## Network Troubleshooter Agent Team

## Abstract

BlueCom, a telecom operator serving over 10 million customers across Middle Eastern countries, faces challenges in resolving customer service tickets within SLA targets due to high ticket volumes and rapidly evolving technology. Manual processes, where engineers sift through historical tickets and on-premise technical manuals, lead to prolonged Mean Time to Resolve (MTTR) and SLA breaches.

The BlueCom Network Troubleshooter addresses this with a two-agent system: Agent 1, a cloud-based AutoGen agent leveraging Weaviate for real-time ticket resolution via Retrieval-Augmented Generation (RAG), and Agent 2, an on-premise LangGraph fallback agent querying technical manuals when Agent 1's confidence is low. This solution reduces MTTR by 50%, increases first-contact resolution by 30%, keeps SLA breaches below 1%, and ensures fallback success coverage of at least 95%.

Background -

BlueCom operates a customer service interface where customers and service executives raise tickets for issues like "latency spikes on router X." Service engineers aim to resolve tickets within 24 hours, but high ticket volumes and reliance on manual searches through historical tickets and technical manuals result in delays, increasing MTTR and breaching SLAs. Key challenges include::

* **High Ticket Volume**: Overwhelms Tier-1 support, delaying resolutions.
* **Complex Technology**: Rapidly changing telecom infrastructure complicates troubleshooting.
* **Manual Processes**: Engineers manually search historical tickets and on-premise manuals, slowing resolution.
* **SLA Compliance**: Delays lead to frequent SLA breaches, impacting customer satisfaction.

To address these, BlueCom requires an automated Network Troubleshooter that rapidly ingests tickets, retrieves relevant historical data, and proposes solutions, enabling engineers to focus on complex issues and improving SLA compliance.

## Objective

* Build an AI-powered Network Troubleshooter to automatically analyze customer tickets and propose accurate resolutions.
* Leverage Weaviate with Google Cloud for sub-second similarity search over historical tickets and technical manuals.
* Achieve:
  + 50% reduction in MTTR.
  + 30% increase in first-contact resolution.
  + SLA breach rate below 1%.
  + Fallback success coverage of at least 95%.

# Problem Statement

BlueCom's service engineers struggle with:

1. High MTTR due to manual ticket resolution processes.
2. Low first-contact resolution rates from inconsistent access to relevant historical data.
3. SLA breaches caused by delays in handling ticket surges.
4. Limited scalability of on-premise manual searches during peak demand.

**Your task:** Design and implement a dual-agent Network Troubleshooter system that:

* Ingests and indexes ticket datasets (src\_incident\_records.csv, src\_tech\_records.csv) and technical manuals (metadata\_incident\_records.csv, metadata\_tech\_records.csv).
* Uses Weaviate on Google Cloud for vector storage and retrieval of ticket and manual embeddings.
* Agent 1 performs similarity search on ProblemDescription embeddings from src\_incident\_records.csv. (implemented using Autogen/Semantic Kernel -- to be deployed on Google Cloud)
* Agent 2 performs fallback similarity search on step\_description embeddings from src\_tech\_records.csv if Agent 1's response lacks accuracy. (implemented using LangGraph -- to be deployed locally/Google Cloud).
* Implements Agent 2 Agent Communication between two agents.
* Deploys on Cloud Functions/GKE (Agent 1) and on-premise (Agent 2).
* Monitors metrics (accuracy, latency, fallback rate, token usage) via Cloud Monitoring and Cloud Logging.

# Data

|  |  |  |
| --- | --- | --- |
| Dataset | Records | Purpose |
| src\_incident\_records.csv | 50 incident tickets with ProductID and ProblemDescription | Source tickets for RAG retrieval and evaluation |
| metadata\_incident\_records.csv | 50 enriched incident records with solutions, tags, timestamps | Ground-truth corpus with resolution details |
| src\_tech\_records.csv | 50 technical documentation steps | Technical knowledge base for resolution guidance |
| metadata\_tech\_records.csv | 50 enriched technical records with solution steps and tags | Comprehensive technical documentation corpus |

**1.** **Source Incident Records Dataset (src\_incident\_records.csv)**

This dataset contains 50 customer incident tickets with basic information including unique ticket identifiers, product identifiers, and detailed problem descriptions from customer service interactions.

