**Worklog - Ngày 13/05/2025**

**📅 Thông tin cơ bản**

* **Ngày: 13/05/2025**
* **Thứ: Thứ Ba**
* **Tuần thực tập: Tuần thứ 1/12**
* **Thời gian làm việc: 9:00 - 17:00**
* **Mood: 🔍 + Tò mò khám phá CI/CD và các dịch vụ DevOps trên AWS**

**🎯 Mục tiêu ngày hôm nay**

* **Khám phá dịch vụ AWS CodePipeline và CodeBuild**
* **Triển khai một quy trình CI/CD đơn giản từ GitHub lên S3**
* **Viết CloudFormation template để tái sử dụng pipeline**
* **Hiểu cách tự động hoá quy trình build và deploy ứng dụng**

**💼 Công việc đã thực hiện**

**1. Làm quen với CodePipeline và CodeBuild ⏱️ 3 giờ**

* **Mô tả: Tìm hiểu cách thiết lập pipeline kết nối GitHub → CodeBuild → S3**
* **Kết quả: Đã tạo pipeline hoạt động tự động hoá toàn bộ quá trình build + deploy**
* **Tools/Tech: AWS CodePipeline, CodeBuild, GitHub**
* **Links: [CodePipeline User Guide](https://docs.aws.amazon.com/codepipeline)**

**2. Tạo CloudFormation Template cho CI/CD ⏱️ 2 giờ**

* **Mô tả: Viết YAML template định nghĩa pipeline, buildspec, IAM Role**
* **Kết quả: Dễ dàng khởi tạo lại hệ thống pipeline chỉ trong vài phút**
* **Tools/Tech: AWS CloudFormation, YAML**
* **Links: [CI/CD CloudFormation Examples](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/)**

**3. Fix lỗi permission khi CodeBuild push lên S3 ⏱️ 1 giờ**

* **Mô tả: Debug IAM Role do CodeBuild không thể upload artifact lên S3**
* **Kết quả: Cập nhật policy s3:PutObject, pipeline hoạt động bình thường**
* **Tools/Tech: IAM Policy, CloudTrail logs**

**📚 Kiến thức học được**

**🔧 Technical Skills**

* **Thiết lập quy trình CI/CD từ GitHub tới S3 sử dụng AWS CodePipeline**
* **Viết buildspec.yml cho CodeBuild để kiểm soát quá trình build**
* **Áp dụng CloudFormation để tự động hóa CI/CD**

**💡 Concepts & Theory**

* **CI/CD là gì và vai trò trong quy trình DevOps hiện đại**
* **Pipeline gồm các stages: Source → Build → Deploy**
* **Tư duy “Infrastructure as Code” áp dụng cả cho automation pipelines**

**🤝 Soft Skills**

* **Kỹ năng debug log và phân tích IAM Policy**
* **Tư duy hệ thống: nối các dịch vụ AWS với nhau thành một quy trình khép kín**
* **Kiên nhẫn và chi tiết khi viết template YAML dài**

**🚧 Khó khăn và giải pháp**

**Vấn đề 1: CodeBuild không nhận đúng cấu hình buildspec**

* **Mô tả: Dù có file buildspec.yml nhưng build step báo lỗi**
* **Impact: Build bị fail liên tục, tốn thời gian debug**
* **Root Cause: File buildspec.yml không nằm đúng thư mục gốc**
* **Solution: Move file lên root và commit lại**
* **Result: Build hoạt động trơn tru**
* **Lesson: AWS rất nghiêm ngặt về cấu trúc file và quyền**

**Vấn đề 2: CodePipeline không trigger khi push code**

* **Mô tả: Đã push code mới nhưng pipeline không tự động chạy**
* **Impact: Mất tính tự động hóa của CI/CD**
* **Root Cause: Chưa bật webhook GitHub hoặc thiếu quyền GitHub token**
* **Solution: Cấu hình lại kết nối GitHub trong pipeline**
* **Result: Pipeline hoạt động tự động khi có commit mới**
* **Lesson: Cần kiểm tra kỹ token, webhook và event trigger**

**💭 Reflection & Insights**

**What went well today?**

* **Thiết lập thành công pipeline đầu tiên**
* **Viết được CloudFormation template đầy đủ cho quy trình CI/CD**
* **Khả năng debug IAM permission được cải thiện rõ rệt**

**What could be improved?**

* **Nên thử build pipeline phức tạp hơn có thêm bước test**
* **Cần luyện thêm cách viết buildspec.yml cho các ngôn ngữ khác nhau**
* **Nên tổ chức lại template thành nested stack cho dễ bảo trì**

**Key Insights**

* **CI/CD không chỉ là DevOps, mà còn là chiến lược tăng hiệu suất làm việc nhóm**
* **CloudFormation rất hữu ích trong việc tái sử dụng quy trình CI/CD**
* **Việc thiết kế IAM policy chuẩn là điều kiện tiên quyết để các dịch vụ AWS hoạt động ăn khớp**

***Worklog created by: Tran Quang Trong   
 Next review: 15/05/2025***

**Worklog - Ngày 15/05/2025**

**📅 Thông tin cơ bản**

* **Ngày: 15/05/2025**
* **Thứ: Thứ Năm**
* **Tuần thực tập: Tuần thứ 1/12**
* **Thời gian làm việc: 9:00 - 17:00**
* **Mood: 🤯 + Căng não vì học về Auto Scaling & Monitoring nâng cao**

**🎯 Mục tiêu ngày hôm nay**

* **Tìm hiểu Auto Scaling Group (ASG) và cấu hình Scaling Policy**
* **Thực hành tạo Launch Template và Auto Scaling cho EC2**
* **Làm quen với CloudWatch Alarm, Log và Dashboard**
* **Viết CloudFormation để triển khai ASG và giám sát**

**💼 Công việc đã thực hiện**

**1. Auto Scaling Group và Launch Template ⏱️ 3 giờ**

* **Mô tả: Tạo launch template và cấu hình Auto Scaling dựa trên CPU utilization**
* **Kết quả: EC2 tự động tăng giảm số lượng instance khi tải thay đổi**
* **Tools/Tech: EC2, Launch Template, ASG, CloudWatch Metrics**
* **Links: AWS Auto Scaling Guide**

**2. CloudWatch Alarms và Logs ⏱️ 2 giờ**

* **Mô tả: Thiết lập các CloudWatch Alarm cho EC2 CPU > 70%, gửi SNS notification**
* **Kết quả: Nhận cảnh báo kịp thời khi hệ thống quá tải**
* **Tools/Tech: CloudWatch Alarms, Logs, SNS**
* **Links: AWS CloudWatch Alarms Tutorial**

**3. Triển khai bằng CloudFormation ⏱️ 2 giờ**

* **Mô tả: Viết template tự động tạo ASG, Alarm, và các resource giám sát**
* **Kết quả: Toàn bộ hệ thống scaling + monitoring có thể tái sử dụng và chỉnh sửa dễ dàng**
* **Tools/Tech: CloudFormation, YAML**
* **Links: Template mẫu ASG và CloudWatch Alarm**

**📚 Kiến thức học được**

**🔧 Technical Skills**

* **Tạo Auto Scaling Group hoạt động theo thời gian thực**
* **Triển khai cảnh báo hiệu suất bằng CloudWatch Alarms**
* **Viết CloudFormation template cho hệ thống có khả năng tự mở rộng**

**💡 Concepts & Theory**

* **Horizontal scaling và lợi ích với ứng dụng web có lưu lượng biến động**
* **Định nghĩa Launch Template vs Launch Configuration**
* **Cloud Monitoring giúp chủ động phát hiện vấn đề và tối ưu vận hành**

**🤝 Soft Skills**

* **Rèn luyện tính kiên nhẫn khi debug template YAML dài và phức tạp**
* **Kỹ năng tổ chức tài nguyên theo nhóm để dễ giám sát**
* **Giao tiếp tốt hơn với mentor để xin giải thích rõ về scaling thresholds**

**🚧 Khó khăn và giải pháp**

**Vấn đề 1: Auto Scaling không hoạt động dù CPU vượt ngưỡng**

* **Mô tả: Load test tăng CPU nhưng không có instance nào được launch thêm**
* **Impact: Hệ thống không scale kịp dẫn đến chậm**
* **Root Cause: Chưa gắn Scaling Policy đúng cho ASG**
* **Solution: Kiểm tra lại step scaling policy, update chính xác CloudWatch alarm**
* **Result: Auto Scaling hoạt động đúng khi CPU tăng cao**
* **Lesson: Scaling policy cần cấu hình chính xác metric và threshold**

**Vấn đề 2: Alarm gửi thông báo liên tục dù CPU bình thường**

* **Mô tả: Sau khi CPU giảm, vẫn nhận spam cảnh báo từ SNS**
* **Impact: Gây nhiễu loạn thông tin, dễ bỏ sót cảnh báo thật**
* **Root Cause: Alarm thiếu cấu hình period và evaluation interval phù hợp**
* **Solution: Tăng thời gian đánh giá và đặt threshold logic hợp lý hơn**
* **Result: Thông báo chỉ gửi khi thực sự có tình trạng bất thường kéo dài**
* **Lesson: Monitoring hiệu quả cần kết hợp giữa kỹ thuật và logic cảnh báo thông minh**

**💭 Reflection & Insights**

**What went well today?**

* **Triển khai được hệ thống Auto Scaling hoàn chỉnh**
* **CloudFormation giúp tiết kiệm thời gian test cấu hình**
* **Hiểu được tầm quan trọng của cảnh báo và tự động phản ứng với sự cố**

**What could be improved?**

* **Cần học thêm về scaling based on custom metrics**
* **Nên thực hành thêm trường hợp scale-in (giảm tải) và cooldown period**
* **Có thể tích hợp thêm CloudWatch Dashboard để theo dõi trực quan hơn**

**Key Insights**

* **Auto Scaling giúp hệ thống AWS linh hoạt, tiết kiệm chi phí và tăng độ tin cậy**
* **CloudWatch không chỉ giám sát mà còn là công cụ tự động hóa phản ứng với sự cố**
* **Việc kết hợp tốt giữa CloudFormation, Alarm, và Scaling Policy là nền tảng của hạ tầng tự vận hành**

***Worklog created by: Tran Quang Trong*   
 *Next review: 16/05/2025***

**Worklog - Ngày 16/05/2025**

**📅 Thông tin cơ bản**

* **Ngày: 16/05/2025**
* **Thứ: Thứ Sáu**
* **Tuần thực tập: Tuần thứ 1/12**
* **Thời gian làm việc: 9:00 - 17:00**
* **Mood: 🌐 + Hào hứng khi thấy load balancer phân phối traffic đều đặn**

**🎯 Mục tiêu ngày hôm nay**

* **Tìm hiểu các loại Elastic Load Balancer trong AWS**
* **Thực hành cấu hình Application Load Balancer (ALB) với Auto Scaling Group**
* **Tạo Target Group và Rule để route traffic theo path**
* **Viết CloudFormation template để triển khai toàn bộ hệ thống cân bằng tải**

