Development and Cloud-Native Migration of a Complaint Management System for GE Healthcare

By: Sana Alinia

29 August 2024, Paris

Institution: L'École La Passerelle des Métiers du Numérique (La PMN)

Advisors: Magalie Chatellard, Magali Wissocq

Agenda

- Introduction and Context
- Problem Statement
- Methodologies
- IBTool Development
- Cloud Migration
- CI/CD Pipeline
- Security and Performance
- Results and Benefits
- Challenges and Lessons
- Future Improvements and Conclusion

Introduction and Context

- Context:
  - GE Healthcare provides medical technology solutions, including devices and software that require efficient customer complaint management.
  - Problem: The previous system was fragmented across multiple tools, leading to inefficiencies and slow responses to customer complaints.
- Objective:
  - Develop a centralized complaint management system (**IBTool**) to streamline workflows.
  - Migrate IBTool to the cloud to improve scalability and performance.
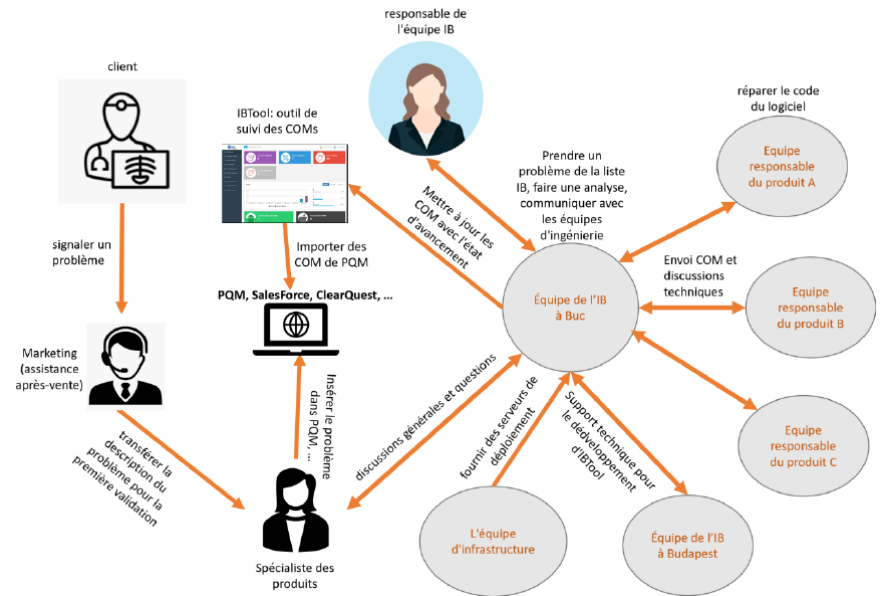
Project Overview

- IBTool: A web-based platform developed to centralize and streamline GE Healthcare's customer complaint management.
  - Before: Complaints managed through multiple disconnected systems (TWD, PQM, SFDC, ClearQuest).
  - After: All complaints handled within a single, unified interface.
- Key Contributions:
  - Development of IBTool: Addressed fragmented data and inefficient workflows.
  - Cloud Migration: Improved scalability, reduced operational costs, and enhanced performance.
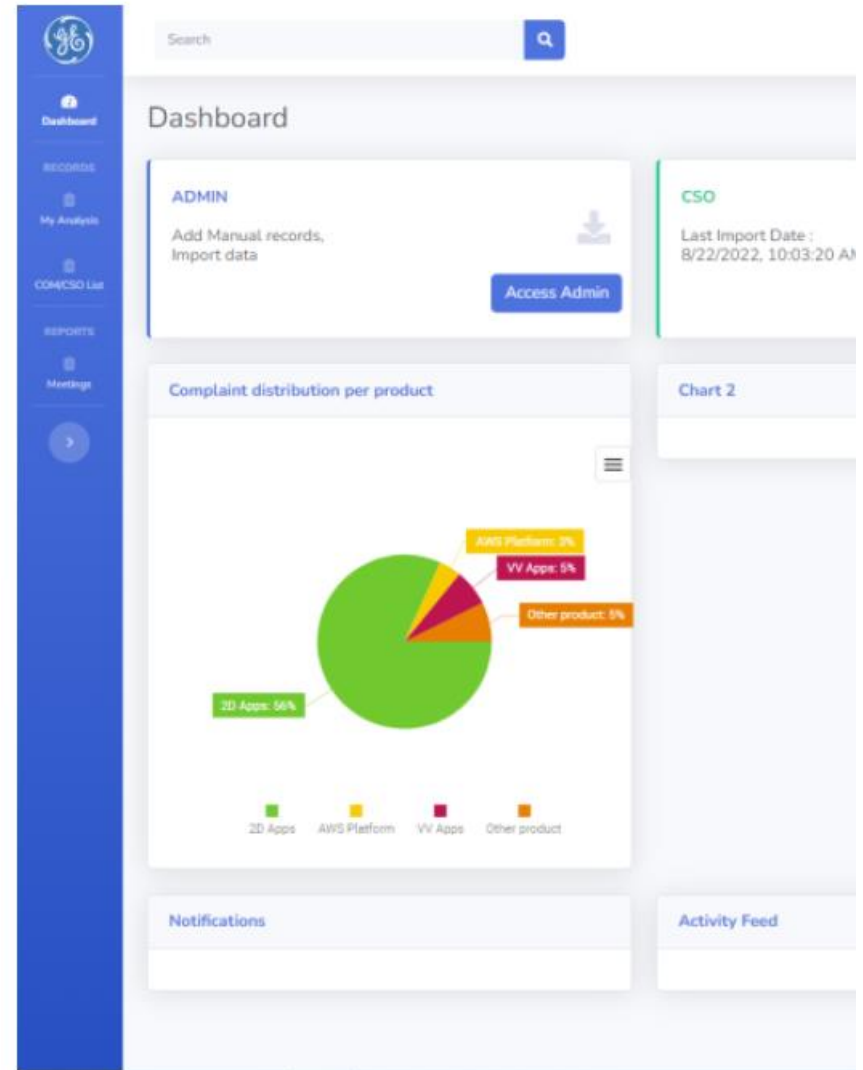
Problem Statement

