



Building Agentic Generative AI Apps with MongoDB Atlas on AWS

Aychin Gasimov

Senior Partner Solutions Architect
Amazon Web Services (AWS)

Better Together: AWS and MongoDB

Competencies/Programs



- Automotive Software Competency
- Data & Analytics Software Competency
- Financial Services Software Competency
- Generative AI Competency
- Government ISV Competency
- Small and Medium Business Software Competency
- AWS Marketplace Seller
- AWS Public Sector Partner
- AWS PrivateLink Ready

Product

Joint Integrations



AWS PrivateLink



Amazon EventBridge



AWS Key Management Service



AWS Lambda



AWS IAM Identity Center



Amazon Bedrock



Amazon Kinesis Data Firehose



Amazon Simple Storage Service



Amazon SageMaker



Amazon Managed Streaming for Apache Kafka

Harness the power of generative AI with MongoDB and AWS

Speed up application development with MongoDB's flexible developer data platform, allowing you to stay at the forefront of AI innovation, deliver unparalleled user experiences, and remain agile in a rapidly changing technological landscape, all while harnessing the capabilities of AWS and MongoDB Atlas.



Amazon Q Developer with MongoDB

Streamlines development workflow, integrates with popular IDEs, bolsters developer productivity by offering intuitive code snippets, pinpoint bugs, and upholding best practices for effective coding process.

MongoDB Vector Search and Amazon SageMaker Jumpstart

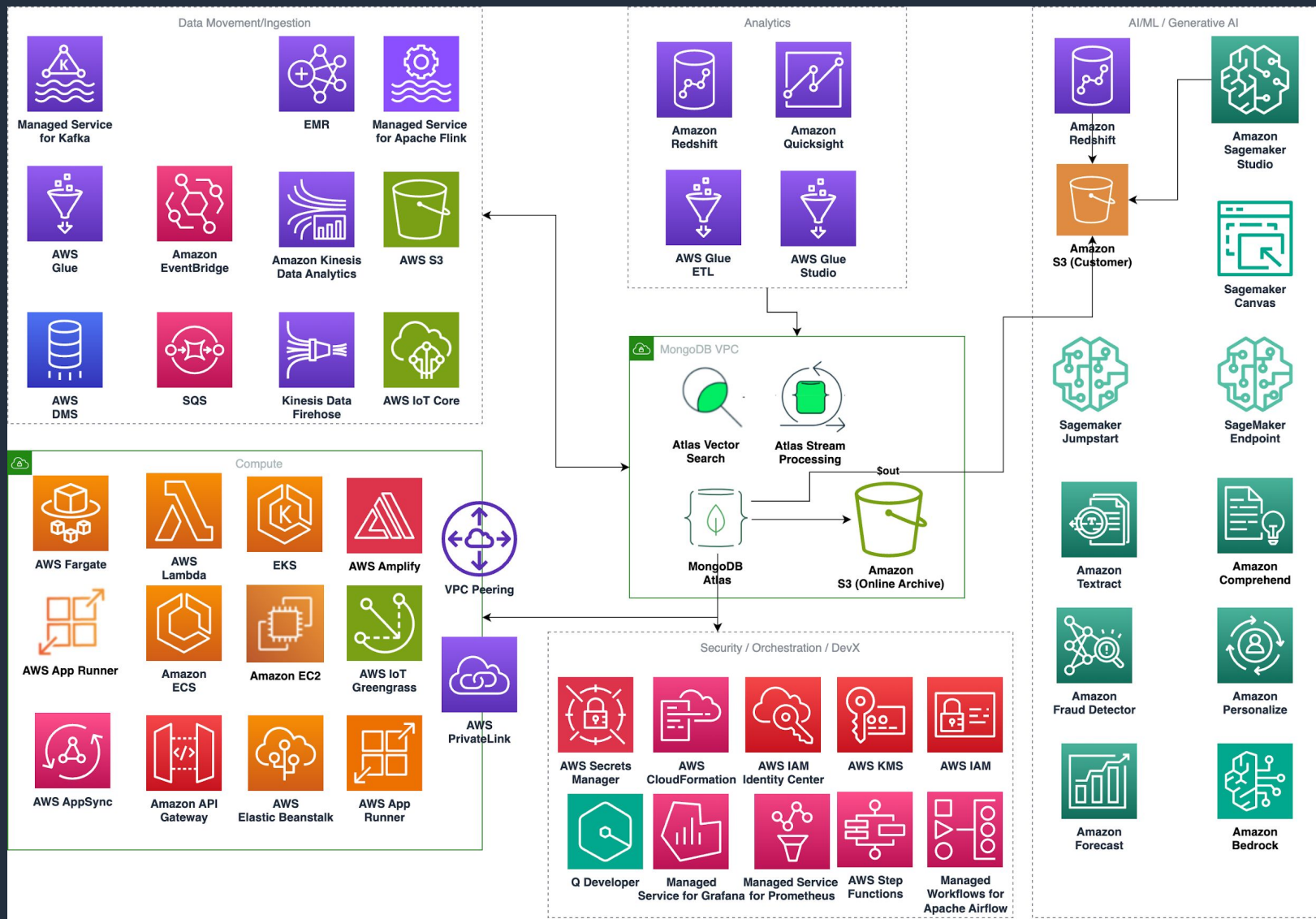
Deploy variety of open-source, public, and private models to power your GenAI applications with MongoDB Atlas Vector Search on AWS.

MongoDB Vector Search and Amazon BedRock

Build your GenAI applications easily with Amazon Bedrock and MongoDB Atlas Vector Search on AWS. Take advantage of Bedrock's privately customizable FMs to power your generative AI applications. Use Titan Embeddings and MongoDB Atlas for personalization and semantic search.



