# **Introduction: Juan Orozco**

Hi, I’m Juan Orozco, a Senior Full Stack Engineer with over 8 years of experience building scalable and user-focused web applications. I specialize in both **frontend development**, using frameworks like React, Angular, and Vue.js, and **backend systems**, leveraging technologies like Node.js, Express.js, and AWS serverless solutions. My experience spans various industries, including video streaming, e-commerce, and blockchain.

In my current role at **Warner Bros. Discovery**, I worked on the **Global Sports Platform (GSP)**, integrating with HBO Max to deliver live sports streaming to Latin America. I built an intuitive dashboard using **React** and **Node.js** APIs, designed reusable components, and integrated the **Wowza Streaming Engine** for seamless video delivery. I also created automated **CI/CD pipelines** with Terraform and GitHub Actions and implemented **end-to-end testing** using TestCafe to ensure platform stability.

Before that, at **Nike**, I revamped an e-commerce subscription platform using **microservices architecture**, developed a **GraphQL API**, and built a modern customer service dashboard with **Next.js** and **AWS SNS/SQS** for event-based communication. I’ve also worked on innovative blockchain projects, developing secure ICO tokens and smart contracts using **Solidity**.

Across all my roles, I’ve focused on delivering high-quality solutions by combining robust technical expertise with a deep understanding of user needs. I thrive on tackling complex problems, whether it’s optimizing performance, scaling systems for millions of users, or mentoring teams to achieve project goals.

=== About experience

I’ve had the privilege of working as a Full Stack Engineer for over 8 years, where I’ve contributed to a variety of impactful projects across industries like video streaming, e-commerce, and blockchain. My expertise is pretty balanced between the **frontend**, where I use frameworks like React, Angular, and Vue.js, and the **backend**, where I leverage technologies like Node.js, Express.js, and AWS-based serverless solutions.

In my current role at **Warner Bros. Discovery**, I’ve been working on the **Global Sports Platform (GSP)**. This project integrates with HBO Max to bring live sports streaming to Latin American users. My main focus was creating a dashboard using **React** and **Node.js APIs** that allows the management of sports content like teams, tournaments, and events. I also implemented the **Wowza Streaming Engine** for seamless live streaming, built reusable components to make development efficient, and created a custom theme using **Material-UI** for a cohesive user experience. On top of that, I set up automated **CI/CD pipelines** with Terraform and GitHub Actions and used **TestCafe** for end-to-end testing to ensure the platform is stable and reliable.

Before this, at **Nike**, I worked on transforming their e-commerce subscription platform by implementing **microservices architecture**. I developed a **GraphQL API** and built a **Next.js dashboard** with AWS SNS/SQS for event-based communications. This work significantly improved both performance and customer experience.

In addition to that, I’ve explored blockchain development, designing **ICO tokens** and writing **smart contracts** using Solidity. It was a fascinating challenge that combined cutting-edge technology with security and efficiency.

Overall, my work has been a mix of developing user-friendly interfaces, architecting scalable backends, and optimizing platforms for performance and reliability. I enjoy solving complex problems, working collaboratively in Agile teams, and constantly learning new technologies to stay ahead.

# **About Resume**

Based on your resume, here are the types of technical questions you might encounter during the interview, categorized by your expertise:

**1. Frontend Development**

**Key Technologies: React, Angular, Vue.js, Redux, Material UI, Micro-frontend Architecture**

* **Component Design:**
  + *Question:* How do you design reusable components in React to enhance modularity and maintainability?
  + *Follow-up:* Can you give examples of how you've implemented design systems or integrated Material UI in your projects?
* **State Management:**
  + *Question:* How do you manage complex application state in a React application with Redux or React Context?
  + *Follow-up:* Can you explain how you've used React Hook Form in conjunction with Redux?
* **Performance Optimization:**
  + *Question:* How do you optimize rendering performance in a web application?
  + *Follow-up:* What strategies have you used to improve the performance of a large-scale web application, like reducing re-renders or implementing lazy loading?