- Challenges:
  - Fragmented Data: Spread across multiple systems.
  - Manual Processes: High error rates and inefficiencies.
  - Scalability Issues: On-premises couldn't handle variable loads.
  - Limited Visibility: Real-time tracking challenges.
- Solution Needed: Unified system to streamline, automate, and scale complaint management.
- Placeholder for Image: [Fragmented Data/Systems Illustration]

Problem Details

Why It Matters:

- Delayed product improvements and compliance risks.
- Efficiency impacts quality, compliance, and satisfaction.

Core Issues:

- Data Fragmentation: Need for unified system.
- Manual Workflows: Automate for efficiency.
- Scalability: Dynamic scaling required.

Methodology Overview

Two Key Methodologies:

|	
IBTool Development: Centralized, web-based platform.	Cloud Migration: Address scalability and cost using AWS.

Agile Approach: Iterative development, continuous feedback, ongoing testing.

Agile Development Process

- Agile Process:
  - Iterative Development: Based on feedback.
  - Phases:
    - Planning: Core features.
    - Sprints: Feature-specific cycles.
    - Testing: Continuous integration.
    - Feedback: Regular reviews.
- Outcome: Continuous incremental improvements tailored to GE Healthcare's needs.
- Placeholder for Image: [Agile Process/Sprint Cycle Image]
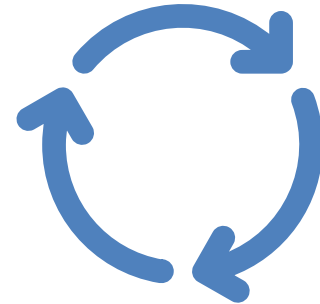
Cloud Migration Methodology

- Cloud Migration:
  - Lift and Shift: Initial migration to AWS.
  - Post-Migration: Optimize with auto-scaling and managed services.
- Steps:
  - ECS Deployment: Orchestrating containerized services with ALB distributing traffic.
  - CodeCommit, CodePipeline, and CodeBuild automate the CI/CD pipeline.
  - CloudWatch monitors performance, and KMS ensures data encryption.

