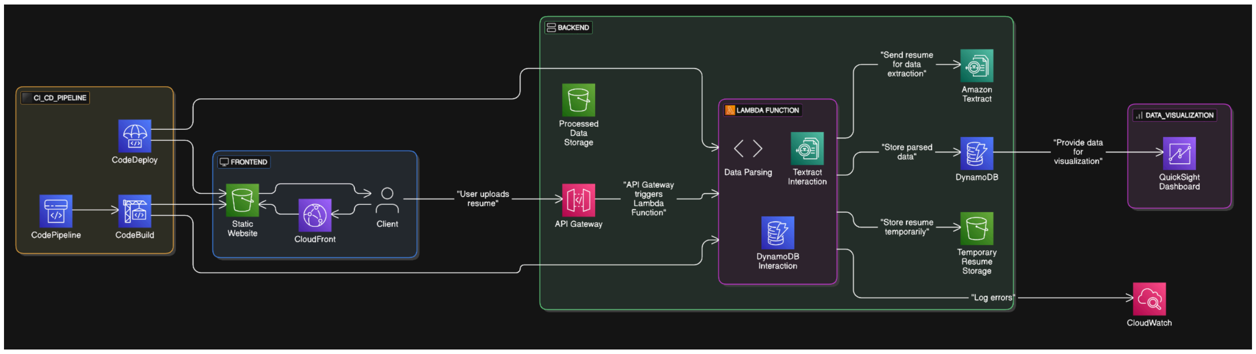
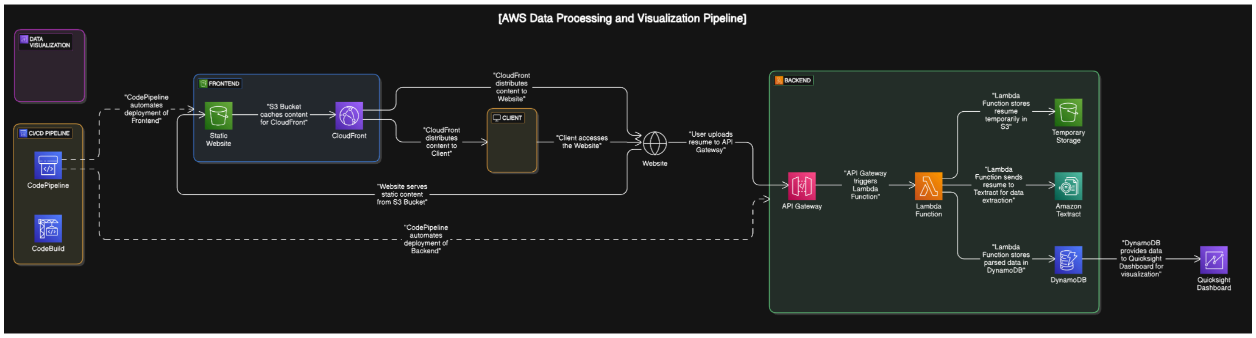
Comparing the two diagrams, the **first diagram** is superior for its clarity and better representation of the system's modularity. The second diagram, while showing more steps, is visually cluttered and harder to follow. The first diagram effectively breaks down the system into manageable parts.




[AWS Data Processing and Visualization Pipeline]
DATA VISUALIZATION
CI/CD PIPELINE
CodePipeline
CodeBuild
"CodePipeline automates deployment of Frontend"
FRONTEND
Static Website
"S3 Bucket caches content for CloudFront"
CloudFront
"CloudFront distributes content to Website"
"CloudFront distributes content to Client"
CLIENT
"Client accesses the Website"
Website
"Website serves static content from S3 Bucket"
"User uploads resume to API Gateway"
"CodePipeline automates deployment of Backend"
BACKEND
API Gateway
"API Gateway triggers Lambda Function"
Lambda Function
"Lambda Function stores resume temporarily in S3"
Temporary Storage
"Lambda Function sends resume to Textract for data extraction"
Amazon Textract
"Lambda Function stores parsed data in DynamoDB"
DynamoDB
"DynamoDB provides data to Quicksight Dashboard for visualization"
Quicksight Dashboard

Here's a script for Yanga to explain the first diagram as the final architecture of his project:

(Slide 1: Title Slide – "Serverless Resume Parser & Portfolio: Final Architecture")

"Good morning/afternoon, everyone. I'm Yanga Mgudwa, and I'm pleased to present the final architecture of my serverless resume parser project. This diagram illustrates the complete system design, emphasizing its modularity, scalability, and security."

(Slide 2: Full Diagram – Overview)

"This diagram clearly shows the system's three main components: the frontend (the user interface), the backend (where the resume processing occurs), and the data visualization dashboard. Each section is clearly delineated, illustrating the modular design. This architecture takes advantage of serverless components offered by AWS, making it highly scalable and cost-effective."

(Slide 3: Frontend – Pointing to Frontend components.)

"Let's begin with the frontend. *(Points to the Client, Website, CloudFront, and Static Website boxes)* The user interacts with a visually appealing website built using HTML, CSS, and

JavaScript. The website's static content is efficiently hosted on Amazon S3 (Simple Storage Service). To ensure fast loading times for users globally, Amazon CloudFront distributes the content via a CDN (Content Delivery Network). Automated deployments are handled by CodePipeline and CodeBuild. CodePipeline monitors the code repository (GitHub) for updates. When changes are detected, CodePipeline orchestrates the build process (using CodeBuild), which compiles and builds the frontend code. Finally, CodePipeline automatically deploys the updated website files to S3."

(Slide 4: Backend – Pointing to Backend components.)

"Now, let's look at the backend—the core of the resume processing. *(Points to the API Gateway, Lambda Function, Amazon Textract, DynamoDB boxes)* When a user uploads a resume through the website, the request goes to the API Gateway, which serves as the secure entry point. API Gateway triggers the Lambda function, a serverless compute service. This function first stores the resume temporarily in an S3 bucket. Next, it sends the resume to Amazon Textract for data extraction. After Textract processes the resume, the Lambda function parses the extracted data—extracting key information like skills and experience—and stores this structured data in DynamoDB."

(Slide 5: Lambda Function Internal Steps - Zooming in on the Lambda Function box)

"Let's look inside the Lambda function more closely. *(Points to the internal blocks within the Lambda Function box)* The Lambda function performs the following steps: first it receives the resume from API Gateway; it stores the resume in temporary storage (S3); then, it sends the resume to Textract for data extraction; it performs data parsing (cleaning, formatting, extracting key information); and finally it stores the processed data in DynamoDB. Crucially, logging is done at each stage, allowing for detailed debugging and error handling."

(Slide 6: Data Visualization – Pointing to QuickSight Dashboard and CloudWatch.)

"Finally, the parsed data in DynamoDB is used to create a compelling and informative QuickSight dashboard. *(Points to the DynamoDB and QuickSight Dashboard boxes)* This dashboard visualizes key metrics, providing recruiters and potential clients with clear insights into Yanga's skills and experience. The CloudWatch logs *(Points to CloudWatch box)* provide valuable data for monitoring the health and performance of the entire system."

(Slide 7: Conclusion – Showing the whole diagram.)

"In summary, this architecture showcases a robust, scalable, and secure system. The modular design, use of serverless components, and automated deployment process all contribute to its efficiency and maintainability. Thank you for your time. I'm happy to answer any questions."

This script provides a clear and concise explanation of the architecture. Remember to practice your presentation and be prepared to answer questions about specific technologies and design choices. Consider including a live demo if possible.