**💼 Công việc đã thực hiện**

**1. Khám phá ELB và các loại Load Balancer ⏱️ 2 giờ**

* **Mô tả: Đọc tài liệu, so sánh ALB, NLB và CLB**
* **Kết quả: Nắm rõ đặc điểm kỹ thuật và use case phù hợp cho từng loại**
* **Tools/Tech: AWS Console, AWS Docs**
* **Links: [Elastic Load Balancing Overview](https://docs.aws.amazon.com/elasticloadbalancing)**

**2. Cấu hình ALB với Auto Scaling Group ⏱️ 3 giờ**

* **Mô tả: Kết nối Auto Scaling Group với ALB để phân phối traffic đến các instance**
* **Kết quả: Người dùng truy cập qua DNS ALB được route đến các EC2 khác nhau tùy tải**
* **Tools/Tech: Application Load Balancer, ASG, Target Group**
* **Links: [ALB and ASG Integration Guide](https://docs.aws.amazon.com/autoscaling/latest/userguide/attach-load-balancer-asg.html)**

**3. Routing Rule theo URL Path + CloudFormation ⏱️ 2 giờ**

* **Mô tả: Thiết lập rule để route /api/\* về một nhóm EC2 riêng biệt**
* **Kết quả: Phân chia traffic hiệu quả và có thể mở rộng theo nhu cầu microservices**
* **Tools/Tech: ALB Listener Rules, Target Group, CloudFormation**
* **Links: Template YAML example ALB + Path Routing**

**📚 Kiến thức học được**

**🔧 Technical Skills**

* **Sử dụng Application Load Balancer để tối ưu phân phối tải**
* **Gắn Auto Scaling Group với ALB để scale + balance cùng lúc**
* **Triển khai hạ tầng cân bằng tải hoàn toàn qua CloudFormation**

**💡 Concepts & Theory**

* **Cân bằng tải layer 7 và các use case phổ biến (web, API, microservices)**
* **Cơ chế health check và failover trong hệ thống phân tán**
* **Routing theo path giúp tổ chức ứng dụng dạng module hóa**

**🤝 Soft Skills**

* **Kỹ năng phân tích kiến trúc phù hợp cho từng loại service**
* **Giao tiếp hiệu quả khi hỏi mentor để phân biệt ALB vs NLB**
* **Ghi chú chi tiết từng bước triển khai để tránh lỗi lặp lại**

**🚧 Khó khăn và giải pháp**

**Vấn đề 1: ALB không route đúng path /api/\***

* **Mô tả: Dù đã tạo rule nhưng truy cập /api/xyz vẫn trả về 404**
* **Impact: Người dùng không thể gọi API backend**
* **Root Cause: Thiết lập rule sai priority và không match chính xác path**
* **Solution: Điều chỉnh lại listener rule và priority order**
* **Result: Traffic /api/\* được route đúng target group**
* **Lesson: Rule trong ALB phải test kỹ để tránh conflict hoặc missmatch**

**Vấn đề 2: EC2 bị đánh dấu Unhealthy trong Target Group**

* **Mô tả: Một số instance bị out khỏi load balancer dù hoạt động bình thường**
* **Impact: Traffic chỉ phân phối vào vài instance, mất cân bằng**
* **Root Cause: Health check port không đúng (check 80, EC2 chạy 8080)**
* **Solution: Chỉnh lại health check path và port cho phù hợp với app backend**
* **Result: Tất cả instance hoạt động ổn định, cân bằng tải tốt hơn**
* **Lesson: Health check cần xác định kỹ theo ứng dụng thực tế, không để mặc định**

**💭 Reflection & Insights**

**What went well today?**

* **Thành công trong việc triển khai ALB routing theo path**
* **Tích hợp được ALB với Auto Scaling Group qua CloudFormation**
* **Hiểu rõ hơn về cách phân phối traffic thông minh với nhiều microservice**

**What could be improved?**

* **Nên thực hành thêm với Network Load Balancer (NLB) cho ứng dụng TCP**
* **Cần luyện viết nested template cho phần Load Balancer riêng**
* **Tăng khả năng monitor target group thông qua CloudWatch Alarm**

**Key Insights**

* **Load Balancer là trung tâm phân phối giúp hệ thống ổn định và dễ mở rộng**
* **Routing theo path phù hợp cho kiến trúc microservices hiện đại**
* **ALB có thể tích hợp sâu với Auto Scaling, CloudWatch, và Certificate Manager để tạo thành một nền tảng ứng dụng web linh hoạt và an toàn**

***Worklog created by: Tran Quang Trong*   
 *Next review: 19/05/2025 (Tuần thứ 2)***

**Worklog - Ngày 19/05/2025**

**📅 Thông tin cơ bản**

* **Ngày: 19/05/2025**
* **Thứ: Thứ Hai**
* **Tuần thực tập: Tuần thứ 2/12**
* **Thời gian làm việc: 9:00 - 17:00**
* **Mood: 🧠 + Tập trung cao độ để hiểu RDS và High Availability**

**🎯 Mục tiêu ngày hôm nay**

* **Tìm hiểu các engine cơ sở dữ liệu được AWS RDS hỗ trợ**
* **Triển khai Amazon RDS với MySQL có Multi-AZ**
* **Kết nối ứng dụng EC2 với RDS thông qua security group**
* **Backup, snapshot và cấu hình tự động sao lưu**

**💼 Công việc đã thực hiện**

**1. Tìm hiểu tổng quan về Amazon RDS ⏱️ 1 giờ**

* **Mô tả: Đọc docs về RDS, các engine hỗ trợ (MySQL, PostgreSQL, MariaDB, SQL Server, Oracle, Aurora)**
* **Kết quả: Hiểu rõ lợi ích khi sử dụng managed database**
* **Tools/Tech: AWS Docs, AWS Console**
* **Links: [RDS Overview](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Welcome.html)**

**2. Triển khai RDS MySQL Multi-AZ ⏱️ 3 giờ**

* **Mô tả: Tạo RDS instance với cấu hình HA (Multi-AZ), bật backup, enable deletion protection**
* **Kết quả: Instance hoạt động ổn định, có thể failover sang AZ phụ nếu AZ chính gặp sự cố**
* **Tools/Tech: Amazon RDS, MySQL, Subnet Group, Security Group**
* **Links: [Multi-AZ deployment](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Concepts.MultiAZ.html)**

**3. Kết nối EC2 đến RDS và kiểm thử ⏱️ 2 giờ**

* **Mô tả: Tạo EC2, gắn vào cùng VPC/Subnet, chỉnh security group cho phép truy cập cổng 3306**
* **Kết quả: Kết nối thành công RDS MySQL từ EC2 qua CLI và MySQL Workbench**
* **Tools/Tech: EC2, SSH, MySQL CLI, Security Group Rules**
* **Links: [Connect EC2 to RDS Guide](https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_ConnectToInstance.html)**

**📚 Kiến thức học được**

**🔧 Technical Skills**

* **Tạo Amazon RDS MySQL instance với HA**
* **Cấu hình network, subnet group, security group để kết nối từ EC2**
* **Tự động hóa snapshot và backup recovery**

**💡 Concepts & Theory**

* **Multi-AZ giúp tăng tính sẵn sàng, không phải để scale (scale là Read Replica)**
* **IAM Database Authentication vs Traditional User/Pass**
* **Deletion Protection và Backup Retention là các yếu tố an toàn dữ liệu quan trọng**

**🤝 Soft Skills**

* **Cẩn trọng trong quá trình cấu hình RDS để tránh bị tính phí cao do sai region/type**
* **Kỹ năng tìm lỗi kết nối database và hiểu networking trong cloud**
* **Kiên trì troubleshoot lỗi Access Denied và timeout**

**🚧 Khó khăn và giải pháp**

**Vấn đề 1: EC2 không kết nối được RDS**

* **Mô tả: Dù dùng đúng endpoint nhưng vẫn báo lỗi “host unreachable”**
* **Impact: Không test được ứng dụng liên kết với DB**
* **Root Cause: Security group chưa mở port 3306 từ EC2 đến RDS**
* **Solution: Thêm inbound rule vào security group RDS cho phép IP của EC2**
* **Result: Kết nối thành công sau khi mở port**
* **Lesson: Luôn kiểm tra kỹ security group và subnet/VPC setup**

**Vấn đề 2: Không tạo được snapshot thủ công**

* **Mô tả: Lỗi khi tạo manual snapshot vì thiếu quyền**
* **Impact: Không sao lưu kịp RDS để backup trước test**
* **Root Cause: IAM role EC2 không có quyền thao tác với RDS**
* **Solution: Gán thêm quyền rds:CreateDBSnapshot vào IAM policy**
* **Result: Snapshot tạo thành công và lưu trữ được**
* **Lesson: IAM policy cần cụ thể cho từng thao tác, tránh gán FullAccess bừa bãi**

**💭 Reflection & Insights**

**What went well today?**

* **Triển khai thành công Amazon RDS Multi-AZ**
* **Kết nối EC2 đến RDS đúng chuẩn network và bảo mật**
* **Hiểu sâu về backup và HA trong cơ sở dữ liệu cloud**

**What could be improved?**

* **Cần thực hành thêm với Read Replica và scale-out**
* **Nên test failover bằng cách simulate AZ failure**
* **Tìm hiểu thêm về Aurora để so sánh với RDS truyền thống**

**Key Insights**

* **RDS giúp giảm tải rất nhiều công việc vận hành database truyền thống**
* **Multi-AZ là lựa chọn tuyệt vời để tăng tính sẵn sàng, nhưng không giúp tăng performance**
* **IAM + Security Group + VPC/Subnet là bộ ba quan trọng để đảm bảo kết nối thành công giữa các dịch vụ AWS**

***Worklog created by: Tran Quang Trong*   
 *Next review: 20/05/2025***

**Worklog - Ngày 20/05/2025**

**📅 Thông tin cơ bản**

* **Ngày: 20/05/2025**
* **Thứ: Thứ Ba**
* **Tuần thực tập: Tuần thứ 2/12**
* **Thời gian làm việc: 9:00 - 17:00**
* **Mood: ⚡ + Hứng thú với ứng dụng serverless đầu tiên**

**🎯 Mục tiêu ngày hôm nay**

* **Tìm hiểu kiến trúc AWS Lambda và các use case**
* **Viết hàm Lambda đơn giản bằng Python**
* **Kích hoạt Lambda bằng Amazon S3 Event**
* **Tạo CloudWatch Logs để giám sát Lambda**

**💼 Công việc đã thực hiện**

**1. Tìm hiểu AWS Lambda và kiến trúc Serverless ⏱️ 1.5 giờ**

* **Mô tả: Đọc tài liệu AWS về Lambda và các event trigger**
* **Kết quả: Nắm được kiến trúc, cách hoạt động và mô hình tính phí**
* **Tools/Tech: AWS Lambda, IAM Role, S3 Event, CloudWatch**
* **Links: [Lambda Overview](https://docs.aws.amazon.com/lambda/latest/dg/welcome.html)**

**2. Viết hàm Lambda xử lý file upload từ S3 ⏱️ 2 giờ**

* **Mô tả: Tạo Lambda function đọc metadata của file vừa upload vào S3**
* **Kết quả: Lambda tự động chạy khi có file mới → log ra tên file và size**
* **Tools/Tech: Python, Boto3, S3 Trigger**
* **Links: Sample Lambda function code**

