CS476 Tech Feasibility
October 22, 2024
StratoSplit

Client:
General Dynamics Mission Systems

Mentor:
Brian Donnelly

Team Members:
Sam Cain
Elliot Hull
Nolan Newman
Dallon Jarman

Overview:

The purpose of this Technology Feasibility document is to identify key technological challenges and design decisions. With this document, we can construct a plan for well structures, and data-driven analysis to make informed decisions and avoid late development stage setbacks.

Introduction

StratoSplit is developing a Web application aimed at modernizing the United States Coast Guard Search and Rescue (SAR) Operational System. This project, sponsored and supported by General Dynamics Mission Systems, will enhance operational efficiency by introducing state-of-the-art audio processing and management capabilities into the SAR communication infrastructure.

One of the primary challenges identified by General Dynamics is the need to simultaneously manage and distinguish multiple audio streams in a browser-based environment. Specifically, the goal is to enable the system to receive, process, and play back distinct audio streams through a desktop headset or speaker configuration. This must be done while ensuring that each audio stream is easily distinguishable. This capability is critical in high-stakes SAR operations, where quick and clear communication between units can save lives.

To address this, StratoSplit's proposed solution involves creating a secure Amazon Web Services (AWS)-based Web Audio Console using a Node application. A key component of this solution is an integrated audio stream generator that will simulate these concurrent audio streams for testing and operational purposes. The generator will feed the audio streams into a web application, which will then apply spatial audio processing techniques to position each stream in a virtual 3D space.This ensures that users can differentiate between streams based on their perceived directionality or spatial positioning.

By leveraging spatial audio technology, the system can create an immersive audio experience where each stream is positioned as if coming from a unique direction, making it intuitive for users to identify and prioritize different audio sources. This approach, along with a multiple-speaker configuration, ensures that audio output remains clear, non-overlapping, and easily interpretable, even in noisy or high-pressure environments.

Additionally, the system will take advantage of AWS's cloud infrastructure to provide secure, scalable, and reliable audio streaming capabilities, ensuring real-time communication without latency issues. The browser-based nature of the application makes it accessible from any standard web browser, reducing hardware dependencies and enabling deployment across a range of operational environments.

This paper outlines the technical feasibility of implementing such a solution, covering the necessary architecture, audio stream generation and management, spatial audio processing techniques, cloud-based infrastructure, and security considerations to meet the rigorous demands of SAR operations.

Technological Challenges

Programming Language

Performance and Efficiency

- Real-time execution
- Low overhead

Concurrency and Scalability

- Parallel stream processing
- Handle multiple connections
- Optimized resource use

Ease of Integration

- Audio tools
- Cloud service support

Simplicity

- Quick debugging
- Large library support
- Troubleshooting resources

Database

Management

- Scalability
- Data integrity
- Security
- Recovery

Performance and Efficiency

- Indexing and Caching
- Real-Time data processing
- Query optimization

Account Management

- Password Management
- Session Management
- Multi-factor authentication
- Password reset and recovery

Compliance and Governance Challenges

- Regulatory compliance
- Data privacy
- Access control
- Audit logging

- Compliance reporting

Audio Streaming

Low Latency

- Real-time
- Minimal delay
- Instant Response

Multi-Stream Support

- Simultaneous streams
- Independent processing

High Audio Quality

- Clear communication
- Noise reduction
- Intelligibility

Secure and Reliable

- AWS GovCloud integration
- Encryption support
- Continuous uptime

Cross-Platform Compatibility

- Desktop browsers
- Mobile devices
- Flexibility

Audio Reception

Low Latency

- Immediate ingestion
- Real-time
- Synchronized streams

Multi-Stream Support

- Simultaneous input streams
- Independent routing

Real-Time Processing

- Quality adjustments
- Pre-forwarding prep

Scalability

- High capacity
- Adaptive resources

Monitoring and Diagnostics

- Stream tracking
- Alerts/logging
- Troubleshooting support

Sounds and Speakers

Audio quality

- High quality streaming
- Wide Volume option

Small

- Not expensive
- Fit in any space
- Travelable

Hosting

Containerization

High Performance

- Quickly boot up
- Quickly shut down

Easy to Use

- Small learning curve

Scalable

- Able to deploy many containers at once
- Containers can be of various sizes

Compatible

- Works with tools we use
- Many possible workflows

Operating System

Startup Time

- Quick Spin up

Required Resources

- RAM required to run
- Storage limitations
- CPU limitations

Responsiveness
- Usability
- Maintained at scale

Software Compatibility
- Works with many tools
- Minimal Setup

Ease of Use
- Learning curve
- Stability

Security
- Data stays secure
- Well maintained
- Updated often

Zero Trust

Network and Infrastructure Challenges
- Network Segmentation
- Micro-segmentation
- Network Device Compatibility

Device and Endpoint Challenges
- Device Authentication
- Endpoint Security
- Bring your device

Application and Data Challenges
- Application Authentication
- Data Encryption
- Data Loss Prevention

Monitoring and Analytics Challenge
- Real-Time Monitoring
- Anomaly Detection
- Security Information and Event management

Change Management Challenges
- User Adoption
- Process Changes
- Cultural Shift

Technical Analysis

1. Programming Language

Desired Characteristics

In selecting programming languages for this project, it is crucial to balance performance and efficiency with the need for concurrency and scalability. The languages must be capable of handling real-time audio processing, ensuring low latency while efficiently managing system resources. Given the need to process multiple audio streams simultaneously, strong support for concurrency and the ability to scale are essential. Ease of integration with existing tools such as FFmpeg, the G.711 audio codec, and AWS is crucial. Simplicity will be the key to reducing development time and maintaining reliability, which is necessary for rapid iteration and easy debugging. Given the choice between Python and JavaScript with Node.js, these characteristics must be carefully evaluated for each language.