Infrastructure as Code with Terraform

- Terraform for IaC:
  - IaC: Automate consistent infrastructure deployment.
  - Why Terraform: Cross-cloud, version control, and efficient provisioning.
- Key Elements:
  - EC2: Provisioned by Terraform for backend compute.
  - ECS & ALB: Docker containers deployed with load balancing.
  - CodePipeline, CodeCommit, and ECR: CI/CD with container storage.
  - CloudWatch: Real-time monitoring of infrastructure.
  - KMS & ACM: Automated encryption and SSL certificate management.
- Benefits: Faster, consistent, and collaborative deployments.
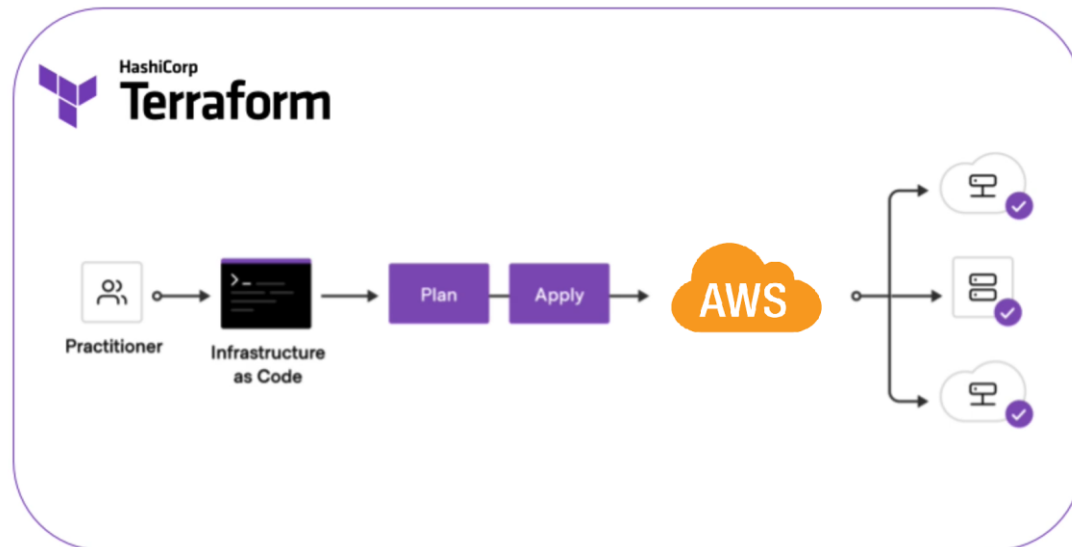
Overview of IBTool Architecture

- On-Premises Setup: Hosted on local servers using Docker containers for frontend, backend, and database.
- Monolithic Design: All services running together.
- Tech Stack:
    - Frontend: HTML, CSS, JS (DevExtreme).
    - Backend: Node.js, Express.js.
    - Database: MongoDB.

Key Features of IBTool

- Core Functions:
  - Complaint Management: Centralized tracking.
  - CRUD Operations: Create, read, update, delete complaints.
  - Automated Workflows: Notifications based on complaint status.

Technical Stack and Components
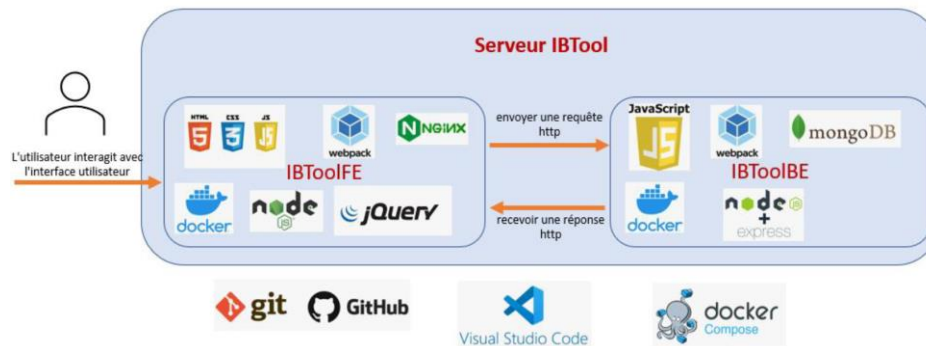
- Frontend:
  - Initial: HTML, CSS, JS (DevExtreme).
  - Improvement: Migrated to React for better performance.
- Backend:
  - Node.js, Express.js handling API requests.
  - Enhancements: Middleware, caching (Redis).
- Database:
  - MongoDB for scalable data storage.
  - Improvements: Indexing for faster queries.
- Placeholder for Image: [Technical Stack Diagram or Component Architecture Diagram]

Motivation for Cloud Migration

- On-Premises Challenges:
  - Scalability Issues: Difficulty handling spikes.
  - High Costs: Fixed infrastructure expenses.
  - Maintenance Overhead: Resource-heavy monitoring.
- Cloud Benefits:
  - Dynamic Scaling: Auto-scaling resources.
  - Cost Efficiency: Pay-per-use model.
  - Resilience: Improved fault tolerance.