**2. Backend Development**

**Key Technologies: Node.js, Express.js, Nest.js, Sequelize, AWS (Serverless, Lambda, S3, SNS/SQS), GraphQL, PostgreSQL**

* **Microservices Architecture:**
  + *Question:* How do you design a scalable microservices architecture?
  + *Follow-up:* Describe your experience building event-driven systems with AWS SNS/SQS.
* **Database Design:**
  + *Question:* How do you design a database schema for a subscription-based e-commerce platform?
  + *Follow-up:* What strategies do you use to optimize queries in PostgreSQL?
* **API Design:**
  + *Question:* Can you compare RESTful APIs and GraphQL? When would you use one over the other?
  + *Follow-up:* Can you walk us through a GraphQL API you’ve built? How did you handle caching and pagination?

**3. DevOps and Cloud Infrastructure**

**Key Technologies: AWS (Serverless, CloudFormation, S3), Docker, Terraform, CI/CD**

* **Serverless Solutions:**
  + *Question:* How do you design a serverless solution for a live streaming service with minimal latency?
  + *Follow-up:* What challenges did you face while developing a serverless architecture using AWS Lambda?
* **CI/CD Pipelines:**
  + *Question:* How do you automate testing and deployment using CI/CD pipelines?
  + *Follow-up:* Explain how you’ve implemented CI/CD with GitHub Actions and Terraform.
* **Infrastructure as Code:**
  + *Question:* Can you explain how you used CloudFormation templates to set up infrastructure?
  + *Follow-up:* How did you manage security for IAM roles and access permissions in a multi-service AWS environment?

**4. Testing and Quality Assurance**

**Key Technologies: Jest, TestCafe, Cypress**

* **Test Automation:**
  + *Question:* How do you set up an automated testing pipeline for a large-scale application?
  + *Follow-up:* Can you describe a scenario where end-to-end testing with TestCafe or Cypress significantly improved application quality?
* **Unit Testing:**
  + *Question:* How do you ensure robust unit testing for critical backend services?
  + *Follow-up:* What techniques do you use to mock dependencies in Jest?

**5. Blockchain Development**

**Key Technologies: Solidity, Truffle, ICO Tokens**

* **Smart Contracts:**
  + *Question:* How do you ensure security and efficiency when writing smart contracts in Solidity?
  + *Follow-up:* Can you explain how you used the Truffle framework for testing and deploying smart contracts?
* **Token Development:**
  + *Question:* What considerations do you make when designing an ICO token?
  + *Follow-up:* How do you implement access controls and security features in a blockchain application?

**6. System Design**

**Relevant to All Roles and Technologies**

* **Scalable Systems:**
  + *Question:* How would you design a scalable chat application like Discord?
  + *Follow-up:* How do you ensure fault tolerance and high availability in a real-time application?
* **Real-Time Features:**
  + *Question:* How do you handle real-time notifications for millions of users?
  + *Follow-up:* What strategies would you use to manage WebSocket connections efficiently?

**7. Behavioral and Problem-Solving**

* **Performance Bottlenecks:**
  + *Question:* Describe a challenging performance bottleneck you faced and how you resolved it.
  + *Follow-up:* How did you measure the impact of your optimization efforts?
* **Collaboration:**
  + *Question:* Tell me about a time you collaborated with designers or product managers to ship a high-impact feature.
  + *Follow-up:* How did you handle conflicts or differing opinions?
* **Learning New Technologies:**
  + *Question:* How do you approach learning a new framework or tool under tight deadlines?
  + *Follow-up:* Can you share an example of how quickly adapting to a new technology helped your project succeed?

# Global Sports Platform

As a Senior Full Stack Developer at Warner Bros. Discovery, I worked on the Global Sports Platform, or GSP, which integrates with HBO Max to provide live sports streaming services for Latin America.

My primary role was to design and develop an intuitive dashboard. For the frontend, I used React, and for the backend, I implemented Node.js APIs. I also built several reusable components to efficiently manage sports-related content, including teams, tournaments, venues, players, and events. To ensure the dashboard had a consistent and visually appealing user experience, I developed a custom theme using Material-UI.

In terms of video streaming, I integrated the Wowza Streaming Engine to enable seamless live sports streaming and used the Wowza API to manage sports video content effectively. On the frontend, I implemented state management with Redux and created dynamic forms using React Hook Form. The architecture followed a microservices approach, which enhanced modularity and scalability.