Alternatives

Python is widely appreciated for its simplicity and community support. It is not the best option for efficiency, however, due to its high level of abstraction. Python excels in easy integration, thanks to a large collection of libraries and tools, including those that can interact with FFmpeg and various codecs. It is limited in concurrency due to the Global Interpreter Lock only allowing for one thread to hold control of the Python interpreter, which is not ideal. Despite this it excels in environments where quick development is key.

Node.js has an asynchronous event-driven architecture, making it a great choice for concurrency and scalability. This suggests that it can efficiently handle multiple audio streams, making it suitable for this task. Node.js excels in a balance of scaling and simplicity due to its lightweight syntax and non-blocking event loops. It integrates well with external tools and services like FFmpeg and WebSockets. JavaScript is generally an excellent choice for web-based applications where concurrency is important.

Analysis

Python is best suited for high-level integration and automation where simplicity and easy integration are key factors. It has a rich ecosystem that allows for seamless interaction with various tools. This makes it great for things like managing workflows. However, it has limited performance and constraints regarding concurrency that make it less than ideal for real-time, multi-stream audio processing. While Python can manage concurrency through various libraries, it still underperforms alternatives in situations where performance is critical. This indicates that it would scale poorly for the needs of this application and might induce extra latency when compared to other programming languages.

JavaScript (Node.js) integrates smoothly with web applications, making this an ideal choice for audio forwarding and web-based interfaces. It stands out particularly due to its

asynchronous architecture making it superior for concurrency. This language being event-driven is optimal for handling concurrent audio streams, allowing for efficient web communication. Overall, this language is a stronger choice for performance-critical applications and is highly suited for this project's core tasks, especially those requiring low latency.

Chosen Approach

For this project, JavaScript emerges as the ideal choice for handling real-time audio forwarding and management of concurrent streams. It provides efficient, scalable, and low-latency performance. These factors make it a natural fit ensuring a smooth workflow and integration with other system components.

Each possible technology has been evaluated on a scale of 1-5 based on the requirements of the audio streaming. The following chart displays a comparison of each of these technologies:

	Python	JavaScript
Performance and Efficiency	2	3
Concurrency and Scalability	2	5
Ease of Integration	5	5
Simplicity	5	5
Total Score / 20	14	18

Proving Feasibility

To prove the feasibility of using JavaScript for these applications, a test environment will be set up to simulate the core functionalities needed in this project. This test environment will be used to handle incoming audio streams, process them using FFmpeg in Node, and forward them to a web interface with WebSockets. It will be deployed on AWS to test scalability and performance under increasing loads. Various metrics will be measured and be used to validate the system's ability to perform as needed and the suitability of this technology will be confirmed.

2. Database

Desired Characteristics

MongoDB is a NoSQL, document-oriented database designed for scalability, high performance, and ease of use. The database will store user login data to allow for quick and efficient user authentication to the platform. When selecting our database, our clients highly suggested using MongoDB, a document-based database. With many people on our team already having experience with MongoDB, we have decided to pursue MongoDB as our database of choice. Looking at the characteristics of MongoDB, we learned that it is an effective choice for storing and retrieving credentials and user information. MongoDB is most known for it's horizontal scaling and load balancing capabilities which allows for more flexibility and scalability as compared to the conventional SQL and NoSQL databases. Speed in a database is a priority and it essential for ensuring that no user is waiting for longer than a couple of seconds to log in. MongoDB counteracts this by saving its data in binary-encoded JSON format. This means that all the data is stored in binary format which MongoDB claims to be much faster than JSON. The benefit of BSON is that the database can save images, videos, and audio. When it comes to security and reliability, MongoDB offers authentication services such as limiting users that can access the database and limiting roles on which data they can see. Limiting the IP's that can connect to the database to ensure no service outside of the organization can access the data. With these security implementation, this is how we can ensure confidentiality within the database. MongoDB offers a service called Database-as-a-Service that allows the database to replicate itself to other servers or automatically load balance the data and workload to ensure reliability.

Alternatives

Some alternatives that exist are MySQL, PostgreSQL, Firebase Authentication, and Amazon Cognito. After evaluating our options, we decided that due to our teams prior knownlegde and our clients wants, we are going to pursue MongoDB.

Analysis

Benchmarks	Mongo	MySQL
Inserting 5000 users	87.13s	210.15s
Adding an address to the user	6.76ms	4.18ms
Delete a user by id	5.66ms	10.21ms
Get 5000 users	35.95s	39.42s

For our analysis, we created a script that allows us to insert 5000 users into a local mongodb and mysql database. The purpose behind using local database rather than cloud hosted it we did not want our results to be skewed from network latency and network congestion. When we timed how long it took to insert all the data, we saw that inserting data into

mongodb is much faster than inserting data into mysql. The purpose behind using 5000 as compared to only inserting 1 user is because it becomes inconsistent and the times between both databases unnoticable to an average person.

The next data approach we implemented is adding an address to the users. We learned that when adding a street address to a single user, it is roughly 50% faster using MySQL as compared to MongoDB. This is not as huge of a problem as we do not anticipate the need to add data to users constantly. A user may add their email address or their phone number to their account but outside of that, a speed different does not convey our approach to using MongoDB.

As we analyze deleting a user from their id, we see that MongoDB is 50% faster than MySQL. This is not a major difference as it is only by 5 ms but can come in use if our clients will ever need to bulk delete users from their database.

Lastly, we see that when it comes to display a list of our 5000 users, we see that MongoDB is only a little bit faster. Roughly 2% faster than MySQL which will come in handy from an administrative side when an admin needs to go in an change roles and view who can access which communication lines.