Cloud-Native Architecture Overview

- New Setup:
  - Frontend: S3 with potential CloudFront for fast delivery.
  - Backend: Dockerized services on ECS with ALB for traffic distribution.
  - MongoDB: Managed NoSQL database.
  - CI/CD: CodePipeline and CodeBuild manage continuous deployments.
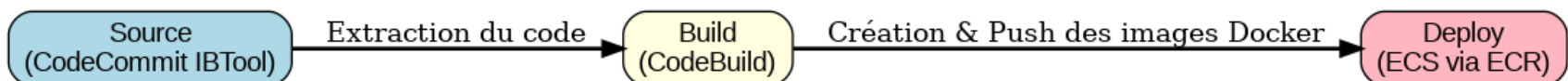 - CloudWatch & KMS: Monitor performance and manage encryption.
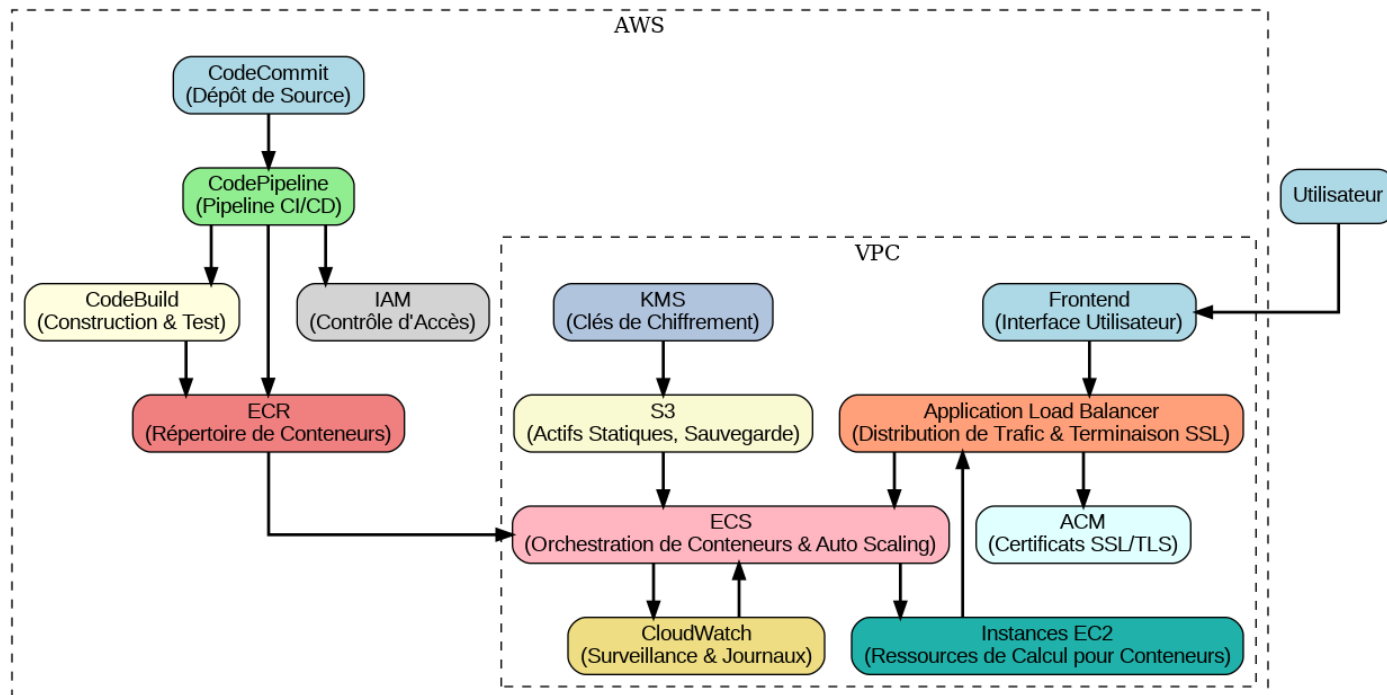
AWS Services Used

- Key AWS Services:
  - S3: Static file storage for the frontend.
  - ECS: Container orchestration for backend.
  - MongoDB: Managed database (if applicable).
  - ALB: Distributes traffic to backend services.
  - CodePipeline: Automates deployment workflow.
  - CloudWatch: Monitors performance.
 - KMS & ACM: Ensures data encryption and certificate management.

