**1. Application Overview**

The Group36 application is a cloud-based microservices solution designed to handle video processing and provide API services to users. It is structured to process video data efficiently while offering a web-based API that enables seamless interaction for end-users. Leveraging Amazon Web Services (AWS) infrastructure, this application ensures scalability, reliability, and cost-effectiveness, addressing both CPU-intensive tasks and frequent API requests. Built with a microservices architecture, the application consists of two primary components: a web client API and a video processing service, both hosted on Amazon Elastic Container Service (ECS) with Fargate for streamlined container management. This design empowers the application to dynamically adjust its resource usage based on workload demands, making it a robust solution for enterprise-grade deployments.

**2. Application Architecture**

**Architecture Diagram and Overview**

The architecture of the Group36 application is a microservices-based model hosted entirely on AWS, leveraging cloud-native services to ensure reliability and scalability. Below is a breakdown of each major component used in the architecture:

* **Elastic Container Service (ECS) with Fargate**: ECS manages and runs the Docker containers for both the web client API and video processing microservices. ECS Fargate abstracts the need to manage underlying server infrastructure, allowing for seamless scaling and reducing the operational overhead associated with container orchestration.
* **Application Load Balancer (ALB)**: The ALB distributes incoming HTTP requests across ECS tasks, ensuring that both the web client API and video processing services can handle multiple concurrent requests efficiently. This load balancing capability enables fault tolerance, ensuring continuous service even if individual instances fail.
* **Elastic Container Registry (ECR)**: ECR serves as a secure Docker image repository for both services, simplifying deployment updates and version control of container images. With ECR, deploying updated services is straightforward, allowing for quick iteration without complex repository management.
* **Amazon Route 53**: Route 53 provides domain name services, routing user requests through the ALB for enhanced accessibility and reliability. This allows users to interact with the application via a user-friendly domain name, masking the complexities of the underlying infrastructure.
* **Amazon Secrets Manager**: Secrets Manager securely stores and manages credentials for accessing private repositories on ECR. This integration ensures that credentials are stored securely and are easily accessible to authorized services, enhancing the application's security.
* **Amazon CloudWatch**: CloudWatch monitors and logs performance metrics for ECS services, offering insights into CPU, memory usage, and error rates in real-time. This allows for proactive management and troubleshooting of service issues.

The architecture is designed to be modular, with each service performing a specific role within the application ecosystem, promoting reliability, scalability, and efficient resource utilization.

A diagram of a process

Description automatically generated

**3. Justification of Architecture**

The architecture of the Group36 application was designed with scalability, security, and performance in mind, aligning with the requirements of a cloud-based microservices solution.

**Division into Microservices**

The decision to split the application into two primary microservices—the web client API and the video processing service—was driven by the unique requirements of each service. The web client API is optimized for handling HTTP requests from end-users, while the video processing service is CPU-intensive, focusing on resource-demanding tasks. This separation enables each service to be managed, scaled, and optimized independently, promoting modularity and maintainability.

**Choice of Compute: ECS with Fargate**

Amazon ECS with Fargate was chosen to manage the Docker containers due to its serverless infrastructure approach, which eliminates the need to maintain the underlying servers. Fargate allows for automatic scaling based on the defined CPU and memory thresholds, which is essential for the CPU-heavy video processing tasks. The serverless model of Fargate also reduces operational complexity, allowing the team to focus on application development rather than infrastructure management.

**Load Distribution and Communication**

The Application Load Balancer (ALB) plays a crucial role in distributing incoming requests among ECS tasks. The ALB supports HTTP/HTTPS protocols and offers automatic routing based on health checks, ensuring that user requests are handled by healthy instances. This approach enhances fault tolerance and improves the resilience of the application.

**Service Abstractions**

Service abstractions like ALB, ECR, and Secrets Manager streamline the application’s infrastructure by simplifying common tasks. The ALB manages routing and load distribution, ECR provides a managed Docker registry for storing container images, and Secrets Manager securely stores credentials for accessing ECR. These abstractions reduce complexity, improving the application's performance and security.

These design choices reflect a balance of performance, cost, and ease of management, ensuring that the application remains flexible, scalable, and adaptable to changing requirements.

**4. Response to Project Criteria**

This section outlines how each project criterion is fulfilled by the application deployment, architecture, and implementation on AWS, demonstrating that the solution meets the functional, security, scalability, and sustainability requirements for effective cloud deployment.

