LLM Hackathon Infrastructure Proposal

DRAFT DOCUMENT: This proposal is a draft offering and all details are subject to change

Overview

This document outlines the proposed infrastructure setup for the OMC hackathon focusing on LLM exploration. The architecture allows OMC employees to experiment with three different LLM models while authenticating through the existing OMC ADFS system, with added RAG (Retrieval Augmented Generation) capabilities.

LLM Models

We propose offering the following three models to showcase different capabilities and use cases:

1. **Llama 3 70B** - Meta's flagship large language model, offering strong general-purpose capabilities across a wide range of tasks including reasoning, code generation, and creative writing.
2. **CodeLlama 34B** - Specialized for code generation and understanding, optimized for programming tasks and technical assistance. Ideal for developers wanting to explore code completion, bug fixing, and technical documentation.
3. **DeepSeek Coder 7B** - A specialized coding model known for its efficiency and strong performance despite its smaller size. Excellent for programming tasks, debugging, and technical documentation, with a different approach than CodeLlama. The model will be deployed in a network-isolated environment to prevent any external communications, ensuring complete privacy for OMC use.

Infrastructure Architecture

Components

1. **Authentication Layer**
   - Integration with OMC ADFS
   - Domain-restricted access
   - Self-service registration for all OMC employees
2. **User Interface**
   - Open WebUI (formerly ollama-webui)
   - Single interface for accessing all three models
   - Persistent chat history and user preferences
3. **Backend Infrastructure**
   - Three separate model-specific Auto Scaling Groups
   - Load balancers for each model cluster
   - Centralized session and data storage
   - Vector database for RAG capabilities

4. **RAG Components**
   - Document processing pipeline
   - Vector embeddings generation
   - Vector database (PostgreSQL with pgvector)
   - Retrieval middleware integration with Open WebUI

Technical Implementation

Authentication Flow

1. Register Open WebUI as a new application in OMC ADFS
2. Configure claim rules to pass OMC email information
3. Implement authentication proxy to validate ADFS assertions
4. Users authenticate via familiar company login screens

AWS Architecture

```
[OMC ADFS Authentication]
            ↓
      [Load Balancer]
            ↓
    [Open WebUI Instances]
            ↓
 ┌─────────────┬─────────────┬─────────────┬─────────────┐
 ↓             ↓             ↓             ↓             ↓
[Llama 3 70B] [CodeLlama 34B] [DeepSeek 7B] [DynamoDB]
[PostgreSQL/pgvector]
[GPU ASG]     [GPU ASG]      [GPU ASG]     [Sessions]   [Vector Store]
                                                              ↑
                                                              ↓
                                                     [Document Processor]
                                                     [Lambda Functions]
```

AWS Resources Required

1. **EC2 Instances**
   - Open WebUI: t3.medium instances (2-3 for redundancy)
   - Llama 3 70B: g5.2xlarge instances (2-3 nodes)
   - CodeLlama 34B: g5.xlarge instances (2-3 nodes)
   - DeepSeek Coder 7B: g4dn.xlarge instances (2-3 nodes)
2. **Supporting Services**
   - Application Load Balancers (4 total)
   - DynamoDB table for session management
   - RDS PostgreSQL with pgvector extension for vector storage
   - S3 bucket for model storage and document repository

- Lambda functions for document processing
- SQS for document processing queue

3. **Networking**
   - VPC with public and private subnets
   - Security groups for access control
   - Connection to OMC network for ADFS
   - Network isolation for models (especially DeepSeek) to prevent external communication
   - Egress controls to ensure no model can "phone home" or access the internet
   - Private endpoints for all AWS services to maintain complete data isolation

Scaling Considerations

Each model is isolated in its own Auto Scaling Group to prevent resource contention. For the hackathon, we recommend starting with a fixed number of instances rather than implementing complex auto-scaling:

- 3 instances for Llama 3 70B (most resource-intensive)
- 3 instances for CodeLlama 34B (specialized model)
- 2 instances for DeepSeek Coder 7B (efficient coding model)

This configuration can support approximately 50-75 concurrent users during the hackathon.