**3. Gắn sự kiện S3 trigger và kiểm thử ⏱️ 2 giờ**

* **Mô tả: Cấu hình bucket S3 gọi Lambda mỗi khi có đối tượng mới**
* **Kết quả: Kiểm thử thành công bằng cách upload ảnh → Lambda log ra thông tin file**
* **Tools/Tech: S3 Event Notification, Lambda Trigger**
* **Links: AWS Lambda-S3 Integration**

**4. Monitor qua CloudWatch Logs ⏱️ 1 giờ**

* **Mô tả: Xem log output của Lambda function trên CloudWatch**
* **Kết quả: Có thể theo dõi hoạt động của function, thời gian thực thi, lỗi**
* **Tools/Tech: CloudWatch Logs, Metrics**
* **Links: Log Group - /aws/lambda/s3-file-processor**

**📚 Kiến thức học được**

**🔧 Technical Skills**

* **Viết Lambda function bằng Python tích hợp với AWS SDK (boto3)**
* **Tạo trigger từ S3 Event để tự động hóa xử lý dữ liệu**
* **Theo dõi hoạt động Lambda qua CloudWatch Logs**

**💡 Concepts & Theory**

* **Serverless giúp giảm gánh nặng vận hành, chỉ lo viết code**
* **Mỗi function nên nhỏ gọn, thực hiện một nhiệm vụ duy nhất**
* **Cloud-native architecture = event-driven + stateless + scalable**

**🤝 Soft Skills**

* **Kỹ năng debug log và kiểm tra context execution**
* **Viết code đơn giản, clean để dễ kiểm tra log**
* **Suy nghĩ theo hướng tự động hóa thay vì thao tác thủ công**

**🚧 Khó khăn và giải pháp**

**Vấn đề 1: Lambda không nhận được sự kiện từ S3**

* **Mô tả: Upload file nhưng Lambda không chạy**
* **Impact: Không xử lý được file tự động**
* **Root Cause: Thiếu quyền cho bucket gửi sự kiện đến Lambda**
* **Solution: Gán IAM policy cho S3 để invoke Lambda**
* **Result: Lambda nhận được trigger như mong đợi**
* **Lesson: S3 → Lambda cần thiết lập quyền invoke chính xác**

**Vấn đề 2: Lambda bị timeout khi xử lý file lớn**

* **Mô tả: Khi upload file > 5MB, Lambda chạy lâu rồi timeout**
* **Impact: Không xử lý được các file lớn**
* **Root Cause: Thời gian xử lý vượt quá timeout default 3 giây**
* **Solution: Tăng timeout lên 30 giây trong cấu hình Lambda**
* **Result: Lambda xử lý được file đến 10MB thành công**
* **Lesson: Cần tối ưu code hoặc tăng timeout hợp lý tùy use case**

**💭 Reflection & Insights**

**What went well today?**

* **Tự triển khai thành công hệ thống xử lý ảnh bằng serverless**
* **Nắm được cách kết hợp giữa Lambda, S3 và CloudWatch**
* **Tự viết hàm Python đơn giản nhưng hiệu quả cao**

**What could be improved?**

* **Nên tách logic thành nhiều function để dễ maintain hơn**
* **Cần tìm hiểu thêm cách sử dụng Layers để tái sử dụng thư viện**
* **Tập luyện việc kiểm soát chi phí với AWS Lambda khi gọi nhiều lần**

**Key Insights**

* **Lambda giúp thực hiện logic ngắn gọn mà không cần quản lý server**
* **Event-driven design là xu hướng tương lai trong kiến trúc cloud**
* **Kết hợp Lambda với S3, SNS, SQS... có thể tạo workflow mạnh mẽ và tự động hóa cao**

***Worklog created by: Tran Quang Trong*   
 *Next review: 21/05/2025***

**Worklog - Ngày 21/05/2025**

**📅 Thông tin cơ bản**

* **Ngày: 21/05/2025**
* **Thứ: Thứ Tư**
* **Tuần thực tập: Tuần thứ 2/12**
* **Thời gian làm việc: 9:00 - 17:00**
* **Mood: 💡 + Hào hứng vì đã deploy được REST API serverless đầu tiên**

**🎯 Mục tiêu ngày hôm nay**

* **Tìm hiểu cách hoạt động của Amazon API Gateway**
* **Tạo REST API sử dụng Lambda integration**
* **Test endpoint với các method GET, POST**
* **Deploy API Gateway và quan sát log qua CloudWatch**

**💼 Công việc đã thực hiện**

**1. Làm quen với Amazon API Gateway ⏱️ 1 giờ**

* **Mô tả: Tìm hiểu khái niệm REST API, stage, deployment, integration với Lambda**
* **Kết quả: Hiểu rõ cách API Gateway hoạt động như “cánh cửa” bảo vệ và routing request**
* **Tools/Tech: API Gateway Console, Docs**
* **Links: [API Gateway Overview](https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html)**

**2. Tạo Lambda function xử lý JSON request ⏱️ 1.5 giờ**

* **Mô tả: Viết một hàm nhận input JSON và trả về phản hồi JSON đơn giản**
* **Kết quả: Function hoạt động chính xác với event['body']**
* **Tools/Tech: Python, Lambda Console, Postman**
* **Code:**

**import json**

**def lambda\_handler(event, context):**

**data = json.loads(event['body'])**

**name = data.get('name', 'guest')**

**return {**

**'statusCode': 200,**

**'body': json.dumps({'message': f'Hello, {name}!'})**

**}**

**3. Kết nối API Gateway với Lambda ⏱️ 2 giờ**

* **Mô tả: Tạo REST API, cấu hình method POST, gán Lambda làm integration**
* **Kết quả: Triển khai API thành công, test bằng Postman trả về đúng kết quả**
* **Tools/Tech: API Gateway REST, Lambda Integration**
* **Endpoint Sample: https://abc123.execute-api.us-east-1.amazonaws.com/dev/hello**

**4. Enable logging và debug lỗi từ CloudWatch ⏱️ 1.5 giờ**

* **Mô tả: Cấu hình stage logging và bật tracing để theo dõi request**
* **Kết quả: Có thể xem chi tiết log mỗi lần gọi API → giúp dễ debug logic**
* **Tools/Tech: CloudWatch Logs, Execution Logs, Metrics**

**📚 Kiến thức học được**

**🔧 Technical Skills**

* **Tạo và deploy REST API bằng API Gateway**
* **Viết Lambda function xử lý request và trả về response JSON**
* **Kết hợp logging từ Lambda và API Gateway để debug**

**💡 Concepts & Theory**

* **API Gateway đóng vai trò là "entry point" bảo mật và route request**
* **Lambda integration dùng Proxy Mode giúp truyền toàn bộ request**
* **Stage Deployment giúp quản lý phiên bản API theo môi trường**

**🤝 Soft Skills**

* **Kỹ năng đọc log và tìm lỗi theo flow từ API → Lambda**
* **Tư duy theo hướng kiến trúc RESTful**
* **Biết cách cấu trúc input/output JSON cho frontend/backend dễ làm việc**

**🚧 Khó khăn và giải pháp**

**Vấn đề 1: Gửi request POST nhưng Lambda không nhận body**

* **Mô tả: Lambda event['body'] là None dù đã gửi JSON**
* **Impact: Không xử lý được dữ liệu người dùng**
* **Root Cause: API Gateway chưa bật Lambda Proxy integration**
* **Solution: Bật tùy chọn “Use Lambda Proxy Integration” trong Method**
* **Result: Lambda nhận đúng toàn bộ request từ client**
* **Lesson: Lambda Proxy giúp truyền toàn bộ header, body, path**

**Vấn đề 2: Không test được API vì lỗi CORS**

* **Mô tả: Gọi API từ frontend thì bị browser chặn**
* **Impact: Không thể tích hợp với ứng dụng web**
* **Root Cause: Chưa bật Access-Control-Allow-Origin**
* **Solution: Thêm CORS headers vào response của Lambda và enable CORS trong API Gateway**
* **Result: Gọi API thành công từ trình duyệt**
* **Lesson: CORS là vấn đề phổ biến cần xử lý kỹ trong API public**

**💭 Reflection & Insights**

**What went well today?**

* **Deploy thành công REST API serverless đầu tiên**
* **Viết được Lambda function trả lời theo input JSON**
* **Test thành công với nhiều công cụ (Postman, curl, web)**

**What could be improved?**

* **Nên tạo nhiều endpoint (GET, PUT, DELETE) để hoàn thiện RESTful**
* **Thử sử dụng OpenAPI để mô tả API rõ ràng hơn**
* **Tìm hiểu thêm về throttling, caching và authentication trong API Gateway**

**Key Insights**

* **API Gateway + Lambda là một giải pháp serverless mạnh mẽ để xây dựng backend**
* **Việc debug API đòi hỏi hiểu cách dữ liệu đi từ client → API Gateway → Lambda**
* **Serverless không chỉ giúp tiết kiệm chi phí mà còn tăng tốc độ triển khai hệ thống**

***Worklog created by: Tran Quang Trong*   
 *Next review: 26/05/2025***

**Worklog - Ngày 26/05/2025**

**📅 Thông tin cơ bản**

* **Ngày: 26/05/2025**
* **Thứ: Thứ Hai**
* **Tuần thực tập: Tuần thứ 3/12**
* **Thời gian làm việc: 9:00 - 17:00**
* **Mood: 📦 + Thích thú với tốc độ xử lý nhanh của DynamoDB**

**🎯 Mục tiêu ngày hôm nay**

* **Tìm hiểu kiến trúc và mô hình dữ liệu của Amazon DynamoDB**
* **Tạo bảng DynamoDB với partition key và sort key**
* **Tích hợp DynamoDB vào Lambda function**
* **Thực hiện các thao tác CRUD cơ bản (PutItem, GetItem, DeleteItem)**

**💼 Công việc đã thực hiện**

**1. Khám phá DynamoDB và mô hình NoSQL ⏱️ 1.5 giờ**

* **Mô tả: Đọc docs về kiến trúc key-value và bảng dữ liệu phân tán**
* **Kết quả: Hiểu rõ về partition key, sort key, read/write capacity**
* **Tools/Tech: DynamoDB Console, AWS Docs**
* **Links: [DynamoDB Overview](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Welcome.html)**

**2. Tạo bảng DynamoDB và thử PutItem ⏱️ 1 giờ**

* **Mô tả: Tạo bảng Users với userId là partition key**
* **Kết quả: Thêm dữ liệu người dùng thành công bằng console và AWS CLI**
* **Tools/Tech: AWS Console, DynamoDB, CLI**
* **Example:**

**{**

**"userId": "123",**

**"name": "Alice",**

**"email": "alice@example.com"**

**}**

**3. Tích hợp DynamoDB vào Lambda function ⏱️ 2.5 giờ**

* **Mô tả: Cập nhật Lambda function hôm qua để ghi nhận thông tin người dùng vào bảng Users**
* **Kết quả: Khi gọi POST API → dữ liệu được lưu vào DynamoDB**
* **Tools/Tech: Python (boto3), Lambda, IAM Role with DynamoDB access**
* **Links: IAM Policy: dynamodb:PutItem, dynamodb:GetItem**

