

## **CS486 Final Capstone Report**

Revision 1.0

**April 15, 2025**

**StratoSplit**

**Client:**

General Dynamics Mission Systems

**Mentor:**

Brian Donnelly, Savannah Chappus

**Team Members:**

Sam Cain

Nolan Newman

Dallon Jarman

Elliot Hull

### **Overview:**

The purpose of this final report is to summarize this project into a single coherent narrative. The intent of this document is to provide someone an understanding of this project within a single reading, starting from motivations and vision, summarizing the development process, and describing the resulting product.

1. Introduction.....	3
2. Process Overview.....	4
3. Requirements.....	5
4. Architecture and Implementation.....	7
5. Testing.....	10
6. Project Timeline.....	12
7. Future Work.....	14
8. Conclusion.....	14
9. Glossary.....	16
10. Appendix A.....	16
11. Production cycle.....	27

## **1) Introduction**

Effective and reliable communication is a cornerstone of operations within the defense, public safety, and intelligence sectors. In high-stakes environments where timely decisions can mean the difference between mission success and failure, communication systems must be fast, secure, and easy to operate under pressure. However, the current infrastructure and tools are often outdated, hardware-intensive, and cumbersome. These limitations introduce inefficiencies in data transmission, audio clarity, and the handling of attachments or supplementary data.

To address this gap, StratoSplit is developing a web-based application design to simulate real-time audio generation and transmission over a secure dashboard. This project is sponsored by General Dynamics Mission Systems (GDMS), a renowned defense contractor known for delivering robust communication and surveillance technologies. GDMS has spearheaded the transformative program *Rescue 21*, the U.S. Coast Guard's advanced distress location and communication system. Our collaboration with GDMS is focused on designing a next-generation solution that not only reflects their standards of technical excellence but also pushes the boundaries of modern web-based communication interfaces.

StratoSplit is being built to support real-time audio streaming, simulate realistic communication environments, and provide an accessible yet powerful monitoring dashboard. One of the primary objectives is to replicate the complex communication dynamics between Coast Guard vessels and command stations, enabling simulations of field communications that mirror real-world operations. By integrating audio generation and transmission capabilities within a browser-accessible interface, StratoSplit removes the need for excessive hardware and complicated user interfaces.

The core vision behind StratoSplit is to revolutionize how secure audio data is generated, transmitted, and monitored in operational contexts. The system will allow operators and sailors to engage in simulated communications with high fidelity and real-time feedback. Furthermore, the dashboard will offer a centralized view of all transmissions, enabling mission commanders and operators to make informed decisions based on accurate and timely audio streams.

Ultimately, this project stands to significantly improve operational efficiency and communication reliability within organizations that depend on mission-critical interactions. With the support and direction of General Dynamics Mission Systems, our team is dedicated to building a high-quality, scalable solution that meets the rigorous

demands of defense and public safety communications. StratoSplit represents not just a technical achievement, but a step forward in supporting those who protect and serve.

## **2) Process Overview**

Our team followed an iterative development lifecycle, structured around multiple sprints to allow for continuous progress, regular feedback incorporation, and incremental delivery of features. This agile-inspired approach enabled us to adapt quickly to client and mentor feedback, refine our priorities, and maintain steady momentum throughout the development of this project.

To support and organize our development process, we utilized the following tools:

- **Git** for version control, enabling collaborative code development, branch management, and tracking of all code changes.
- **Excel** spreadsheets for task organization, sprint planning, and progress tracking. Tasks were broken down by week and assigned to individual team members.
- **Discord** for team communication, both for quick day-to-day coordination and for organizing asynchronous updates and file sharing.
- **Zoom** for all internal and external meetings.

Each team member adopted clearly defined roles based on individual strengths and areas of responsibility:

- **Sam** took on the role of **Team Lead** and **3D Audio Development**, overseeing project coordination while also handling the processing of audio input/output.
- **Dallon** focused on **Database Management, Authentication, and Release Management**, ensuring secure user access, data persistence, and proper handling of sensitive communication.
- **Elliot** was responsible for **Unit Testing** and **Meeting Recording**, writing automated tests to ensure functionality and quality while taking note of meeting details.
- **Nolan** served as the **System Architect** and **Deployment Lead**, managing AWS infrastructure, deployment, and the overall design of the system architecture.

Our development process was structured around recurring meetings and standardized team procedures:

- **Client Meetings** were held weekly on Tuesdays at 6:00 PM, where we presented our weekly progress, addressed feedback, and posed a curated list of questions at the end of each discussion to ensure alignment with the client's vision.
- **Mentor Meetings** occurred weekly on Fridays at 2:20 PM, providing technical and strategic guidance as well as feedback.

- **Internal Team Meetings** followed mentor meetings. These followed a structured agenda:
  - A minimum 2-minute report from each member to summarize personal progress.
  - A minimum 20-minute team discussion to review challenges and coordinate tasks.
  - A 15-minute planning session to discuss the upcoming week's goals and assignments.
  - A final summary overview to clarify the week's direction and distribute tasks.

Our decision making process was democratic. In cases of disagreement, a team vote was held. If a time occurred, the Team Lead (Sam) had final decision making authority.

To promote continuous improvement and team cohesion, we incorporated a peer and self-feedback component into our internal meetings:

- Each team member provided self-reflection on their work during their individual report.
- The group gave constructive feedback to one another during discussions
- Feedback was delivered verbally and in an open format to encourage clarity, transparency, and collaborative problem solving.

This structured yet flexible process enabled us to work efficiently as a team, ensure consistent progress, and stay aligned with the expectations of our client and mentor throughout the development lifecycle.

### **3) Requirements**

The requirements for the Separation of Cloud Generated Audio Streams project were gathered through a combination of weekly client meetings and collaborative internal brainstorming sessions. Each Tuesday, the team met with the client to discuss progress and clarify the evolving scope of the project. These meetings allowed for consistent feedback, scope adjustments, and alignment with stakeholder expectations.

Additionally, the team held structured brainstorming sessions using sticky notes to generate, categorize, and prioritize features based on client input and user needs. This dual approach ensured that both high-level system goals and specific technical requirements were well-understood and documented.