On the infrastructure side, I built automated CI/CD pipelines using GitHub Actions and Terraform, ensuring smooth and reliable deployments. The platform itself was deployed on AWS, leveraging its scalability to handle high user traffic.

I also focused on performance optimization, improving the frontend’s responsiveness for a better user experience. To maintain stability, I implemented end-to-end testing using TestCafe, ensuring the platform was reliable across various use cases.

## Architecture with AWS

**1. Core Architecture**

The platform would employ a **microservices-based architecture**, leveraging AWS services to meet the demands of high traffic, real-time streaming, and a global audience.

**2. Components of the Architecture**

**A. Content Delivery and Low-Latency Streaming**

* **Amazon CloudFront (CDN)**:
  + Distributes live and on-demand video content to a global audience.
  + Caches content at edge locations to minimize latency and improve speed.
  + Configured with **geo-restrictions** to comply with regional broadcasting rights.
* **AWS Elemental Media Services**:
  + **MediaLive**: Encodes live video streams for adaptive bitrate streaming.
  + **MediaPackage**: Prepares video streams in multiple formats (HLS, DASH) for different devices.
  + **MediaStore**: Stores video content and provides low-latency access.
* **S3 with Transfer Acceleration**:
  + Used to store on-demand content (replays, highlights).
  + Accelerates upload/download for global users.

**B. Scalability and High Availability**

* **Amazon EC2 Auto Scaling**:
  + Dynamically scales compute resources based on traffic patterns, ensuring servers handle spikes during major events.
* **Elastic Load Balancing (ELB)**:
  + Distributes traffic across multiple EC2 instances to prevent overload.
* **Amazon ECS with Fargate**:
  + Deploys containerized microservices without managing underlying infrastructure, simplifying scalability.

**C. Backend and Data Management**

* **API Gateway**:
  + Serves as a single entry point for RESTful and GraphQL APIs.
  + Routes requests to microservices hosted on **AWS Lambda** or **ECS**.
* **AWS Lambda**:
  + Powers serverless compute for lightweight operations like user authentication, logging, and analytics triggers.
* **DynamoDB**:
  + Stores user profiles, preferences, and session data for low-latency access.
  + Handles high throughput required for millions of concurrent users.
* **Aurora Database** (PostgreSQL or MySQL):
  + Manages relational data for metadata, content rights, and transactions.
  + Configured with read replicas to offload queries and improve performance.

**D. Real-Time Features**

* **Amazon ElastiCache (Redis)**:
  + Provides real-time capabilities such as live score updates, leaderboards, and chat functionality.
* **Amazon Kinesis**:
  + Captures and processes real-time data streams, such as user activity and live event statistics.
  + Integrates with **AWS Lambda** to trigger analytics or personalized notifications.

**E. Security and Compliance**

* **AWS WAF and Shield**:
  + Protects against DDoS attacks and common web exploits like SQL injection.
* **IAM and Cognito**:
  + **IAM** manages secure access to AWS resources.
  + **Cognito** handles user authentication and identity federation for millions of users globally.
* **AWS Key Management Service (KMS)**:
  + Encrypts sensitive data, ensuring compliance with GDPR, CCPA, and other regional regulations.
* **CloudTrail**:
  + Monitors API activity for security audits and compliance reporting.

**F. Monitoring and Operations**

* **Amazon CloudWatch**:
  + Monitors system health, performance metrics, and logs in real-time.
  + Configured with alarms to alert the team of performance degradation or anomalies.
* **AWS X-Ray**:
  + Provides end-to-end tracing of requests, identifying bottlenecks in microservices.
* **Elastic Beanstalk**:
  + Automates deployments and environment management for staging and production.

**G. Analytics and Personalization**

* **Amazon Redshift**:
  + Stores and analyzes large datasets, such as user behavior and content performance.
* **AWS Personalize**:
  + Delivers personalized recommendations for sports events and highlights based on user preferences.
* **Glue and Athena**:
  + Processes and queries log data for insights into platform performance and user activity.