**4. Thực hiện GetItem và DeleteItem qua Lambda ⏱️ 1.5 giờ**

* **Mô tả: Viết thêm 2 Lambda cho GET và DELETE endpoint → thao tác với bảng Users**
* **Kết quả: Có thể lấy thông tin user và xoá dữ liệu qua API Gateway**
* **Tools/Tech: Lambda Proxy Integration, API Gateway, DynamoDB SDK**

**📚 Kiến thức học được**

**🔧 Technical Skills**

* **Cách thiết kế bảng NoSQL cho truy vấn nhanh**
* **Tích hợp DynamoDB với Lambda thông qua SDK Boto3**
* **Tạo các hàm CRUD để tương tác toàn diện với bảng dữ liệu**

**💡 Concepts & Theory**

* **DynamoDB dùng partition key để phân mảnh dữ liệu theo vùng**
* **So sánh giữa provisioned vs on-demand capacity mode**
* **NoSQL không có schema nên cần định hình dữ liệu rõ từ đầu**

**🤝 Soft Skills**

* **Tư duy tổ chức dữ liệu theo truy vấn thay vì quan hệ**
* **Kỹ năng gỡ lỗi API → Lambda → DynamoDB theo dòng execution**
* **Ghi chú và comment code kỹ càng cho các thao tác database**

**🚧 Khó khăn và giải pháp**

**Vấn đề 1: Lambda bị lỗi “AccessDeniedException”**

* **Mô tả: Lambda không thể ghi vào bảng Users**
* **Impact: API trả về lỗi 500**
* **Root Cause: Thiếu quyền dynamodb:PutItem trong IAM Role**
* **Solution: Gán thêm policy vào execution role của Lambda**
* **Result: Lỗi biến mất, dữ liệu được lưu thành công**
* **Lesson: Cấu hình quyền chi tiết cho từng resource cụ thể là bắt buộc**

**Vấn đề 2: Dữ liệu trả về bị null khi dùng GetItem**

* **Mô tả: Gọi API nhưng không thấy user dù đã lưu**
* **Impact: Truy xuất sai → mất trải nghiệm người dùng**
* **Root Cause: Truy vấn sai partition key (dùng id thay vì userId)**
* **Solution: Sửa lại key trong lệnh get\_item() thành đúng tên**
* **Result: Truy vấn thành công**
* **Lesson: NoSQL rất nhạy cảm với key, cần đồng nhất giữa code và dữ liệu**

**💭 Reflection & Insights**

**What went well today?**

* **Tạo bảng DynamoDB và lưu được thông tin từ API**
* **Lambda tích hợp hoàn chỉnh với DynamoDB**
* **Cấu hình đúng IAM role để tránh lỗi truy cập**

**What could be improved?**

* **Cần học về Index (GSI, LSI) để hỗ trợ truy vấn nâng cao**
* **Nên thêm tính năng phân trang và scan toàn bộ bảng**
* **Viết thêm hàm cập nhật (UpdateItem) để hoàn thiện CRUD**

**Key Insights**

* **DynamoDB là lựa chọn lý tưởng cho serverless backend vì scale tốt và chi phí thấp**
* **Không giống như SQL, thiết kế bảng NoSQL phải xoay quanh truy vấn chính**
* **IAM, tên key, và region đều cần chính xác tuyệt đối khi dùng dịch vụ database**

***Worklog created by: Tran Quang Trong*   
 *Next review: 27/05/2025***

**Worklog - Ngày 27/05/2025**

**📅 Thông tin cơ bản**

* **Ngày: 27/05/2025**
* **Thứ: Thứ Ba**
* **Tuần thực tập: Tuần thứ 3/12**
* **Thời gian làm việc: 9:00 - 17:00**
* **Mood: 🧩 + Phấn khích khi hiểu rõ cấu trúc Index trong DynamoDB**

**🎯 Mục tiêu ngày hôm nay**

* **Tìm hiểu về Global Secondary Index (GSI) và Local Secondary Index (LSI)**
* **Tạo GSI cho bảng Users để truy vấn theo email**
* **Sử dụng Scan và kết hợp Pagination để lấy dữ liệu toàn bảng**
* **Viết thêm hàm tìm kiếm user theo email qua API**

**💼 Công việc đã thực hiện**

**1. Hiểu về GSI vs LSI trong DynamoDB ⏱️ 1 giờ**

* **Mô tả: Đọc tài liệu và ví dụ để phân biệt hai loại index phụ**
* **Kết quả: GSI độc lập với partition key chính, LSI dùng chung key nhưng sort khác**
* **Tools/Tech: AWS Docs, Console**
* **Links: [DynamoDB Index Guide](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/SecondaryIndexes.html)**

**2. Tạo GSI trên bảng Users để truy vấn theo email ⏱️ 2 giờ**

* **Mô tả: Tạo GSI tên EmailIndex với email là partition key**
* **Kết quả: Có thể query user theo địa chỉ email dễ dàng**
* **Tools/Tech: AWS Console, Python (boto3), Query API**
* **Code mẫu:**

**response = table.query(**

**IndexName="EmailIndex",**

**KeyConditionExpression=Key('email').eq('alice@example.com')**

**)**

**3. Sử dụng Scan kết hợp phân trang ⏱️ 2 giờ**

* **Mô tả: Thử lệnh scan bảng lớn và xử lý nhiều page kết quả**
* **Kết quả: Hiểu cách LastEvaluatedKey hoạt động và viết vòng lặp xử lý phân trang**
* **Tools/Tech: Boto3, Scan API, Pagination logic**
* **Code:**

**response = table.scan()**

**items = response['Items']**

**while 'LastEvaluatedKey' in response:**

**response = table.scan(ExclusiveStartKey=response['LastEvaluatedKey'])**

**items.extend(response['Items'])**

**4. Viết thêm hàm Lambda tìm user theo email ⏱️ 2 giờ**

* **Mô tả: Viết API GET /user?email= trả về thông tin user qua GSI**
* **Kết quả: Triển khai thành công, test bằng Postman với nhiều user khác nhau**
* **Tools/Tech: Lambda, API Gateway, DynamoDB GSI**
* **Output: JSON { "userId": "123", "email": "alice@example.com" }**

**📚 Kiến thức học được**

**🔧 Technical Skills**

* **Cấu hình và truy vấn GSI trong DynamoDB**
* **Hiểu rõ giới hạn của Scan và cách dùng Pagination**
* **Viết Lambda function filter dữ liệu bằng index**

**💡 Concepts & Theory**

* **GSI giúp tạo truy vấn linh hoạt, nhưng cần quản lý throughput hợp lý**
* **Scan không hiệu quả với bảng lớn → nên kết hợp với filter và phân trang**
* **LSI phải được tạo lúc tạo bảng, GSI có thể thêm sau đó**

**🤝 Soft Skills**

* **Tư duy tổ chức dữ liệu theo hướng dễ scale**
* **Biết đánh đổi giữa hiệu năng (query) và chi phí (GSI)**
* **Viết code dễ maintain, hỗ trợ cả query và phân trang**

**🚧 Khó khăn và giải pháp**

**Vấn đề 1: Không thể tạo LSI sau khi bảng đã tạo**

* **Mô tả: Cố gắng thêm LSI thì bị lỗi**
* **Impact: Không mở rộng được truy vấn như mong muốn**
* **Root Cause: LSI phải được định nghĩa khi khởi tạo bảng**
* **Solution: Sử dụng GSI thay thế, hoặc tạo lại bảng mới nếu cần**
* **Result: Tạo GSI thành công với chức năng tương đương**
* **Lesson: Cần thiết kế kỹ từ đầu khi dùng NoSQL**

**Vấn đề 2: Scan quá chậm với bảng nhiều dữ liệu**

* **Mô tả: Thời gian xử lý kéo dài khi bảng có >1000 bản ghi**
* **Impact: API phản hồi chậm, gây trải nghiệm kém**
* **Root Cause: Scan duyệt toàn bộ bảng, không có key condition**
* **Solution: Sử dụng Query thay vì Scan khi có thể, hoặc dùng paginated scan**
* **Result: API chạy nhanh hơn khi dùng đúng truy vấn**
* **Lesson: Hạn chế dùng Scan trong production backend**

**💭 Reflection & Insights**

**What went well today?**

* **Tạo được GSI và truy vấn user theo email mượt mà**
* **Hiểu sâu cách DynamoDB hoạt động khi bảng lớn**
* **API truy vấn hoạt động đúng, dễ dàng scale nếu có nhiều user**

**What could be improved?**

* **Cần luyện tập thêm về update/delete với condition expression**
* **Tìm hiểu thêm về TTL (Time to Live) và stream events từ DynamoDB**
* **Nên học cách backup và restore bảng lớn**

**Key Insights**

* **GSI giúp DynamoDB linh hoạt hơn nhiều so với tưởng tượng ban đầu**
* **Scan chỉ dùng khi thật sự cần, còn lại nên tối ưu hóa bằng Query**
* **DynamoDB tuy không có schema, nhưng cần tư duy thiết kế bảng kỹ càng từ đầu**

***Worklog created by: Tran Quang Trong*   
 *Next review: 29/05/2025***

**Worklog - Ngày 29/05/2025**

**📅 Thông tin cơ bản**

* **Ngày: 29/05/2025**
* **Thứ: Thứ Năm**
* **Tuần thực tập: Tuần thứ 3/12**
* **Thời gian làm việc: 9:00 - 17:00**
* **Mood: 🔄 + Phấn khích khi hiểu rõ event-based architecture trên DynamoDB**

**🎯 Mục tiêu ngày hôm nay**

* **Tìm hiểu về DynamoDB Streams và cách hoạt động**
* **Bật stream cho bảng Users**
* **Tạo Lambda xử lý dữ liệu khi có record mới/updated**
* **Kiểm thử hệ thống thông qua thao tác PutItem, UpdateItem**

**💼 Công việc đã thực hiện**

**1. Nắm rõ kiến trúc DynamoDB Streams ⏱️ 1 giờ**

* **Mô tả: Đọc tài liệu về cách hoạt động của Streams, các view: NEW\_IMAGE, OLD\_IMAGE, NEW\_AND\_OLD\_IMAGES**
* **Kết quả: Hiểu rõ cách ghi nhận các thay đổi của bảng DynamoDB vào stream**
* **Tools/Tech: AWS Docs, DynamoDB Console**
* **Links: [DynamoDB Streams Overview](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Streams.html)**

**2. Bật DynamoDB Stream cho bảng Users ⏱️ 1 giờ**

* **Mô tả: Cấu hình chế độ NEW\_AND\_OLD\_IMAGES để lấy dữ liệu cũ và mới khi record thay đổi**
* **Kết quả: Mỗi thay đổi trên bảng được ghi vào stream**
* **Tools/Tech: DynamoDB Console**

**3. Tạo Lambda để xử lý sự kiện thay đổi từ Stream ⏱️ 2.5 giờ**

* **Mô tả: Tạo Lambda function nhận event từ stream, log thông tin record mới/cũ**
* **Kết quả: Mỗi lần cập nhật hoặc thêm user → Lambda chạy và log ra dữ liệu**
* **Tools/Tech: Python (boto3), Lambda, CloudWatch Logs**
* **Code Snippet:**