As a result of this acquisition process, the following function and non-functional requirements were defined:

**Functional:**

- Passwordless login for secure and convenient user authentication
- User login and session management
- Detection of user logout events
- Log user access incidents for auditing and security
- Delete user account functionality
- Reconnect without re-authentication to support seamless transitions
- Assign channels to users or teams
- Manage user accounts (create, edit, delete, assign teams)
- View available audio channels
- Mix audio streams in a 3D audio environment
- Mute/unmute individual or all channels
- Save, load, clear, and delete configurations
- Display alerts for active audio transmission
- Log system and transmission errors
- Perform unit testing to validate functionality and prevent regressions

**Non-functional:**

- Performance Benchmarks
  - HTTPS Connection  $\leq 1s$
  - Authentication Response  $\leq 500ms$
  - Concurrent user support  $\geq 10$
  - Concurrent audio streams  $\geq 10$
  - 500ms latency transmission
  - Audio processing latency  $\leq 200\ ms$
  - Mute/unmute response time  $\leq 100\ ms$
  - Volume adjustment latency  $\leq 100\ ms$
  - Database query time  $\leq 2\ s$
- Quality Attributes
  - Unit test coverage  $\geq 70\%$
  - User-friendly, responsive UI for ease of use in critical environments
  - Cross platform compatibility for accessibility
- Platform and Security Constraints
  - Must use U.S. Sourced Software
  - Implement a Zero Trust architecture model
  - Deployment must be based on AWS infrastructure
  - Utilize a NoSQL database for flexibility and scalability

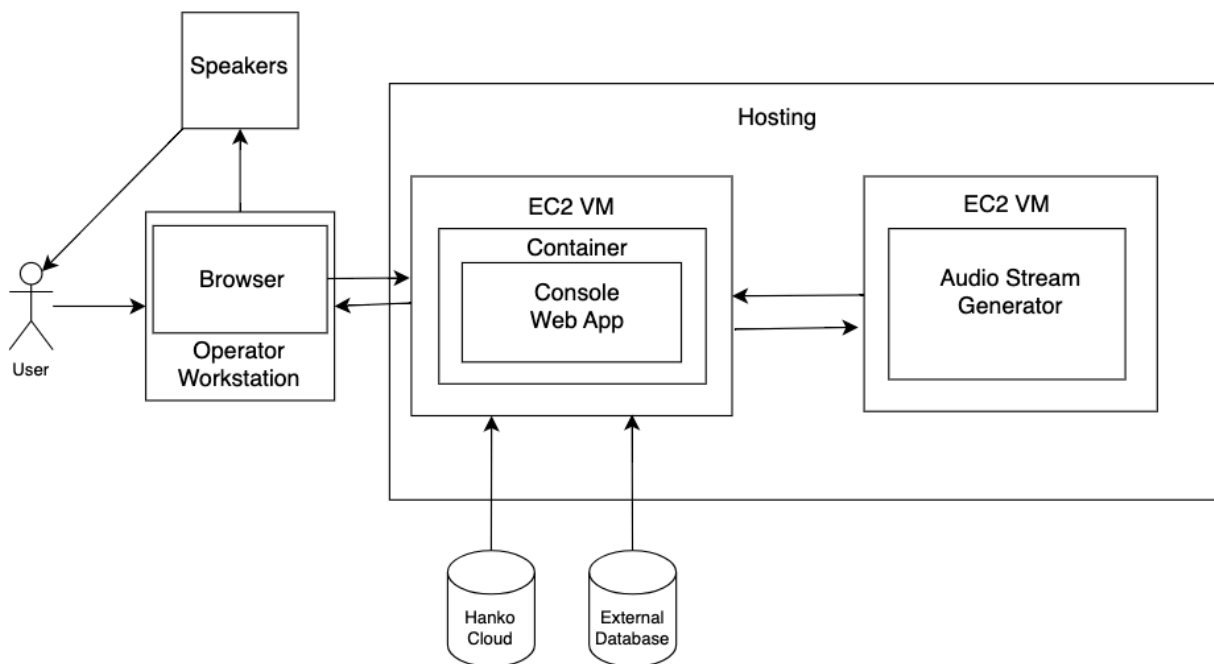
These requirements formed the foundation for design, implementation, and validation of the system. They guided architectural decisions, technology selection, and project

planning to ensure that the final product would meet both user expectations and the strict demands of mission-critical communications.

## **4) Architecture and Implementation**

### **4.1 System Overview**

StratoSplit is a web-based dashboard for simulating and managing spatial audio communications, designed primarily for Coast Guard operational scenarios. The system is cloud-hosted on AWS, comprising distinct modules for the web console, audio stream generation, and persistent storage. The software was built with flexibility, maintainability, and rapid user responsiveness in mind, following a microservices-inspired client-server model.



**Figure 1:** High-Level System Architecture

The system architecture consists of several key components, as shown in the above diagram:

- **Operator Workstation (Client):** Users interact with the system through a modern browser interface that handles authentication, stream controls, and audio playback. Output is routed to local speakers for real-time monitoring.
- **Console Web APP (Containerized EC2 VM):** Hosts the main Node.js/Express application, handling routing, API logic, user state, authentication enforcement, and database operations.

- **Audio Stream Generator (Separate EC2 VM):** A standalone Python service responsible for generating and broadcasting audio streams via RTP over multicast.
- **External MongoDB Database:** Stores persistent configuration data, user accounts, team data, and channel information.
- **Hanko Cloud:** Provides passwordless, token-based authentication. The console verifies these tokens before permitting access to secured features.

## **4.2 Component Responsibilities and Communication**

The system follows a modular service-oriented structure and utilizes standardized protocols for inter-component communication:

- The frontend (browser) sends commands to the backend (web app) using authenticated HTTPS requests.
- The backend processes these commands, interacts with MongoDB, or sends instructions to the audio stream generator to start or stop streams.
- Once activated, the generator broadcasts the audio via RTP multicast, and the frontend is configured to listen for this stream using browser-based playback tools.
- Session control and user identity are enforced via integration with Hanko Cloud's passwordless auth system.