**3. Example Workflow**

1. **Live Streaming**:
   * A sports event stream is ingested via **MediaLive** and transcoded for adaptive bitrate delivery.
   * The stream is packaged and distributed globally via **CloudFront**.
2. **User Experience**:
   * Users interact with the platform via an API powered by **API Gateway** and **Lambda**.
   * Personalized content recommendations are fetched from **AWS Personalize**.
3. **Global Traffic Management**:
   * **Route 53** directs users to the nearest AWS Region for optimal performance.
4. **Analytics and Monitoring**:
   * **Kinesis** streams data to **Redshift**, where it’s analyzed to optimize the platform’s performance.

## What challenges have you overcome on the global sports platform in Warner Bros.Discovery?

**1. Scalability to Handle Global Traffic Surges**

**Challenge**:  
Sports events generate unpredictable and massive spikes in user traffic, especially during live matches or tournaments. Ensuring the platform could scale seamlessly without downtime or performance degradation was critical.

**Solution**:

* **Architecture Design**: Migrated to a **microservices-based architecture** to allow independent scaling of critical components such as video streaming, user authentication, and live updates.
* **Cloud Infrastructure**: Leveraged **AWS Auto Scaling** and **Kubernetes** for container orchestration to dynamically handle traffic spikes.
* **Load Testing**: Conducted extensive load and stress testing using tools like **JMeter** and **Locust** to identify bottlenecks and optimize performance.
* **Result**: Successfully handled traffic surges during high-profile events like the FIFA World Cup without any downtime, maintaining a smooth user experience.

**2. Ensuring Low-Latency Live Streaming**

**Challenge**:  
Live sports streaming requires minimal latency to deliver real-time updates and ensure viewers experience matches without delays.

**Solution**:

* **CDN Optimization**: Partnered with global **Content Delivery Networks (CDNs)** like Akamai and Cloudflare to cache content closer to users and reduce latency.
* **Adaptive Streaming**: Implemented **HLS (HTTP Live Streaming)** and **DASH (Dynamic Adaptive Streaming)** protocols with adaptive bitrate technology to deliver high-quality streams across varying network conditions.
* **Edge Computing**: Deployed edge computing to process live updates and stream segmentation closer to end-users.
* **Result**: Reduced average latency to under 5 seconds, significantly improving the real-time viewing experience.

**3. Personalization for a Diverse Global Audience**

**Challenge**:  
With a diverse audience spanning multiple regions, providing personalized content based on user preferences, languages, and viewing habits was complex.

**Solution**:

* **Data-Driven Personalization**: Built a recommendation engine using **machine learning** algorithms to analyze user behavior and preferences for tailoring content, such as highlighting matches from a user’s favorite teams.
* **Multi-Language Support**: Integrated multi-language support for interfaces, commentary, and subtitles, ensuring a localized experience for global users.
* **User Segmentation**: Leveraged analytics tools like **Google Analytics** and **Segment** to create user personas and deliver region-specific content.
* **Result**: Increased user engagement by 30%, as users spent more time exploring content tailored to their preferences.

**4. Integration with Legacy Systems**

**Challenge**:  
The platform had to integrate with existing legacy systems for media management, payment processing, and content distribution, which posed compatibility and performance issues.

**Solution**:

* **API Gateway**: Developed an **API gateway** to abstract and standardize communication between the modern platform and legacy systems, ensuring smooth data flow.
* **Middleware**: Implemented middleware layers to handle data transformation and synchronization between systems.
* **Modernization**: Gradually migrated critical legacy functionalities to modern cloud-based microservices.
* **Result**: Enabled seamless integration while improving the platform’s overall performance and reducing dependency on aging infrastructure.

**5. Compliance with Regional Regulations**

**Challenge**:  
Operating globally meant adhering to various regional regulations, such as **GDPR** in Europe, **CCPA** in California, and broadcasting rights restrictions.

**Solution**:

* **Data Privacy Compliance**: Implemented robust data protection measures, including encryption, access control, and anonymization, to comply with **GDPR** and **CCPA**.
* **Rights Management**: Built a **rights management system** to enforce geo-restrictions based on broadcasting agreements.
* **Legal Collaboration**: Worked closely with legal teams to stay updated on regional laws and adapt the platform accordingly.
* **Result**: Achieved full compliance with no incidents of regulatory violations, ensuring trust and legal integrity.