Chosen Approach

When considering all of the requirements and needs, the ideal solution for user authentication in a MongoDB-based application would be to leverage MongoDB's built-in authentication mechanisms. This approach offers a seamless and efficient way to manage and verify user identities within the database itself.

MongoDB supports a range of authentication methods, including SCRAM (default), X.509 certificate authentication, and Kerberos. SCRAM provides a strong and secure authentication mechanism that is well-suited for most applications. By utilizing MongoDB's authentication features, developers can ensure the confidentiality and integrity of user data while streamlining the authentication process.

Proving Feasibility

To prove the feasibility of MongoDB, we will create a sample user data schema to prove how simple it is but effective it is to create a simple and useful user schema to add users to the database and authenticate them. From there, we will prove that MongoDB is efficient at query capabilities for authentication. Next, we will prove that the database can handle a large user base and an application such as puppeteer can help create hundreds of users within seconds to ensure that our database can handle large amounts of users. From there, we will test the security of the database to ensure that passwords are properly encrypted with NIST standards such as salting the passwords and using SHA3 as the hash.

To prove the effective of our proof of concept, the success criteria is that users can successfully log in and access authorized data. The users are able to authenticate and authorize their accounts. All of the user data is securely stored and protected from malicious attackers and attacks like rainbow tables. Lastly, our database is scalable and can handle hundreds of accounts being made at once will still effectively allowing our users to sign into their accounts.

3. Audio Streaming

Desired Characteristics

Several characteristics are important for ensuring efficient and reliable audio streaming for SAR operations:

- **Low Latency:** The streaming system must operate with minimal delay to ensure real-time communication.
- **Multi-Stream Support:** The system should support simultaneous audio streams, ensuring that each stream can be processed independently and mixed as needed.
- **High Audio Quality:** Audio must remain clear and intelligible
- **Secure and Reliable:** Given the sensitivity of SAR operations, the streams must be securely handled within AWS.
- **Cross-Platform Compatibility:** The solution must work across platforms including desktop browsers and mobile devices, ensuring flexibility in how audio streams are accessed.

Alternatives

One solution would be to use FFmpeg with RTP. FFmpeg is a powerful multimedia framework that supports real-time audio streaming, low-latency transmission, built-in security, and server-side mixing. It is well-suited for handling multiple streams and forwarding them to clients via HTTPS, including mobile devices. Flexibility in command-line configuration allows for fine control over audio processing, making it a strong choice for SAR operations. FFmpeg is highly compatible with cloud platforms like AWS and can scale effectively.

Another potential solution would be to use WebRTC. WebRTC is a peer-to-peer communication technology that supports low-latency audio transmissions between browsers. It provides built-in encryption for secure audio communications. WebRTC does not natively support server-side mixing, which would result in additional infrastructure being needed to manage multiple audio streams centrally, which may introduce complexities with mobile applications and the implementation of a Zero Trust security methodology.

One final solution would be to use GStreamer. Gstreamer is an open-source multimedia framework designed to build audio and video processing pipelines. It supports server-side mixing of multiple audio streams and real-time streaming over RTP. GStreamer offers significant customization, but setup can be complex and time-consuming. The complexity of this software might be overkill for straightforward audio streaming needs.

Analysis

FFmpeg offers extensive control over audio streams, allowing real-time mixing and forwarding. Its ability to handle both desktop and mobile platforms makes it highly flexible for SAR operations. It is an excellent choice for real-time, multi-stream environments where server-side control is key, thus supporting a Zero Trust security system. FFmpeg's compatibility with mobile platforms and cloud environments makes it the most robust option for centralized audio streaming and mixing.

WebRTC excels in delivering low-latency audio transmissions with built-in security features. However, it lacks server-side mixing, which is crucial for managing multiple audio streams. Implementing WebRTC would require building additional infrastructure for centralized stream management, leading to more complexity. While this may be ideal for browser-based applications, WebRTC is less suitable for multi-stream environments.

GStreamer offers powerful multimedia processing with a high degree of customization. It is particularly suitable for advanced, complex setups requiring fine-tuned control over multiple audio streams. However, its complexity can lead to longer setup and may require specialized knowledge. While GStreamer is flexible, it is not the most efficient option for straightforward SAR applications.

Chosen Approach

When considering all of the requirements and needs, the ideal solution for audio streaming in SAR operations would be to use FFmpeg with RTP according to the scores found in Table

WebRTC may offer advantages in secure, low-latency communication and GStreamer provides flexibility in building audio pipelines, but FFmpeg is the most suitable for our specific needs. Its combination of low-latency performance, multiple stream support, server-side mixing, and mobile compatibility make it the best fit for real-time audio streaming in SAR environments. FFmpeg has ease of integration with cloud platforms such as AWS ensuring scalability as demands grow.

Each possible technology has been evaluated on a scale of 1-5 based on the requirements of the audio streaming. The following chart displays a comparison of each of these technologies:

	FFmpeg	WebRTC	GStreamer
Latency	5	4	3
Multi-Stream Support	5	3	4
High Audio Quality	5	4	4
Secure and Reliable	5	2	4
Cross-Platform Compatibility	5	4	3
Total Score / 25	25	17	18

Proving Feasibility

FFmpeg has been evaluated to ensure it meets the desired characteristics for audio streaming. A test environment using two EC2 instances has been set up to accept multiple audio streams and forward them to various ports, confirming the system's ability to handle server-side mixing and stream management. Latency tests have shown the system provides low-latency. Initial results demonstrate the system's potential for secure streaming. Multicast streaming capabilities have been validated, ensuring delivery of streams to multiple clients. Mobile testing has not yet been performed, but cross-platform compatibility will be assessed in future phases of development.