**Data Dictionary - Source Records:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Column Name** | **Data Type** | **Description** | **Sample Value** |
| TicketID | String | Unique identifier for each incident ticket | "TKT-001", "TKT-002" |
| ProductID | String | Product/service identifier related to the incident | "RTR-X1000", "SWT-Core-01" |
| ProblemDescription | String | Detailed description of the customer-reported issue | "Experiencing latency spikes on router X during peak hours" |

**2. Metadata Incident Records Dataset (metadata\_incident\_records.csv)**

This enriched dataset contains 50 incident records with comprehensive resolution information, including customer details, product information, solution details, status tracking, and metadata tags.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column Name** | **Data Type** | **Description** | **Sample Value** |
| TicketID | String | Unique ticket identifier | "TKT-001" |
| CustomerID | String | Anonymous customer identifier | "CUST-12345" |
| ProductID | String | Product/service identifier | "RTR-X1000" |
| ProductInformation | String | Detailed product specifications and context | "Cisco Router X1000, Firmware v2.1.4" |
| SolutionDetails | String | Comprehensive resolution steps and outcomes | "Updated QoS configuration, reset buffer limits" |
| Status | String | Current ticket status | "Resolved", "In Progress", "Pending" |
| Tags | String | Categorization tags for incident classification | "network, latency, router, performance" |
| Timestamp | Date | Incident creation and resolution timestamps | "2024-03-15 14:30:00" |
| DocID | String | Reference to related technical documentation | "DOC-RTR-001" |

**3. Source Technical Records Dataset (src\_tech\_records.csv)**

This dataset contains 50 technical documentation entries outlining procedural steps for network troubleshooting and maintenance.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column Name** | **Data Type** | **Description** | **Sample Value** |
| DocID | String | Unique documentation identifier | "DOC-RTR-001" |
| ProductID | String | Related product identifier | "RTR-X1000" |
| step\_description | String | Detailed technical procedure description | "Configure QoS parameters for optimal traffic prioritization" |

**4. Metadata Technical Records Dataset (metadata\_tech\_records.csv)**

This enriched dataset includes 50 comprehensive technical documentation records with detailed solution steps, technical tags, and document categorization.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column Name** | **Data Type** | **Description** | **Sample Value** |
| DocID | String | Unique documentation identifier | "DOC-RTR-001" |
| ProductID | String | Related product identifier | "RTR-X1000" |
| ProductInformation | String | Detailed product specifications | "Cisco Router X1000 Series Configuration Guide" |
| SolutionSteps | String | Step-by-step resolution procedures | "1. Access router CLI 2. Configure QoS 3. Verify settings" |
| TechnicalTags | String | Technical categorization tags | "routing, QoS, configuration, performance" |
| DocumentType | String | Document classification | "Configuration Guide", "Troubleshooting Manual" |

# Solution Design & Detailed Phases

**Phase 1: Data Preparation & Ingestion**

1. **Data Profiling & Cleaning**
   * Inspect src\_incident\_records.csv and src\_tech\_records.csv for missing values, inconsistent formats, or duplicates.
   * Normalize ProblemDescription and step\_description by removing extra whitespace and standardizing text encoding.
   * Segment long descriptions (>2 KB) into chunks for efficient embedding.
2. **Metadata Validation**
   * Validate metadata\_incident\_records.csv and metadata\_tech\_records.csv for consistency with source datasets.
   * Ensure TicketID and DocID mappings align across datasets.

**Phase 2: Embedding Generation & Vector Indexing**

1. **Embedding Pipeline**
   * Generate 1,024-dimensional embeddings for ProblemDescription (src\_incident\_records.csv) using Vertex AI Text Embeddings API.
   * Generate embeddings for step\_description (src\_tech\_records.csv) using the same API.
   * Attach metadata: TicketID, ProductID, DocID, and text snippets..
2. **Weaviate Index Creation**
   * Configure Weaviate on Google Cloud GKE with cosine distance metric for vector storage.
   * Create separate indexes for ProblemDescription (incident tickets) and step\_description (technical manuals).
   * Upload embeddings and metadata using Weaviate's Python client.
3. **Validation & Benchmarking**
   * Run sample nearest-neighbor queries to verify sub-200 ms response times.
   * Compare retrieved TicketID/DocID against known similar records for accuracy.

**Phase 3: Agent 1 - Cloud RAG Retrieval (AutoGen)**

1. **RetrieverAgent (AutoGen on Cloud Functions/GKE)**
   * Input: New ticket text (ProblemDescription).
   * Tasks:
     + Embed input text using Vertex AI Text Embeddings API.
     + Query Weaviate for top-K (e.g., K=5) similar ProblemDescription records.
     + Retrieve associated SolutionDetails from metadata\_incident\_records.csv.
   * Aggregation Logic:
     + Weight retrieved solutions by normalized similarity scores.
     + Select solutions with cumulative weight exceeding 60%.
     + Compute confidence score (average of top-K similarities).
     + Output: Proposed solution with confidence score.
2. **Threshold Decision**
   * If confidence ≥ 0.75, forward solution to ResponseFormatter.
   * Else, trigger A2A Broker to route to Agent 2.