**def lambda\_handler(event, context):**

**for record in event['Records']:**

**event\_name = record['eventName']**

**new\_image = record['dynamodb'].get('NewImage')**

**old\_image = record['dynamodb'].get('OldImage')**

**print(f"{event\_name} detected")**

**print("Old:", old\_image)**

**print("New:", new\_image)**

**4. Kiểm thử và quan sát log ⏱️ 2 giờ**

* **Mô tả: Thêm user, update user info → quan sát log từ Lambda**
* **Kết quả: Mỗi sự kiện đều được log chi tiết, phản hồi nhanh và chính xác**
* **Tools/Tech: Postman, API Gateway, CloudWatch Logs**
* **Output mẫu:**

**MODIFY detected**

**Old: {'userId': {'S': '123'}, 'email': {'S': 'old@example.com'}}**

**New: {'userId': {'S': '123'}, 'email': {'S': 'new@example.com'}}**

**📚 Kiến thức học được**

**🔧 Technical Skills**

* **Bật DynamoDB Streams và thiết lập cấu hình chính xác**
* **Tạo Lambda trigger tự động từ DynamoDB Stream**
* **Xử lý các loại sự kiện: INSERT, MODIFY, REMOVE trong Lambda**

**💡 Concepts & Theory**

* **Stream = chuỗi các log ghi nhận thay đổi dữ liệu**
* **Event-driven architecture giúp hệ thống phản ứng tức thời**
* **NEW\_AND\_OLD\_IMAGES phù hợp để so sánh thay đổi của dữ liệu**

**🤝 Soft Skills**

* **Khả năng gỡ lỗi log với nhiều trường hợp khác nhau**
* **Thiết kế hệ thống phi đồng bộ để giảm tải backend chính**
* **Tư duy hướng sự kiện giúp mở rộng hệ thống dễ dàng hơn**

**🚧 Khó khăn và giải pháp**

**Vấn đề 1: Lambda không tự kích hoạt**

* **Mô tả: Sau khi update dữ liệu, không thấy Lambda log chạy**
* **Impact: Mất khả năng phản ứng với thay đổi**
* **Root Cause: Chưa bật trigger DynamoDB Stream cho Lambda**
* **Solution: Thêm trigger thủ công, đảm bảo IAM đúng**
* **Result: Lambda hoạt động như mong đợi**
* **Lesson: Mỗi trigger cần được gắn rõ ràng, cần kiểm tra kỹ sau cấu hình**

**Vấn đề 2: Không phân biệt được loại sự kiện**

* **Mô tả: Khi log ra dữ liệu, không rõ là thêm mới hay chỉnh sửa**
* **Impact: Khó kiểm soát logic xử lý trong Lambda**
* **Root Cause: Không đọc đúng eventName từ record**
* **Solution: Thêm điều kiện if theo eventName**
* **Result: Log rõ ràng từng loại thao tác**
* **Lesson: Xử lý event-driven cần nhận diện đúng loại sự kiện để định hướng hành động**

**💭 Reflection & Insights**

**What went well today?**

* **Kết nối thành công DynamoDB Stream với Lambda**
* **Xử lý event INSERT, MODIFY ổn định và nhanh chóng**
* **Quan sát toàn bộ thay đổi của bảng một cách tự động**

**What could be improved?**

* **Cần phân loại logic rõ hơn trong Lambda cho từng event**
* **Tìm hiểu thêm cách xử lý event REMOVE (khi xoá dữ liệu)**
* **Nên thêm retry logic cho các trường hợp Lambda lỗi**

**Key Insights**

* **DynamoDB Stream mở ra khả năng realtime processing trên backend**
* **Lambda kết hợp stream phù hợp để audit, logging, trigger workflow**
* **Serverless event-driven là mô hình hiện đại, nhẹ, phản ứng nhanh và tiết kiệm tài nguyên**

***Worklog created by: Tran Quang Trong*   
 *Next review: 02/06/2025***

**Worklog - Ngày 02/06/2025**

**📅 Thông tin cơ bản**

* **Ngày: 02/06/2025**
* **Thứ: Thứ Hai**
* **Tuần thực tập: Tuần thứ 4/12**
* **Thời gian làm việc: 9:00 - 17:00**
* **Mood: ☁️ + Hào hứng với CI/CD đầu tiên trên AWS CodePipeline**

**🎯 Mục tiêu ngày hôm nay**

* **Tìm hiểu dịch vụ CodePipeline và CodeBuild**
* **Thiết lập pipeline đơn giản cho dự án web tĩnh**
* **Kết nối GitHub với CodePipeline**
* **Triển khai tự động lên S3 sau mỗi lần push code**

**💼 Công việc đã thực hiện**

**1. Khởi tạo CodePipeline từ GitHub đến S3 ⏱️ 2 giờ**

* **Mô tả: Tạo pipeline mới kết nối GitHub, CodeBuild và deploy lên S3**
* **Kết quả: Web tĩnh được cập nhật tự động mỗi lần push**
* **Tools/Tech: GitHub, CodePipeline, S3**

**2. Viết buildspec.yml cho CodeBuild ⏱️ 1 giờ**

* **Mô tả: Tạo file hướng dẫn build từ source code**
* **Kết quả: Tự động build, xử lý file và đẩy lên bucket**
* **Tools/Tech: YAML, AWS CodeBuild**

**3. Test và debug quá trình CI/CD ⏱️ 2 giờ**

* **Mô tả: Gỡ lỗi permissions, role IAM và trigger từ GitHub**
* **Kết quả: Code thay đổi → build + deploy nhanh chóng**
* **Tools/Tech: IAM Role, GitHub Webhook, CloudWatch Logs**

**📚 Kiến thức học được**

**🔧 Technical Skills**

* **Sử dụng CodePipeline để tự động hoá triển khai**
* **Viết buildspec.yml phù hợp cho ứng dụng tĩnh**
* **Gỡ lỗi IAM role và quyền CodeBuild**

**💡 Concepts & Theory**

* **CI/CD là phần quan trọng trong DevOps flow**
* **IAM role cần phân quyền rõ ràng cho từng stage**
* **Webhook từ GitHub sẽ trigger build tự động**

**🤝 Soft Skills**

* **Tư duy hệ thống theo từng giai đoạn pipeline**
* **Ghi log và phân tích quá trình để debug nhanh**
* **Quản lý project deployment một cách rõ ràng hơn**

**🚧 Khó khăn và giải pháp**

**Vấn đề 1: CodeBuild không có quyền ghi vào S3**

* **Mô tả: Build thành công nhưng deploy failed**
* **Impact: Không tự động hoá được**
* **Root Cause: Thiếu policy s3:PutObject**
* **Solution: Thêm policy vào IAM role cho CodeBuild**
* **Result: Triển khai tự động thành công**
* **Lesson: Gắn chính xác policy IAM là yếu tố quyết định**

**Vấn đề 2: Webhook GitHub không kích hoạt được**

* **Mô tả: Push code không kích hoạt pipeline**
* **Impact: Mất tính tự động**
* **Root Cause: Thiếu quyền OAuth khi kết nối GitHub**
* **Solution: Reconnect GitHub với quyền repo**
* **Result: Build auto chạy đúng mỗi push**
* **Lesson: Tích hợp CI/CD cần kiểm tra kỹ quyền API**

**💭 Reflection & Insights**

**What went well today?**

* **Thiết lập pipeline CI/CD đầu tiên hoàn chỉnh**
* **Code thay đổi được deploy lên môi trường production test**
* **Gỡ lỗi nhanh nhờ CloudWatch Logs**

**What could be improved?**

* **Nên thử tích hợp thêm test tự động vào stage build**
* **CI/CD cho Lambda function cần chuẩn bị sớm**
* **Nên log mỗi lần build thành công/thất bại**

**Key Insights**

* **CI/CD không chỉ tiết kiệm thời gian mà còn giảm lỗi thủ công**
* **IAM đúng là yếu tố then chốt trong pipeline hoạt động mượt**
* **Một pipeline tốt cần theo dõi logs chặt chẽ để tối ưu**

***Worklog created by: Tran Quang Trong*   
 *Next review: 03/06/2025***

**Worklog - Ngày 03/06/2025**

**📅 Thông tin cơ bản**

* **Ngày: 03/06/2025**
* **Thứ: Thứ Ba**
* **Tuần thực tập: Tuần thứ 4/12**
* **Thời gian làm việc: 9:00 - 17:00**
* **Mood: 🧠 + Tập trung cao độ để làm quen với AWS Step Functions**

**🎯 Mục tiêu ngày hôm nay**

* **Học về AWS Step Functions và cách hoạt động của state machine**
* **Tạo workflow đơn giản xử lý chuỗi Lambda functions**
* **Kết hợp Step Functions với DynamoDB và SNS**
* **Viết State Machine definition sử dụng Amazon States Language (ASL)**

**💼 Công việc đã thực hiện**

**1. Làm quen với AWS Step Functions ⏱️ 3 giờ**

* **Mô tả: Đọc tài liệu và hướng dẫn để hiểu state machine, transitions và task**
* **Kết quả: Hiểu cơ bản các trạng thái Task, Choice, Parallel, Wait**
* **Tools/Tech: AWS Step Functions, AWS Docs, State Visualizer**
* **Links: [Amazon States Language Guide](https://docs.aws.amazon.com/step-functions/latest/dg/concepts-amazon-states-language.html)**

**2. Tạo workflow xử lý Lambda ⏱️ 2 giờ**

* **Mô tả: Tạo state machine gồm 3 bước: validate → xử lý → thông báo**
* **Kết quả: Workflow chạy tuần tự chính xác theo thiết kế**
* **Tools/Tech: Lambda, Step Functions, CloudWatch Logs**
* **Code Snippet:**

**{**

**"StartAt": "ValidateData",**

**"States": {**

**"ValidateData": {**

**"Type": "Task",**

**"Resource": "arn:aws:lambda:validate",**

**"Next": "ProcessData"**

**},**

**"ProcessData": {**

**"Type": "Task",**

**"Resource": "arn:aws:lambda:process",**

**"Next": "Notify"**

**},**

**"Notify": {**

**"Type": "Task",**

**"Resource": "arn:aws:sns:notify",**

**"End": true**

**}**

**}**

**}**

**3. Tích hợp với DynamoDB và SNS ⏱️ 2 giờ**

* **Mô tả: Trong quá trình xử lý, ghi kết quả vào DynamoDB và gửi thông báo qua SNS**
* **Kết quả: Log được lưu, user nhận email khi hoàn tất**
* **Tools/Tech: Step Functions, Lambda, DynamoDB, SNS**
* **Links: AWS SDK, IAM Role cho Step Functions**

**📚 Kiến thức học được**

**🔧 Technical Skills**

* **Thiết kế và triển khai state machine với Step Functions**
* **Kết nối nhiều dịch vụ AWS thông qua workflow tự động**
* **Sử dụng ASL để định nghĩa logic workflow phức tạp**

**💡 Concepts & Theory**

* **AWS Step Functions hoạt động theo mô hình state machine**
* **Trạng thái như Choice, Parallel, Wait giúp kiểm soát logic chi tiết**
* **Mỗi task có thể gọi Lambda hoặc tích hợp với dịch vụ khác qua SDK**

**🤝 Soft Skills**

* **Tư duy logic theo hướng luồng xử lý tự động**
* **Kỹ năng thiết kế workflow có thể mở rộng trong tương lai**
* **Kỹ năng debug từng bước bằng log và visual execution graph**

