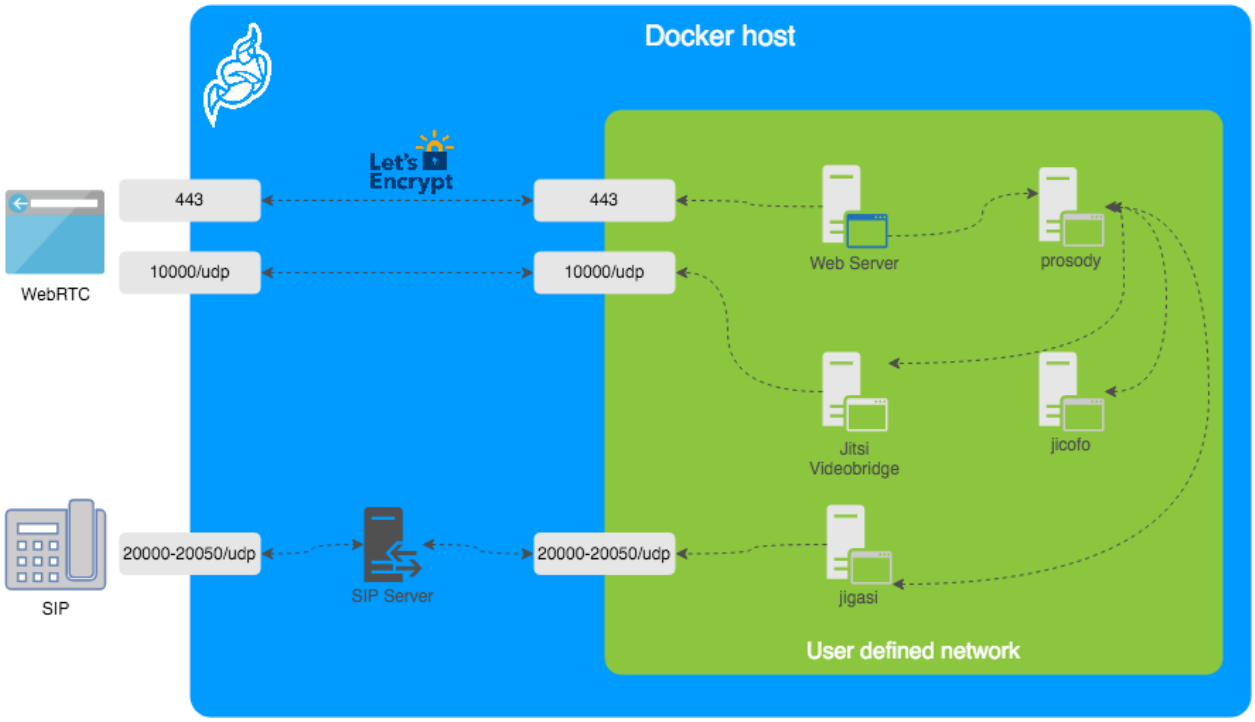

Abbildung 1: Jitsi Komponente

Jitsi besteht mehreren Komponenten. Für unser Use Case wurden die folgenden verwendet:

- **Nginx**: Webserver
- **Prosody**: XMPP Server für Signaling
- **Jitsi Videobridge (jvb)**: WebRTC kompatibler Server für Videostreaming
- **Jitsi Conference Focus (jcofo)**: Verwaltet Sessions

(Jitsi, 2024)

2 Vorgehen

Gemäss unseren Recherchen lässt sich jitsi auf einem Hyperscaler auf primär zwei Arten installieren:

1. Manuelle Installation auf eine provisionierte EC2-Instanz
2. Via Docker-Compose auf einem Cluster wie ECS

Da wir die erste Variante für nicht wirklich «cloud-native» hielten, probiert wir zuerst das Aufsetzen eines ECS-Clusters.

Dazu folgten wir einer Anleitung vom «AWS Open Source Blog» (Sueiras, 2020). Entsprechend der Anleitung wurde das ECS-Cluster, das SSL-Zertifikat, die Security-Groups und der Loadbalancer aufgesetzt.