MongoDB Atlas on AWS



Generative AI: Trends through the years

2023

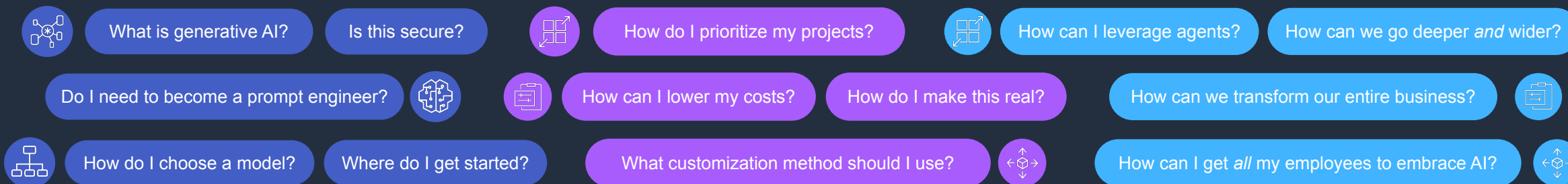
The year of POCs

2024

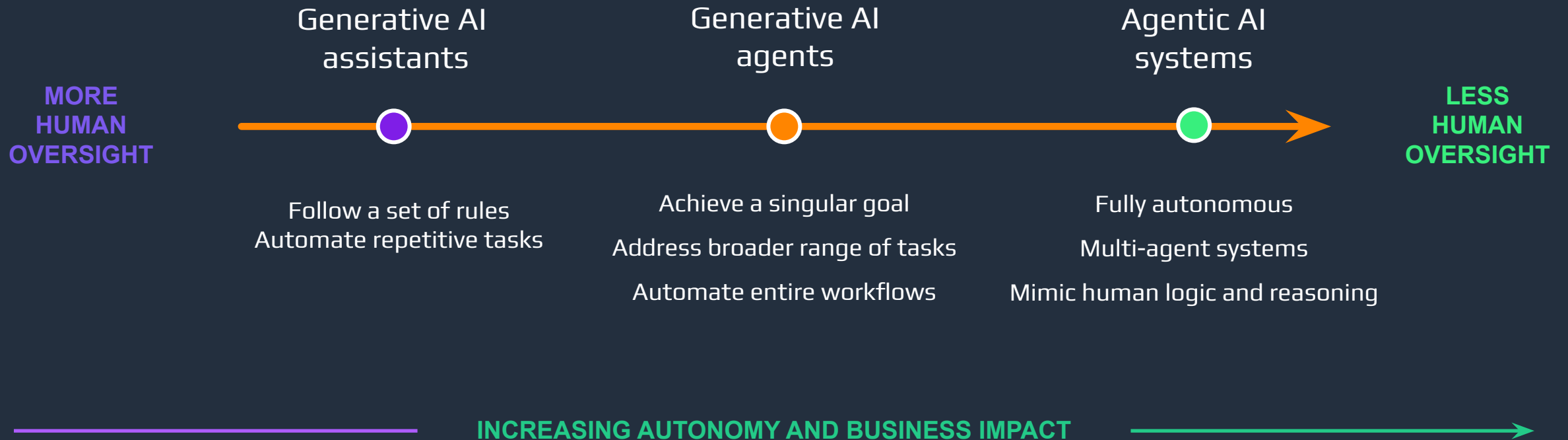
The year of Production

2025

The year of Business Value



The evolution into Agentic AI



AWS AI Stack

APPLICATIONS

Kiro

Amazon Q

AWS Transform

Amazon Connect

AWS Marketplace

AI & AGENT DEVELOPMENT SOFTWARE & SERVICES

SDKS FOR AGENTS

Vertically Integrated

Nova Act

Flexible/OSS

Strands Agents

AMAZON BEDROCK

Models

Amazon Nova

3P Models

Capabilities

Optimization

Guardrails

Customization

AgentCore

Runtime

1P Tools

Gateway

Identity

Memory

Observability

Knowledge Bases

INFRASTRUCTURE

AMAZON SAGEMAKER AI

Model building

Deployment

Model training

MLOps

Fine-tuning

Governance

AI COMPUTE

AWS Trainium

AWS Inferentia

GPUs

INTERFACES & PROTOCOLS
(MCP/A2A)