**🚧 Khó khăn và giải pháp**

**Vấn đề 1: Không parse được kết quả từ Lambda**

* **Mô tả: State kế tiếp không nhận dữ liệu từ bước trước**
* **Impact: Workflow không chạy đúng logic**
* **Root Cause: Output từ Lambda không đúng format JSON**
* **Solution: Sửa lại return value theo chuẩn { "result": "OK" }**
* **Result: Workflow nhận đúng dữ liệu và tiếp tục xử lý**
* **Lesson: Phải kiểm soát input/output rõ ràng khi dùng Step Functions**

**Vấn đề 2: Step Functions không có quyền gọi SNS**

* **Mô tả: Bước thông báo thất bại với lỗi AccessDenied**
* **Impact: Không gửi được thông báo hoàn thành**
* **Root Cause: Thiếu quyền sns:Publish trong IAM role**
* **Solution: Thêm action sns:Publish vào policy cho Step Functions**
* **Result: Gửi thông báo thành công**
* **Lesson: Luôn kiểm tra kỹ IAM role cho từng dịch vụ được gọi trong workflow**

**💭 Reflection & Insights**

**What went well today?**

* **Hiểu được cơ chế hoạt động và thiết kế của Step Functions**
* **Triển khai thành công workflow đầu tiên có tương tác với Lambda + SNS**
* **Biết cách debug chi tiết từng bước trong flow**

**What could be improved?**

* **Cần luyện thêm với các trạng thái như Choice, Parallel**
* **Nên thử error handling flow bằng Catch, Retry**
* **Workflow nên tách nhỏ từng task ra file riêng để dễ maintain**

**Key Insights**

* **Step Functions là cách tuyệt vời để điều phối nhiều service AWS mà không cần orchestration code**
* **Phối hợp với các dịch vụ khác như SNS, DynamoDB cho phép workflow thực hiện tác vụ backend hoàn chỉnh**
* **Visual workflow và log giúp dễ theo dõi và debug hơn so với Lambda đơn lẻ**

***Worklog created by: Tran Quang Trong*   
 *Next review: 04/06/2025***

**Worklog - Ngày 04/06/2025**

**📅 Thông tin cơ bản**

* **Ngày: 04/06/2025**
* **Thứ: Thứ Tư**
* **Tuần thực tập: Tuần thứ 4/12**
* **Thời gian làm việc: 9:00 - 17:00**
* **Mood: ⚙️ + Hào hứng với khả năng tự động hoá workflow phức tạp**

**🎯 Mục tiêu ngày hôm nay**

* **Làm việc với các trạng thái nâng cao trong AWS Step Functions**
* **Thêm error handling vào workflow với Retry và Catch**
* **Tích hợp thêm Lambda xử lý logic phân nhánh với Choice**
* **Thực hành Parallel State để chạy đồng thời nhiều bước**

**💼 Công việc đã thực hiện**

**1. Retry và Catch trong Step Functions ⏱️ 2 giờ**

* **Mô tả: Thêm cơ chế thử lại và bắt lỗi cho các bước Lambda**
* **Kết quả: Workflow vẫn tiếp tục dù một bước gặp lỗi, giảm crash toàn hệ thống**
* **Tools/Tech: AWS Step Functions, Error Handling Patterns**
* **Links: [Error Handling in Step Functions](https://docs.aws.amazon.com/step-functions/latest/dg/handling-errors.html)**

**2. Xử lý phân nhánh với Choice ⏱️ 2 giờ**

* **Mô tả: Thêm trạng thái điều kiện để phân nhánh logic theo dữ liệu đầu vào**
* **Kết quả: Workflow có thể đi các hướng khác nhau tùy vào kết quả xử lý**
* **Tools/Tech: Amazon States Language, Choice Rules**
* **Code mẫu:**

**{**

**"Type": "Choice",**

**"Choices": [**

**{**

**"Variable": "$.status",**

**"StringEquals": "OK",**

**"Next": "NotifySuccess"**

**},**

**{**

**"Variable": "$.status",**

**"StringEquals": "ERROR",**

**"Next": "NotifyFailure"**

**}**

**]**

**}**

**3. Parallel State để xử lý đồng thời ⏱️ 2 giờ**

* **Mô tả: Triển khai Parallel state để chạy hai Lambda xử lý dữ liệu khác nhau cùng lúc**
* **Kết quả: Rút ngắn thời gian xử lý tổng thể, tận dụng tài nguyên tốt hơn**
* **Tools/Tech: Step Functions Parallel, Lambda**
* **Scenario: Một nhánh ghi log, một nhánh xử lý dữ liệu → hợp nhất kết quả cuối**

**📚 Kiến thức học được**

**🔧 Technical Skills**

* **Error handling: Retry, Catch, giới hạn lần thử và loại lỗi**
* **Branching logic với Choice, sử dụng biến trạng thái đầu vào**
* **Chạy song song task với Parallel để tăng hiệu năng**

**💡 Concepts & Theory**

* **AWS Step Functions hỗ trợ mô hình workflow phức tạp một cách rõ ràng và có kiểm soát**
* **Khả năng bắt lỗi từng bước giúp workflow ổn định hơn**
* **Phân nhánh cho phép thực thi logic tùy biến dựa trên giá trị trả về**

**🤝 Soft Skills**

* **Tư duy hệ thống khi thiết kế các luồng xử lý phân nhánh**
* **Kỹ năng tổ chức workflow logic thành mô hình dễ maintain**
* **Biết cách đọc log debug theo hướng trạng thái và theo dõi trace rõ ràng**

**🚧 Khó khăn và giải pháp**

**Vấn đề 1: Loop vô hạn do Retry sai cấu hình**

* **Mô tả: Một Lambda liên tục bị retry khiến chi phí tăng**
* **Impact: Gây tắc nghẽn workflow và vượt giới hạn chi phí**
* **Root Cause: Không giới hạn số lần retry và không catch lỗi cụ thể**
* **Solution: Thêm MaxAttempts và catch đúng loại lỗi cần xử lý**
* **Result: Workflow retry có kiểm soát và dừng đúng cách**
* **Lesson: Retry cần đi kèm với giới hạn và định nghĩa lỗi rõ ràng**

**Vấn đề 2: Trạng thái Choice sai cú pháp**

* **Mô tả: Step Functions báo lỗi khi deploy state machine**
* **Impact: Không thể tạo mới workflow**
* **Root Cause: Cấu trúc Choices thiếu biến đầu vào $**
* **Solution: Xem lại example trên docs và sửa chính xác Variable**
* **Result: Workflow deploy thành công, chạy đúng điều kiện**
* **Lesson: Định nghĩa logic trong Choice cần chính xác từng ký tự**

**💭 Reflection & Insights**

**What went well today?**

* **Đã làm chủ được nhiều trạng thái đặc biệt của Step Functions**
* **Có thể thiết kế workflow có phân nhánh và song song rõ ràng**
* **Debug lỗi state machine nhanh hơn nhờ đọc log chính xác**

**What could be improved?**

* **Nên thực hành thêm với State Map để lặp qua danh sách**
* **Cần thử tích hợp thêm với SQS hoặc EventBridge**
* **Documentation riêng cho từng state cần rõ hơn để tránh nhầm**

**Key Insights**

* **Step Functions mở rộng khả năng backend không cần server, dễ giám sát và bảo trì**
* **Tách logic theo workflow giúp dễ debug, dễ viết test case và mở rộng về sau**
* **Retry và Catch là công cụ hữu ích nhưng cần hiểu rõ để tránh phát sinh chi phí không mong muốn**

***Worklog created by: Tran Quang Trong*   
 *Next review: 11/06/2025***

**Worklog - Ngày 11/06/2025**

**📅 Thông tin cơ bản**

* **Ngày: 11/06/2025**
* **Thứ: Thứ Tư**
* **Tuần thực tập: Tuần thứ 5/12**
* **Thời gian làm việc: 9:00 - 17:00**
* **Mood: 🔄 + Hứng thú với luồng sự kiện bất đồng bộ trong hệ thống phân tán**

**🎯 Mục tiêu ngày hôm nay**

* **Làm quen với Amazon EventBridge và khái niệm sự kiện**
* **Tạo rule chuyển sự kiện từ S3 sang Lambda**
* **So sánh EventBridge với SNS, SQS**
* **Triển khai hệ thống nhỏ sử dụng event-driven architecture**

**💼 Công việc đã thực hiện**

**1. Làm quen EventBridge + Rule tạo sự kiện từ S3 ⏱️ 2.5 giờ**

* **Mô tả: Tạo rule EventBridge bắt sự kiện PutObject từ S3 và gọi Lambda**
* **Kết quả: Mỗi khi file mới được upload lên bucket → Lambda tự động xử lý**
* **Tools/Tech: Amazon EventBridge, S3, Lambda**
* **Links: [EventBridge S3 Integration](https://docs.aws.amazon.com/eventbridge/latest/userguide/eventbridge-s3.html)**

**2. Triển khai kiến trúc xử lý sự kiện bất đồng bộ ⏱️ 2 giờ**

* **Mô tả: Upload ảnh → trigger Lambda resize ảnh → gửi metadata tới SQS**
* **Kết quả: Kiến trúc phi đồng bộ, các bước tách biệt và có thể scale riêng**
* **Tools/Tech: Lambda, EventBridge, SQS**
* **Scenario: Phù hợp cho hệ thống xử lý ảnh/video real-time**

**3. So sánh EventBridge vs SNS vs SQS ⏱️ 1.5 giờ**

* **Mô tả: Tìm hiểu ưu nhược điểm từng dịch vụ pub/sub**
* **Kết quả: Hiểu rõ khi nào dùng push (SNS), queue (SQS), hay routing rule (EventBridge)**
* **Docs: AWS Whitepaper Messaging and Eventing**

**📚 Kiến thức học được**

**🔧 Technical Skills**

* **Cấu hình rule EventBridge trigger theo sự kiện cụ thể**
* **Thiết kế kiến trúc sự kiện kết hợp Lambda + SQS**
* **Quản lý message flow bất đồng bộ**

**💡 Concepts & Theory**

* **Event-driven architecture giúp giảm coupling và dễ scale**
* **EventBridge hỗ trợ routing rule theo điều kiện rất linh hoạt**
* **SNS là push-based, SQS là pull-based, EventBridge là rule-based**

**🤝 Soft Skills**

* **Phân tích hệ thống theo hướng async**
* **Đặt tên rule, pattern và queues có logic rõ ràng**
* **Hiểu giá trị của việc tách rời các service qua message bus**

**🚧 Khó khăn và giải pháp**

**Vấn đề 1: EventBridge không nhận sự kiện từ S3**

* **Mô tả: Đã cấu hình rule đúng nhưng không thấy Lambda được gọi**
* **Impact: Không thực hiện được trigger**
* **Root Cause: Bucket S3 chưa bật EventBridge notifications**
* **Solution: Vào bucket → Properties → Bật EventBridge Notifications**
* **Result: Sự kiện được gửi qua đúng rule, Lambda được gọi**
* **Lesson: Một số nguồn cần enable gửi sự kiện trước khi EventBridge nhận được**

