### Slide 1: Title Slide (1 minute)

* **Title**: IBTool: Development and Cloud-Native Migration of a Complaint Management System for GE Healthcare
* **Your Name**: Sana Alinia
* **Date**: [Defense Date]
* **Institution**: L’École La Passerelle des Métiers du Numérique (La PMN)
* **Advisors and Mentors**: Magalie Chatellard, Magali Wissocq

### Slide 2: Agenda (1 minute)

* **Agenda**:
  1. Introduction and Context
  2. Problem Statement
  3. Methodologies to Solve the Problem
  4. Contribution 1: Development of IBTool
  5. Contribution 2: Migration of IBTool to Cloud
  6. CI/CD Pipeline Implementation
  7. Security and Performance Enhancements
  8. Results and Benefits
  9. Challenges and Lessons Learned
  + Future Improvements and Conclusion

### Slide 3: Introduction and Context (1 minute)

* **Context**:
  + GE Healthcare is a global leader in medical technology and solutions.
  + A key challenge is efficiently managing customer complaints, ensuring product quality and client satisfaction.
  + IBTool was developed to centralize and streamline complaint management, then later migrated to the cloud for better scalability and performance.
* **Key Focus of the Thesis**:
  + **Contribution 1**: The development of IBTool.
  + **Contribution 2**: Migration to the cloud using AWS to enhance scalability, performance, and security.
* **Placeholder for Image**: [Context of GE Healthcare or IBTool Overview Image]

### Slide 4: Project Overview (1 minute)

* **IBTool**:
  + A web-based application developed to manage and track GE Healthcare customer complaints.
  + Initially hosted on-premises but later migrated to a cloud-native architecture.
* **Thesis Contributions**:
  + **Development of IBTool**: Solving the problem of fragmented data and inefficient workflows.
  + **Cloud Migration**: Addressing scalability, cost efficiency, and performance challenges by leveraging AWS services.
* **Placeholder for Image**: [Project Overview Image - IBTool Workflow or Application Screenshot]

### Slide 5: Problem Statement (2 minutes)

* **Challenges in Existing System**:
  + **Fragmented Data**: Customer complaints spread across multiple systems (TWD, PQM, SFDC, ClearQuest), leading to inefficiencies.
  + **Manual Processes**: High error rates and inefficiencies due to the manual management of complaints.
  + **Scalability and Performance Issues**: The on-premises infrastructure couldn’t handle variable loads efficiently, leading to slowdowns and downtime.
  + **Limited Visibility**: Difficulty in tracking the status and severity of complaints in real-time.
* **Need for a Solution**:
  + The need to create a unified system to streamline complaint management, automate processes, and scale efficiently with growing data.
* **Placeholder for Image**: [Problem Overview Image - Illustration of Fragmented Data/Systems]

### Slide 6: Problem Details (1 minute)

* **Why It Matters**:
  + Inefficient complaint management can lead to delayed product improvements and potential compliance risks in the healthcare sector.
  + Improving the efficiency and visibility of complaint management directly impacts product quality, regulatory compliance, and customer satisfaction.
* **Core Issues to Address**:
  + **Data Fragmentation**: Consolidate data into a single, unified system.
  + **Manual Workflows**: Automate processes to reduce errors and improve response times.
  + **Scalability**: Implement a system that can scale dynamically with demand.
* **Placeholder for Image**: [Data Fragmentation Diagram or Complaints Management Flowchart]

### Slide 7: Methodology Overview (1 minute)

* **Two Key Methodologies**:
  1. **Development of IBTool**: Focused on solving the problem of inefficient complaint management by creating a centralized, web-based platform.
  2. **Cloud Migration**: Aimed at addressing scalability, performance, and operational costs by migrating IBTool to a cloud-native architecture using AWS.
* **Agile Development**: Both phases followed an agile approach with iterative development, regular feedback loops, and ongoing testing.
* **Placeholder for Image**: [Agile Development Cycle Image or Methodology Overview Diagram]