DATA



Amazon Q

Reinvent work with AWS'
generative AI-powered assistant

Unites your data sources, repositories, and
enterprise systems

Available where you work

Delivers industry-leading assistance

Reinvent work with **Amazon Q**

BUSINESS



Amazon Q
Business

KNOWLEDGE SEARCH

SUMMARIZATION

CONTENT CREATION

EXTRACT INSIGHTS

RESEARCH & ANALYSIS



Amazon Q in
QuickSight

UNDERSTAND DATA

BUILD & REFINE VISUALS

BUILD CALCULATIONS

EXECUTIVE SUMMARIES

CREATE DATA STORIES

DEVELOPERS



Amazon Q
Developer

PLAN APPLICATION

CODE GENERATION

UNIT TESTING

SECURITY SCANNING

CODE REMEDIATION

CODE MIGRATION

TROUBLESHOOTING

DEVELOPER KNOWLEDGE

SPECIALIZED USERS



Amazon Q in
Connect

AGENT ASSIST

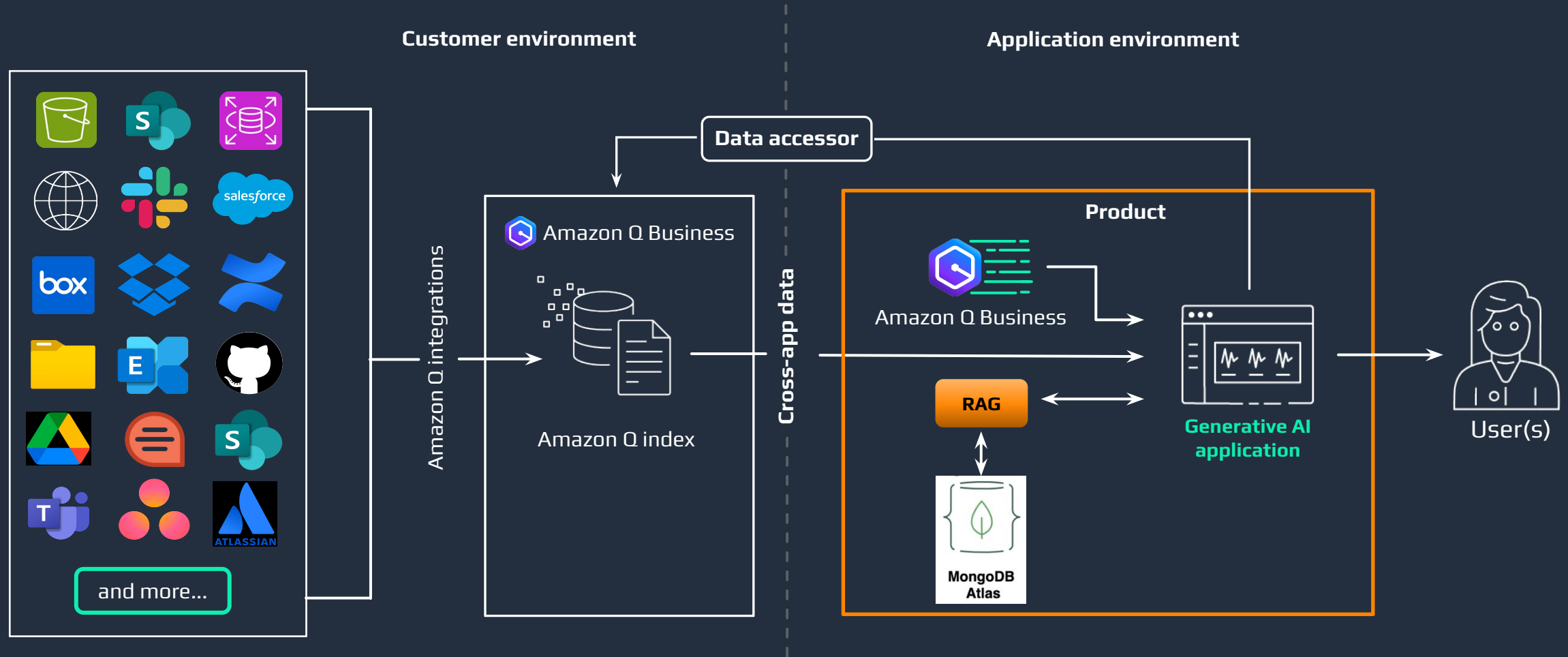


Amazon Q in
AWS Supply Chain

SUPPLY CHAIN

The Amazon Q index

ADD CROSS-APP INSIGHTS TO YOUR GENERATIVE AI APPLICATIONS



The broadest choice for building and deploying agents

Agent Powered Applications



For Builders



For Business
Users

Tools for Building AI Agents



Marketplace



Bedrock Agents



Strands



Nova Act SDK

Agentic AI Foundation



Amazon Bedrock AgentCore



Amazon Bedrock

The easiest way to build and scale generative AI applications with powerful tools and foundation models

Choice of leading FMs through a single API

Optimization for cost, latency and accuracy

Customization with your data

Safety and responsible AI checks

Agents that execute complex tasks

Amazon Bedrock

BROAD CHOICE OF MODELS

		ANTHROPIC				∞ Meta			
Effective reasoning & rapid analysis for long context windows	Frontier intelligence & industry leading price performance, high-quality image & video generation	Advanced reasoning & coding capabilities, including computer use skills	Multimodal search & advanced retrieval powering multilingual knowledge agents	Advanced reasoning, comprehension, scientific & mathematical analysis, & code generation	High-quality video generation from text & images	Advanced image & language reasoning	Knowledge summarization , expert agents, & code completion	Software engineering AI for large enterprises	High-quality AI image generation, easily deployable at scale
JAMBA	AMAZON NOVA	CLAUDE	COMMAND EMBED RERANK	DEEPSEEK-R1	LUMA RAY 2	LLAMA	MISTRAL MIXTRAL	MALIBU POINT	STABLE DIFFUSION STABLE IMAGE

Coming soon

Amazon Bedrock Marketplace enables developers to discover, test, and use over 100 popular, emerging, and specialized foundation models (FMs) alongside the current selection of industry-leading models in Amazon Bedrock. **DeepSeek-R1** model is now available in Amazon Bedrock Marketplace.



Amazon Nova Foundation Models

State-of-the-art foundation models that deliver frontier intelligence and industry leading price performance.

Understanding models

Amazon Nova Micro

Our text only model that delivers the lowest latency responses at very low cost

GENERALLY AVAILABLE

Amazon Nova Lite

Our lowest cost multimodal model that is lightning fast for lightweight tasks

GENERALLY AVAILABLE

Amazon Nova Pro

Our highly capable multimodal model with best combination of accuracy, speed, and cost for a wide range of tasks

GENERALLY AVAILABLE

Amazon Nova Premier

Our most capable multimodal model for complex reasoning tasks and for use as the best teacher for distilling custom models

COMING SOON



Creative content generation models

Amazon Nova Canvas

State-of-the-art image generation model

GENERALLY AVAILABLE

Amazon Nova Reel

State-of-the-art video generation model

GENERALLY AVAILABLE

Amazon Nova Sonic

State-of-the-art speech-to-speech model

GENERALLY AVAILABLE



Voyage.ai Embeddings Model in Bedrock Marketplace

AMAZON BEDROCK MARKETPLACE INCREASES AMAZON BEDROCK MODEL CHOICE WITH 100+ PUBLICLY AVAILABLE AND PROPRIETARY EMERGING, POPULAR, AND SPECIALIZED MODELS THROUGH A SERVERFUL OFFERING

The screenshot displays the Amazon Bedrock Marketplace interface. On the left, a navigation sidebar lists categories like 'Getting started', 'Foundation models', 'Playgrounds', and 'Builder tools'. The main content area features the 'voyage-3 Embedding Model' by Voyage AI, with buttons for 'Open in playground', 'View subscription options', and 'Deploy'. Below this, the 'voyage-3.5 Embedding Model' by MongoDB, Inc. is shown, including a 'Continue to Subscribe' button and a 'Save To List' button. A 'Product Overview' section at the bottom explains that text embedding models are neural networks that transform texts into numerical vectors. The interface also includes a search bar and various navigation links at the top.

Amazon Bedrock

- Getting started
 - Overview
 - Providers
- Foundation models
 - Model catalog [New](#)
 - Marketplace deployments [New](#)
 - Custom models (Distillation, Fine-tuning, Continued pre-training)
 - Imported models
 - Prompt Routers
- Playgrounds
 - Chat / Text
 - Image / Video
- Builder tools
 - Agents
 - Flows
 - Knowledge Bases

voyage-3 Embedding Model
By: [Voyage AI](#) | [Not subscribed](#) | Deployments: 0

[Open in playground](#)
[View subscription options](#)
[Deploy](#)

aws marketplace

[About](#) [Categories](#) [Delivery Methods](#) [Solutions](#) [Resources](#) [Your Saved List](#) [Become a Channel Partner](#) [Sell in AWS Marketplace](#) [Amazon Web Services Home](#)

voyage-3.5 Embedding Model
By: [MongoDB, Inc](#) Latest Version: 0.0.1
[Deployed on AWS](#)

Text embedding model optimized for general-purpose (including multilingual) retrieval/search and AI applications. 32K context length.

[Continue to Subscribe](#)
[Save To List](#)