CI/CD Pipeline Overview

- CI/CD:
  - CI: Automate testing/builds after code changes.
  - CD: Automate deployment to production.
- CI/CD Pipeline:
  - CodeCommit: Source code repository.
  - CodeBuild: Automated builds and tests.
  - CodePipeline: Orchestrates the workflow from commit to deployment.
  - ECR: Stores Docker images for ECS deployment.
  - CloudWatch: Monitors the pipeline and system performance.
- Goal: Faster, more reliable updates through automated pipelines.

| Source (CodeCommit IBTool) | → Extraction du code → | Build (CodeBuild) | → Création & Push des images Docker → | Deploy (ECS via ECR) |

CodeCommit and CodeBuild

- CodeCommit: Secure Git repository for code management.
  - Benefits: Version control, collaboration, AWS integration.
- CodeBuild: Managed build service for compiling, testing, and producing Docker images.
  - Process: Fetch code → Test → Build Docker images.
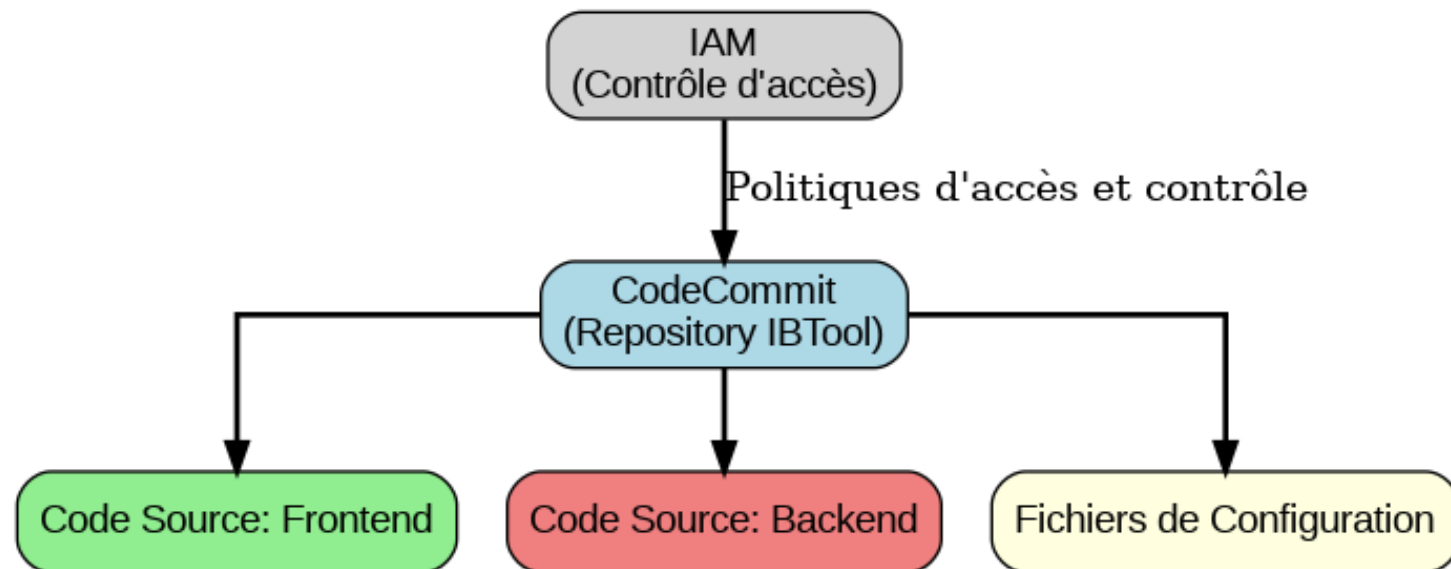
CodePipeline and Deployment

- CodePipeline: Automates the deployment workflow.
  - Steps:
    - Code pushed to CodeCommit.
    - CodeBuild compiles and tests.
    - Successful builds are deployed to ECS.
- Benefits: Reduces manual effort, ensures reliable deployments.
- Placeholder for Image: [CI/CD Pipeline Diagram or Workflow Image]

Automated Testing in CI/CD

Automated Testing:

Unit Tests: Test individual components.

Integration Tests: Test interactions between services.

E2E Tests: Simulate real-world scenarios.

Tools:

Backend: Mocha, Chai.

Frontend: Jest, Enzyme.

E2E: Playwright.

CI/CD Pipeline Benefits

- Benefits of CI/CD:
  - Faster Delivery: Continuous updates without downtime.
  - Reduced Risk: Automated tests validate changes before deployment.
  - Scalability: Pipeline scales with project needs, ensuring up-to-date environments.

