## 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 Azure 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

TechNet support team needs a system that:

1. **Speeds up resolution** - Find answers in seconds, not minutes
2. **Improves consistency** - Same problem always gets same solution
3. **Handles peak times** - Works even during busy periods
4. **Learns from experience** - Gets better as more cases are added

Your task: Build a dual-agent troubleshooter using Azure Cloud that automatically suggests solutions based on similar past tickets and technical documentation.

* **Dual-Agent System** - Two AI agents working together (Agent 1 for quick search, Agent 2 for complex reasoning)
* Azure Cloud Integration - Use Azure tools to power AI agents and store/search data
* **Automatic Solution Suggestions** - System analyzes problems and suggests solutions from past tickets and technical docs

# 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 simple\_incident\_records.csv and simple\_tech\_records.csv for missing values, inconsistent formats, or duplicates using Azure Machine Learning Workbench
* Normalize ProblemDescription and step\_description by removing extra whitespace and standardizing text encoding
* Segment long descriptions (>1 KB) into chunks for efficient embedding
* Remove any customer identifying information and standardize ticket formats

**2. Data Storage Setup**

* Upload cleaned data to Azure Blob Storage buckets with organized folder structure:
  + gs://technet-data/incidents/simple\_incident\_records.csv
  + gs://technet-data/technical/simple\_tech\_records.csv
* Configure bucket permissions and access controls
* Set up data validation pipelines for ongoing data quality

**Tools Used:**

* Azure Azure Blob Storage (file storage and organization)
* Azure Machine Learning Workbench (data profiling and cleaning)
* Azure Active Directory (AAD) (access control)

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

**1. Embedding Pipeline**

* Generate 1,024-dimensional embeddings for ProblemDescription from simple\_incident\_records.csv using Azure Machine Learning Text Embeddings API
* Generate embeddings for step\_description from simple\_tech\_records.csv using the same API
* Attach metadata: ticket\_id, product, doc\_id, and original text snippets
* Store embeddings in Azure Blob Storage organized by data type

**2. Weaviate Index Creation**

* Configure Weaviate on Azure Cloud GKE with cosine distance metric for vector storage
* Create separate collections for ProblemDescription (incident tickets) and step\_description (technical guides)
* Upload embeddings and metadata using Weaviate's Python client
* Configure index parameters for optimal performance with 50-record datasets

**3. Validation & Benchmarking**

* Run sample nearest-neighbor queries to verify sub-200ms response times using Azure Machine Learning Workbench
* Compare retrieved ticket\_id/doc\_id against known similar records for accuracy validation
* Test query performance with various similarity thresholds

**Tools Used:**

* Azure Machine Learning Embeddings API (text-to-vector conversion)
* Weaviate on Azure Cloud GKE (vector database)
* Azure Blob Storage (embedding storage)
* Azure Machine Learning Workbench (testing and validation)

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

1. RetrieverAgent (AutoGen on Azure Functions)

* Input: New ticket text (ProblemDescription)
* Tasks:
  + Embed input text using Azure Machine Learning Text Embeddings API
  + Query Weaviate for top-K (e.g., K=5) similar ProblemDescription records
  + Retrieve associated solution details from simple\_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 Agent-to-Agent communication to route to Agent 2
* Log decision rationale and confidence metrics

**Tools Used:**

* AutoGen framework (AI agent orchestration)
* Azure Functions (serverless deployment)
* Azure Machine Learning Embeddings API (text processing)
* Weaviate Python client (vector search)

**Phase 4: Agent 2 - Fallback Technical Reasoning (LangGraph)**

**1. ReasonerAgent (LangGraph)**

* Input: Ticket text and Agent 1's low-confidence output
* Tasks:
  + Embed input text using Azure Machine Learning Text Embeddings API
  + Query Weaviate for top-K similar step\_description records from simple\_tech\_records.csv
  + Retrieve solution steps from technical documentation
  + Craft few-shot prompt including:
    - Original ticket text
    - Top 3-5 similar step\_description records with solutions
    - Instructions: "Return a concise resolution with up to 5 technical steps"
  + Invoke Azure Machine Learning Azure OpenAI model for technical reasoning
  + Output: Structured technical resolution with step-by-step guidance

**2. Error Handling & Retry Logic**

* Retry up to 2 times on malformed output or timeout with exponential back-off
* Log all fallback invocations and failures to Azure Monitor Logs
* Implement graceful degradation if both agents fail

**Tools Used:**

* LangGraph (workflow orchestration)
* Azure Machine Learning Azure OpenAI (AI reasoning engine)
* Azure Machine Learning Embeddings API (text processing)
* Azure Monitor Logs (error tracking)

**Phase 5: Agent-to-Agent Communication & Workflow Composition**

**1. Agent Communication Broker**

* Route incoming ticket requests to Agent 1 (AutoGen on Azure Functions)
* Perform health checks on Agent 1 (latency <1s, error rate <5%)
* Trigger failover to Agent 2 if Agent 1 breaches confidence threshold (<0.75)
* Log routing decisions and performance metrics to Azure Monitor Logs

**2. LangGraph Workflow Orchestration**

* Define workflow graph: RetrieverAgent → A2A Broker → conditional ReasonerAgent → ResponseFormatter
* Implement node-based execution with clear state management
* Use descriptive node names and inline documentation for maintainability

**3. ResponseFormatter**

* Merge solutions and confidence scores into structured JSON payload:

{

"ticket\_id": "TKT-001",

"proposed\_solution": ["Step 1: Check router power connection", "Step 2: Reset network settings"],

"confidence": 0.82,

"used\_fallback": false,

"agent\_source": "agent\_1",

"processing\_time\_ms": 245

}

**4. REST API Service**

* Wrap workflow in FastAPI application
* Deploy Agent 1 to Azure Functions, Agent 2 to GKE (or on-premise)
* Secure with Azure Active Directory (AAD) authentication and API key validation

**Tools Used:**

* FastAPI (API framework and agent communication)
* Azure Functions (Agent 1 hosting)
* Azure Kubernetes Service (AKS) (Agent 2 hosting)
* Azure Active Directory (AAD) (authentication and authorization)
* Azure Monitor Logs (communication tracking)

**Phase 6: Simple Web Interface**

**User Interface**

* Simple form: paste customer problem, click "Get Solution"
* Shows: suggested solution, confidence level, which agent responded
* Allow staff to mark if solution worked or needs adjustment

**Tools Used:**

* Azure App Service (web hosting)
* React/Next.js (frontend)

**Phase 7: Monitoring Dashboard**

**Performance Tracking**

* Count daily ticket resolutions
* Track average response time
* Monitor which agent is used more often
* Alert if system is slow or down

**Tools Used:**

* Azure Monitor (system metrics)
* Azure Monitor Logs (detailed logs)

# 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 Azure Resource Manager templates for all Azure resources.
   * Container image URIs in Azure Container Registry.
   * Azure Functions/GKE endpoints and on-premise server URLs.
   * Azure Active Directory (AAD) policy snippets and Azure Key Vault references.
3. **Monitoring & Alerts**
   * Azure Monitor dashboard configurations.
   * Azure Monitor alert policy definitions.
4. **Documentation**
   * Architecture diagrams (Mermaid format).
   * Google API Discovery Service for /troubleshoot endpoint.
   * Runbook: setup, scaling, troubleshooting, and extension guidelines.