Nachdem auf der Domäne der Loadbalancer per CNAME-Record eingestellt wurde, konnte man Jitsi über das Web erreichen. Sobald man aber einer Konferenz beitreten wollte, kam eine Fehlermeldung zu WebSockets.

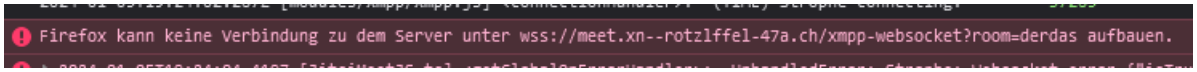

Abbildung 2 WebSocket Fehlermeldung

Nach einer Recherche im Internet fanden wir heraus, dass wir nicht die einzigen mit solchen Problemen sind:

- GitHub Issue 1: https://github.com/jitsi/docker-jitsi-meet/issues/1475
- GitHub Issue 2 https://github.com/jitsi/docker-jitsi-meet/issues/1531
- GitHub Issue 3: https://github.com/jitsi/docker-jitsi-meet/issues/1154
- StackOverflow Forum: https://stackoverflow.com/questions/70397053/jitsi-behind-alb-connection-to-websocket
- Serverfault Forum: https://serverfault.com/questions/1046931/jitsi-meet-in-docker-container-websocket-error

Wir konnte ausschliessen, dass das Problem am Loadbalancer lag, da es auch bei direktem Zugriff auf die EC2-Instanzen per IP auftrat. An den Security-Groups lag es auch nicht, da wir auch testweise alle TCP und UDP-Ports öffneten. Die verschiedenen Lösungsvorschläge der obigen Issues haben wir ebenfalls ohne Erfolg probiert.

Nach mehreren Stunden Troubleshooting entschieden wir uns dafür, die Installation manuell auf den EC2-Instanzen zu machen.

Dazu folgten wir auch zuerst der Anleitung des AWS-Blogs. Das Ganze haben wir dann erweitert mit Loadbalancer und Skripts, wie im Kapitel «Implementierung» erklärt wird.

3 Architektur

Folgende AWS-Services werden für die Architektur verwendet:

- **EC2-Instanzen**: Als Jitsi-Server
- **Auto-Scaling**: Erstellt eine neue EC2-Instanz auf anderer AZ, falls die Main-Instanz abstürzt
- **Load Balancer**: Leitet Traffic auf die entsprechende EC2-Instanz, nutzt SSL-Zertifikat von ACM
- **Certificate Manager**: Importiert und verwaltet das SSL-Zertfikat
- **Cloudwatch**: Monitoring der Instanzen
- **IAM**: Verwaltung der Benutzer und Berechtigungen

Die ganze Architektur befindet sich in einer VPC, wobei diese aus zwei Subnetzen in je einer AZ besteht, wie auf dem Architekturdiagramm zu sehen ist.