Security Overview

- Cloud-Native Security:
    - Objective: Protect sensitive data and ensure compliance with GDPR, HIPAA.
    - Practices:
        - Role-based access control.
   - Data encryption (in transit and at rest).
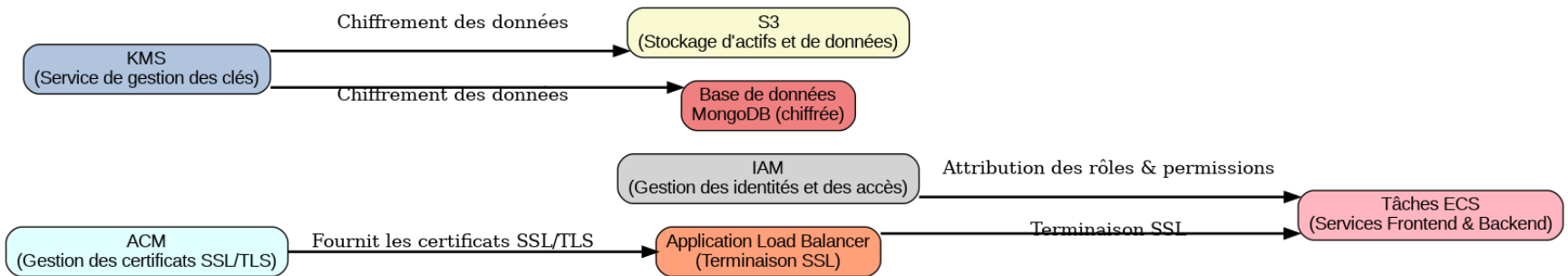
IAM, KMS, and Encryption

- AWS IAM: Fine-grained permissions to control resource access.
- AWS KMS:
  - Encryption: Protect data in MongoDB and S3.
  - Key Rotation: Automated, ensuring long-term security.
- Placeholder for Image: [IAM and KMS Flow Diagram]
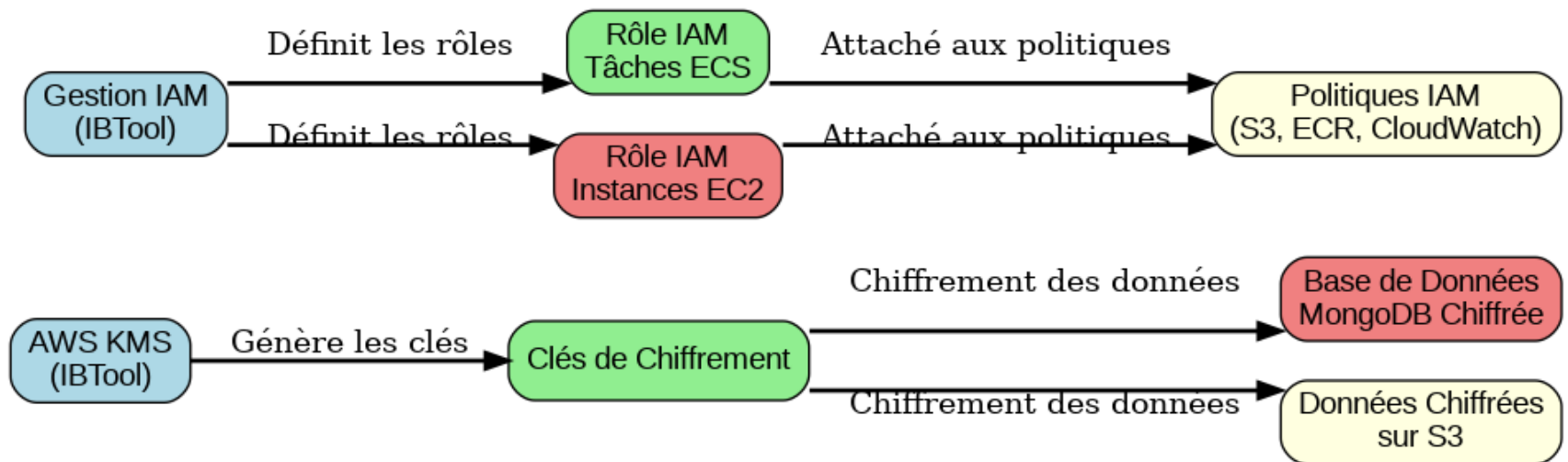
SSL/TLS and ACM