### Slide 8: Agile Development Process (2 minutes)

* **Agile Methodology**:
  + **Iterative Development**: Incremental updates based on feedback from stakeholders at GE Healthcare.
  + **Key Phases**:
    1. **Planning**: Identification of core features and prioritization of development tasks.
    2. **Development Sprints**: Iterative coding cycles focusing on specific features (e.g., user management, complaint tracking).
    3. **Testing**: Continuous integration and testing of new features to ensure stability and functionality.
    4. **Feedback Loop**: Regular reviews and adjustments based on user feedback.
* **Outcome**:
  + Continuous delivery of incremental improvements, ensuring that IBTool evolved to meet the exact needs of GE Healthcare over time.
* **Placeholder for Image**: [Agile Process Diagram - Sprint Cycle Image]

### Slide 9: Cloud Migration Methodology (1 minute)

* **Cloud Migration Approach**:
  + **Lift and Shift Strategy**: Migrating the core functionalities of IBTool from on-premises infrastructure to AWS without major changes in application code initially.
  + **Post-Migration Optimization**: Leveraging cloud-native features like auto-scaling and managed services to optimize performance and costs.
  + **Key Steps**:
    1. **Assessment**: Identifying which components would benefit most from cloud migration.
    2. **Containerization**: Using Docker to containerize the application for easier deployment and scaling.
    3. **Deployment on AWS ECS**: Migrating containers to AWS Elastic Container Service (ECS) for orchestration and scaling.
* **Placeholder for Image**: [Cloud Migration Steps Diagram or Lift and Shift Overview]

### Slide 10: Infrastructure as Code with Terraform (2 minutes)

* **Terraform for IaC**:
  + **What is IaC?**: Infrastructure as Code allows for automated, consistent deployments of infrastructure using declarative configuration files.
  + **Why Terraform?**:
    - Cross-cloud compatibility.
    - Version control and rollback for infrastructure changes.
    - Efficient, repeatable provisioning of AWS services.
* **Key Elements in the Terraform Setup**:
  + **EC2 Instances**: Automatically provision and configure virtual machines.
  + **ECS Cluster**: Set up and scale Docker containers on AWS.
  + **Networking**: Create and manage security groups, VPCs, and load balancers.
* **Benefits**:
  + Faster deployment times.
  + Consistency across environments (development, testing, production).
  + Improved collaboration via version-controlled infrastructure.
* **Placeholder for Image**: [Terraform Diagram - IaC Flowchart]

### Slide 11: Overview of IBTool Architecture (2 minutes)

* **Initial Architecture**:
  + **On-Premises Setup**: IBTool initially hosted on local servers using Docker containers for frontend, backend, and database services.
  + **Monolithic Approach**: The application was built as a monolithic system, with all services running together.
  + **Technologies Used**:
    - **Frontend**: HTML, CSS, JavaScript (with DevExtreme library for UI components).
    - **Backend**: Node.js and Express.js to handle API requests.
    - **Database**: MongoDB for storing customer complaints.
* **Placeholder for Image**: [Figure 1 - On-Premises Architecture Diagram]

### Slide 12: Key Features of IBTool (1 minute)

* **Core Functionalities**:
  + **Centralized Complaint Management**: All customer complaints are tracked in one system.
  + **CRUD Operations**: Ability to create, read, update, and delete complaints efficiently.
  + **Automated Workflows**: Notifications and workflows triggered based on complaint status and severity.
* **Placeholder for Image**: [Figure 2 - IBTool User Interface Screenshot or Workflow Diagram]

### Slide 13: Technical Stack and Components (2 minutes)

* **Frontend**:
  + **Technologies**: HTML, CSS, JavaScript, and DevExtreme for building interactive UI components.
  + **Improvement**: Later migrated to React for better performance, maintainability, and dynamic UI.
* **Backend**:
  + **Node.js and Express.js**: REST API for handling client requests, processing data, and communicating with the MongoDB database.
  + **Improvements**: Enhanced data processing using middleware and caching (Redis) for faster responses.
* **Database**:
  + **MongoDB**: Document-based NoSQL database for flexible and scalable data storage.
  + **Improvements**: Indexing strategies for faster query performance.
* **Placeholder for Image**: [Figure 3 - Technical Stack Diagram or Component Architecture Diagram]