4. Audio Reception

Desired Characteristics

There are multiple considerations to take into account when picking an audio reception technology for this type of application:

- **Low Latency:** The system must operate with minimal delay to ensure real-time communication.
- **Multi-Stream Support:** The system must support receiving multiple simultaneous audio streams, allowing them to be processed independently and routed to the correct endpoint.
- **Real-Time Processing:** The system must offer real-time audio processing capabilities to optimize audio quality and assign spatial audio positioning.
- **Scalability:** As operations grow, the system must scale effectively to handle an increasing number of audio streams.
- **Monitoring and Diagnostics:** Continuous monitoring and diagnostics are essential to stream health, ensuring efficient resolution of performance issues.

Alternatives

A potential technology for audio reception is FFmpeg. FFmpeg is a powerful tool that supports multiple low-latency audio streams simultaneously. It provides robust real-time processing features critical to the program such as encoding and decoding and is highly scalable in cloud environments. FFmpeg does not offer monitoring and diagnostic tools without integration of external tools. This technology excels in core reception and processing tasks, especially when combined with additional monitoring tools.

Nginx with RTMP is also a good choice for audio reception. It is a lightweight, efficient solution for handling real-time audio reception. It is capable of managing multiple streams effectively, making it well-suited for scalability. It lacks built-in real-time audio processing features, so it would need to be paired with external tools for preprocessing of audio data. It has basic built-in monitoring and diagnostic capabilities making it more appropriate for simple setups. Nginx is particularly powerful for forwarding audio streams efficiently, but is not particularly good at complex audio manipulation or processing.

Analysis

FFmpeg and Nginx together form a powerful combination for audio reception and forwarding which is crucial for the application. FFmpeg handles the core reception tasks, providing strong real-time processing capabilities after the audio is received. Its multi-stream support and low-latency performance make it highly adaptable in environments where time-sensitive communication is essential. Additionally, its scalability allows it to grow with operational demands, despite it lacking tools for monitoring and diagnostics.

Nginx with RTMP serves as an excellent compliment to FFmpeg, efficiently managing the forwarding of audio streams to the web application. It excels in scalability and low-latency performance, making it ideal. Nginx lacks built-in audio processing, but when used with FFmpeg for processing it can monitor these audio streams for active connections, bitrate and stream health, and stream metadata. These technologies working together would ensure that the processed streams are transmitted reliably.

Chosen Approach

The ideal approach for audio reception in SAR operations would involve combining FFmpeg for real-time processing and Nginx with RTMP for efficient forwarding. FFmpeg will handle the critical tasks of audio reception and preprocessing, after which it will pass them to Nginx to ensure the audio streams are forwarded to the web application with minimal latency. This mixing of technologies provides flexibility, scalability, and low-latency performance needed for an effective application.

Each possible technology has been evaluated on a scale of 1-5 based on the requirements of the audio reception. The following chart displays a comparison of each of these technologies:

	FFmpeg	Nginx	Both
Latency	5	4	5
Multi-Stream Support	5	4	5
Real-Time Processing	5	1	5
Scalability	5	5	5
Monitoring and Diagnostics	0	4	4
Total Score / 25	20	18	24

Proving Feasibility

FFmpeg and Nginx have been evaluated to ensure they meet the desired characteristics of audio reception. Multicast streaming capabilities have been validated through the EC2 test environment handling incoming streams and forwarding them to their respective destinations. Latency tests suggest real-time processing and streaming capabilities and gradual increasing of audio streams suggests scalability. Stress testing will be performed and cross-platform capabilities will be assessed further into development. Continuous uptime testing will be used in order to evaluate the long-term stability of the system.

5. Sounds and Speakers

Desired Characteristics

The system should allow an operator to monitor multiple radio signals simultaneously with high-quality audio. It should provide the ability to switch between monitoring signals through speakers or isolate a single signal via headphones. The setup must ensure clear audio reproduction, support for multichannel inputs, and offer flexible signal management. The hardware must perform consistently in an indoor environment, providing accurate sound output for extended listening periods, which is essential for effective radio signal monitoring.

Alternatives

The LSR6 speakers are known for their high-quality audio performance, utilizing weatherized magnetic planar drivers for clear and precise sound reproduction. They feature a woven glass-fiber woofer that enhances low-frequency response and overall tonal balance. Pros include excellent sound clarity suitable for critical listening tasks, high output without distortion, making them ideal for multi-signal monitoring, and durable construction capable of handling various environments. Cons include a premium price point that may exceed budget constraints for larger setups and the possibility of requiring additional amplification or processing for optimal performance in larger arrays.

The Spatial Audio Lab S-Series offers affordable yet high-quality speakers designed for consistent sound reproduction. The S-Series is tailored for long listening sessions, providing a balanced audio profile that enhances clarity and reduces listening fatigue. Pros include cost-effectiveness, making it easier to acquire multiple units for an array, reliable sound quality ensuring clear reproduction of multiple signals, and lightweight, compact design suitable for desk setups. Cons include that while affordable, they may not deliver the same high-end fidelity as more expensive options like the LSR6, and the bass response may not be as robust as premium models, which could affect the overall audio experience.

The Genelec 8010A is a compact studio monitor renowned for its excellent sound quality and small footprint. It is designed for professional use and is highly regarded for its precise audio reproduction, making it suitable for critical listening environments. Pros include exceptional audio fidelity ideal for monitoring nuanced signals, a compact design that allows for easy arrangement in a desk setup suitable for an array, and built-in room response controls that enable optimal sound in various environments. Additionally, it fits well within the budget, costing approximately $350 per unit. Cons include a slightly higher price than some budget options, but

justified by superior performance, and limited low-frequency extension compared to larger monitors, which may not be ideal for all audio applications.