This approach decouples user interaction, audio logic, and data management while maintaining a seamless, real-time user experience.

## **4.3 Typical Use Case: Starting a Stream**

1. The user logs into the dashboard using Hanko authentication
2. They select a configuration and activate an audio channel
3. An admin sends an API request to the web app to start streaming
4. The web app validates the request and contacts the Python-based audio generator
5. The generator parses the message and begins transmitting audio streams over RTP multicast
6. The browser listens to the multicast group and plays the stream through the user's speakers

This design minimized latency and scales efficiently, as one stream can serve many listeners without creating individual audio sessions.

## **4.4 Component Breakdown**



### **Console Web App**

- Tech Stack: Node.js, Express.js
- Functionality: Serves frontend files, manages user sessions, handles authenticated API routes, and connects to MongoDB
- Deployment: Runs on an AWS EC2 instance

### **Audio Stream Generator**

- Tech Stack: Python
- Functionality: Takes MP3 inputs and broadcasts them using RTP over multicast, supports multiple streams and durations, and includes loop safeguards
- Deployment: Runs on an AWS EC2 instance

### **MongoDB**

- Functionality: Persists user/team data, saved configurations, and team data.
- Integration: Accessed via Mongoose ODM within the web application

### **Hanko Cloud**

- Functionality: Manages passwordless authentication and issues tokens; tokens are verified server-side on each request
- Security: Central to implementing Zero Trust

## **4.5 Communication Protocols and Flow**

HTTPS (Browser -> Web App): Used for session validation.

RTP Multicast (Audio Generator <-> Web App): Efficient one-to-many audio streaming protocol; initiated by backend API call, received and passed to browser listener.

MongoDB Queries: Store and fetch configuration data, user data, and team data.

This layered and protocol-specific design ensures high responsiveness, clear separation of concerns, and secure operation under demanding conditions.

## **4.6 Architectural Influences**

StratoSplit's design reflects several architectural patterns:

- Client-Server: Traditional API-driven control from frontend to backend
- Service-Oriented: The audio generator functions independently and can be scaled or replaced without modifying the core application

- Event-Driven: Multicast transmission allows audio data to propagate passively to all listeners reducing network overhead.

## **4.7 As-Built vs. As-Planned**

Feature	Originally Planned	As Built	Reason for Change
Audio Transport	RTP Multicast	RTP Multicast	N/A
RTP Generation	External API / Tool	Python	More control
User Interface	Mobile optimized	Browser optimized	Time constraints
Operating System	Amazon Linux 2	Ubuntu	Kernel access

## **5) Testing**

A robust multi-layered testing strategy was central to the development of our project, ensuring the system not only performed as expected in isolated components but also behaved reliably in real-world use cases. Our testing process included **unit testing**, **integration testing**, and **usability testing**, each contributing to a broader goal of delivering a resilient, user-friendly and maintainable application.

### **5.1 Unit Testing**

Our unit testing efforts focused on validating core logic in both our JavaScript and Python codebases. We used **Jest** for JavaScript and **unittest** for Python to test individual functions such as authentication routes, audio processing utilities, configuration logic, and UI toggle behavior. These tests helped identify edge cases and ensured the code behaved correctly across a variety of inputs.

Notably, our Python module responsible for audio transmission was tested to ensure header formatting, payload integration, and proper handling of sequence boundaries. Meanwhile, Jest allowed us to test UI response logic by mocking virtual DOM elements and simulating button clicks or stream toggles.

We targeted a minimum of 70% test coverage, and our Jest reports provided detailed breakdowns of test quality across files. Through this foundation, we were able to detect regressions early and enforce consistent code behavior across updates.

### **5.2 Integration Testing**

Given the multi-stack nature of our system - spanning Node.js (Express.js), MongoDB, Hanko Authentication, and a Python-based audio stream generator - integration testing was crucial. These tests focused on validating the interactions between components, particularly around user login flows, protected routes, audio stream triggering, and configuration persistence.

For example, tests ensured that Hanko tokens were correctly verified by middleware and that only authenticated users could access protected APIs. We also validated that audio stream requests were passed from the Node.js backend to the Python audio service and returned valid stream endpoints. MongoDB operations (CRUD for users, teams, configurations, and logs) were tested for error handling and concurrency.

### **5.3 Usability and End-User Testing**

Our usability testing was designed to evaluate how real users interact with the system under simulated operational conditions. These sessions included structured evaluations with our client, mentor, and internal team meetings, using guided tasks such as starting/stopping streams, saving configurations, and adjusting channel audio settings.

We observed first-time use behavior, collected feedback via think-aloud techniques, and ran post-session interviews to measure clarity, efficiency, and responsiveness. These tests confirmed that the UI was generally intuitive, but they also exposed smaller usability issues such as misaligned icons, dropdown state confusion, and unclear stream toggling behavior - many of which were corrected in response to feedback.

These tests also helped uncover one critical issue within our mentor meetings: when testing streams of various audio lengths, we found that playing multiple audio files of various lengths resulted in unexpected termination of the transmission when the shortest of the files ended. This occurred because the audio files were not configured to loop if they were different lengths. This bug was discovered live during a demo, underscoring the importance of user testing. We resolved this by enforcing a loop flag on the generator and updating the handler to manage multiple stream resets more gracefully.

### **5.4 Reflections and Final Results**

Across all three testing types, we followed a layered approach: unit tests to validate correctness, integration tests to verify system-level reliability, and usability tests to ensure real-world readiness. This process uncovered functional bugs, integration gaps, and interface inconsistencies - most of which were addressed during spring cycles.

By the time of final testing, we had achieved:

- Consistent performance under concurrent audio streams
- Authenticated session control across Hanko and Express
- Streamlined UI behavior based on user feedback
- Stable API and database performance, including error-handling under load
- Audio stream bugs resolved, including looping and concurrency edge cases

This multifaceted testing effort gave us high confidence that our software is ready for deployment in a mission-critical setting. More importantly, it laid the groundwork for future scalability and maintainability by embedding testing into our development culture.