- SSL/TLS Encryption: Encrypts all traffic between clients and the backend.
- AWS ACM: Manages SSL certificates, automating renewals.
- Placeholder for Image: [SSL/TLS Certificate Diagram]
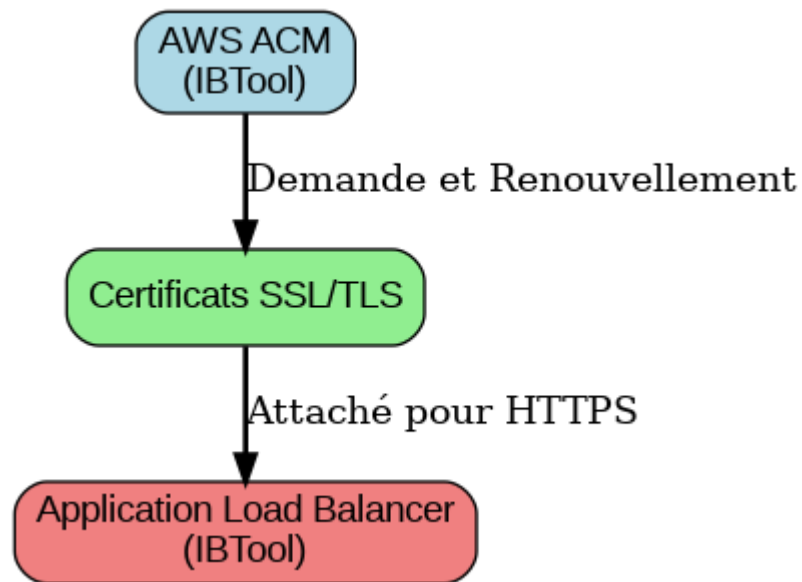
Performance Monitoring with CloudWatch

- CloudWatch:
  - Monitoring: Real-time performance metrics.
  - Alerts: Triggered by thresholds (e.g., high CPU usage).
- Use Case: Proactively monitoring latency and error rates.
- Placeholder for Image: [CloudWatch Metrics Dashboard]
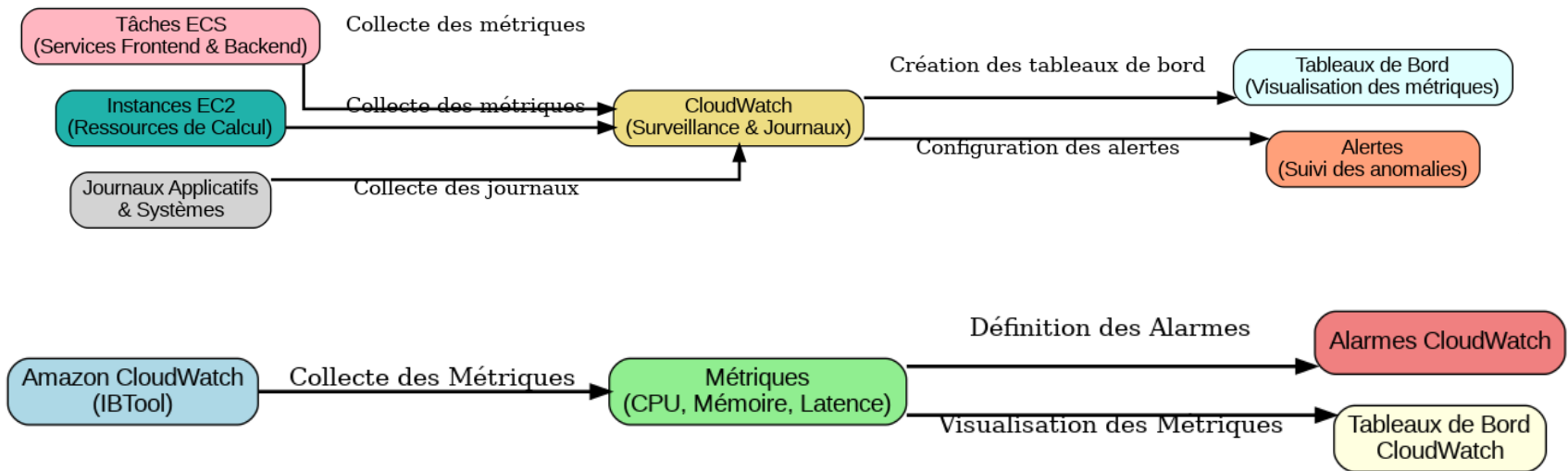
Auto-Scaling and Elasticity