Analysis

The Landscape Ribbon Six (LSR6) speakers deliver superior sound clarity, ideal for the nuanced requirements of monitoring multiple signals. However, their premium pricing may limit their feasibility for larger setups. The Spatial Audio Lab S-Series offers a more budget-friendly alternative without sacrificing performance, ensuring clear signal reproduction during extended listening sessions. The Genelec 8010A stands out as an excellent choice, combining professional-grade audio quality with a compact design, making it a highly suitable option for monitoring in constrained spaces.

Chosen Approach

Rated on a scale from 1-10 where larger is better

	Genelec 8010A	Spatial Audio Lab S-Series	LSR6 speakers
Price	8	4	1
Size	10	7	5
Audio Quality	6	8	10

Proving Feasibility

The Genelec 8010A speakers are chosen for their exceptional audio fidelity and compact design, ensuring accurate sound reproduction for monitoring multiple signals over extended periods. While the quality of the audio is lower than that of other options, the size makes it accessible for most operators regardless of space restrictions. Additionally, their built-in room response controls allow operators to tailor the sound to their environment, making them versatile for various listening situations. With a 45-day return policy for testing and evaluation, operators can assess their suitability before full commitment, costing approximately $350 per unit.

Connection and Additional Hardware Requirements

To effectively utilize up to eight Genelec 8010A speakers, will require a reliable audio interface, such as the Behringer UMC1820, which can connect directly to a Linux computer via USB. The UMC1820 allows for multichannel audio output, enabling all speakers to function from a single computer. No additional software is needed to operate the UMC1820; it is recognized by the Linux system as an audio device, facilitating straightforward audio routing to the Genelec speakers.leveraging Python libraries such as sounddevice or PyAudio to control audio playback seamlessly. The price for a complete 6 speaker array with the audio interface and will come out to $2400 which is below the cost ceiling.

6. Hosting

Desired Characteristics

When it comes to hosting our application, our only option is AWS GovCloud. AWS is a robust and scalable hosting solution. The main application we will be using on AWS are EC2 instances. EC2 offvers some key advantages:

- **Availability:** EC2 provides options for creating highly available configurations, including load balancing and auto-scaling.
- **Scalability:** EC2 instances can be easily scaled up or down to accommodate fluctuating workloads.
- **Security:** EC2 offers a variety of security features, such as security groups, network access control lists (NACLs), and instance-level security.
- **Performance:** EC2 instances can be configured with different instance types and sizes to optimize performance.
- **Cost-Efficiency:** EC2 offers a variety of pricing options, including on-demand, reserved, and spot instances, to meet different budget requirements.

When selecting the ideal EC2 instances for our application, several key factors must be carefully considered.Compute Power is paramount. We need to assess the required CPU and memory resources to ensure our application can handle its workload efficiently. This includes considering the peak usage and the expected growth of the application.

Storage is another critical factor. We must determine the amount of storage necessary for the application's data and logs. This includes both the initial storage requirements and the projected future growth.

Networking plays a vital role in application performance. We need to evaluate the bandwidth and network performance requirements to ensure smooth communication and data transfer. This includes considering the network latency and throughput needs of the application.

Finally, Cost is a significant consideration. We must balance the performance requirements with the budget constraints to select the most cost-effective EC2 instance type. This involves considering factors such as on-demand pricing, reserved instances, and spot instances.

Based on the previous factors, an EC2 instance will fit all of our needs and will allow us to achieve any workload needed such as:

- **General-purpose:** Suitable for a wide range of workloads, including web servers, application servers, and small databases.
- **Compute-optimized:** Designed for compute-intensive workloads, such as high-performance computing and machine learning.
- **Memory-optimized:** Ideal for applications that require large amounts of memory, such as databases and in-memory data stores.
- **Storage-optimized:** Optimized for storage-intensive workloads, such as data warehousing and big data analytics.

7. Containerization

Desired Characteristics

The containers used by our project have to be:

- **High Performance:** Able to quickly initialize, so that the user doesn't experience long delays when trying to access a new, dynamically created container.
- **Easy to Use:** The development team, and end user alike should be able to interact with this container easily.
- **Scalable:** The container system should be able to support many instances, in order to support many users trying to use the service at once.
- **Compatible:** The container shouldn't interfere with other tools or systems that are used with this project.

Alternatives

For the first solution considered by our group, we looked at Docker, one of the more popular container systems. This seems like a fairly good solution, though another possible solution would be Podman. This container system is fully open source, and seen as viable, if less popular and not as well supported.

Analysis

Docker has a lead in a couple of key ways over Podman. Being the more popular choice, there is a surplus of information about how to use Docker, in many different environments, and help for how to fix errors. Podman has these forums too, though not to the same degree. Docker is generally considered to be easier to use, with a shallower learning curve, and is very performant. Its prevalence also lends itself to compatibility with many different softwares.

However, Podman does pull ahead in the security segment. Podman has been optimized for security first, which may have led to its deficiencies elsewhere. This focus on security is a nice touch, though maybe not enough to have it pull ahead of Docker's many upsides.

	Docker	Podman
Performance	4	3
Difficulty	4	3
Scalability	5	4
Compatibility	5	4
Total Score / 20	18	14

Chosen Approach

We chose to go with Docker over Podman. Podman's security features would be nice, but the improved ease of use, and compatibility of Docker proved to be more important. In terms of security, with the server itself being very secure, and Docker still being fairly trustworthy in terms of security, Padman's advantages did not overwhelm the advantages of Docker.