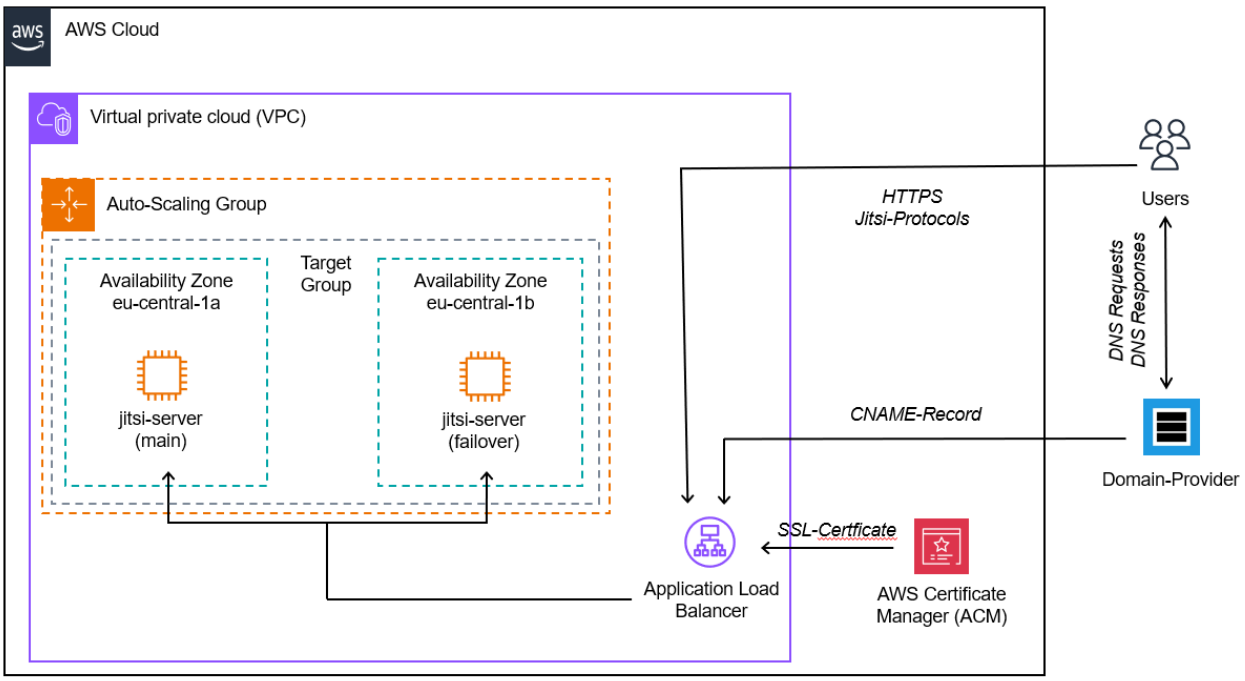

Abbildung 3 Architekturdiagramm

Der Jitsi-Service läuft auf einer EC2-Instanz, welche sich in einer Auto-Scaling-Gruppe befindet. Da Jitsi den State bei so einer Installation nicht teilen kann, betreibt das Auto-Scaling nur eine Instanz. Falls diese Instanz ausfällt, wird eine neue in der anderen AZ gestartet.

Die EC2-Instanzen werden vom Auto-Scaling automatisch in die Zielgruppe für den Application Load Balancer registriert. Dieser leitet dann den HTTPS-Traffic auf die entsprechende EC2-Instanz. Falls es dann ein Failover gibt, wird der Traffic auf die neue Instanz geleitet. Der Load Balancer fügt auch das SSL-Zertifikat vom ACM dem HTTPS-Traffic hinzu, so dass die Nutzer eine verschlüsselte und authentisierte Verbindung haben.

Im AWS Certificate Manger wird das SSL-Zertifikat für den Domainnamen («meet.domain.tld») importiert. Das Zertifikat wurde zuvor bereits mit «certbot» durch eine CA signiert.

Beim Domain-Provider (in diesem Fall Hostpoint) muss dann noch ein CNAME-Record erstellt werden, welcher auf den Application Load Balancer zeigt.

Per Cloudwatch-Monitoring wird bei einem Ausfall der EC2-Instanz per E-Mail alarmiert.

Das ganze Setup wird mit einem eigens kreierten User durchgeführt, der mit einem minimalen Privilegienset läuft, um grösstmögliche Isolation zu gewährleisten.

4 Implementierung

Der Code ist auf diesem GitHub-Repository abrufbar: https://github.com/ProgrammiererEFZ/pcls-jitsi

Das Aufsetzen und Abbauen der Umgebung wird mit Bash-Skripts gemacht, da wir darin am meisten Kenntnisse haben. Zudem sind die Skripte logisch aufgebaut und somit einfach zum Verstehen.

Das ursprüngliche ECS-Cluster haben wir mit zuerst mit Terraform und dann noch mit Cloudformation probiert.

4.1 00_create_iam.sh

Das Skript legt folgende Parameter fest:

- Name der Policy, definiert die notwendigen Berechtigungen
- Name des Users
- Name des EC2 Keypairs

Das Skript erstellt einen eigenen User, um die nachfolgenden Aktionen durchzuführen. Dafür verwendet er eine Policy, welche zuerst erstellt wird. Anschliessend wird ein Schlüsselpaar generiert, um sich mit der EC2 Instanz zu verbinden. Zuletzt werden Access und Secret Key für das AWS-Tool auf den neuen User gesetzt.