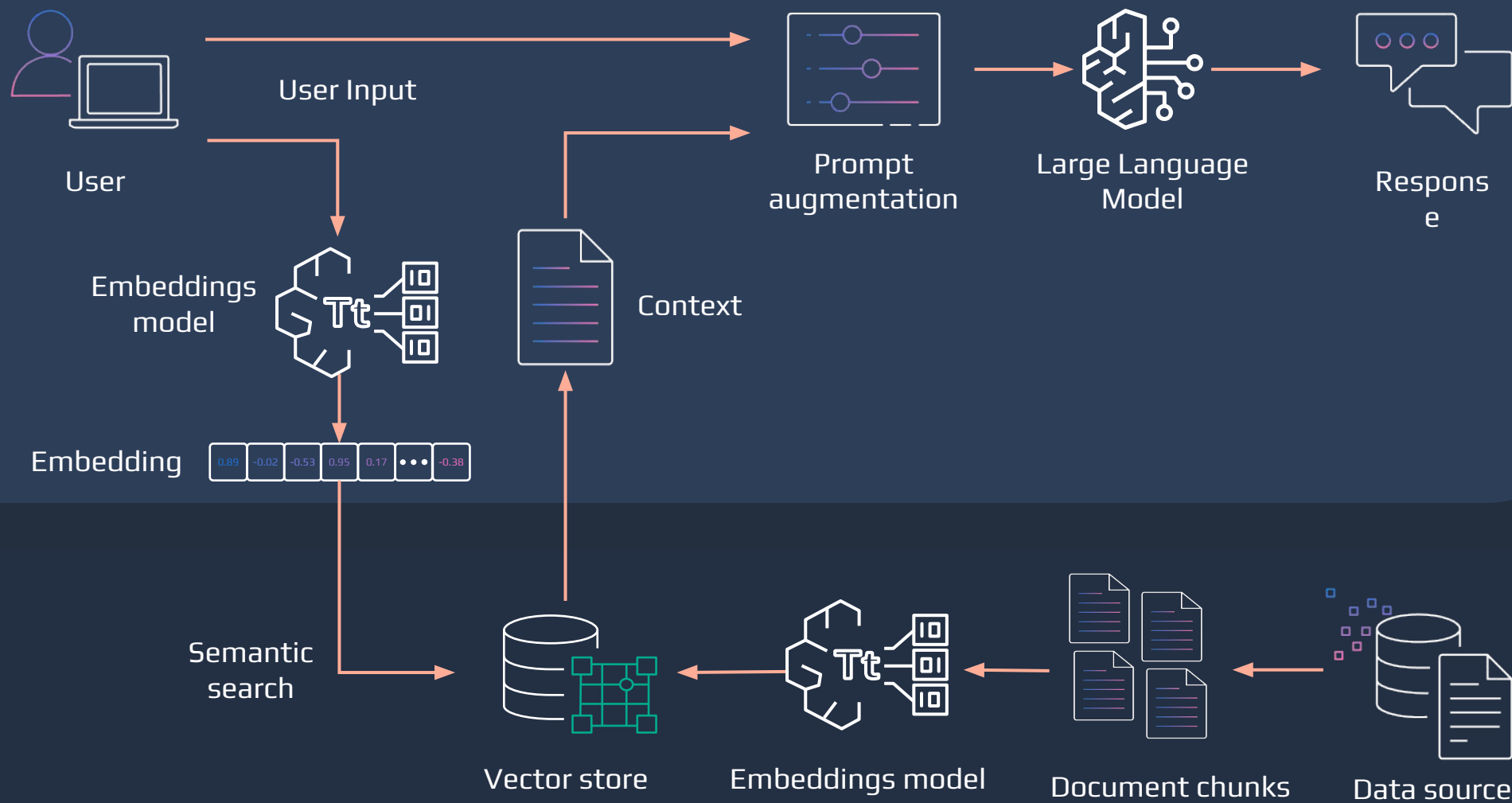
Product Overview
Text embedding models are neural networks that transform texts into numerical vectors. They are a crucial building block for semantic search/retrieval systems and retrieval-augmented

Highlights

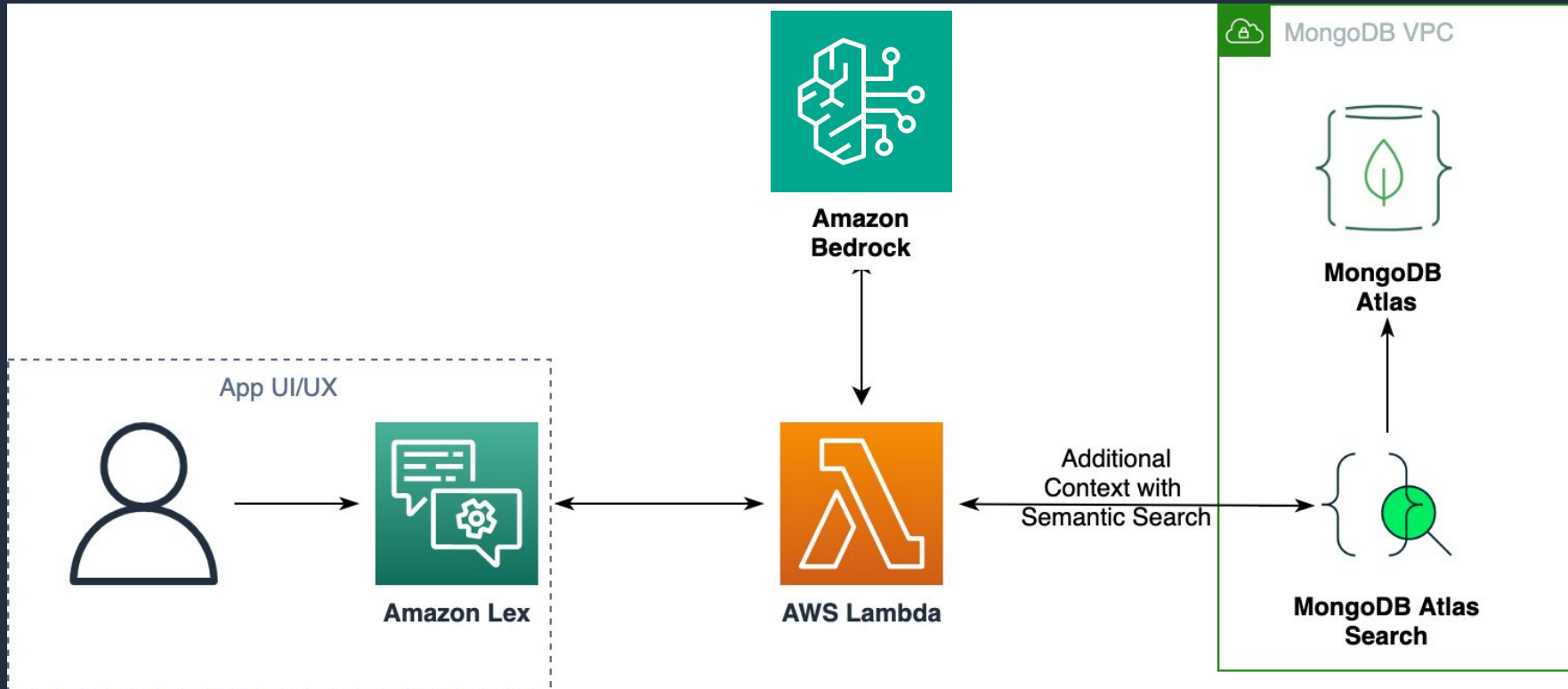
- Optimized for general-purpose and multilingual retrieval quality
- Smaller embedding dimension (1024)

Everybody loves RAG!