**Phase 4: Fallback LLM Reasoning Agent**

1. **ReasonerAgent (LangGraph)**
   * Input: Ticket text and Agent 1's low-confidence output.
   * Tasks:
     + Embed input text using Vertex AI Text Embeddings API.
     + Query Weaviate for top-K similar step\_description records from src\_tech\_records.csv.
     + Retrieve SolutionSteps from metadata\_tech\_records.csv.
     + Craft a few-shot prompt including:
       - Ticket text.
       - Top 3-5 similar step\_description records with SolutionSteps.
       - Instructions: "Return a concise resolution with up to 5 steps."
     + Invoke Vertex AI Gemini model for reasoning.
     + Output: Final resolution parsed into structured steps.
   * Error Handling:
     + Retry up to 2 times on malformed output or timeout with exponential back-off.
     + Log fallback invocations to Cloud Logging.

**Phase 5: Workflow Composition & API Exposure**

1. **Agent Communication**
   * Route incoming ticket requests to Agent 1.
   * Perform health checks on Agent 1 (latency, error rate).
   * Trigger failover to Agent 2 if Agent 1 breaches latency (>1 s) or error SLA (>5 errors/10 min).
   * Log routing decisions to Cloud Logging.
2. **LangGraph Workflow**
   * Define a graph sequencing: RetrieverAgent → A2A Broker → conditional ReasonerAgent → ResponseFormatter.
   * Use descriptive node names and inline comments for maintainability.
3. **ResponseFormatter**
   * Merge solutions and confidences into a JSON payload:

**{**

**"TicketID": "INC999",**

**"proposed\_solution": ["Step 1: Verify router configuration...", "Step 2: Check logs..."],**

**"confidence": 0.82,**

**"used\_fallback": false**

**}**

1. **REST API Service**
   * Wrap workflow in a FastAPI app.
   * Deploy Agent 1 to Cloud Functions/GKE, Agent 2 to on-premise servers.
   * Secure with Cloud IAM authentication and token validation.

**Phase 6: Containerization & Deployment**

1. **Dockerization**
   * Create Dockerfiles for Agent 1 (Cloud Functions) and Agent 2 (on-premise).
   * Install AutoGen, LangGraph, Weaviate client, and FastAPI dependencies.
   * Use multi-stage builds for minimal image size.
   * Push Docker images to Artifact Registry.
2. **Deployment**
   * Deploy Agent 1 to Cloud Functions/GKE.
   * Deploy Agent 2 to on-premise servers.
   * Configure Artifact Registry for secure image access.

**Phase 7: Observability & Monitoring**

1. **System Monitoring (Cloud Logging)**
   * Log embedding calls, Weaviate queries, and Gemini invocations.
   * Track metrics: latency, token counts, similarity scores, fallback rate.
2. **Custom Metrics (Cloud Monitoring)**
   * Monitor: fallback\_rate, avg\_confidence, resolution\_latency, error\_count.
   * Create dashboards for real-time visualization.
3. **Alerting**
   * Configure Cloud Monitoring alerts:
     + fallback\_rate > 10% over 1 hour.
     + resolution\_latency > 1 s over 5 min.
     + error\_count > 5 over 10 min.
   * Route alerts to Slack/PagerDuty via webhooks.

**Phase 8: UI Development & Stakeholder Demo**

1. **Web UI (React/Next.js)**
   * Features:
     + Single-ticket input (text or file upload).
     + Batch ticket upload via CSV.
     + Display retrieved tickets, proposed solutions, confidence, and fallback status.
     + Feedback form for manual corrections.
   * Deploy on App Engine.
2. **Demo Script & Training**
   * Prepare a guide showing:
     + High-confidence scenario (Agent 1 resolution).
     + Low-confidence scenario (Agent 2 fallback).
     + Real-time dashboard updates and alert triggers.

# Expected Deliverables

1. **Code Assets**
   * AutoGen and LangGraph workflow definitions.
   * Python scripts for data ingestion, embedding generation, and API server.
   * Prompt templates for RetrieverAgent and ReasonerAgent.
   * Dockerfiles for Agent 1 and Agent 2.
2. **Deployment Artifacts**
   * Cloud Deployment Manager templates for all GCP resources.
   * Container image URIs in Artifact Registry.
   * Cloud Functions/GKE endpoints and on-premise server URLs.
   * Cloud IAM policy snippets and Secret Manager references.
3. **Monitoring & Alerts**
   * Cloud Monitoring dashboard configurations.
   * Cloud Monitoring alert policy definitions.
4. **Documentation**
   * Architecture diagrams (Mermaid format).
   * OpenAPI spec for /troubleshoot endpoint.
   * Runbook: setup, scaling, troubleshooting, and extension guidelines.