- AWS Auto Scaling:
  - Elasticity: Adjusts resources based on traffic.
  - Scaling Scenarios:
    - Scale Up: Auto-add resources during traffic spikes.
    - Scale Down: Auto-remove resources during low activity.
- Placeholder for Image: [Auto-Scaling Flow Diagram]
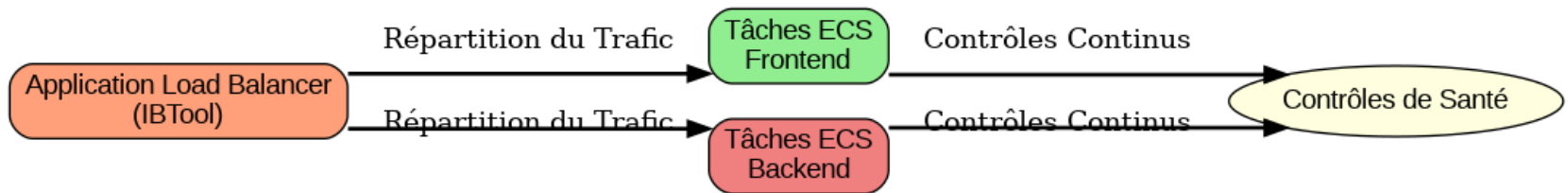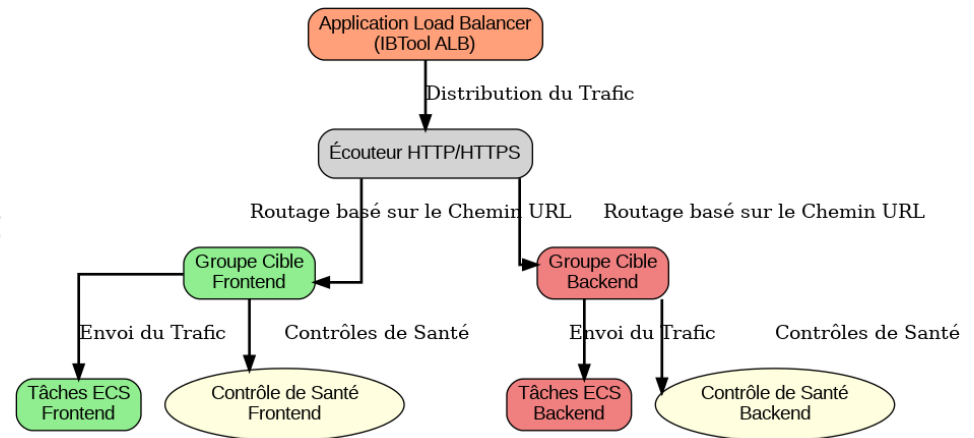
Key Benefits from Development

- Development Outcomes:
  - Improved Management: Centralized, automated complaint workflows.
  - Better UX: React-based frontend for better performance.
  - Error Reduction: Automation minimizes human errors.

Key Benefits from Cloud Migration

- Cloud Migration Benefits:
  - Scalability: Dynamically adjusts resources to demand.
  - Cost Savings: Pay-per-use model reduced operational expenses.
  - High Availability: Better uptime and disaster recovery.
- Placeholder for Image: [Cloud Migration Benefits - Uptime, Scalability, Costs]

Quantitative Improvements

- Performance Gains:
  - Latency: 30% reduction from backend optimization and load balancing.
  - Cost: 25% operational cost savings via auto-scaling.
  - Uptime: 99.9% uptime due to cloud architecture.
- Placeholder for Image: [Performance Metrics Dashboard]

Key Challenges

- Challenges:
  - Cloud Migration Complexity: Integrating AWS services (ECS, ALB, RDS, S3) required expertise.
  - Security and Compliance: Balancing performance with HIPAA, GDPR compliance.
  - Data Migration: Safely migrating large, sensitive data to the cloud.
- Placeholder for Image: [Challenges Flowchart - Cloud, Security, Data Migration]

Lessons Learned

- Lessons:
  - Automate: Infrastructure, CI/CD pipelines, and monitoring reduce errors and improve efficiency.
  - Continuous Monitoring: Proactive management through alerts and performance insights.
  - Agile Development: Iterative improvements based on real-time feedback improved outcomes.
- Placeholder for Image: [Lessons Learned Diagram - Automation, Monitoring, Agile]

Future Improvements

- Potential Enhancements:
  - Serverless: AWS Lambda to reduce costs and simplify scaling.
  - Predictive Analytics: Use machine learning to predict complaint trends.
  - CI/CD: Blue/green deployments for safer, smoother updates.
- Placeholder for Image: [Future Roadmap - Serverless, Machine Learning, CI/CD]

Conclusion

- Summary:
  - IBTool: Centralized complaint management with automated workflows.
  - Cloud Migration: Enhanced scalability, security, and efficiency.
  - CI/CD: Faster, reliable deployments with minimal downtime.
- Final Thought: Cloud-native architecture provides a strong foundation for healthcare application innovation.
- Placeholder for Image: [Key Achievements Visualization]

THANK YOU

Open for Questions.