User Experience

1. **Authentication**
   - Users navigate to the hackathon URL
   - They're redirected to the familiar OMC ADFS login if not already authenticated
   - After authentication, they're taken to the Open WebUI interface
2. **Model Selection**
   - Users select which model they want to interact with via a dropdown
   - Behind the scenes, requests route to the appropriate model cluster
   - Users can switch models at any time during their session
3. **RAG Capabilities**
   - Users can upload documents (PDF, DOCX, TXT, etc.) to create knowledge bases
   - Documents are automatically processed, chunked, and embedded
   - Users can query models with context from their uploaded documents
   - System retrieves relevant information to augment model responses
4. **Persistent Experience**
   - Chat history is maintained across sessions
   - User preferences and custom prompts are saved
   - Knowledge bases and documents remain available for future sessions
   - Experience remains consistent even if backend instances change

Cost Considerations

Estimated daily cost for the proposed setup (Australian region):

- EC2 GPU Instances: ~$600-800 AUD per day
- Supporting services: ~$100-150 AUD per day
- RAG infrastructure: ~$50-80 AUD per day
- Total estimated cost for a 2-day hackathon: ~$1,500-2,100 AUD

Implementation Timeline

Task	Timeframe
AWS infrastructure setup	2-3 days
ADFS integration	1-2 days
Open WebUI configuration	1 day
RAG components setup	2-3 days
Model deployment and testing	1-2 days
Total preparation time	7-11 days

Business Value

This hackathon setup will allow OMC employees to:

1. **Explore different model capabilities** - Understand the trade-offs between model size, speed, and capabilities
2. **Experiment with real-world use cases** - Test potential applications within OMC workflows
3. **Compare model performances** - Evaluate which models excel at different types of tasks
4. **Build technical familiarity** - Gain hands-on experience with LLM infrastructure
5. **Test RAG capabilities** - Explore how retrieval augmented generation can enhance LLMs with company-specific knowledge
6. **Prototype potential applications** - Create proof-of-concepts for using LLMs with OMC documents

Next Steps

1. Finalize model selection and sizing requirements
2. Create detailed implementation plan
3. Set up development environment for testing
4. Configure ADFS test application
5. Develop deployment automation
6. Prepare sample document sets for RAG demonstrations

RAG Implementation Details

Document Processing Pipeline

1. **Upload Interface**

- Users upload documents through Open WebUI
- Files stored in S3 with appropriate access controls

2. **Processing**

    - Lambda functions triggered by S3 events
    - Documents parsed based on file type (PDF, DOCX, TXT, etc.)
    - Text extracted and chunked into appropriate segments

3. **Embedding Generation**

    - Text chunks processed by embedding model
    - We recommend using all-MiniLM-L6-v2 for efficiency
    - Vector embeddings generated for each text chunk

4. **Storage**

    - Vectors stored in PostgreSQL with pgvector extension
    - Metadata and source information preserved
    - Indexed for efficient similarity search

Query Flow

1. User submits a question through the interface
2. System generates embedding for the question
3. Vector store searched for relevant chunks
4. Retrieved context added to the prompt
5. LLM generates response informed by the retrieved content
6. Response displayed to user with source citations

This RAG implementation enables users to get answers grounded in their own documents, making the LLM capabilities directly applicable to OMC-specific knowledge and use cases.

Network Security and Data Privacy Controls

In addition to the end-to-end encryption and zero-knowledge architecture, we implement strict network isolation:

1. **Complete Network Isolation**

    - All LLM instances run in private subnets with no internet access
    - Special attention to DeepSeek and other models to prevent any "phoning home"
    - Network Activity Control Lists block all outbound traffic to public internet
    - VPC endpoints used for all AWS service communication

2. **Internal-Only Accessibility**

    - Models only accessible to OMC staff through the internal network
    - No external API endpoints exposed
    - Multi-layered security ensuring models cannot be reached from outside OMC

3. **Communication Monitoring**

- Network flow logs monitor for any unauthorized communication attempts
- Alerts configured for any unexpected network traffic
- Regular security audits to verify isolation effectiveness

This network isolation strategy ensures that even models like DeepSeek, which might otherwise have privacy concerns in some deployments, are completely contained within OMC's environment with no ability to transmit data externally.

Data Security and Privacy

A key feature of this implementation is its robust security model that ensures all user data remains private:

1. **End-to-End Encryption**
    - All conversations with the LLMs are encrypted in transit
    - Encryption keys are managed per user
    - Neither DevOps nor IT teams can access the actual conversation content
2. **RAG Document Security**
    - Uploaded documents are processed in secure environments
    - Access controls ensure users only see their own documents and conversations
3. **Zero Knowledge Architecture**
    - System is designed on zero knowledge principles
    - Administrative functions do not require or permit content access
    - Monitoring and logging exclude sensitive content
4. **Technical Implementation**
    - Client-side security measures for document uploads
    - Separate access control mechanisms for system maintenance vs content access

Note: For the hackathon implementation, we will focus on network isolation, access controls, and transport encryption. Full end-to-end encryption of stored data at rest, including client-side encryption and column-level encryption in PostgreSQL, could be implemented as a future enhancement but will not be available for the initial hackathon.