4.2 01_create_instances.sh

Das Skript legt folgende Parameter fest:

- Name der VPC
- Name der Security-Gruppe
- Name der Zielgruppe
- Name der Auto-Scaling Gruppe
- Name des Launch-Templates
- EC2 Setup-Skript
- EC2 Instanztyp
- EC2 AMI-ID (Betriebssystem)
- EC2 Keypair

Das Skript fügt folgende Schritte durch:

1. VPC erstellen.
2. Internet-Gateway erstellen.
3. Security Gruppe mit benötigten Ports (TCP: 22, 80, 443, 4443, UDP: 10000) erstellen.
4. Zwei Subnetze in verteilten AZs erstellen.
5. Route zwischen Subnetzen und Internet-Gateway erstellen.
6. Zielgruppe für EC2-Instanzen und Load Balancer erstellen und konfigurieren auf HTTPS.
7. Launch-Template für das Auto-Scaling erstellen und konfigurieren mit dem Setup-Skript.

8. Auto-Scaling-Gruppe erstellen, welche dann die EC2-Instanz hochfährt.

Auf den EC2-Instanzen wird Jitsi via dem mitgegebenen Setup-Skript («jitsi_setup.sh») installiert und konfiguriert. Der Domain Name muss direkt im Skript angepasst werden.

Für Testzwecke hat der Gratis-Instanz-Typ «t2.micro» ausgereicht. Für eine Grösse Anzahl Nutzer muss die Instanz entsprechend mehr Ressourcen haben.

Nachdem die EC2-Instanz hochgefahren ist, wird sie automatisch der Zielgruppe hinzugefügt. Das Hochfahren und die Installation von Jitsi dauern zwei bis drei Minuten. Danach kann das nächste Skript ausgeführt werden.

4.3 02_create_load_balancer.sh

Das Skript legt folgende Parameter fest:

- Name der VPC (identisch wie aus 4.2)
- Name der Security-Gruppe (identisch wie aus 4.2)
- Name der Zielgruppe (identisch wie aus 4.2)
- Name des Load Balancers
- Domain Name des SSL-Zertifikats

Nachdem das Skript die nötigen IDs für VPC, Subnets und der Security Group gefunden hat, erstellt es einen Application Load Balancer für HTTPS-Anfragen. Wir haben uns für einen Application Load Balancer entschieden, da dieser das SSL-Zertifikat einfach an den Nutzer weiterleiten kann. Zudem waren wir mit der Anwendung des Load Balancer bereits vertraut, da wir ihn beim Versuch mit dem ECS-Cluster brauchten. Eine alternative Lösung wäre mit Elastic IPs möglich, wie auch im AWS Blogpost erklärt.

Das Erstellen, Signieren und Importieren (in ACM) des SSL-Zertifikates wurde manuell ohne Skript gemacht, da es auch eine manuelle Bestätigung auf der Seite des Domänen-Providers braucht.

Am Schluss des Skriptes wird der DNS-Name des Load Balancers ausgegeben. Dieser muss dann per CNAME-Record bei Domänen-Provider referenziert werden. Der Load-Balancer braucht ebenfalls zwei bis drei Minuten, bis er funktioniert.

4.4 03_activate_monitoring.sh

Das Skript legt folgende Parameter fest:

- E-Mail-Adressen, an welche alarmiert werden soll
- Name des Monitoring Topics
- Name der Monitoring Rule

In diesem Skript werden die Regeln festgelegt und die Mails, an welche alarmiert werden soll. Für Testzwecke wird alarmiert, wenn eine EC2-Instanz auf «terminated» geht.

Um das Monitoring-Setup abzuschliessen, muss die Aktivierung via E-Mail bestätigt werden.