## 6) Project Timeline

The development of our project followed a structured timeline with overlapping work streams aligned to the project’s modular architecture. The team adopted a short-spring agile model, allowing for flexibility and continuous integration as individual components progressed. The project formally began in September 2024 and concluded development in April 2025, with final documentation and testing wrapping up in early May 2025.

Name	Start Date	End date	Days	September	October	November	December	January	February	March	April	May
Documentation	9/2/2024	5/9/2025	180									
Audio Generator	10/14/2024	12/31/2024	57									
Database	11/8/2024	1/30/2025	60									
User Interface	11/8/2024	3/31/2025	102									
Spatial Audio	11/8/2024	2/27/2025	80									
Testing	12/2/2024	4/29/2025	107									
Zero Trust	12/16/2024	4/16/2025	88									

**Figure 2:** Gantt Chart of Project Timeline

September to October 2024 consisted of documentation and planning. The team began by outlining requirements, user stories, and architectural plans. Early documentation included the project proposal, initial system architecture, and team standards. The documentation stream continued in parallel with technical development to ensure consistent alignment between planning and implementation.

Starting in October and ending in December 2024 we made our Audio Generator. Early development efforts focused on creating a simulated audio environment using MP3 file input and audio packet generation logic. This component, led by Nolan, formed the core

engine of the system and was essential for validating audio transmission throughout the backend.

Beginning in November 2024 and ending in January 2025 we built our database and integrated it with the backend. Dallon led the integration of MongoDB with Express.js, handling user/team management, audio configuration storage, and logging. This phase included schema design and connection testing and laid the foundation for interaction with Hanko authentication.

From November 2024 until March 2025 we worked on user interface development. UI implementation began once core backend endpoints were functional. This work, primarily handled by Dallon and Nolan, included dynamic stream controls, mute/unmute logic, and a responsive layout. Feedback from initial demos helped guide design iterations.

Our last key feature, spatial audio mixing began development in November 2024 and ended in February 2025. We developed our mixing engine in tandem with UI features. It allowed users to simulate directional audio environments and adjust stream volume and panning in real time. This feature was critical in order to reduce the hardware overhead associated with single audio channels per speaker. This development was led by Sam.

Testing efforts ramped up in December 2024 and are ongoing through April 2025. These tests included unit testing, integration testing, and usability testing. Elliot was the lead for unit testing. Multiple bugs were discovered through mentor demo sessions and these were crucial for integration. Testing continued in parallel with feature integration to validate system-wide reliability.

Lastly, starting in December 2024 and ending in April 2025, Zero Trust policies were progressively integrated into the system architecture, ensuring all protected routes and configuration required authenticated access. This included Hanko middleware integration, session validation, and API hardening led by Dallon. Nolan managed the AWS deployment in line with secure-by-default principles.

As of April 2025, all functional components have been implemented, and the system has passed key acceptance tests. Final documentation and usability feedback are being consolidated into the final client deliverables. The codebase is stable with verified performance benchmarks. By following this timeline, the team ensured parallel development with coordinated milestones, enabling steady progress and continuous improvement throughout the project lifecycle.

## **7) Future Work**

While our application achieves its core objectives of simulating audio generation and providing a real-time monitoring dashboard, several valuable extensions have been identified that would further enhance the system's usability, realism, and deployment potential. These ideas emerged during development, client discussions, and internal reflections and are worth considering for future iterations of the project.

While the current application is accessible on a mobile device, it is optimized for desktop and browser-based use. However, in real-world scenarios such as Coast Guard field operations or rapid deployment environments, personnel often rely on mobile devices for communication and monitoring. Future development should focus on implementing mobile-responsive design and potentially creating a dedicated mobile app for iOS and Android. This would allow field users to monitor streams, mute/unmute channels, and receive alerts without relying on laptops, drastically improving accessibility and flexibility in high mobility use cases.

The current system simulates audio generation and stream monitoring in one direction - from simulated field units to command. To more accurately mirror operational environments, two way audio communication should be supported. Implementing full-duplex communication would allow operators not only to monitor but to respond in real-time, facilitating realistic command-and-response workflows. This would require stream multiplexing, more advanced audio routing logic, and potentially a push to talk interface.

Future work could also include the development of a reporting module that logs key metrics such as connection times, alert frequency, stream activity levels, and mute/unmute behavior. Providing these metrics in an exportable format would be beneficial for post-mission analysis, training review, or audit compliance.

These future work items represent meaningful opportunities to evolve this project into a full-featured operational tool. While they extend beyond the initial project scope, they align closely with the real-world needs of defense, public safety, and intelligence operations and reflect the kinds of enhancements that could transform this project from a prototype into a deployable system.

## **8) Conclusion**

StratoSplit was conceived from the need for a modern, secure, and flexible communication system for mission-critical environments such as search and rescue,

military, and public safety operations. Traditional communication solutions are hardware-intensive, antiquated, and unscalable. Our solution filled these gaps by developing a web-based console that simulates real-time audio transmission with 3D audio mixing, centralized monitoring, and secure access.

Key features include:

- Secure, passwordless login via Hanko
- Real-time audio streaming over multicast
- A responsive, user-friendly web console
- audio mixing for spatial communication simulation
- AWS-based containerized deployment
- MongoDB backend
- Unit test coverage exceeding 70%
- Zero Trust security implementation

System performance benchmarks include sub-500ms authentication response and successful concurrent support of many audio streams and users. This enables seamless functionality under realistic field conditions, minimizes downtime, and simplifies updates via centralized architecture and microservices.

The system has extensive use in Coast Guard operations, disaster coordination, or communications deployment. Scalability, redundancy, and ease of use make it a strong response solution, particularly in scenarios that call for little hardware but maximum operational integrity.

Reflecting on the course, this capstone demanded technical depth, collaborative agility, and iterative design in the face of client feedback. We gained experience in deployment, secure authentication, systems integration, and adaptive problem-solving. The team operated with high cohesion, dividing tasks while maintaining collective ownership of the outcome. Our partnership with General Dynamics held us to high standards and maintained our work on practical, meaningful goals.