8. Operating System

Desired Characteristics

The most important characteristics of the operating system will be:

- **Startup Time:** The time it takes to "spin up" the OS while starting it within a container, especially since containers will need to be spun up dynamically on the server.
- **Required Resources:** The system requirements for running the OS, as this will impact how many containers we can expect to be able to feasibly run at any given time.
- **Responsiveness:** How quickly the OS will respond to any user input, especially if it is only provided limited resources while running on the server.
- **Software Compatibility:** The compatibility of the OS with tools that we may need to use in our project, for sending and receiving audio streams, and implementing the interface.
- **Ease of Use:** How difficult will it be for our team to implement the workflow into the operating system, and will it require manual installation of tools.
- **Security:** Is the operating system secure, in how it handles data?

Alternatives

The first solution we looked at was an Ubuntu distribution of linux. This is a very mainstream distribution of linux, and is often rated one of the easiest to use. Its compatibility and security are both very robust, as it is built for less experienced users, and is often updated. However, the startup, necessary resources, and responsiveness all suffer due to included tools.

Another option to consider would be Debian. Debian shares the advantage of being a mainstream linux distribution with Ubuntu, with the added advantage of having less "fluff" included by default. Instead, Debian requires more tools to be installed, creating a bit of friction in the ease of use category. Though, this is mostly offset by the improved startup, resource use, and responsiveness that comes with a "lighter" install, that is overall easier for a computer to run.

An additional option for us to consider was the Amazon-specific linux distribution. Since the team will be using AWS, it would be easy to implement Amazon's specific linux distribution that works well with AWS services "out of the box". Since this is also optimized to be run, containerised, within AWS, it also offers great performance. However, since it's use cases are also more limited, and its support restricted to specifically Amazon, it may have a steeper learning curve, and less community information to go off of.

Analysis

Ubuntu and Debian are likely the easiest for our group to use. However, since we are likely to need to modify our work environment anyway, and since containerizing the OS will help normalize any changes we need, the in-built AWS solution for an OS, through Amazon Linux will likely be the best option to take. Our group's prior experience with Linux makes the ease of use category less important, which results in Amazon Linux 2's better performance, and focus on security is a good pick for our operating system.

	Ubuntu	Debian	Amazon Linux 2
Startup Time	3	4	5
Required Resources	3	4	5
Responsiveness	3	4	5
Software Compatibility	5	4	4
Ease of Use	5	4	3
Total Score / 25	19	20	22

Chosen Approach

We have chosen the Amazon Linux 2 OS to be the operating system that we build off of for our application. The OS is immediately available, and is ensured to be focused primarily on both security and performance. Also, being designed to run on the AWS servers, its compatibility should be an asset while working on getting each container up and running.

9. Zero Trust

Desired Characteristics

A robust Zero Trust architecture is essential for securing sensitive information. Our proposed solution aims to meet the following key characteristcs. FedRAMP and NIST compliance are essential for government agencies handling sensitive information. FedRAMP ensures that cloud service providers meet rigorous security standards, protecting data from unauthorized access and cyber threats. NIST compliance provides a comprehensive framework for managing cybersecurity risks, including identifying, assessing, and mitigating vulnerabilities. By adhering to these standards, we can ensure our application is securly transmitting data and is following all modern security standards.

Role-based access control (RBAC) is a fundamental security measure that restricts user access to specific resources based on their assigned roles. This approach minimizes the risk of unauthorized access, data breaches, and insider threats. By limiting user privileges to only what is necessary for their job function, our applcation can effectively protect sensitive information.

Access logging is a critical component of security monitoring and incident response. By tracking user activity, we can detect anomalies, investigate security incidents, and identify potential threats. Access logs provide valuable insights into system usage, helping to identify unauthorized access attempts, data breaches, and other security incidents. By implementing robust access logging practices, we can strengthen the security of our application and ensure no data is getting leaked.

HTTPS, a secure communication protocol, is essential for protecting sensitive data transmitted over the internet. By encrypting data, HTTPS prevents unauthorized interception and ensures the confidentiality and integrity of information. This is particularly crucial for government agencies handling classified information. Without HTTPS, sensitive data could be exposed to malicious actors, leading to data breaches, identity theft, and reputational damage.

Preventing lateral movement is a critical security measure that limits an attacker's ability to move within a network once they have gained initial access. Lateral movement allows attackers to access additional systems and data, escalating the impact of a security breach. By implementing strong network segmentation, limiting administrative privileges, and using network access control lists (ACLs), organizations can significantly reduce the risk of lateral movement.

Server-side access logging provides valuable insights into system activity, enabling security teams to identify and investigate potential threats. By monitoring user logins, failed login attempts, and unusual activity, organizations can detect and respond to security incidents promptly. Server-side access logs can also be used to conduct forensic analysis in the event of a security breach, helping to identify the root cause and take corrective action.

Alternatives

Several network connectivity options were explored to ensure secure and reliable access to AWS resources. Each option offers distinct advantages and disadvantages. AWS Direct Connect provides a dedicated, private network connection between the on-premises network and AWS. This option offers high bandwidth and low latency, making it ideal for organizations with significant data transfer needs. However, it can be more complex to set up and manage

compared to other options. AWS VPN offers a virtual private network (VPN) solution, creating a secure, encrypted connection to AWS resources. This option is more flexible and can be used for various use cases. However, it may have performance limitations, especially for large-scale deployments. WireGuard VPN is a modern and efficient VPN protocol, known for its simplicity and performance. It offers strong security and can be easily configured. However, it may require additional management and configuration, especially for complex network environments.

Chosen Approach

For our infrastructure, we've chosen AWS GovCloud due to it being a requirement from our clients.To automate the provisioning and configuration of our infrastructure, we'll utilize Terraform, ensuring consistency and efficiency in our deployment process.

AWS Network Firewall is a robust security tool that provides a powerful layer of protection for our network. By implementing Network Firewall, we can filter and monitor network traffic, blocking malicious activity and unauthorized access. This includes the ability to inspect traffic for threats, such as malware, viruses, and exploits. By establishing granular control over network traffic, we can significantly reduce the risk of breaches and protect our sensitive data.