4.5 99_delete_everything.sh

Das Skript legt folgende Parameter fest:

- Name der VPC (identisch wie aus 4.2)
- Name der Security-Gruppe (identisch wie aus 4.2)
- Name der Zielgruppe (identisch wie aus 4.2)
- Name der Auto-Scaling Gruppe (identisch wie aus 4.2)
- Name des Launch-Templates (identisch wie aus 4.2)
- Name des Load Balancers (identisch wie aus 4.3)
- Name des Monitoring Topics (identisch wie aus 4.4)
- Name der Monitoring Rule (identisch wie aus 4.4)

Das Skript baut alle Ressourcen der vorherigen Skripte wieder ab. Um das VPC zu löschen, wird jeweils 30 Sekunden gewartet und dann geprüft, ob die EC2-Instanzen bereits vollständig runtergefahren wurden. Das VPC kann nur dann gelöscht werden. Das Skript kann nicht vom jitsi-user ausgeführt werden, da dieser Benutzer gelöscht wird. Wegen Abhängigkeiten muss das Skript eventuell ein zweites Mal durchgeführt werden.

4.6 Anwendung von Jitsi

Wenn der Load Balancer gestartet ist und der DNS-Record konfiguriert wurde, kann Jitsi verwendet werden. Dazu können die Nutzer im Webbrowser auf die entsprechende Adresse (https://meet.domain.tld). Dort kann deine eine Meeting-ID eingegeben und beigetreten werden.

4.7 Testen des HA/Autoscaling

Um HA und Auto-Scaling zu testen, wird die EC2-Instanz manuell gelöscht. Danach erstellt das Auto-Scaling automatisch eine neue Instanz. Sobald diese fertig mit der Installation ist, leitet der Load Balancer der Verkehr auf die neue Instanz. Das Auto-Scaling ist auf zwei AZs eingestellt.

Die neue Instanz wird teilweise auch auf der gleichen AZ wieder hochgefahren, da wir den Ausfall einer AZ nicht simulieren können. Falls jedoch eine AZ ausfällt, sollte die neue Instanz auf einer anderen AZ hochgefahren werden.

5 Erkenntnisse und Fazit

Am Anfang wollten wir Jitsi unbedingt über ECS auf dem Hyperscaler zum Laufen bringen. Wir dachten uns, das kann doch nicht so schwer sein, und das Docker Compose-File gab es ja auch schon.

Leider mussten wir aber irgendwann feststellen, dass diese Lösung überhaupt nicht stabil läuft und keineswegs «bullet-proof» zum deployen ist. Anfangs haben wir noch versucht, das Deployment mit kleinen Fixes zum Laufen zu bringen, aber spätestens als uns der Bug aus Kapitel 2 um die Ohren flog, mussten wir uns geschlagen geben.

Das Vorgehen von einer simplen Installation auf einer EC2-Instanz war dagegen unglaublich einfach und lief anschliessend auch sehr stabil. Einzig die Flexibilität und Skalierbarkeit war mit diesem Ansatz nicht mehr so fest dynamisch und erfordert jeweils eine Anpassung im Deployment-Skript. Eine vertikale Skalierung wäre möglich, indem man eine Spot-Instanz nehmen würde, oder automatisch den Instanz-Typ der Nachfrage anpasst. Eine horizontale Skalierung ist mit der aktuellen Installation nicht möglich, da Jitsi so nicht in der Lage ist den State zu teilen. Mit ECS wäre dies möglich gewesen.

Zuerst haben wir versucht, unsere Skripte in Terraform (und später in Cloudformation) zu schreiben, aber nach einigen Misserfolgen mit dem ECS-Deployment sind wir irgendwann dazu übergegangen, einfach awscli-Kommandos mittels Bash-Skripte auszuführen.

Abschliessend können wir sagen, dass wir froh sind, doch noch eine lauffähige Jitsi-Instanz hinbekommen zu haben. Als Verbesserung könnte man das nächste Mal die Bash-Skripte aber beispielsweise noch nach Terraform portieren, da dies vor allem beim Teardown des Systems viel zuverlässiger agiert.

6 Literaturverzeichnis

Jitsi. (6. Januar 2024). *Architecture*. Von Jitsi Meet Handbook: https://jitsi.github.io/handbook/docs/devops-guide/devops-guide-docker/ abgerufen

Sueiras, R. (7. März 2020). *AWS Open Source Blog*. Von Getting started with Jitsi, an open source web conferencing solution: https://aws.amazon.com/de/blogs/opensource/getting-started-with-jitsi-an-open-source-web-conferencing-solution/ abgerufen