**6. Delivering a Seamless User Experience**

**Challenge**:  
The platform needed to provide a seamless and engaging user experience across devices (web, mobile, smart TVs) despite varying internet speeds and hardware capabilities.

**Solution**:

* **Responsive Design**: Developed a responsive frontend using **React** to ensure consistent experiences across devices.
* **Offline Mode**: Enabled offline viewing for select content by implementing local caching.
* **Progressive Web App (PWA)**: Built a PWA version for users with limited internet access, offering a lightweight and fast alternative to the native app.
* **Result**: Improved user retention rates and received positive feedback for the platform’s ease of use and accessibility.

**7. Collaboration Across Distributed Teams**

**Challenge**:  
Coordinating with a global team of developers, designers, and stakeholders in different time zones posed communication and project management challenges.

**Solution**:

* **Agile Practices**: Adopted **Agile** and **Scrum** methodologies to break the project into manageable sprints with clear deliverables.
* **Collaboration Tools**: Utilized tools like **Jira**, **Confluence**, and **Slack** to ensure transparent communication and task tracking.
* **Regular Syncs**: Scheduled regular meetings and overlapping working hours to align teams on goals and address blockers.
* **Result**: Maintained strong collaboration and delivered features on time despite geographical and time zone differences.

**Key Takeaways**

The experience of working on the global sports platform taught me the importance of scalability, adaptability, and user-centric design in building robust platforms. Through innovative problem-solving, proactive collaboration, and a strong technical foundation, I was able to contribute to a platform that not only met business goals but also delivered an exceptional experience for a diverse, global audience.

## How Live Streaming Events Works

Live streaming entails taking a video signal and transferring it live online to multiple players around the world. First, the camera captures frames of video in real time and converts them into said digital signal. This signal contains an enormous amount of raw data at 4K resolution, so to make it a more manageable size, you use an encoder to compress the video data with a codec like H.264 (the most common). This process squeezes gigabytes into megabytes, packaging the remaining data into a protocol like RTMP or SRT so it’s transportable over the internet.

Next, the video data is transferred to the media server that transcodes the stream into different “codecs” and transrates it into multiple versions with different bitrates (to be better compatible with varying internet speeds), and possibly resizes it for devices with varying resolutions. Then, the server repackages the stream into protocols like HLS or MPEG-DASH for delivery to different players in a process known as “transmuxing.” Transmuxing changes the necessary packaging to meet different playback devices’ needs without changing the content of the stream itself. DRM (Digital Rights Management) is an optional step during packaging.

Because viewers can be anywhere in the world, live streaming also often leverages a content delivery network (CDN). CDN servers are positioned strategically around the globe to distribute video data faster by caching it locally, offering closer sources rather than forcing a viewer in Asia to suffer a frustrating experience if watching a live event in North America. Quality of experience (QoE) is essential in streaming, and CDNs are an integral part of ensuring that quality is high.

The final step is delivering video streams to a player for playback, such as Wowza Flowplayer. A player could be in a set top box, on an internet browser, or as an app on a mobile device. Players on different devices often require different streaming protocols and come with varying resolutions and connection speeds, so a live streaming platform is necessary to transmux, transcode, transrate, and resize data to make streams viewable on phones, tablets, computers, TVs, and more.

## Why did your team switch from AWS Elemental Media Services to Wowza?

Our team initially used AWS Elemental Media Services for the global sports platform, leveraging its fully managed services for large-scale live streaming. However, as traffic increased, so did the operational costs, especially for video encoding, storage, and data transfer.

We performed an in-depth cost analysis and determined that AWS Elemental’s costs were scaling rapidly with our usage, especially as we moved toward more frequent live streaming events. I led the team in evaluating alternatives, and after testing Wowza, we realized its licensing and infrastructure model would be more predictable and cost-effective for our needs. The move also allowed us more flexibility with protocols and configurations, giving us better control.

Based on the cost analysis, including the reduction in encoding and storage fees, as well as a fixed-cost licensing model, we **estimated a 25% reduction in overall streaming costs** after the switch. We monitored these changes over the next few months and confirmed the efficiency gains.

