**Revised Course Outline: TypeScript to Cloud Training with AI Integration**

**Duration**: 15 Full Days

|  |  |  |
| --- | --- | --- |
| **Module** | **Topic Areas** | **AI Integration (Where Relevant)** |
| **TypeScript Fundamentals** | Types, interfaces, enums, generics, type inference | Building type-safe AI data models (e.g., NLP result interfaces, prediction output schemas) |
| **TypeScript for Backend Developers** | Type guards, utility types, declaration merging | Validating and shaping AI response payloads using advanced types |
| **Migration from JavaScript** | Refactoring Node.js apps to TypeScript, tsconfig, ts-node, module resolution | Migrating AI pipelines and inference services written in JavaScript to TypeScript |
| **Core Patterns** | Middleware, dependency injection, async error handling | DI for AI modules (e.g., ML service clients), async AI job queues |
| **Advanced Node.js Backend Development** | RESTful APIs with Express & TypeScript, validation with Zod/Joi | Designing AI-driven endpoints (e.g., recommendation, classification APIs) |
| **Logging & Monitoring** | Winston, Morgan, structured logging | Logging AI inference duration, input/output payloads for observability |
| **Testing & TDD** | Jest, Supertest, mocking services, code coverage | Unit tests for AI service wrappers and mocking external AI responses |
| **Microservices Principles** | Bounded contexts, API gateways, service isolation | Designing microservices that consume or expose AI functionalities |
| **Microservices Architecture** | REST, Pub/Sub, Event-Driven (SNS, EventBridge) | Event-driven AI pipelines (e.g., image processing, fraud detection triggered via SQS/EventBridge) |
| **Resilience & Security** | JWT, retries, circuit breakers, timeouts | Secure and fault-tolerant AI API consumption |
| **AWS Lambda & Serverless Node.js** | Function handlers, execution lifecycle, IAM roles | Creating lightweight AI inference functions (e.g., using AWS Bedrock, SageMaker endpoint calls) |
| **Serverless Framework** | serverless.yml, deployment, local testing | Deploying AI-backed Lambda services |
| **Triggers & Events** | API Gateway, DynamoDB Streams, SQS, EventBridge | AI workflows triggered on event streams (e.g., real-time sentiment analysis on message events) |
| **Monitoring** | CloudWatch Logs, X-Ray tracing | Monitoring latency and error rates of AI service integrations |
| **AWS DynamoDB (NoSQL Backend)** | Tables, items, partition/sort keys, indexes | Storing AI results and input/output logs efficiently in a NoSQL setup |
| **Access Patterns** | AWS SDK v3, marshalling/unmarshalling, pagination | Managing batch AI result retrieval and pagination |
| **Best Practices** | Cost optimization, throttling, schema design | Optimizing storage of AI prediction results for scale |
| **Service Composition** | Lambda + API Gateway + DynamoDB integrations | Designing complete AI-driven serverless pipelines |
| **Cloud-Native Backend Engineering** | GitHub Actions, Serverless Framework, AWS CDK basics | CI/CD for AI microservices and automated model deployment |
| **CI/CD & IaC** | Infrastructure as Code and GitOps | Automating AI service deployment pipelines (e.g., retraining jobs via GitHub Actions) |
| **Observability** | Logs, metrics, alerts, tracing with CloudWatch | AI-specific metrics (e.g., model response time, error rates) |
| **Performance & Cost** | Cold start optimization, reserved concurrency | Optimizing performance for AI models served on-demand via Lambda |