**Criterion 1: Functional Requirements**

The application is designed to handle video processing and API requests in a distributed, cloud-based environment. The microservices architecture ensures that CPU-intensive video processing is handled separately from the user-facing web client API, optimizing performance and ensuring that user interactions do not interfere with the backend processing. By using ECS with Fargate for container orchestration, the application scales automatically based on demand, meeting functional requirements for elasticity and performance under varying workloads.

**Criterion 2: Scalability**

Scalability is achieved through the use of ECS with Fargate, which allows the application to dynamically scale containers up or down based on CPU and memory usage. The Application Load Balancer (ALB) manages incoming traffic, distributing requests across healthy instances and ensuring even load distribution. The architecture supports the potential for high concurrency, with Fargate’s serverless model allowing additional containers to launch quickly in response to increased demand.

**Criterion 3: Security**

The application follows best practices for security, leveraging AWS IAM roles to enforce least privilege access across services. IAM roles are assigned specifically to ECS tasks to ensure that each microservice only has access to the resources it needs. Data transmitted through the ALB can be encrypted with HTTPS, ensuring secure user interactions. Secrets Manager is utilized to store sensitive information, such as ECR credentials, securely. Additionally, CloudWatch monitoring provides insights into access patterns, enabling quick identification of potential security incidents.

**Criterion 4: Cost Efficiency**

The architecture is cost-effective, leveraging managed services like ECS with Fargate, which reduces the need for manual infrastructure management and minimizes idle resource costs. The AWS Pricing Calculator was used to estimate monthly costs based on 50 concurrent users, with provisions for further scalability if demand increases. By using Fargate, the application incurs charges based on resource usage rather than a fixed capacity, ensuring that costs are aligned with demand.

**Criterion 5: Sustainability**

The application design is optimized for sustainability by using serverless computing (ECS with Fargate), which reduces resource waste by only allocating resources when needed. AWS’s commitment to renewable energy also contributes to the sustainability of this solution, as it operates on energy-efficient cloud infrastructure. In addition, the application could be further optimized by implementing code improvements and container optimization to reduce CPU and memory usage.

**Criterion 6: Maintainability**

The modular architecture enhances maintainability, as each microservice operates independently. This structure allows individual services to be updated, tested, and redeployed without affecting the entire application. Using ECR to manage Docker images simplifies the process of updating services, as new versions of container images can be pushed to ECR and automatically deployed to ECS tasks.

**Criterion 7: Monitoring and Logging**

CloudWatch is used to monitor application performance, track resource usage, and log errors. These logs provide valuable insights into application behavior, helping to quickly identify and resolve issues. CloudWatch alerts can be configured to notify the team of any critical issues, such as service downtime or high error rates, ensuring proactive monitoring and maintenance.

**5. Cost Estimate**

Using the AWS Pricing Calculator, the estimated monthly cost for running the Group36 application with 50 concurrent users is approximately as follows:

* **ECS with Fargate**: $210
  + This includes charges for CPU and memory usage for two microservices running with moderate compute requirements.
* **Application Load Balancer (ALB)**: $55
  + Covers load balancing for incoming traffic across ECS services.
* **Elastic Container Registry (ECR)**: $5
  + Storage for container images of the application services.
* **Secrets Manager**: $0.40
  + Storage and retrieval for securely managing credentials and access keys.
* **CloudWatch Monitoring**: $12
  + Logs and performance monitoring for ECS tasks and other AWS services.
* **Route 53 Domain Costs**: $1
  + Basic domain registration and DNS management for directing traffic to the application.

**Estimated Total Monthly Cost: $283.40**

This estimate reflects typical usage and operational costs for handling moderate traffic of 50 concurrent users.

**6. Scaling Up the Application**

To scale the Group36 application to support 10,000 concurrent users, several architectural enhancements are required:

**Microservices**

The current microservices design may benefit from further subdivision, especially within the video processing service. Additional microservices can be created to handle specific video processing tasks, distributing the load more effectively and minimizing latency.

**Compute**

ECS with Fargate can remain viable, but enhanced auto-scaling configurations would be necessary to accommodate large spikes in demand. Alternatively, high-performance EC2 instances could be used to optimize costs and improve resource utilization.

**Load Distribution**

At higher scales, a multi-tiered load balancing approach may be required. Introducing regional load balancers and using Amazon CloudFront as a content delivery network (CDN) for static content can alleviate pressure on the ALB, improve latency, and deliver faster response times globally.