### Slide 14: Motivation for Cloud Migration (2 minutes)

* **Challenges with On-Premises Setup**:
  + **Scalability Issues**: The on-premises infrastructure couldn’t easily handle sudden spikes in complaint volumes.
  + **High Operational Costs**: Fixed infrastructure costs regardless of usage, leading to inefficiencies.
  + **Maintenance Overhead**: Manual monitoring and scaling required significant resources.
* **Why Cloud Migration?**:
  + **Dynamic Scaling**: Ability to scale resources up or down automatically based on demand.
  + **Cost Efficiency**: Pay-per-use pricing model reduces waste and optimizes spending.
  + **Resilience and Availability**: Improved fault tolerance and uptime guarantees using cloud-native services.
* **Placeholder for Image**: [Figure 6 - Cloud Migration Motivation Diagram]

### Slide 15: Cloud-Native Architecture Overview (2 minutes)

* **New Cloud-Native Architecture**:
  + **Frontend**: Hosted on S3 with CloudFront for faster content delivery and global distribution.
  + **Backend**: Containerized services running on AWS ECS, orchestrated and automatically scaled based on load.
  + **Database**: MongoDB migrated to a managed service, reducing the overhead of database maintenance and ensuring availability.
  + **Load Balancer**: Application Load Balancer (ALB) to distribute traffic evenly across the backend services.
* **Placeholder for Image**: [Figure 6 - Cloud-Native Architecture Diagram]

### Slide 16: AWS Services Used (1 minute)

* **Key AWS Services**:
  + **Amazon S3**: Static file storage and hosting for the frontend.
  + **Amazon ECS**: Orchestration of Docker containers for the backend.
  + **Amazon RDS**: Managed database service to host MongoDB (or DynamoDB, if applicable).
  + **AWS ALB**: Application Load Balancer for traffic distribution and SSL termination.
  + **AWS IAM**: For managing access control and security roles.
* **Placeholder for Image**: [AWS Architecture Diagram Highlighting Key Services]

### Slide 17: CI/CD Pipeline Overview (1 minute)

* **What is CI/CD?**:
  + **Continuous Integration (CI)**: Automating the process of testing and building the application after every code change.
  + **Continuous Deployment (CD)**: Automating the deployment of the application to production after successful testing and builds.
* **Goal**: Ensure faster, more reliable updates to the IBTool application by automating the build, test, and deployment pipeline.
* **Placeholder for Image**: [CI/CD Process Flow Image]

### Slide 18: CodeCommit and CodeBuild (2 minutes)

* **CodeCommit**:
  + A secure and scalable Git repository service used to host and manage the IBTool source code.
  + **Benefits**: Version control, collaboration features, and integration with other AWS services.
* **CodeBuild**:
  + A fully managed build service that compiles source code, runs tests, and produces deployment artifacts.
  + **Build Process**:
    1. Fetch source code from CodeCommit.
    2. Run automated tests and linting.
    3. Build Docker images for the frontend and backend services.
* **Placeholder for Image**: [CodeCommit and CodeBuild Process Diagram]

### Slide 19: CodePipeline and Deployment (2 minutes)

* **CodePipeline**:
  + Automates the workflow from code changes to production deployment, integrating CodeCommit, CodeBuild, and deployment stages.
  + **Deployment Flow**:
    1. Code is committed to CodeCommit.
    2. CodeBuild runs tests and builds the application.
    3. If successful, CodePipeline triggers the deployment of the updated services to ECS.
* **Benefits**:
  + Reduces manual intervention.
  + Ensures consistent and reliable deployment of new features and bug fixes.
* **Placeholder for Image**: [CI/CD Pipeline Diagram or CodePipeline Workflow Image]

