# 🌐 Deployment Guide: Qdrant + Chatbot on AWS ECS

## **1. Push Your Chatbot App to AWS ECR**

1. **Build Docker image**  
   From your project root:
2. docker build -t rag-chatbot .
3. **Tag the image for ECR**  
   Replace <AWS\_ACCOUNT\_ID> and <REGION>:
4. docker tag rag-chatbot:latest <AWS\_ACCOUNT\_ID>.dkr.ecr.<REGION>.amazonaws.com/rag-chatbot:latest
5. **Login to ECR**
6. aws ecr get-login-password --region <REGION> | docker login --username AWS --password-stdin <AWS\_ACCOUNT\_ID>.dkr.ecr.<REGION>.amazonaws.com
7. **Push to ECR**
8. docker push <AWS\_ACCOUNT\_ID>.dkr.ecr.<REGION>.amazonaws.com/rag-chatbot:latest

## **2. Deploy Qdrant on AWS ECS**

### Option A: Use Qdrant Cloud (managed)

* Go to https://cloud.qdrant.io, create a cluster.
* Get the **URL + API key** → set them as environment variables in your ECS service.

### Option B: Self-host Qdrant on ECS

1. Use the official Qdrant Docker image:
2. version: "3.9"
3. services:
4. qdrant:
5. image: qdrant/qdrant
6. ports:
7. - "6333:6333"
8. volumes:
9. - qdrant\_storage:/qdrant/storage
10. volumes:
11. qdrant\_storage:
12. Deploy via **ECS task definition** or **Fargate** with persistent **EBS/EFS volume** attached to /qdrant/storage.

## **3. Update Chatbot to Use Qdrant on AWS**

In your chatbot\_app.py, replace local Qdrant client:

from qdrant\_client import QdrantClient

client = QdrantClient(

host=os.getenv("QDRANT\_HOST", "localhost"),

port=int(os.getenv("QDRANT\_PORT", 6333)),

api\_key=os.getenv("QDRANT\_API\_KEY", None) # if using Qdrant Cloud

)

Then set these environment variables in ECS task definition:

{

"name": "QDRANT\_HOST",

"value": "your-qdrant-endpoint.amazonaws.com"

},

{

"name": "QDRANT\_PORT",

"value": "6333"

},

{

"name": "QDRANT\_API\_KEY",

"value": "xxxxx"

}

## **4. Run Both Services in ECS**

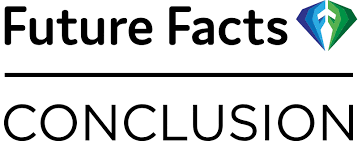
* Create **two ECS services**:
  + **rag-chatbot-service** → runs your chatbot container (Streamlit, port 8501).
  + **qdrant-service** → runs Qdrant (port 6333, with persistent storage).
* Use an **Application Load Balancer (ALB)**:
  + Route yourdomain.com/chat → chatbot ECS service
  + Route yourdomain.com/qdrant → Qdrant (optional, usually internal only).

## **5. Make It Scalable & Reliable**

* **Autoscaling**: Configure ECS Fargate to scale chatbot tasks based on CPU/memory usage.
* **Persistence**: Attach **EBS/EFS** volume to Qdrant to persist vectors across restarts.
* **Security**: Restrict Qdrant to private subnets (only chatbot can talk to it).
* **Monitoring**: Enable CloudWatch logs for both services.

## ✅ End Result

* Your chatbot runs on **AWS ECS (Fargate)**, fully containerized.
* Qdrant runs **persistently with storage** (self-hosted or cloud).
* Retrieval is faster, more reliable, and supports **hybrid search + metadata filtering**.
* Easy to scale horizontally (add more chatbot replicas).



***Welcome to Future Facts!***

*We are very excited to have you as part of our team and work on an innovative Data Engineering Project. This document will take you through the details of your graduation project and what you can expect.*

***Table of Contents***

[Your Graduation Project Mentor 2](#_Toc183692031)

[Additional Mentors 2](#_Toc183692032)

[Expectations from Future Facts 2](#_Toc183692033)

[Project Timeline 3](#_Toc183692034)

[The Project 5](#_Toc183692035)

[Your Onboarding Schedule: Week 1 7](#_Toc183692036)

# Your Graduation Project Mentor

Your graduation project mentor will be **Victor Milewski.**

**Victor’s** role will be to guide you through the next months and ensure you get the most out of the project. We will set up a first meeting with your mentor to set the goals of the projects. Then every week we will have a check in to ensure we are on the right track.