**7. Securing the Application**

As the Group36 application will be commercially deployed, securing the cloud infrastructure and application from potential cyber threats is critical. The following security measures are recommended:

* **IAM Roles with Least Privilege**: Each ECS task and associated AWS resources are assigned specific IAM roles with the minimal permissions required to operate. This approach adheres to the principle of least privilege, which limits access to only necessary permissions, reducing the risk of unauthorized access. For instance, the ECS tasks can only retrieve images from ECR and access required AWS services, minimizing exposure.
* **Secrets Manager for Credential Management (Confidentiality)**: AWS Secrets Manager is utilized to securely store and retrieve sensitive credentials, such as the ECR repository credentials. This enhances confidentiality by ensuring that sensitive data remains encrypted and accessible only to authorized ECS tasks. By storing credentials securely, we minimize risks associated with hard-coded secrets in code, reducing the potential for data leaks.
* **Network Security (Defense-in-Depth)**: Security Groups and network configurations are designed to control inbound and outbound traffic. Only HTTP and HTTPS traffic from the ALB is allowed into the ECS tasks, following a defense-in-depth strategy to prevent unauthorized network access. This minimizes exposure to attacks like unauthorized access and limits communication to necessary services only.
* **CloudWatch for Monitoring and Alerting (Continuous Monitoring)**: AWS CloudWatch monitors ECS performance, network activity, and application health. By setting up alerts on suspicious activity or unexpected behavior, the team can quickly respond to potential security incidents. Continuous monitoring allows for proactive identification and mitigation of security threats.
* **SSL/TLS Encryption (Data Integrity and Confidentiality)**: SSL/TLS encryption is enforced on the ALB using a certificate from ACM. This ensures that all data transmitted between users and the application is encrypted, protecting it from interception or tampering. Implementing HTTPS on the ALB protects sensitive user data, fostering trust and maintaining data integrity.

**8. Sustainability**

Sustainability is a key consideration in cloud-based application design. The Group36 application incorporates various choices to support environmental sustainability across software, hardware, data center, and resource levels.

* **Software Optimization**: The application is containerized, using Docker images optimized for size and efficiency. This reduces the amount of computational power required, lowering energy consumption during runtime. Additionally, serverless components like ECS Fargate allow the application to dynamically allocate resources based on demand, minimizing idle resource usage and, consequently, energy waste.
* **Hardware Efficiency**: By leveraging AWS’s cloud infrastructure, the application benefits from high-efficiency hardware managed by AWS. This infrastructure uses cutting-edge technology designed to maximize performance per watt, ensuring that resources are efficiently utilized. Further optimization can be achieved by monitoring usage metrics and selecting instance types with energy-efficient profiles.
* **Data Center Sustainability**: AWS data centers are optimized for sustainability, with a commitment to achieving 100% renewable energy usage by 2025. Running the application on AWS’s infrastructure aligns with sustainable data center goals, as AWS employs advanced cooling, power efficiency practices, and renewable energy sources to reduce carbon emissions. This alignment supports sustainability at the data center level without requiring additional effort from the application team.
* **Resource Allocation and Auto-Scaling**: The application leverages ECS Fargate’s auto-scaling capabilities, which dynamically adjusts resource allocation based on demand. This reduces unnecessary energy consumption and hardware strain by ensuring that only the required resources are used at any given time. For future sustainability, the application could implement more granular auto-scaling policies to adjust workloads precisely based on metrics such as CPU and memory usage.
* **Future Sustainability Enhancements**: To further improve sustainability, the application could implement features like workload scheduling, which runs non-urgent tasks during off-peak times when data center loads are lower. This approach can leverage AWS’s efficient scheduling capabilities and reduce the environmental footprint by optimizing resource consumption patterns.

Through these efforts, the Group36 application contributes to sustainability by optimizing resource usage, aligning with AWS’s renewable energy initiatives, and adopting efficient operational practices. These choices not only reduce environmental impact but also improve cost efficiency, ensuring that the application aligns with sustainable practices in the cloud.

**Conclusion**

The Group36 application is designed for scalability, security, and efficiency within AWS’s cloud ecosystem. The architecture leverages AWS’s managed services to reduce complexity, ensuring that the application can scale dynamically while maintaining cost efficiency and sustainability. The design considerations, security measures, and sustainability practices reflect an application that is ready for enterprise-scale deployment with minimal operational overhead.