### Slide 20: Automated Testing in CI/CD (2 minutes)

* **Automated Testing in CI/CD Pipeline**:
  + **Unit Tests**: Ensure individual components of the application (backend functions, frontend components) behave as expected.
  + **Integration Tests**: Test interactions between different services (e.g., frontend interacting with backend API).
  + **End-to-End (E2E) Tests**: Simulate real-world usage to ensure the application works as a whole.
* **Tooling Used**:
  + **Mocha** and **Chai** for backend testing.
  + **Jest** and **Enzyme** for React frontend testing.
  + **Cypress** for end-to-end testing of the entire application.
* **Placeholder for Image**: [Automated Testing Process Diagram]

### Slide 21: CI/CD Pipeline Benefits (1 minute)

* **Key Benefits of CI/CD**:
  + **Faster Delivery**: Enables continuous delivery of features, improvements, and fixes without downtime.
  + **Reduced Risk**: Automated tests ensure that code changes are validated before deployment.
  + **Scalability**: The pipeline scales with the needs of the project, ensuring that development and production environments are always up-to-date.
* **Placeholder for Image**: [Pipeline Benefits Visualization - Faster Delivery, Reduced Risk, and Scalability]

Great! Let’s continue with the remaining slides, covering **Security and Performance Enhancements**, **Results and Benefits**, **Challenges and Lessons Learned**, and **Future Improvements and Conclusion**.

### Slide 22: Security Overview (1 minute)

* **Security in Cloud-Native Architecture**:
  + **Objective**: Protect sensitive customer complaint data while ensuring compliance with healthcare regulations like GDPR and HIPAA.
  + **Security Practices**:
    - Role-based access control.
    - Data encryption both in transit and at rest.
* **Placeholder for Image**: [Security Overview Diagram - Showing Layered Security Architecture]

### Slide 23: IAM, KMS, and Encryption (1 minute)

* **AWS IAM (Identity and Access Management)**:
  + Enforce fine-grained permissions to control access to AWS resources (e.g., ECS, S3).
  + Assign roles and policies to different users and services.
* **AWS KMS (Key Management Service)**:
  + **Data Encryption**: Encrypt sensitive data stored in MongoDB and S3 using customer-managed encryption keys.
  + **Automated Key Rotation**: Ensures data remains secure over time without manual intervention.
* **Placeholder for Image**: [IAM and KMS Flow Diagram]

### Slide 24: SSL/TLS and ACM (1 minute)

* **SSL/TLS Encryption**:
  + **Secure Data Transmission**: All traffic between clients and the IBTool backend is encrypted using SSL/TLS.
  + **AWS ACM (Certificate Manager)**: Manages SSL certificates for securing the application without manual renewal.
* **Placeholder for Image**: [SSL/TLS Certificate Diagram - Connection Secured with HTTPS]

### Slide 25: Performance Monitoring with CloudWatch (1 minute)

* **Amazon CloudWatch**:
  + **Monitoring**: Collects and visualizes real-time performance metrics for ECS, ALB, and other AWS services.
  + **Alarms and Alerts**: Automatically triggers alerts when thresholds are breached (e.g., high CPU usage, low memory).
* **Use Case**: Monitoring request latency and error rates to proactively address performance bottlenecks.
* **Placeholder for Image**: [CloudWatch Metrics and Dashboard Example]

### Slide 26: Auto-Scaling and Elasticity (1 minute)

* **AWS Auto Scaling**:
  + **Elasticity**: Automatically adjusts the number of EC2 instances or ECS tasks based on the current workload.
  + **Scenarios**:
    - **Scale Up**: When traffic spikes, new instances or containers are provisioned automatically.
    - **Scale Down**: During periods of low activity, unused resources are terminated to save costs.
* **Placeholder for Image**: [Auto-Scaling Process Flow - Traffic Load vs. Resource Allocation]