RAG in action



MongoDB Atlas and Amazon Bedrock architecture



- Simple bot
- Orchestration in Lambda
- Interaction with LLMs via Bedrock APIs
- Ingestion flow is a separate process

Code example for Simple Bot

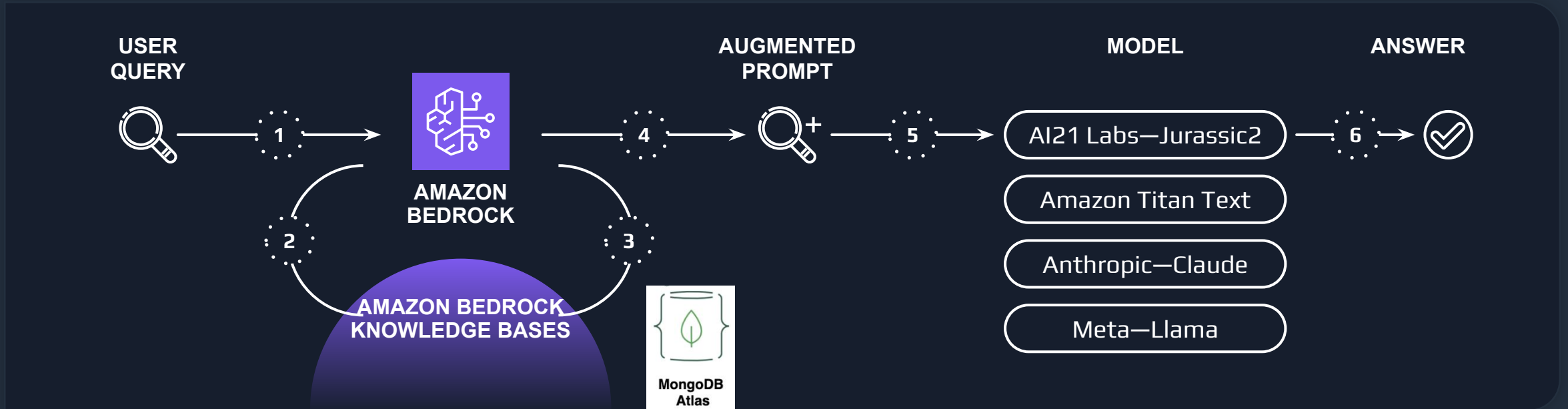
```
def mdb_query(mdbclient, query, kcount):
    text_as_embeddings = embeddings.embed_documents([query])
    embedding_value = text_as_embeddings[0]
    print("embedding size: " + str(len(embedding_value)))

    # get the vector search results based on the filter conditions.
    response = collection.aggregate([
        {
            "$vectorSearch": {
                "index": "vector_index",
                "path": "eg_vector",
                "queryVector": text_as_embeddings[0],
                "numCandidates": 20,
                "limit": 4
            }
        }, {
            '$project': {
                'score': {'$meta': 'searchScore'},
                'field_name_to_be_vectorized': 1,
                'title': 1,
                'claim_id': 1,
                '_id': 0
            }
        }
    ])
    return response
```

- Explicit Embeddings generation
- MongoDB API to search
- Need to handle ingestion

Amazon Bedrock Knowledge Bases

NATIVE SUPPORT FOR RAG



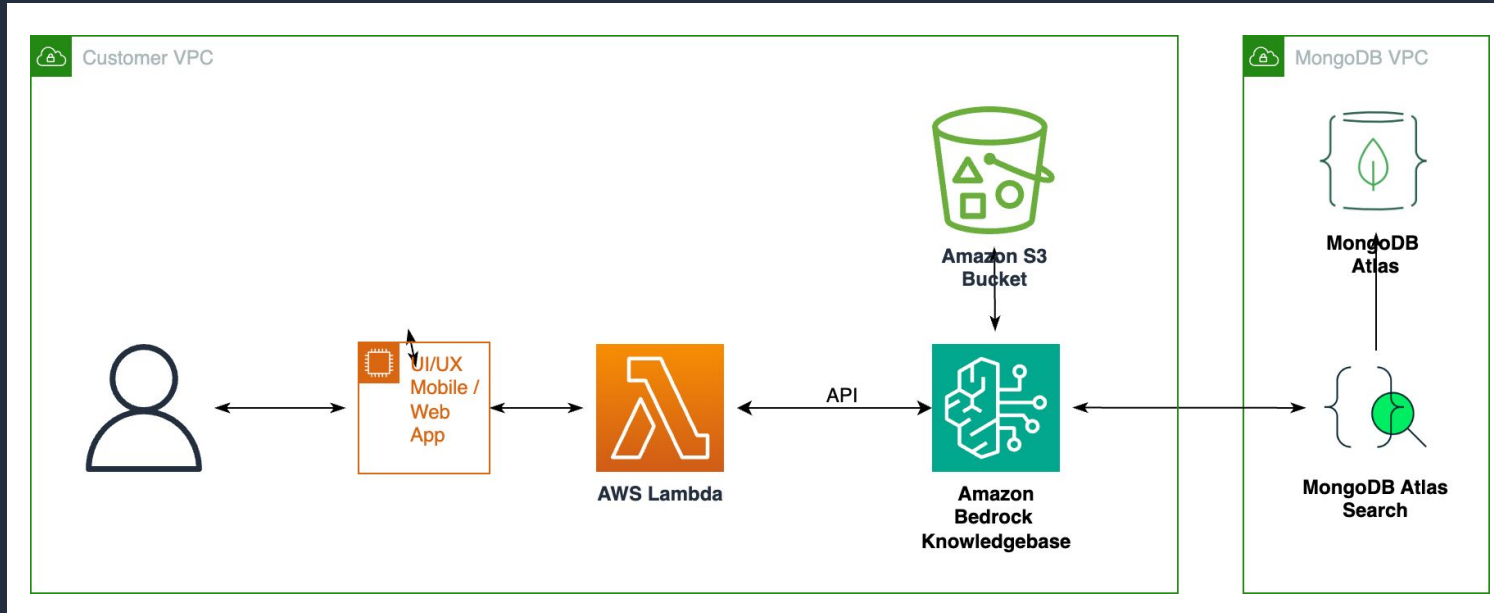
Securely connect FMs to data sources for RAG to deliver more relevant responses

Fully managed RAG workflow including ingestion, retrieval, and augmentation

Built-in session context management for multiturn conversations

Automatic citations with retrievals to improve transparency

MongoDB Atlas Bedrock—Reference architecture KB Only



- Content/Knowledge Ingestion handled by Bedrock
- UX: Web or Mobile App
- Lambda interacts w/ Bedrock APIs
- Semantic Search using Atlas Vector Search
- Text and Hybrid Search with Atlas!