AWS Verified Access is a Zero Trust Network Access solution that offers a more secure and efficient approach to user access compared to traditional VPNs. By eliminating the need for a VPN, AWS Verified Access reduces the attack surface, as there's no longer a permanent connection to the network. Instead of relying on network location as a security factor, AWS Verified Access focuses on user identity, device posture, and application access policies. This fine-grained approach ensures that only authorized users can access specific applications and data, even if they're on an untrusted network. By continuously verifying user identity and device health, AWS Verified Access provides a robust security posture. This helps prevent unauthorized access and reduces the risk of data breaches. Additionally, AWS Verified Access simplifies user access, as users can access applications directly from their web browsers without the need to install VPN clients.

Proving Feasibility

To ensure secure and zero trust access to our application on AWS GovCloud, we will establish a robust connection between our on-premises network and the cloud environment. Rigorous security measures will be implemented to filter and monitor network traffic, safeguarding sensitive data. The application will be deployed to AWS GovCloud, adhering to strict security and performance standards. User access will be strictly controlled through AWS Verified Access, ensuring only authorized individuals can interact with the application. Continuous monitoring will be employed to maintain optimal performance, security, and user experience. Proactive risk identification and mitigation strategies will be implemented to safeguard the system and its data.

10. **Logging**

Desired Characteristics

Comprehensive logging is essential for maintaining the security and reliability of enterprise applications. A robust logging strategy encompasses multiple layers: application, system, network, and centralized log management, along with corresponding alerts and notifications. Application-level logging implements strategic log statements within the codebase to track user actions, system behaviors, and error conditions. This granular logging enables rapid identification and resolution of issues, minimizing application downtime. System-level logging monitors the health and security of infrastructure components, such as EC2 instances. These logs capture critical data including user access patterns, command execution, and file system modifications, providing crucial visibility into system operations. Network-level logging, through tools like iptables and tcpdump, serves as a critical defense mechanism by monitoring network traffic for unauthorized access attempts and potential security threats. This monitoring enables the detection of anomalous patterns and suspicious activities.

Next, centralized log management solutions such as AWS CloudWatch Logs aggregate data from these diverse logging sources. This consolidation facilitates comprehensive analysis, trending, troubleshooting, and swift incident response, enhancing the overall security posture of the application. Finally, implementing a robust alerting and notification system, such as using tools like ntfy, ensures that we are promptly notified of critical events and security incidents. This allows us to take timely action to mitigate risks and minimize downtime.

Alternatives

Several options are available for effective log-based management systems. Each approach offers distinct advantages and disadvantages, making the choice dependent on specific requirements and priorities.

AWS CloudWatch Logs is a powerful and widely-used solution. It seamlessly integrates with other AWS services, making it easy to collect and analyze logs from various sources. Its scalability allows it to handle large volumes of logs and automatically adjust to changing needs. Additionally, CloudWatch Logs offers advanced features like log filtering, alerting, and visualization. However, relying on AWS infrastructure can introduce vendor lock-in, and costs can escalate for large-scale log ingestion and analysis.

ELK Stack, an open-source suite, offers flexibility and customization. Logstash collects logs from diverse sources, Elasticsearch stores and indexes them, and Kibana provides a user-friendly interface for analysis and visualization. While powerful, it requires more complex setup and management compared to CloudWatch Logs. Moreover, it can be resource-intensive, especially for large-scale deployments.

Simple file-based logging is a straightforward approach, suitable for smaller-scale applications. Its ease of implementation and low cost are attractive. However, managing and analyzing large log files can become cumbersome. Log rotation tools like Logrotate can help automate this process.

Considering these alternatives, we've chosen AWS CloudWatch Logs as our primary logging solution. Its seamless integration with our AWS infrastructure,scalability, and advanced features make it an ideal choice. By leveraging CloudWatch Logs, we can efficiently collect, analyze, and monitor logs from various sources. This enables us to identify and troubleshoot

issues, respond to security threats, and gain valuable insights into our application's performance.

Additionally, CloudWatch Logs offers cost-effective pricing models and can be easily scaled to accommodate our growing needs. By utilizing this robust logging solution, we can ensure the reliability and security of our application

Chosen Approach

To ensure robust logging and monitoring of our application, we will implement a layered approach that encompasses application, system, and network levels. This multi-faceted strategy will provide comprehensive visibility into our system's behavior, enabling early detection of issues and proactive resolution.

At the application level, we will incorporate detailed logging statements within our code. This will capture critical information such as user interactions, system errors, and performance metrics. By logging events like user logins, API requests, and database queries, we can gain valuable insights into application behavior and identify potential bottlenecks or security vulnerabilities.

To complement application-level logging, we will implement system-level and network-level monitoring. System-level logging provides crucial information about the underlying operating system and infrastructure, including logs from EC2 instances and operating systems logs. By monitoring these logs, we can identify hardware failures, software vulnerabilities, and security threats. Network-level logging captures information about network traffic, including incoming and outgoing connections. This helps us monitor network performance, detect anomalies, and identify potential security breaches. Tools like iptables and tcpdump can be used to capture network traffic and generate detailed logs.

By combining these layers of logging and monitoring, we can create a comprehensive view of our application's behavior and identify potential issues early on. This proactive approach enables us to maintain optimal performance and security, ensuring a seamless user experience.

Proving Feasibility

To validate the feasibility of our proposed logging strategy, we will implement a series of steps to ensure comprehensive and effective logging. We will leverage the power of CloudWatch Logs to centralize and analyze our logs. By configuring CloudWatch Logs to collect logs from various sources, including EC2 instances, Lambda functions, and our application, we can gain a holistic view of our system's behavior. This involves setting up appropriate log groups and log streams to organize and categorize logs effectively.