To prepare for the planning meeting next week you can think about:

1. What do I want to get out of the AWS RAG Project?
2. Where do I see the next steps in my Career and how can Future Facts help?
3. What do I want to learn from my mentor?
4. What do I expect from my mentor?

# Other Mentors

As additional mentors for the project we have **Caoimhe Ni Mhaonaigh** and **Ramses Kools.**

They will assist in case **Victor** is not available or for technical details in which **Caoimhe and Ramses** have more experience. They will also assist in the code reviews and feedback. You can ask any of the mentors for help at any time of you’re stuck.

When you visit the office, all our colleagues within Future Facts are also available to ask questions to, we have a wealth of knowledge here within the company and you can always ask people for help. Your project mentor can be a point of contact to understand the expertise of different colleagues and who to ask for what. You can always plan a little coffee meeting with any colleague to get to know anyone a little better.

# Expectations from Future Facts

The AWS RAG project is an important Internal project that will be used with future clients. Therefore, there are expectations from Future Facts that when this project is finished it is up to the Future Facts standard.

On the Friday the 10th of October you will present the work you have done over the project in our Friday Session and present the code in the Data Science meeting that day. The project will need to have clear documentation.

# The Project

***Retrieval Augmented Generation (RAG) In AWS***

Purpose of this project: In last years Internship by fellow Spike-Up Graduates, a RAG system was implemented. The internship project focused on testing, validating, and developing the AWS native RAG System. In this project the goal is to add components for a complete data ingestion pipeline. This includes collection of data, cleaning and processing of data, and loading the data into the RAG database.

*Deliverables:* A fully functional Data Ingestion pipeline for a RAG system utilizing primarily AWS native components.

Introduction

Retrieval Augmented Generation (RAG) is an innovative approach to enhancing the capabilities of large language models (LLMs) by incorporating external, up-to-date information. AWS offers several solutions and services to implement RAG applications effectively. Here's an overview of RAG applications in AWS:

What is RAG?

Retrieval Augmented Generation is a process that optimizes the output of LLMs by referencing an authoritative knowledge base outside of their training data sources before generating a response. This approach allows LLMs to access the latest information, improving relevancy, accuracy, and usefulness in various contexts.

A diagram of a computer process

Description automatically generated

Benefits of RAG

* Enhanced accuracy: RAG enables LLMs to provide more precise and up-to-date information by referencing external knowledge bases.
* Domain-specific knowledge: It allows the integration of specialized or proprietary information, making LLMs more effective for specific use cases.
* Improved user trust: RAG enables the presentation of accurate information with source attribution, increasing confidence in the AI solution.
* Cost-effectiveness: RAG extends LLM capabilities without the need for retraining, offering a more economical approach to improving model performance.

Implementing RAG on AWS

AWS provides several services and tools to build RAG applications:

* Amazon Bedrock: Amazon Bedrock is a fully-managed service that offers high-performing foundation models along with capabilities to build generative AI applications. It includes:
  + Knowledge bases for Amazon Bedrock, allowing easy connection of foundation models to data sources for RAG.
  + Automated handling of vector conversions, retrievals, and improved output generation.
* Amazon Kendra: For organizations managing their own RAG, Amazon Kendra offers:
  + An enterprise search service powered by machine learning.
  + Kendra Retrieve API for high-accuracy semantic ranking in RAG workflows.
  + Support for various document formats and data sources.

Serverless RAG Architecture

AWS enables the creation of serverless RAG solutions, combining the power of foundation models with the cost-effectiveness of serverless architecture. This approach utilizes:

* AWS Lambda for event-driven processing.
* Amazon S3 for document storage.
* LanceDB, an open-source vector database, for efficient vector storage and retrieval.

RAG Workflow on AWS

1. Document ingestion: Files are uploaded to an Amazon S3 bucket.
2. Content extraction and embedding: A Lambda function processes the files, extracts content, and generates embeddings using models like Amazon Titan.
3. Vector storage: Embeddings are stored in a vector database, such as Aurora PostgreSQL-Compatible or LanceDB.
4. Query processing: User queries are embedded and compared to stored vectors to retrieve relevant information.
5. Response generation: The retrieved information is used to augment the prompt sent to the LLM, which generates the final response.

AWS Services for RAG Components

* Vector storage: Amazon Aurora PostgreSQL-Compatible, Amazon OpenSearch Service, or serverless options like LanceDB.
* Embedding generation: Amazon Bedrock with models like Amazon Titan.
* LLM integration: Amazon Bedrock with models such as Claude 3.
* Development and deployment: Amazon SageMaker for notebooks and model deployment.