# Glossary

## Acronyms

AWS - Amazon Web Services  
EC2 - Elastic Compute Cloud  
GDMS - General Dynamics Mission Systems  
IGW - Internet gateway  
SAR - Search and Rescue  
USCG - United States Coast Guard  
VPC - Virtual Private Cloud

## Appendix A: Development Environment and Toolchain

Hardware: Two t2 micro EC2 Ubuntu 24.0.4 instances

Toolchain: EC2, Transit Gateway, VPC, Cloudflare, Hanko, MongoDB, Node JS, Python

### AWS Setup:

#### 1. Create VPC:

Navigate to 'VPC > Your VPCs > Create VPC' and create a VPC providing a name and IPv4 CIDR '10.99.0.0/16'. Leave everything else default.

<input checked="" type="checkbox"/>	nau-multicast	<a href="#">vpc-0aed22b7391ace15f</a>	Available	Off	10.99.0.0/16
-------------------------------------	---------------	---------------------------------------	-----------	-----	--------------

#### 2. Create Subnets:

Navigate to 'VPC > Subnets > Create Subnets' and create subnets as follows. Select newly created VPC and provide names, subnet zones, and CIDRs as follows:

Public 1: east 1a - 10.99.0.0/18

Public 2: east 1b - 10.99.64.0/18

Private 1: east 1a - 10.99.128.0/18

Private 2: east 1b - 10.99.192.0/18

<input type="checkbox"/>	nau-public1	<a href="#">subnet-...</a>	Available	<a href="#">vpc-0ae...</a>	Off	10.99.0.0/18
<input type="checkbox"/>	nau-private2	<a href="#">subnet-...</a>	Available	<a href="#">vpc-0ae...</a>	Off	10.99.192.0/18
<input type="checkbox"/>	nau-public2	<a href="#">subnet-...</a>	Available	<a href="#">vpc-0ae...</a>	Off	10.99.64.0/18
<input type="checkbox"/>	nau-private1	<a href="#">subnet-...</a>	Available	<a href="#">vpc-0ae...</a>	Off	10.99.128.0/18

#### 3. Create Internet Gateway:

Navigate to 'VPC > Internet Gateways > Create Internet Gateway' within AWS and create a new internet gateway.



### Create internet gateway [Info](#)

An internet gateway is a virtual router that connects a VPC to the internet. To create a new internet gateway specify the name for the gateway below.

#### Internet gateway settings

##### Name tag

Creates a tag with a key of 'Name' and a value that you specify.

#### Tags - optional

A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources or track your AWS costs.

##### Key

##### Value - optional

[Remove](#)[Add new tag](#)

You can add 49 more tags.

[Cancel](#)[Create internet gateway](#)

Add new IGW to 'Public 1' routing table and 'Public 2' routing table.

<input type="checkbox"/>	nau-igw	<a href="#">igw-0c6d6330b3d80b0cc</a>	<span>✔ Attached</span>	<a href="#">vpc-0aed22b7391ace15f   nau-multicast</a>
<b>Routes (2)</b>				
<input type="text" value="Filter routes"/>				
<div>Both <a href="#">Edit routes</a></div>				
<div>&lt; 1 &gt; <a href="#">Settings</a></div>				
Destination	Target	Status	Propagated	
0.0.0.0/0	<a href="#">igw-0c6d6330b3d80b0cc</a>	<span>✔ Active</span>	No	
10.99.0.0/16	local	<span>✔ Active</span>	No	

#### 4. Create Transit Gateway:

Navigate to 'VPC > Transit Gateways > Create Transit Gateway' within AWS. Check enable multicast support, leave all else default, and confirm.

☒ **Multicast support** [Info](#)

<input type="checkbox"/>	nau-transit-gateway	<a href="#">tgw-061064e44de57c203</a>	<span>✔ Available</span>
--------------------------	---------------------	---------------------------------------	--------------------------

#### 5. Create Transit Gateway Attachment:

Navigate to 'VPC > Transit gateway attachments > Create transit gateway attachment'. Select the new Transit Gateway and the VPC. The subnets should auto populate with the public subnets. Create the transit gateway attachment.

#### VPC ID

Select the VPC to attach to the transit gateway.

vpc-0aed22b7391ace15f

#### Subnet IDs [Info](#)

Select the subnets in which to create the transit gateway VPC attachment.

☒ us-east-1a

subnet-036ae667916ae75c1

☒ us-east-1b

subnet-0d9ec21a1b90972c3

## 6. Create Transit Gateway Multicast Domain:

Navigate to 'VPC > Transit gateway multicast domains > Create transit gateway multicast domain' check enable IGMP2 support and attach transit gateway. Create the multicast domain.

#### Create transit gateway multicast domain [Info](#)

A multicast domain controls how traffic flows for all associated subnets. A multicast domain can only be created in transit gateway with the multicast support option enabled.

##### Details

###### Name tag - *optional*

Creates a tag with the key set to Name and the value set to the specified string.

transit-gateway-multicast-domain-01

###### Transit gateway ID [Info](#)

tgw-061064e44de57c203

#### Configure the transit gateway multicast domain

☒ IGMPv2 support [Info](#)

☐ Static sources support [Info](#)

☐ Auto accept shared associations [Info](#)

## 7. Set up EC2 Host:

Navigate to 'EC2 > Instances > Launch an instance' and select your VPC.

#### ▼ Network settings [Info](#)

##### VPC - *required* [Info](#)

vpc-0aed22b7391ace15f (nau-multicast)  
10.99.0.0/16

##### Subnet [Info](#)

subnet-036ae667916ae75c1 nau-public1  
VPC: vpc-0aed22b7391ace15f Owner: 814304444943 Availability Zone: us-east-1a  
Zone type: Availability Zone IP addresses available: 16376 CIDR: 10.99.0.0/18

##### Auto-assign public IP [Info](#)

Disable

Make sure inbound SSH and HTTPS traffic is enabled.