To capture granular details about our application's behavior, we will integrate logging statements into our code. This will enable us to log user interactions, system errors, and performance metrics, providing valuable insights for troubleshooting and performance optimization.

To monitor the health and security of our infrastructure, we will enable detailed system-level logging on our EC2 instances. This will provide crucial information about system events, security alerts, and resource utilization.

To safeguard our network and detect potential security threats, we will configure network-level logging tools like iptables and tcpdump. These tools will monitor network traffic, identify anomalies, and provide valuable insights into network behavior.

To ensure the reliability and effectiveness of our logging solution, we will conduct rigorous testing. We will verify that logs are being collected, stored, and indexed correctly. Additionally, we will test the functionality of our alerting and notification system to ensure that critical events are promptly reported to relevant stakeholders.

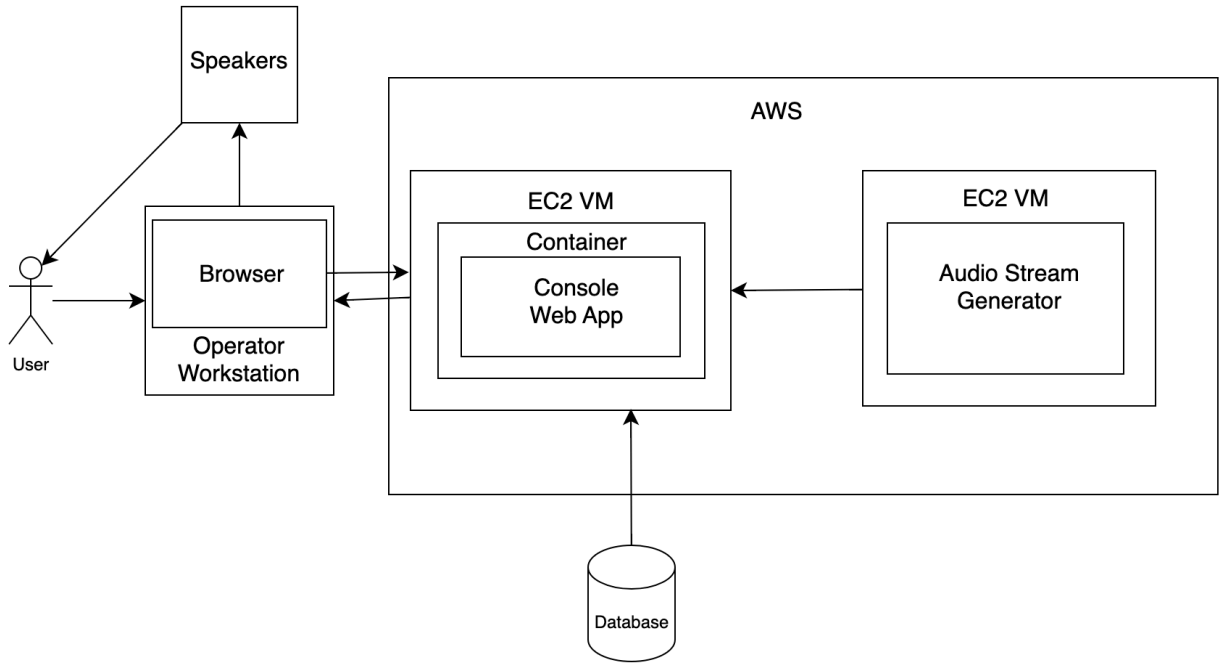
Based on our testing and monitoring efforts, we will continuously refine our logging strategy to optimize its effectiveness. This may involve adjusting log levels, adding new log sources, or fine-tuning alert thresholds to strike the right balance between capturing essential information and minimizing log volume. By following these steps, we can establish a robust, reliable, and effective logging solution that empowers us to maintain optimal performance, security, and operational efficiency.

Technological Integration

System Diagram

Given the discussed solutions, the diagram below shows a comprehensive mapping of how they will all interact. The user will login in through a browser based application using credentials stored in the database. After authentication the server makes a request to start a new container for the session. This server will forward the audio streams assigned to the user to the container further allowing them to mix them into spatial audio through the web application. This audio is then output to a multi speaker set up or headset for the user. The audio stream generator continually sends all streams to the server, but only allows users access to prespecified streams. Both of these machines will be hosted within separate AWS EC2 virtual machines and will communicate via RTP. This may eventually be expanded to include mobile devices further into development.

The figure below is a system diagram of our proposed solution:



Conclusion

In conclusion, our project consists of creating a web application capable of mixing audio streams into spatial audio for General Dynamics Mission Systems. It is aimed at developing an efficient, scalable, and secure system to be used in SAR operations. We have put together a security methodology, audio streaming techniques, hardware, and software in order to complete this. With the requirements of using AWS in addition to Node, we have found the best suited solutions for each challenge and have what is required to begin initial development. These technologies are what is needed to enable low-latency, encoded communication between our audio stream generator, the application, and the client browser while maintaining a Zero Trust security methodology.

For operating systems, we have decided on using Amazon Linux 2 OS. JavaScript is an excellent choice for the programming language given its compatibility with Node. FFmpeg is a powerful tool for the generation of audio streams and when combined with Nginx is fully capable of receiving the audio streams and routing them to the Node app. The Genelec 8010A speaker provides a combination of functionality and a lower price point perfect for the hardware needs. The combination of these components provide an effective solution for the overall requirements. The system will have the ability to handle real-time streaming and have security that will certainly meet the needs of operational demands of both current and future missions. This software will lay a flexible foundation for continued development and assist General Dynamics in their mission of efficient radio communication over the internet.