Code example for Bedrock KB

```
from langchain_aws import AmazonKnowledgeBasesRetriever

# initialize AmazonKnowledgeBaseRetriever
retriever = AmazonKnowledgeBasesRetriever(
    knowledge_base_id=os.environ.get("BEDROCK_KB_ID"),
    region_name="us-west-2",
    retrieval_config={
        "vectorSearchConfiguration": {
            "numberOfResults": 5,
            "overrideSearchType": "SEMANTIC",
        }
    },
)

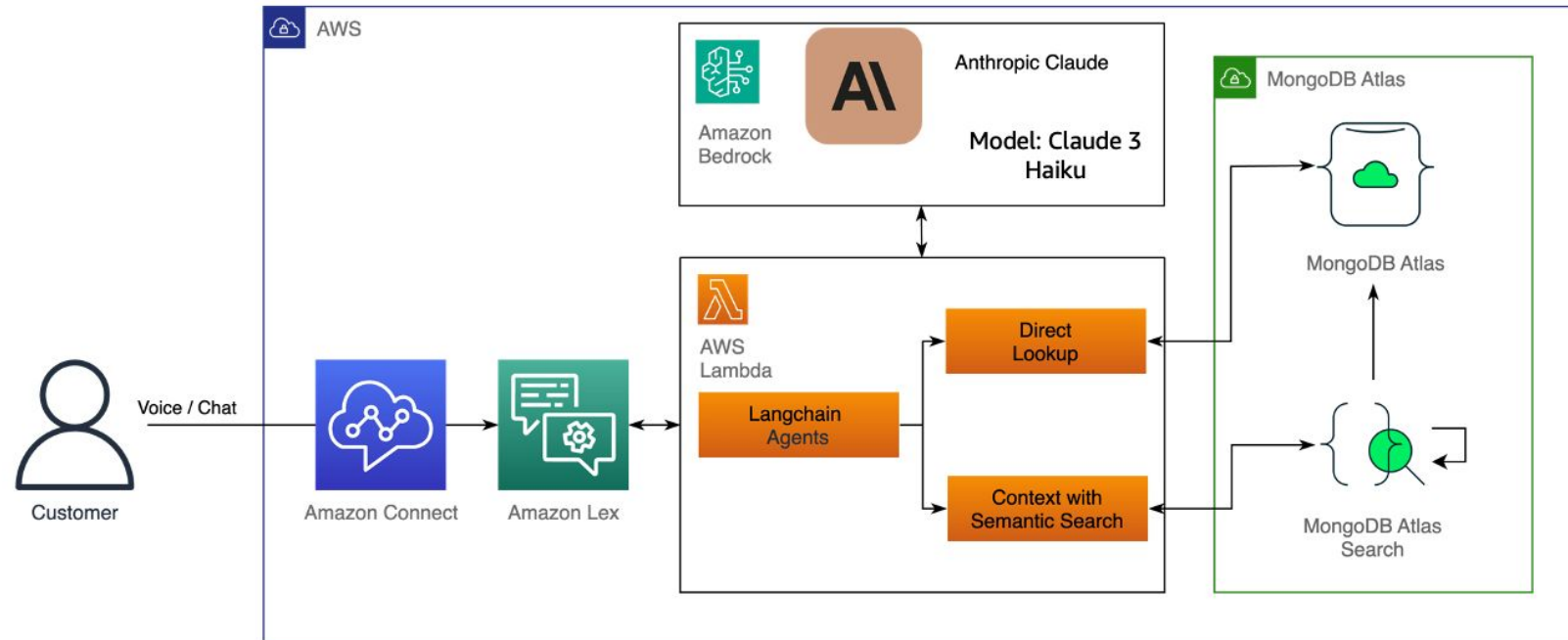
# Streamed response emulator
def response_generator(query_string):
    res = retriever.invoke(query_string, region_name="us-west-2")

    print("finished search...")
```

- No need for explicit Embeddings generation
- Simple API to search

Let's Build with Agents!

MongoDB Atlas, Amazon Bedrock, Langchain, and Anthropic



- More complex bot
- Orchestration in Lambda
- Langchain Agent/Tools
- Direct/Lexographical/Semantic Searches
- Interaction with LLMs via Bedrock APIs
- Ingestion flow is a separate process

<https://github.com/mongodb-partners/QS-Anthropic-1>

Code Example for Langchain Agents

```
def mongodb_place_lookup_by_name(query_str: str) -> str:
    """
    Retrieve place information by place name from the MongoDB database.

    This function connects to the MongoDB Atlas database and searches for a place
    using the provided name query. The search checks both the place name and country,
    using case-insensitive regex matching. The function returns the first matching
    document without the ID field.

    Args:
        query_str (str): The place name to search for. It may include partial matches.

    Returns:
        str: A string representation of the first matching document's details without the ID field.
    """
    res = ""
    filter = {
        "$or": [
            {"Place Name": {"$regex": query_str, "$options": "i"}},
            {"Country": {"$regex": query_str, "$options": "i"}},
        ]
    }
    project = {"_id": 0, "details_embedding": 0}

    res = collection.find_one(filter=filter, projection=project)
    return str(res)
```