### Inbound Security Group Rules

▼ Security group rule 1 (TCP, 22, 0.0.0.0/0)

Remove

Type | Info

ssh

Protocol | Info

TCP

Port range | Info

22

Source type | Info

Anywhere

Source | Info

Q Add CIDR, prefix list or security group

0.0.0.0/0 X

Description - optional | Info

e.g. SSH for admin desktop

▼ Security group rule 2 (TCP, 443, 0.0.0.0/0)

Remove

Type | Info

HTTPS

Protocol | Info

TCP

Port range | Info

443

Source type | Info

Anywhere

Source | Info

Q Add CIDR, prefix list or security group

0.0.0.0/0 X

Description - optional | Info

e.g. SSH for admin desktop

Add an additional inbound rule for 'Custom UDP' over port range '5001 - 5019' from the source '10.99.0.0/16'

sgr-058a0464e12fb06f3

Custom UDP

UDP

5001 - 5019

Cust...

Q

10.99.0.0/16 X

Delete

Navigate to 'VPC > Elastic IP Addresses > Allocate Elastic IP Address' and allocate and associate Elastic IP to the new EC2 Instance.

Configure DNS using the provider of choice with this new Elastic IP Address.

### 8. Set up EC2 Audio Generator:

Navigate to 'EC2 > Instances > Launch an instance' and select your VPC.

▼ Network settings Info

VPC - required Info

vpc-0aed22b7391ace15f (nau-multicast)  
10.99.0.0/16

↻

Subnet Info

subnet-036ae667916ae75c1 nau-public1  
VPC: vpc-0aed22b7391ace15f Owner: 814304444943 Availability Zone: us-east-1a  
Zone type: Availability Zone IP addresses available: 16376 CIDR: 10.99.0.0/18

↻ Create new subnet

Auto-assign public IP Info

Disable

Configure inbound security as per the following rules:

Inbound rules (2)							
<input type="text" value="Search"/> <span>Manage tags</span> <span>Edit inbound rules</span>							
<div> <div>&lt; 1 &gt;</div> <div>⚙</div> </div>							
Name	Security group r...	IP version	Type	Protocol	Port range	Source	
-	sgr-08006af56670e...	IPv4	SSH	TCP	22	0.0.0.0/0	
-	sgr-078553faf5e7d3...	IPv4	Custom UDP	UDP	5000	10.99.0.0/16	

Navigate to ‘VPC > Elastic IP Addresses > Allocate Elastic IP Address’ and allocate and associate Elastic IP to the new EC2 Instance.

9. Create Transit Gateway Multicast Domain Associations:

Navigate to your Transit Gateway Multicast Domain and create 1 association per public subnet within your VPC.

Associations (2) <span>info</span>							
<input type="text" value="Find association by attribute or tag"/> <span>Actions</span> <span>Create association</span>							
<div> <div>&lt; 1 &gt;</div> <div>⚙</div> </div>							
<input type="checkbox"/>	Subnet ID	Attachment ID	Resource type	Resource ID	Resource owner...	State	
<input type="checkbox"/>	<a href="#">subnet-00dd9201208d940...</a>	<a href="#">tgw-attach-0376c9c81953...</a>	VPC	<a href="#">vpc-0aed22b7391ace15f</a>	814304444943	<span>✓</span> <span>A</span>	
<input type="checkbox"/>	<a href="#">subnet-036ae667916ae75c1</a>	<a href="#">tgw-attach-0376c9c81953...</a>	VPC	<a href="#">vpc-0aed22b7391ace15f</a>	814304444943	<span>✓</span> <span>A</span>	

10. Create Multicast Groups:

Navigate to your Transit Gateway Multicast Domain. Select groups and add members based on your specific needs. Select both network interfaces and provide a multicast IP address to enable.

Add group members
info

Adding a member to a multicast group enables the network interface to receive multicast traffic sent by the sources of the multicast group.

Details

Transit gateway ID

tgw-061064e44de57c203

Group IP address

Requires a valid IPv4 or IPv6 IP Address in the 224.0.0.0/4 or ff00::/8 CIDR range.

Available network interfaces (2) info

Create network interface

< 1 >

⚙

<input type="checkbox"/>	Name	Network interface ID	Subnet ID	Availability Zone	Status	Instance ID	VPC ID
<input type="checkbox"/>	-	<a href="#">eni-0b71b9ca4f9d1cf96</a>	<a href="#">subnet-036ae667916ae75c1</a>	us-east-1a	in-use	i-06f5286c4f4fc5ed9	<a href="#">vpc-0aed22b7391ace15f</a>
<input type="checkbox"/>	-	<a href="#">eni-07c7874471e1d4c03</a>	<a href="#">subnet-036ae667916ae75c1</a>	us-east-1a	in-use	i-0835fc3c16abbffd3	<a href="#">vpc-0aed22b7391ace15f</a>

Cancel
Add group members

## 11. Disable Source Destination Check:

Navigate to 'EC2 > Instances' select the 'Actions' dropdown. In the networking tab click 'Change source/destination check', check 'Stop', and save the changes.

The screenshot shows the AWS Management Console 'Instances' page. The instance 'nau-generator' (ID: i-0835fc3c16abbffd3) is selected. The 'Actions' dropdown menu is open, and the 'Networking' tab is selected, showing the 'Change source/destination check' option. Below the console, a modal dialog titled 'Change Source / destination check' is displayed. The dialog explains the purpose of the check and shows the current configuration for the selected instance and network interface. The 'Source / destination checking' section has the 'Stop' option selected with a checked checkbox. At the bottom right of the dialog are 'Cancel' and 'Save' buttons.

**Change Source / destination check**

The source / destination check ensures that the instance is the source or destination of all the traffic it sends and receives. Each EC2 instance performs source and destination checks by default. [Learn more](#)

**Instance ID**  
i-0835fc3c16abbffd3 (nau-generator)

**Network interface**  
eni-07c7874471e1d4c03

**Source / destination checking**  
Stop to allow your instance to send and receive traffic when the source or destination is not itself.

☒ Stop

Cancel Save

Multicast Traffic should now be successfully enabled across your EC2 instances. Note traffic will only come through on ports with inbound traffic enabled via the security wizard. The ports and IPs used in this guide are relevant to the application we created.