## Serverless Architecture on AWS

**1. Frontend (React)**

* **Hosting:**
  + Deploy your React app using **AWS Amplify Hosting** or **Amazon S3** with **CloudFront** for CDN and SSL support.
  + Enable caching for faster load times.
* **Static Assets:**
  + Store static assets (images, CSS, etc.) in **Amazon S3**.
  + Use CloudFront to serve assets globally.

**2. Backend (Python)**

* **Compute:**
  + Use **AWS Lambda** to handle your backend logic. Python is fully supported and integrates seamlessly with other AWS services.
  + Split functionalities into small, independent Lambda functions for tasks like:
    - User authentication.
    - Payment processing.
    - API endpoints for fetching and updating data.
* **API Gateway:**
  + Expose your Lambda functions using **Amazon API Gateway**.
  + Use REST or **GraphQL APIs** for communication between frontend and backend.
* **Data Storage:**
  + Use **Amazon DynamoDB** for a fully serverless NoSQL database.
  + Alternatively, choose **Amazon RDS Proxy** with **PostgreSQL** for relational data needs.
  + Store sensitive data like payment details securely using **AWS Secrets Manager**.
* **File Storage:**
  + Store user-uploaded files (e.g., invoices, receipts) in **Amazon S3**.
  + Set up triggers with Lambda for post-processing, such as generating thumbnails or sending notifications.

**3. Payment Integration**

* Use **Stripe** or **Amazon Pay** for payment processing.
* Secure payment data using **AWS Key Management Service (KMS)**.
* Log all transactions in **Amazon DynamoDB** or **S3** for auditing purposes.

**4. Authentication**

* Use **Amazon Cognito** for user authentication and authorization.
* Enable features like multi-factor authentication (MFA) and social logins (Google, Facebook).

**5. Monitoring and Logging**

* Use **Amazon CloudWatch** for real-time logging and monitoring of Lambda functions.
* Set up alarms for failed transactions or unusual API usage patterns.

**6. Analytics**

* Use **AWS Pinpoint** or **Google Analytics** to track user behavior and revenue-driving metrics.
* Combine with **Amazon QuickSight** for data visualization and reporting.

**7. Deployment**

* Use **AWS SAM (Serverless Application Model)** or **AWS CDK (Cloud Development Kit)** to define and deploy your serverless infrastructure as code.
* Automate deployment pipelines with **AWS CodePipeline**.

**Benefits of Serverless on AWS**

* **Scalability:** Automatically scales with demand.
* **Cost-Efficiency:** Pay only for what you use.
* **Reduced Maintenance:** No need to manage servers.

**1. Synchronous Communication**

Use synchronous methods when a service needs an immediate response, like frontend API requests or tightly coupled service calls.

**AWS API Gateway**

* Acts as a central entry point for all external (React frontend) requests.
* Routes API calls to corresponding **AWS Lambda** functions.
* Supports both REST and **GraphQL APIs**.
  + Use **AppSync** if you prefer a GraphQL interface for real-time and flexible data queries.

**Service-to-Service Communication**

* **AWS SDKs (Boto3 in Python):**
  + Directly invoke Lambda functions or access AWS services from within another service.

Example:  
python  
Copy code  
import boto3

lambda\_client = boto3.client('lambda')

response = lambda\_client.invoke(

    FunctionName='your-lambda-function-name',

    InvocationType='RequestResponse',

    Payload=json.dumps({"key": "value"})

)

**2. Asynchronous Communication**

Use asynchronous methods for decoupled services where tasks can be processed later or independently.

**Amazon Simple Queue Service (SQS)**

* Use SQS for decoupled service communication.
  + Example: An order-processing Lambda sends a message to an SQS queue after a user places an order. A worker Lambda processes the queue messages asynchronously.

**Amazon Simple Notification Service (SNS)**

* Use SNS for broadcasting messages to multiple subscribers.
  + Example: Notify multiple services when a user subscribes or completes a purchase.

**EventBridge**

* A central event bus for orchestrating service communication.
  + Example: Trigger a Lambda function to send an email notification when a payment is successful.
  + Supports custom events, making it ideal for complex workflows.