**Vấn đề 2: Trễ sự kiện khi Lambda gửi metadata vào SQS**

* **Mô tả: Sau khi Lambda xử lý, SQS nhận message trễ vài giây**
* **Impact: Pipeline event mất tính tức thời**
* **Root Cause: Lambda xử lý lâu, không log khi gửi message**
* **Solution: Thêm log debug và tăng timeout Lambda**
* **Result: Message đến đều và đúng lúc hơn**
* **Lesson: Đối với hệ thống async, cần log từng bước rõ để trace flow**

**💭 Reflection & Insights**

**What went well today?**

* **Làm chủ được luồng dữ liệu bất đồng bộ trong hệ thống**
* **Sử dụng EventBridge rất đơn giản để routing event**
* **Workflow tách biệt giúp test và maintain từng phần độc lập**

**What could be improved?**

* **Cần học thêm cách debug EventBridge rule qua CloudWatch**
* **Nên tích hợp thêm Dead Letter Queue cho Lambda và SQS**
* **Tìm hiểu sâu hơn về Event Replay và Schema Registry**

**Key Insights**

* **Kiến trúc bất đồng bộ giúp hệ thống scale tốt và phục hồi lỗi hiệu quả hơn**
* **EventBridge là thành phần linh hoạt nhất trong hệ thống pub/sub**
* **Nên dùng kết hợp cả SNS, SQS và EventBridge tùy theo use-case thực tế để đạt hiệu quả tối ưu**

***Worklog created by: Tran Quang Trong*   
 *Next review: 20/06/2025***

**Worklog - Ngày 20/06/2025**

**📅 Thông tin cơ bản**

* **Ngày: 20/06/2025**
* **Thứ: Thứ Sáu**
* **Tuần thực tập: Tuần thứ 6/12**
* **Thời gian làm việc: 9:00 - 17:00**
* **Mood: 📈 + Tập trung vào giám sát và phản ứng tự động với CloudWatch**

**🎯 Mục tiêu ngày hôm nay**

* **Làm quen với Amazon CloudWatch và CloudWatch Logs**
* **Cấu hình log group và stream cho Lambda**
* **Thiết lập cảnh báo sử dụng CloudWatch Alarms**
* **Tự động phản hồi cảnh báo bằng SNS và Lambda**

**💼 Công việc đã thực hiện**

**1. Giám sát logs của Lambda với CloudWatch Logs ⏱️ 2 giờ**

* **Mô tả: Kiểm tra CloudWatch log stream được tạo tự động từ Lambda function**
* **Kết quả: Có thể theo dõi chi tiết hoạt động, lỗi và độ trễ của Lambda**
* **Tools/Tech: AWS Lambda, CloudWatch Logs, Log Stream**
* **Links: [CloudWatch Logs for Lambda](https://docs.aws.amazon.com/lambda/latest/dg/monitoring-cloudwatchlogs.html)**

**2. Thiết lập CloudWatch Alarms trên chỉ số Lambda ⏱️ 2 giờ**

* **Mô tả: Tạo alarm cảnh báo khi Lambda lỗi vượt ngưỡng hoặc thời gian xử lý tăng cao**
* **Kết quả: Có cảnh báo gửi về qua email khi chức năng bị lỗi hoặc quá tải**
* **Tools/Tech: CloudWatch Alarms, Metric, Lambda Error Rate**
* **Trigger: Nếu Errors > 1 trong 5 phút → gửi cảnh báo**

**3. Tích hợp CloudWatch Alarm với SNS và Lambda phản ứng ⏱️ 2 giờ**

* **Mô tả: Khi alarm kích hoạt → SNS gửi thông báo → Lambda ghi log vào DynamoDB**
* **Kết quả: Hệ thống phản ứng tự động khi xảy ra sự cố**
* **Tools/Tech: SNS, CloudWatch, Lambda, DynamoDB**
* **Scenario: Ứng dụng cho hệ thống giám sát sự cố production**

**📚 Kiến thức học được**

**🔧 Technical Skills**

* **Giám sát logs của Lambda với CloudWatch**
* **Tạo Alarm theo chỉ số hệ thống**
* **Kết nối CloudWatch Alarm với SNS topic và Lambda**

**💡 Concepts & Theory**

* **CloudWatch là trung tâm giám sát logs, metrics, alarms**
* **SNS giúp truyền thông báo theo nhiều kênh (email, HTTP, Lambda...)**
* **Automation response giúp giảm thời gian xử lý sự cố thủ công**

**🤝 Soft Skills**

* **Xử lý logic cảnh báo rõ ràng và phản hồi tự động**
* **Hiểu tư duy observability và root cause analysis**
* **Quản lý dashboard giám sát để theo dõi toàn bộ hệ thống**

**🚧 Khó khăn và giải pháp**

**Vấn đề 1: Không thấy log từ Lambda trong CloudWatch**

* **Mô tả: Log group không được tạo tự động như mong đợi**
* **Impact: Không thể debug chức năng Lambda**
* **Root Cause: IAM Role gán cho Lambda chưa có quyền logs:CreateLogGroup**
* **Solution: Thêm chính sách đầy đủ quyền ghi log**
* **Result: Log group và stream hoạt động đúng**
* **Lesson: Phân quyền log cần cấu hình chính xác cho observability**

**Vấn đề 2: SNS không gửi thông báo email**

* **Mô tả: Đã cấu hình SNS topic nhưng không thấy nhận được email**
* **Impact: Cảnh báo không đến người chịu trách nhiệm**
* **Root Cause: Email chưa xác nhận (confirm subscription)**
* **Solution: Mở email, xác nhận subscription từ liên kết AWS gửi**
* **Result: Email được nhận đầy đủ sau khi xác thực**
* **Lesson: Cần kiểm tra xác nhận khi dùng SNS gửi email**

**💭 Reflection & Insights**

**What went well today?**

* **Thiết lập thành công cảnh báo và phản ứng tự động**
* **Có thể giám sát hoạt động Lambda theo thời gian thực**
* **CloudWatch Alarms giúp phát hiện lỗi nhanh chóng**

**What could be improved?**

* **Cần luyện thêm tạo dashboard tổng hợp từ nhiều nguồn**
* **Thử tích hợp cảnh báo với Slack hoặc Webhook**
* **Tối ưu hóa lại thời gian lấy metrics để tiết kiệm chi phí CloudWatch**

**Key Insights**

* **Monitoring là một phần thiết yếu trong vận hành hệ thống AWS**
* **CloudWatch không chỉ giám sát mà còn giúp phản ứng kịp thời qua SNS và Lambda**
* **Tích hợp các dịch vụ đơn giản có thể tạo ra một hệ thống giám sát hoàn toàn tự động và hiệu quả**

***Worklog created by: Tran Quang Trong*   
 *Next review: 25/06/2025***

**Worklog - Ngày 25/06/2025**

**📅 Thông tin cơ bản**

* **Ngày: 25/06/2025**
* **Thứ: Thứ Tư**
* **Tuần thực tập: Tuần thứ 7/12**
* **Thời gian làm việc: 9:00 - 17:00**
* **Mood: 🧠 + Tò mò và hứng thú khi khám phá cách AWS theo dõi ứng dụng đa tầng**

**🎯 Mục tiêu ngày hôm nay**

* **Tìm hiểu cơ bản về AWS X-Ray**
* **Kích hoạt X-Ray tracing cho Lambda và API Gateway**
* **Phân tích trace để debug lỗi latency trong hệ thống**
* **Trải nghiệm biểu đồ service map của X-Ray**

**💼 Công việc đã thực hiện**

**1. Kích hoạt X-Ray trên Lambda + API Gateway ⏱️ 2 giờ**

* **Mô tả: Mở chế độ tracing cho Lambda function và tích hợp API Gateway**
* **Kết quả: Có thể theo dõi toàn bộ hành trình request từ API đến backend**
* **Tools/Tech: AWS Lambda, API Gateway, X-Ray, IAM**
* **Links: [Enable X-Ray Tracing](https://docs.aws.amazon.com/lambda/latest/dg/services-xray.html)**

**2. Phân tích trace detail và xử lý lỗi độ trễ ⏱️ 2 giờ**

* **Mô tả: Chạy các request giả lập và kiểm tra trace timeline**
* **Kết quả: Xác định Lambda mất thời gian do gọi external API chậm**
* **Tools/Tech: AWS X-Ray Console, Lambda logs, Mock APIs**
* **Output: Phân tích latency per segment → phát hiện điểm nghẽn**

**3. Quan sát và phân tích Service Map ⏱️ 2 giờ**

* **Mô tả: Xem sơ đồ các service backend giao tiếp với nhau qua X-Ray**
* **Kết quả: Có góc nhìn tổng thể về các thành phần đang gọi qua lại**
* **Tools/Tech: AWS X-Ray Service Map, CloudWatch**
* **Scenario: Phù hợp để theo dõi hệ thống nhiều Lambda, API, DB**

**📚 Kiến thức học được**

**🔧 Technical Skills**

* **Bật tracing cho API Gateway và Lambda**
* **Phân tích chi tiết từng segment trong X-Ray**
* **Xác định nguyên nhân gây latency hoặc lỗi hệ thống**

**💡 Concepts & Theory**

* **Distributed tracing là kỹ thuật thiết yếu cho microservices**
* **Mỗi trace gồm nhiều segment đại diện cho các thành phần của hệ thống**
* **X-Ray giúp visual hóa hệ thống từ góc nhìn end-to-end**

**🤝 Soft Skills**

* **Tư duy theo luồng dữ liệu → hiểu hệ thống vận hành như thế nào**
* **Đọc biểu đồ trace và giải thích nguyên nhân lỗi**
* **Sử dụng công cụ giám sát để phục vụ debugging chính xác**

**🚧 Khó khăn và giải pháp**

**Vấn đề 1: Không thấy trace xuất hiện trong X-Ray**

* **Mô tả: Dù đã bật tracing, vẫn không thấy data**
* **Impact: Không thể theo dõi request**
* **Root Cause: IAM Role chưa cho phép xray:PutTraceSegments**
* **Solution: Thêm quyền vào IAM Role của Lambda và API Gateway**
* **Result: Trace hoạt động bình thường, hiển thị đầy đủ**
* **Lesson: Khi dùng dịch vụ nào cần đảm bảo quyền tương ứng đã được cấp**

**Vấn đề 2: Trace bị rời rạc, không gắn kết với full flow**

* **Mô tả: Một số trace chỉ hiển thị Lambda, không có API Gateway hoặc các bước khác**
* **Impact: Khó hình dung luồng request tổng thể**
* **Root Cause: Header trace không được truyền qua các tầng đúng cách**
* **Solution: Đảm bảo truyền trace ID trong headers, và các function đều bật tracing**
* **Result: Trace đầy đủ, bao gồm cả API Gateway, Lambda, và DB**
* **Lesson: Tracing end-to-end cần sự đồng bộ giữa tất cả các thành phần**

**💭 Reflection & Insights**

**What went well today?**

* **Hiểu và cấu hình được AWS X-Ray cho môi trường Lambda**
* **Biết cách đọc trace để tìm bottleneck trong hệ thống**
* **Service Map giúp thấy toàn bộ hệ thống hoạt động ra sao trong thời gian thực**

**What could be improved?**

* **Nên học thêm cách export trace để phân tích ngoài**
* **Thử áp dụng X-Ray cho ứng dụng containerized (ECS/Fargate)**
* **Cần tối ưu quyền IAM để không mở quá rộng**