## Hanko Cloud:

Navigate to [cloud.hanko.io](https://cloud.hanko.io) and create a new Hanko project.

### Select project type

Select the project type based on your requirements. This cannot be changed later, but you can always create another project.

**Hanko**☒

**Passkey API**☐

#### Authentication and user management

- Onboard and authenticate users with a range of login options, including passkeys, social logins, and enterprise SSO
- Customizable UI components for registration, login and the user profile
- Admin dashboard

#### Passkey infrastructure

- Integrate passkeys into any app
- Supports passkey creation, authentication, and transaction use cases
- A managed FIDO2-certified WebAuthn server API

Create project


Set the app URL to the domain name or 'localhost:443'.

**Hanko** Caelum

### App URL

The protocol and hostname of the location where you want to integrate Hanko Elements (i.e., your app's login page). The domain is also used for passkey binding (WebAuthn rpID).


[What are valid app URLs?](#)

 If you want to change the environment (e.g. from development to production), we strongly recommend [creating a new project](#) instead of changing the app URL of this project.

App URL \*

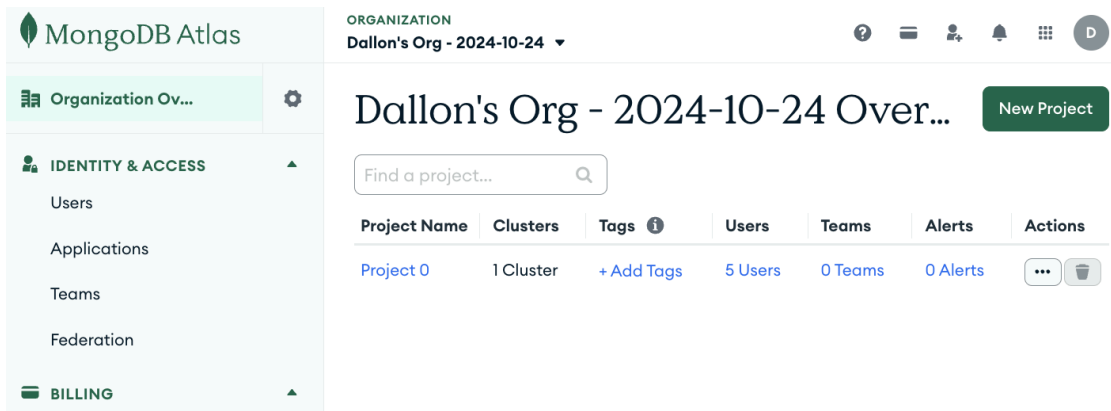
Cancel Save

Store the API url in the Dashboard page for the '.env' file on the host machine.

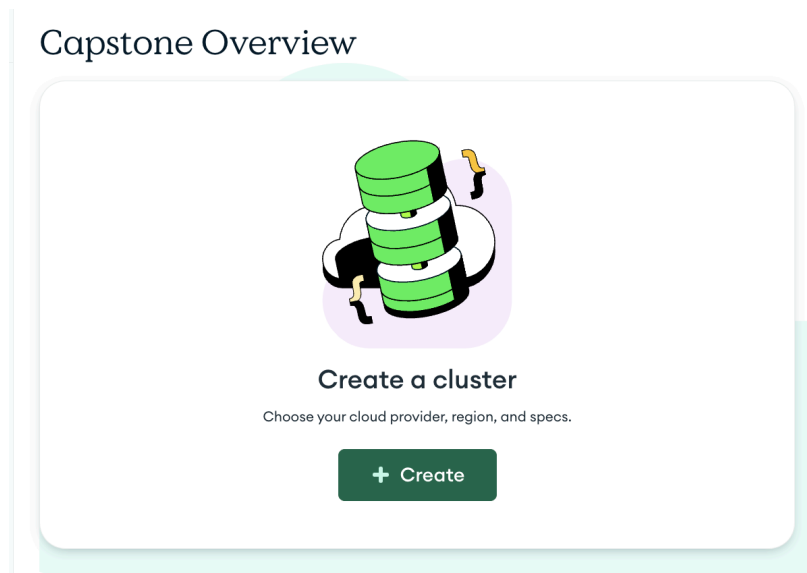
**API URL:** <https://65b795cd-6728-46f7-9d07-55dbb42b3c8a.hanko.io> 

## MongoDB:

1. Navigate to [mongodb.com](https://mongodb.com) and log into the cloud console.
2. Once logged in, navigate to the top right of the page and click on New Project



3. Give the project a useful name such as Capstone and hit next then Create Project.
4. Once the project is created, you will be able to create your database. Click on Create



5. Choose the Free Tier Cluster, name your cluster the name of the application, your preferred provider, and the Region that is closest to the location of server hosting your project. Then hit Create Deployment

## Deploy your cluster

Use a template below or set up advanced configuration options. You can also edit these configuration options once the cluster is created.

☐ M10
 

**\$0.08/hour**  
 Dedicated cluster for development environments and low-traffic applications.

STORAGE	RAM	vCPU
10 GB	2 GB	2 vCPUs

☐ Flex
 

**From \$0.011/hour**  
Up to \$30/month  
 For application development and testing, with on-demand burst capacity for unpredictable traffic.

STORAGE	RAM	vCPU
5 GB	Shared	Shared

☒ Free
 

**Free**  
 For learning and exploring MongoDB in a cloud environment.

STORAGE	RAM	vCPU
512 MB	Shared	Shared

**Free forever!** Your free cluster is ideal for experimenting in a limited sandbox. You can upgrade to a production cluster anytime.

**Configurations**  
 Name  
 You cannot change the name once the cluster is created.

**Quick setup**  
☒ Automate security setup ⓘ  
☐ Preload sample dataset ⓘ

**Provider**

**Region**  

N. Virginia (us-east-1) ★

★ Recommended ⓘ   Low carbon emissions ⓘ

6. Create your user account that will run as the administrator. Make sure to remember this password, it can be reset later if forgotten. Then hit **Create Database User**.