- Langchain constructs the tools for an agent from the functions

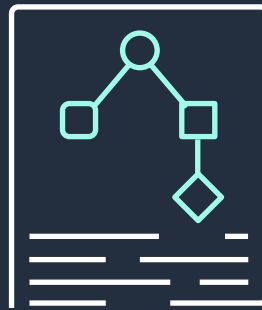
Amazon Bedrock Agents

Amazon Bedrock Agents

ENABLE GENERATIVE AI APPLICATIONS TO EXECUTE MULTISTEP TASKS USING
COMPANY SYSTEMS AND DATA SOURCES



**Decompose into steps
using available actions
and Amazon Bedrock Knowledge
Bases**



**Execute action or
search knowledge base**

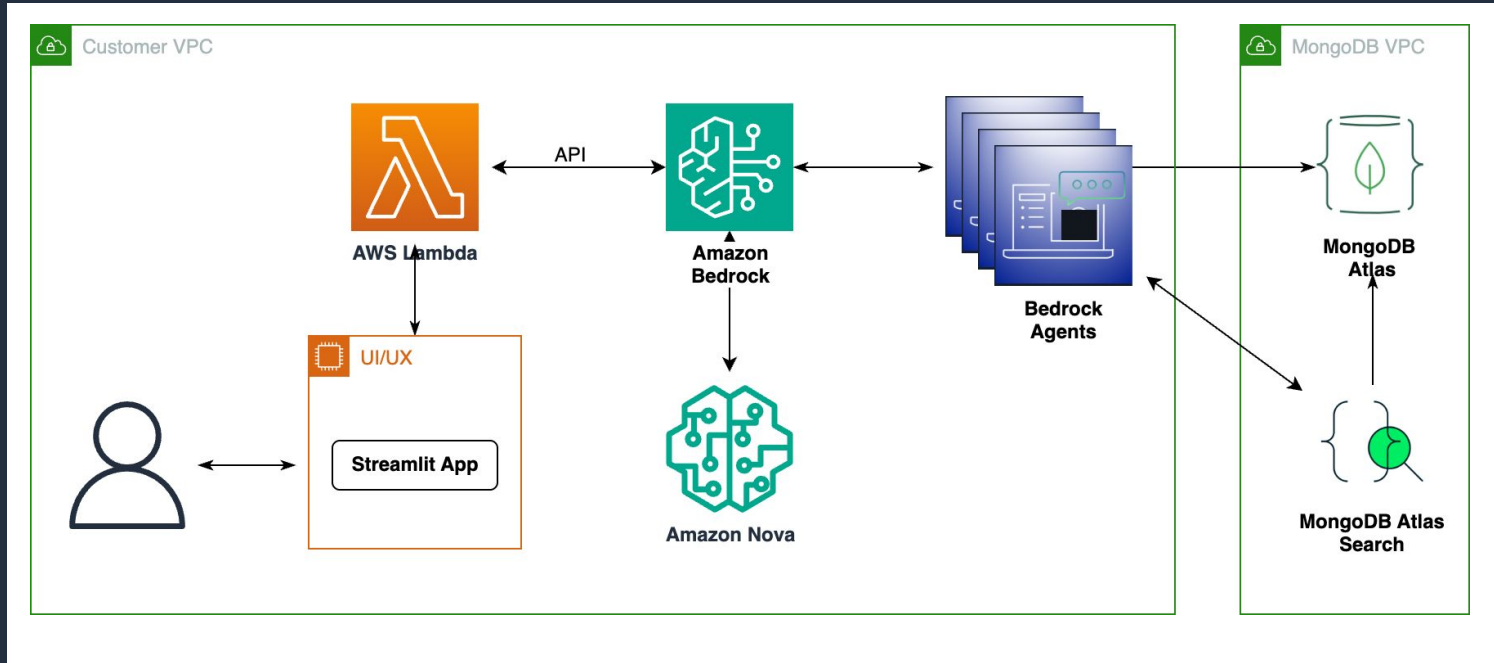
Observe results

Think about next step



Until final answer

MongoDB Atlas Bedrock—Reference architecture Agents Only



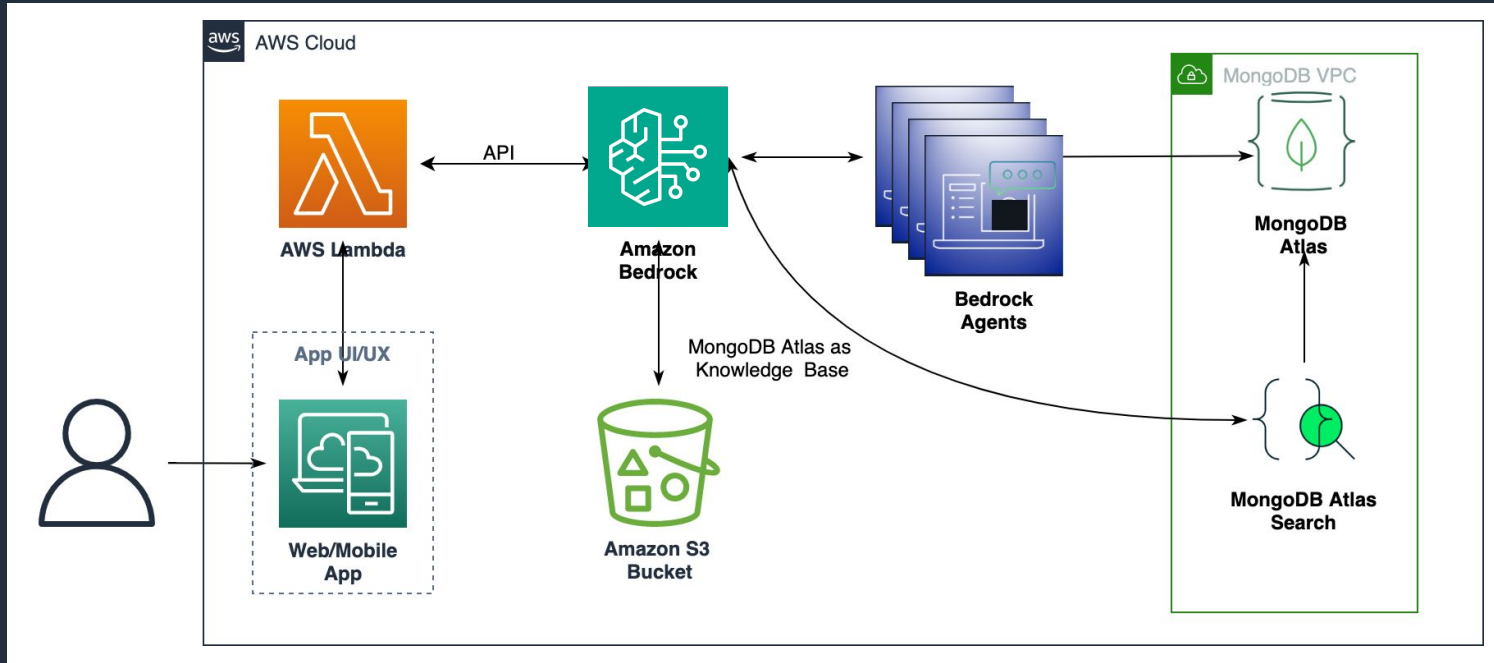
- Content/Knowledge Ingestion handled by Bedrock
- UX: Web or Mobile App
- Lambda interacts w/ Bedrock APIs
- Semantic Search using Atlas Vector Search

Code Example for Bedrock Agents

- Lambda Power Tools framework to simplify development

```
@app.get("/get_place_by_country", description="Retrieve places by country name")
@tracer.capture_method
def place_lookup_by_country(query_str: Annotated[str, Query(description="The country name")])
    ) -> Annotated[str, Body(description="Place Names")]:
    logger.info(f"Looking up places by country: {query_str}")
    client = get_mongo_client()
    # get database and collection
    collection = get_travel_collection(client)
    res = ""
    res = collection.aggregate(
        [
            {"$match": {"Country": {"$regex": query_str, "$options": "i"}}},
            {"$project": {"Place Name": 1}},
        ]
    )
    places = []
    for place in res:
        places.append(place["Place Name"])
    logger.info(f"Found {len(places)} places in country: {query_str}")
    return str(places)
```

MongoDB Atlas Bedrock—Reference architecture Combined



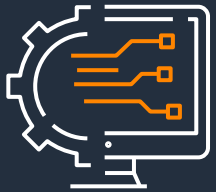
- Content/Knowledge Ingestion handled by Bedrock
- UX: Web or Mobile App
- Lambda interacts w/ Bedrock APIs
- Semantic Search using Atlas Vector Search