**Key Insights**

* **X-Ray là công cụ mạnh mẽ để quan sát, đo lường và debug hệ thống phân tán**
* **Tracing giúp tiết kiệm thời gian khi cần xử lý sự cố latency hoặc lỗi logic**
* **Kết hợp CloudWatch + X-Ray sẽ tạo nên hệ thống giám sát toàn diện cho hệ thống serverless**

***Worklog created by: Tran Quang Trong*   
 *Next review: 30/06/2025***

**Worklog - Ngày 30/06/2025**

**📅 Thông tin cơ bản**

* **Ngày: 30/06/2025**
* **Thứ: Thứ Hai**
* **Tuần thực tập: Tuần thứ 8/12**
* **Thời gian làm việc: 9:00 - 17:00**
* **Mood: 🛠️ + Rất hứng thú khi điều khiển EC2 mà không cần SSH**

**🎯 Mục tiêu ngày hôm nay**

* **Tìm hiểu về AWS Systems Manager và Session Manager**
* **Truy cập EC2 không cần key pair bằng Session Manager**
* **Thực hành với Run Command và Automation Documents**
* **Lưu trữ thông tin cấu hình trong Parameter Store**

**💼 Công việc đã thực hiện**

**1. Kết nối EC2 bằng Session Manager ⏱️ 2 giờ**

* **Mô tả: Cấu hình IAM Role và SSM Agent để truy cập EC2 instance qua AWS Console**
* **Kết quả: Điều khiển máy chủ từ xa không cần SSH/key pair**
* **Tools/Tech: EC2, SSM Agent, IAM Role, Session Manager**
* **Links: [SSM Session Manager Guide](https://docs.aws.amazon.com/systems-manager/latest/userguide/session-manager.html)**

**2. Thực thi Run Command trên nhiều EC2 ⏱️ 1.5 giờ**

* **Mô tả: Gửi lệnh shell đến nhiều instance cùng lúc qua SSM**
* **Kết quả: Tự động cài đặt phần mềm mà không cần login**
* **Tools/Tech: Systems Manager → Run Command, AWS CLI**
* **Ví dụ lệnh: sudo yum update -y**

**3. Sử dụng Parameter Store để quản lý biến cấu hình ⏱️ 2 giờ**

* **Mô tả: Tạo các tham số dạng SecureString để lưu thông tin nhạy cảm**
* **Kết quả: EC2 và Lambda có thể truy xuất biến cấu hình một cách an toàn**
* **Tools/Tech: Parameter Store, IAM, Lambda Environment**
* **Output: Truy xuất thông số DB\_PASSWORD từ ứng dụng**

**4. Tìm hiểu Automation Documents ⏱️ 1.5 giờ**

* **Mô tả: Khám phá các tài liệu sẵn có để automate task như restart EC2, patching**
* **Kết quả: Thấy tiềm năng trong việc tự động hóa vận hành**
* **Tools/Tech: AWS Systems Manager Automation**
* **Links: [SSM Automation Docs Library](https://docs.aws.amazon.com/systems-manager/latest/userguide/automation-documents-reference.html)**

**📚 Kiến thức học được**

**🔧 Technical Skills**

* **Truy cập EC2 mà không cần SSH**
* **Thực thi hàng loạt tác vụ từ xa qua Run Command**
* **Quản lý cấu hình với Parameter Store an toàn và có versioning**

**💡 Concepts & Theory**

* **SSM là công cụ quản lý hệ thống tập trung và bảo mật**
* **Session Manager giúp tránh mở cổng SSH — tăng bảo mật**
* **Automation Documents là cách để viết "playbook" tự động hóa**

**🤝 Soft Skills**

* **Tư duy tự động hóa quy trình quản trị hệ thống**
* **Hiểu tầm quan trọng của bảo mật truy cập từ xa**
* **Quản lý biến môi trường tốt giúp ứng dụng ổn định và dễ scale**

**🚧 Khó khăn và giải pháp**

**Vấn đề 1: EC2 không hiện trong Session Manager**

* **Mô tả: EC2 không connect được dù đã cài SSM Agent**
* **Impact: Không thể truy cập từ xa**
* **Root Cause: IAM Role chưa có quyền ssm:StartSession**
* **Solution: Gắn đúng IAM Role có quyền SSM cho EC2**
* **Result: EC2 có thể điều khiển từ Session Manager**
* **Lesson: Session Manager phụ thuộc vào cả agent, IAM và network**

**Vấn đề 2: Không đọc được biến từ Parameter Store**

* **Mô tả: Lambda không truy cập được SecureString**
* **Impact: Lỗi runtime do thiếu biến cấu hình**
* **Root Cause: IAM Role của Lambda chưa có quyền truy xuất Parameter Store**
* **Solution: Thêm quyền ssm:GetParameter + kms:Decrypt**
* **Result: Ứng dụng hoạt động bình thường**
* **Lesson: Biến cấu hình phải đi kèm quyền chính xác để đảm bảo bảo mật**

**💭 Reflection & Insights**

**What went well today?**

* **Lần đầu điều khiển EC2 mà không cần SSH rất mượt mà**
* **Parameter Store giúp quản lý secrets cực kỳ sạch và hiệu quả**
* **Run Command giúp tiết kiệm thời gian cài đặt hàng loạt**

**What could be improved?**

* **Nên thử tích hợp SSM Automation với CloudWatch Events**
* **Học cách viết Automation Document tùy chỉnh**
* **Cần quản lý tốt hơn các permission giữa các component**

**Key Insights**

* **AWS Systems Manager là một công cụ cực kỳ hữu ích cho DevOps và SysAdmin**
* **Truy cập EC2 không SSH = bảo mật hơn + dễ audit hơn**
* **Automation càng sớm thì vận hành hệ thống càng ổn định và ít lỗi**

***Worklog created by: Tran Quang Trong*   
 *Next review: 02/07/2025***

**Worklog - Ngày 02/07/2025**

**📅 Thông tin cơ bản**

* **Ngày: 02/07/2025**
* **Thứ: Thứ Tư**
* **Tuần thực tập: Tuần thứ 9/12**
* **Thời gian làm việc: 9:00 - 17:00**
* **Mood: 🔄 + Thích thú khi thấy pipeline tự động chạy từ git push đến deploy**

**🎯 Mục tiêu ngày hôm nay**

* **Tạo repository trên CodeCommit**
* **Thiết lập CodeBuild để build web tĩnh**
* **Kết nối CodePipeline để tự động deploy lên S3**
* **Tìm hiểu cơ bản cách viết buildspec.yml**

**💼 Công việc đã thực hiện**

**1. Tạo repository CodeCommit ⏱️ 1 giờ**

* **Mô tả: Tạo repo riêng cho ứng dụng web tĩnh (HTML/CSS/JS)**
* **Kết quả: Source code đã được push lên repo riêng trên AWS**
* **Tools/Tech: CodeCommit, Git CLI**
* **Link: [AWS CodeCommit Quickstart](https://docs.aws.amazon.com/codecommit/latest/userguide/getting-started.html)**

**2. Cấu hình CodeBuild và viết buildspec.yml ⏱️ 2 giờ**

* **Mô tả: Tạo project CodeBuild build mã nguồn và sync lên S3**
* **Kết quả: Build thành công và file đã xuất hiện trên S3**
* **Tools/Tech: CodeBuild, S3, IAM Role**
* **Code mẫu:**

**version: 0.2**

**phases:**

**build:**

**commands:**

**- aws s3 sync . s3://my-static-site-bucket --delete**

**artifacts:**

**files:**

**- '\*\*/\*'**

**3. Thiết lập CodePipeline ⏱️ 2 giờ**

* **Mô tả: Tạo pipeline tự động trigger từ CodeCommit → CodeBuild → Deploy lên S3**
* **Kết quả: Mỗi lần git push sẽ kích hoạt build và deploy tự động**
* **Tools/Tech: CodePipeline, CodeBuild, CodeCommit, S3**
* **Output: Pipeline chạy hoàn chỉnh, deploy nhanh chóng**

**4. Tối ưu hóa IAM Policy và kiểm tra các quyền ⏱️ 2 giờ**

* **Mô tả: Kiểm tra kỹ quyền của các service role dùng trong pipeline**
* **Kết quả: Đảm bảo pipeline không bị lỗi permission denied**
* **Tools/Tech: IAM Policies, Trust Relationships**

**📚 Kiến thức học được**

**🔧 Technical Skills**

* **Thiết lập hệ thống CI/CD cơ bản với CodeCommit → CodeBuild → CodePipeline**
* **Viết file buildspec.yml để định nghĩa quá trình build**
* **Deploy tự động lên S3 mỗi lần push code**

**💡 Concepts & Theory**

* **CI/CD giúp tự động hóa quy trình phát triển và triển khai**
* **Mỗi service cần IAM Role riêng biệt để phân quyền đúng**
* **Artifacts và buildspec giúp kiểm soát pipeline hiệu quả**

**🤝 Soft Skills**

* **Tư duy tổ chức quy trình phát triển theo pipeline**
* **Kiểm tra kỹ quyền của từng service → giảm lỗi runtime**
* **Phân tích và debug lỗi pipeline bằng log từ CodeBuild**

**🚧 Khó khăn và giải pháp**

**Vấn đề 1: CodeBuild không thể sync file lên S3**

* **Mô tả: Lỗi “Access Denied” khi chạy aws s3 sync**
* **Impact: Build thành công nhưng deploy thất bại**
* **Root Cause: IAM Role của CodeBuild thiếu quyền S3**
* **Solution: Thêm quyền s3:PutObject, s3:DeleteObject, s3:ListBucket**
* **Result: File đã được sync lên đúng bucket**
* **Lesson: Luôn kiểm tra kỹ chính sách IAM cho từng dịch vụ**

**Vấn đề 2: CodePipeline không trigger sau khi git push**

* **Mô tả: Push code lên CodeCommit nhưng pipeline không khởi chạy**
* **Impact: Không có quy trình tự động → phải chạy tay**
* **Root Cause: Chưa cấu hình đúng trigger trong pipeline**
* **Solution: Bật trigger trong phần Source (CodeCommit) của pipeline**
* **Result: Mỗi lần push đều tự động trigger build**
* **Lesson: CodePipeline cần cấu hình đúng trigger để hoạt động mượt**

**💭 Reflection & Insights**

**What went well today?**

* **Tự tay tạo pipeline hoàn chỉnh là một milestone lớn**
* **Việc push code rồi thấy hệ thống deploy tự động rất thú vị**
* **Tìm hiểu buildspec.yml giúp kiểm soát chính xác quá trình build**

**What could be improved?**

* **Cần học thêm về việc deploy lên Elastic Beanstalk/ECS thay vì chỉ S3**
* **Có thể thêm bước test tự động vào pipeline**
* **Cải thiện buildspec.yml để dùng nhiều stage hơn**

**Key Insights**

* **CI/CD là phần không thể thiếu trong DevOps hiện đại**
* **AWS cung cấp trọn bộ dịch vụ cho pipeline mà không cần tool bên ngoài**
* **Sự nhất quán và tự động hóa giúp team phát triển ổn định và an toàn hơn**

***Worklog created by: Tran Quang Trong*   
 *Next review: 04/07/2025***