Connect to StratoSplit

1

2

3

Set up connection security

Choose a connection method

Connect

You need to secure your MongoDB Atlas cluster before you can use it. Set which users and IP addresses can access your cluster now. [Read more](#)

1. Add a connection IP address
 

✓ Your current IP address (174.26.138.35) has been added to enable local connectivity. Only an IP address you add to your Access List will be able to connect to your project's clusters. Add more later in [Network Access](#).
2. Create a database user
 

This first user will have [atlasAdmin](#) permissions for this project.

We autogenerated a username and password. You can use this or create your own.

ⓘ You'll need your database user's credentials in the next step. Copy the database user password.

Username

Password  


SHOW

COPY

7. For the connection method, choose Drivers and copy step 3 and paste that into the .env file
8. For the last step, on the left hand side, click on **Network Access** and edit the IP address in the options and change it to either 0.0.0.0/0 or to your server's IP address.



## IP Access List

+ADD IP ADDRESS

ⓘ You will only be able to connect to your cluster from the following list of IP Addresses:

IP Address	Comment	Status	Actions
174.26.138.35/32	Created as part of the Auto Setup process	● Active	<a href="#">EDIT</a> <a href="#">DELETE</a>

## Machine 1 Host:

### 1. Update System Packages:

```
.ssh — ubuntu@ip-10-99-36-135: ~ — ssh -i NAU_SOCGAS.pem ubuntu...
ubuntu@ip-10-99-36-135:~$ sudo apt update && sudo apt upgrade -y
```

### 2. Pull Github Repository:

```
ubuntu@ip-10-99-63-138:~$ git clone https://github.com/StratoSplit/Caelum.git
Cloning into 'Caelum'...
remote: Enumerating objects: 323, done.
remote: Counting objects: 100% (323/323), done.
remote: Compressing objects: 100% (199/199), done.
remote: Total 323 (delta 163), reused 258 (delta 111), pack-reused 0 (from 0)
Receiving objects: 100% (323/323), 18.15 MiB | 37.54 MiB/s, done.
Resolving deltas: 100% (163/163), done.
ubuntu@ip-10-99-63-138:~$
```

### 3. Install Node js and Dependencies:

Install Node

```
ubuntu@ip-10-99-63-138:~$ sudo apt install -y nodejs
```

Install base dependencies and run fix

```
ubuntu@ip-10-99-63-138:~$ cd Caelum/app
ubuntu@ip-10-99-63-138:~/Caelum/app$ npm i
```

```
ubuntu@ip-10-99-63-138:~/Caelum/app$ npm audit fix
```

Install OS specific dependencies

```
.ssh — ubuntu@ip-10-99-63-138: ~/Caelum/app — ssh -i NAU_SOCGAS....
ubuntu@ip-10-99-63-138:~/Caelum/app$ npm install dotenv bcrypt
```

#### 4. Set up .env:

Use your personal API URLs to set up the .env file.

```
GNU nano 7.2 .env
MONGO_URI=mongodb+srv://root:NokkikBSFJJp1WvA@capstone.zgone.mongodb.net/?retryWrites=true&w=majority
MONGO_DB_NAME=Caelum-Dallon
HANKO_API_URL=https://65b795cd-6728-46f7-9d07-55dbb42b3c8a.hanko.io
SSL_KEY_PATH=./key.pem
SSL_CERT_PATH=./cert.pem
```

#### 5. Manually configure first admin account:

\*User data was configured using MongoDB Compass connected to the database.

```
1  _id: ObjectId('67e5acd9a7dea2310b86695c')
2  userId : "ea2eb43f-a7e1-4723-a8d7-f546bafc5861/"
3  username : "nolan/"
4  email : "nolannew259@gmail.com/"
5  lastLogin : 2025-03-27T21:55:26.052+00:00
6  createdAt : 2025-03-27T19:54:01.820+00:00
7  role : "admin/"
8  team : "67da11b0dc8cdfec242b7dff/"
```

Role was manually set to “admin”.

#### 6. Run node server:

```
ubuntu@ip-10-99-63-138:~/Caelum/app$ sudo node server.js
```

### Machine 2 Audio Stream Generator:

#### 1. Update System Packages:

```
.ssh — ubuntu@ip-10-99-36-135: ~ — ssh -i NAU_SOCGAS.pem ubuntu...
ubuntu@ip-10-99-36-135:~$ sudo apt update && sudo apt upgrade -y
```

#### 2. Install Python3 and pip:

```
.ssh — ubuntu@ip-10-99-36-135: ~ — ssh -i NAU_SOCGAS.pem ubuntu...
ubuntu@ip-10-99-36-135:~$ sudo apt install -y python3 python3-pip
```

3. Pull Github Repository:

```
.ssh — ubuntu@ip-10-99-36-135: ~ — ssh -i NAU_SOCGAS.pem ubuntu...
ubuntu@ip-10-99-36-135:~$ git clone https://github.com/StratoSplit/Caelum.git
```

4. Run stream\_audio.py:

```
.ssh — ubuntu@ip-10-99-36-135: ~/Caelum/audio_stream — ssh -i NAU_...
[ubuntu@ip-10-99-36-135:~$ cd Caelum/audio_stream/
[ubuntu@ip-10-99-36-135:~/Caelum/audio_stream$ python3 stream_audio.py
Listening for pings on 239.0.0.11:5000...
```

## Production Cycle:

In order to develop, test, and run this application locally, we need to set up 2 Virtual Machines following the steps found in '**Machine 1 Host**' and '**Machine 2 Audio Generator**' setup. These do not need to be deployed in AWS with Multicast Enabled so long as they are on the same network and your device does not have limited privileges.

### **Machine 1: Host Console (Web Application)**

1. **Clone the repository**
  - a. ``git clone https://github.com/StratoSplit/Caelum.git``
2. **Install dependencies**
  - a. ``npm install``
3. **Start the server**
  - a. ``node server.js``

### **Machine 2: Audio Stream Generator**

1. **In a new terminal, change directories to the audio stream folder**
  - a. ``cd /path/to/project/audio_stream``

2. **Start the Stream Generator**
  - a. ``python3 stream_audio.py``
3. **Navigate to <https://localhost>**