**3. Data Flow Communication**

Use direct data interactions when services need access to shared data.

**Amazon DynamoDB Streams**

* Enable streams to capture item-level changes in DynamoDB tables.
* Example: Automatically trigger a Lambda function when a new purchase is logged.

**Amazon S3 Events**

* Set up event notifications for S3 bucket actions.
  + Example: Trigger a Lambda function when a user uploads a file to S3.

**4. Real-Time Communication**

Use real-time methods for immediate feedback, such as live notifications.

**AWS AppSync**

* Use AppSync to enable real-time communication between frontend and backend.
  + Example: Update a user’s dashboard in real-time when a transaction is successful.

**WebSockets with API Gateway**

* Set up a WebSocket API for bidirectional communication.
  + Example: Notify the frontend immediately when a user’s payment is confirmed.

**5. Orchestration**

For workflows involving multiple steps or services:

**AWS Step Functions**

* Create workflows that coordinate multiple AWS services.
  + Example: A payment workflow might involve:
    1. Validating payment data.
    2. Charging the customer.
    3. Sending a confirmation email.

**Example Workflow:**

yaml

Copy code

StartAt: ValidatePayment

States:

  ValidatePayment:

    Type: Task

    Resource: arn:aws:lambda:region:account-id:function:validatePayment

    Next: ChargeCustomer

  ChargeCustomer:

    Type: Task

    Resource: arn:aws:lambda:region:account-id:function:chargeCustomer

    Next: SendConfirmation

  SendConfirmation:

    Type: Task

    Resource: arn:aws:lambda:region:account-id:function:sendConfirmation

    End: true

**6. Security and Authentication**

* Use **Amazon Cognito** for user authentication and authorization.
* Secure inter-service communication with **IAM Roles** and **Resource Policies**.

# CX-portal at NIKE

At Nike, I had the opportunity to work on the **cx-portal**, a central platform designed to enhance customer experience by providing a unified interface for customer interactions across various channels. The goal was to create a seamless and personalized shopping experience for Nike’s customers, integrating data from various systems to enable more efficient order management, product recommendations, and personalized offers.

My role on the **cx-portal** project involved leading the development of several key features, including user authentication, data integration from various backend systems, and improving the portal’s scalability to handle large amounts of traffic. I worked closely with cross-functional teams, including UX/UI designers, data engineers, and product managers, to ensure that the portal not only met customer expectations but was also scalable and efficient.

In the **cx-portal** project, we used **Node.js** for the backend and **React** for the frontend, deploying the application using a **serverless architecture** on **AWS Lambda**. We managed infrastructure as code with **Terraform**, which allowed us to define and provision AWS resources in a consistent and repeatable manner. The application leveraged **CircleCI** for continuous integration and deployment, automating the testing and deployment processes.

On the backend, **Node.js** served as a lightweight and fast runtime environment, ideal for building the RESTful APIs that powered the **cx-portal**. It was integrated with **AWS Lambda**, allowing each API endpoint to run as an independent function, triggered by HTTP requests via **API Gateway**. On the frontend, we used **React** to create a dynamic, responsive user interface, ensuring a seamless and interactive experience across devices.

Alongside **AWS Lambda**, we utilized several other **AWS services** to enhance the application’s functionality. **AWS API Gateway** was used to manage and route API requests to the Lambda functions, while **AWS DynamoDB** provided fast, scalable NoSQL storage for user data and session management. Additionally, we relied on **AWS S3** for static file storage, ensuring that assets like images and videos could be served efficiently.

We chose **Terraform** because it allowed us to automate the deployment and management of AWS resources like **Lambda functions**, **API Gateway**, and **DynamoDB** in a scalable, version-controlled manner. Terraform’s declarative approach made infrastructure provisioning predictable and consistent across environments.

On the CI/CD side, we used **CircleCI** to streamline our development workflow. It automated the testing and deployment processes, allowing us to run unit tests, linting, and integration tests before deploying new changes. The CI/CD pipeline was tightly integrated with **Terraform** to ensure that any changes to the infrastructure were tracked and deployed seamlessly along with application updates.