### Slide 27: Key Benefits from Development (1 minute)

* **Development Benefits**:
  + **Improved Complaint Management**: Centralized and automated complaint workflows lead to faster response times and reduced manual intervention.
  + **Enhanced User Experience**: Migrating to React and improving frontend performance resulted in a more responsive and user-friendly interface.
  + **Reduced Errors**: Automation significantly lowered the chances of human errors in data entry and complaint tracking.
* **Placeholder for Image**: [Before vs. After Comparison - IBTool Interface and Workflow]

### Slide 28: Key Benefits from Cloud Migration (1 minute)

* **Cloud Migration Benefits**:
  + **Scalability**: Ability to dynamically scale resources up or down based on real-time demand, ensuring high availability even during peak usage.
  + **Cost Optimization**: Pay-per-use pricing model significantly reduced unnecessary spending on underutilized resources.
  + **High Availability**: The use of AWS services ensured better uptime and disaster recovery mechanisms.
* **Placeholder for Image**: [Cloud Migration Benefits - Uptime, Scalability, and Cost Graphs]

### Slide 29: Quantitative Improvements (1 minute)

* **Performance Metrics**:
  + **Latency Reduction**: Achieved a 30% reduction in response times due to backend optimizations and the use of load balancing.
  + **Cost Savings**: Cloud migration led to a 25% reduction in operational costs by leveraging auto-scaling and serverless architecture.
  + **Uptime Improvement**: Reduced downtime by over 99.9%, ensuring continuous availability of the service.
* **Placeholder for Image**: [Performance Metrics Dashboard - Latency, Uptime, and Cost Comparisons]

### Slide 30: Key Challenges (2 minutes)

* **Technical Challenges**:
  + **Complexity of Cloud Migration**: Orchestrating multiple AWS services (ECS, ALB, RDS, S3) required a steep learning curve and careful integration.
  + **Security and Compliance**: Balancing security measures with performance and ensuring compliance with industry standards such as HIPAA and GDPR.
  + **Data Migration**: Moving large volumes of sensitive customer complaint data from on-premises MongoDB to the cloud while ensuring data integrity and minimal downtime.
* **Placeholder for Image**: [Challenges Flowchart - Cloud, Security, and Data Migration]

### Slide 31: Lessons Learned (2 minutes)

* **Key Lessons**:
  + **Automate Everything**: Automation, from infrastructure provisioning with Terraform to CI/CD pipelines, significantly reduced human error and improved deployment efficiency.
  + **Continuous Monitoring**: Implementing robust monitoring and alerting systems allowed us to proactively manage and resolve performance issues.
  + **Iterative Development**: Adopting Agile methodologies ensured that IBTool evolved based on real user feedback, resulting in a more robust and user-centric application.
* **Placeholder for Image**: [Lessons Learned Summary Diagram - Automation, Monitoring, Agile]

### Slide 32: Future Improvements (2 minutes)

* **Potential Enhancements**:
  + **Serverless Architecture**: Implement AWS Lambda for certain backend functions to further reduce costs and simplify scaling.
  + **Predictive Analytics**: Leverage machine learning models on AWS to predict trends in customer complaints and proactively address potential issues.
  + **Advanced CI/CD Techniques**: Introduce blue/green deployments or canary releases to minimize risks during updates and ensure smooth rollouts.
* **Placeholder for Image**: [Future Improvements Roadmap - Serverless, Machine Learning, Advanced CI/CD]

### Slide 33: Conclusion (1 minute)

* **Summary**:
  + **Development of IBTool**: Centralized, efficient complaint management system that automated workflows and improved response times.
  + **Cloud Migration**: Enhanced scalability, security, and cost efficiency, resulting in improved performance and uptime.
  + **CI/CD Pipeline**: Automated testing and deployment enabled faster, more reliable updates with minimal downtime.
* **Final Thought**: The project highlights the power of cloud-native architecture in improving healthcare applications and sets the foundation for future innovations.
* **Placeholder for Image**: [Final Summary Visualization - Key Achievements]

### Slide 34: Questions (1 minute)

* **Thank You for Your Attention**.
* **Open for Questions**.
* **Placeholder for Image**: [Simple Thank You Image or Contact Information Slide]