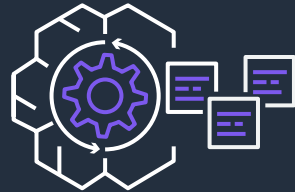
Amazon Bedrock AgentCore



Essential tools and capabilities to build highly effective agents



Deploy securely
at scale



Enhance with tools
and memory



AgentCore Gateway



AgentCore Memory



AgentCore Browser



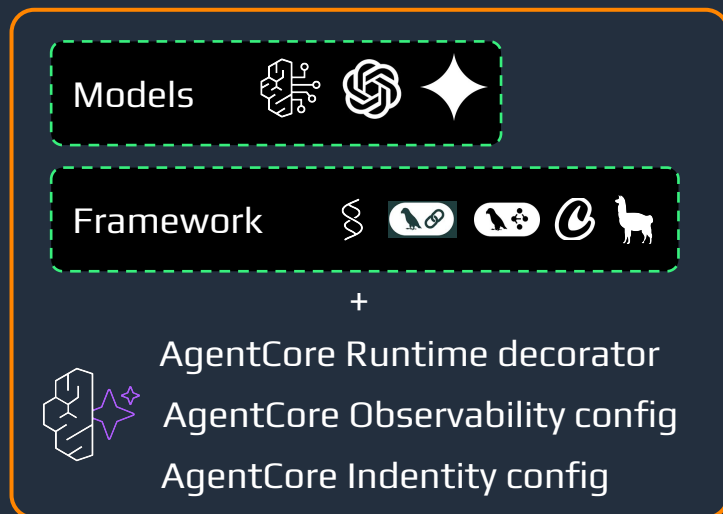
AgentCore Code Interpreter



Monitor

AgentCore Runtime

Agent or tool code



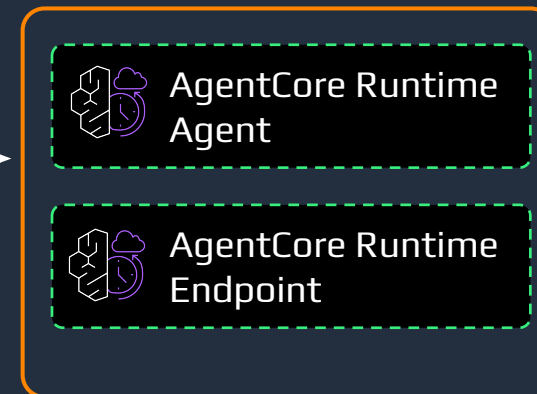
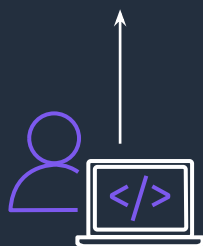
configure



launch

AgentCore Runtime

Amazon ECR Repository



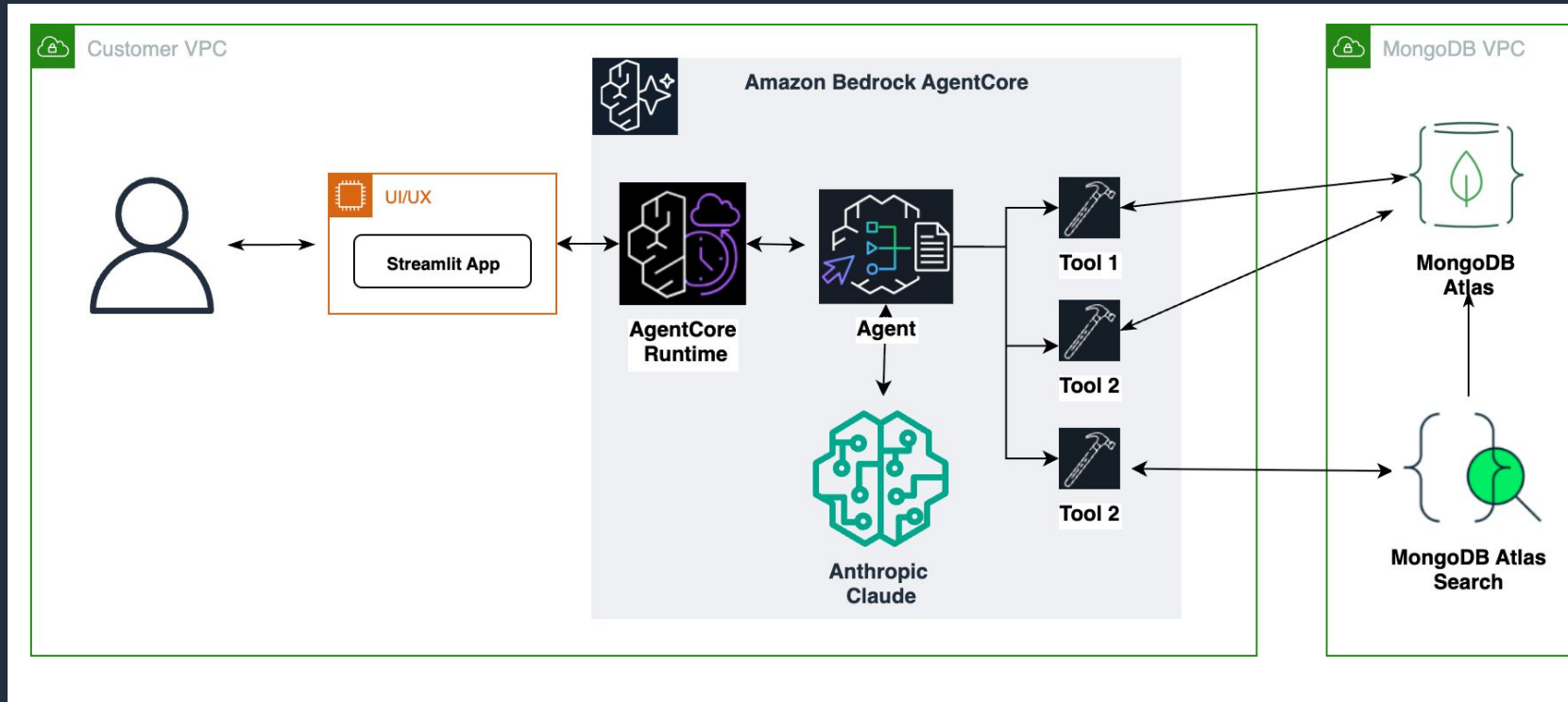
User



Application

invoke

Bedrock AgentCore—Reference architecture



- Compute is managed by AgentCore
- UX: Web or Mobile App
- Semantic Search using Atlas Vector Search

Code Example for Bedrock AgentCore Strands

```
@tool
def place_lookup_by_country(query_str: str) -> str:
    """Retrieve places by country name

    Args:
        query_str: The country name to search for

    Returns:
        List of place names in the specified country
    """
    logger.info(f"Looking up places by country: {query_str}")
    client = get_mongo_client()
    # get database and collection
    collection = get_travel_collection(client)
    res = ""
    res = collection.aggregate(
        [
            {"$match": {"Country": {"$regex": query_str, "$options": "i"}}},
            {"$project": {"Place Name": 1}},
        ]
    )
    places = []
    for place in res:
        places.append(place["Place Name"])
    logger.info(f"Found {len(places)} places in country: {query_str}")
    return str(places)
```

- Annotation to define a tool
- No Lambda
- Similar to other orchestrations
- Easy migration

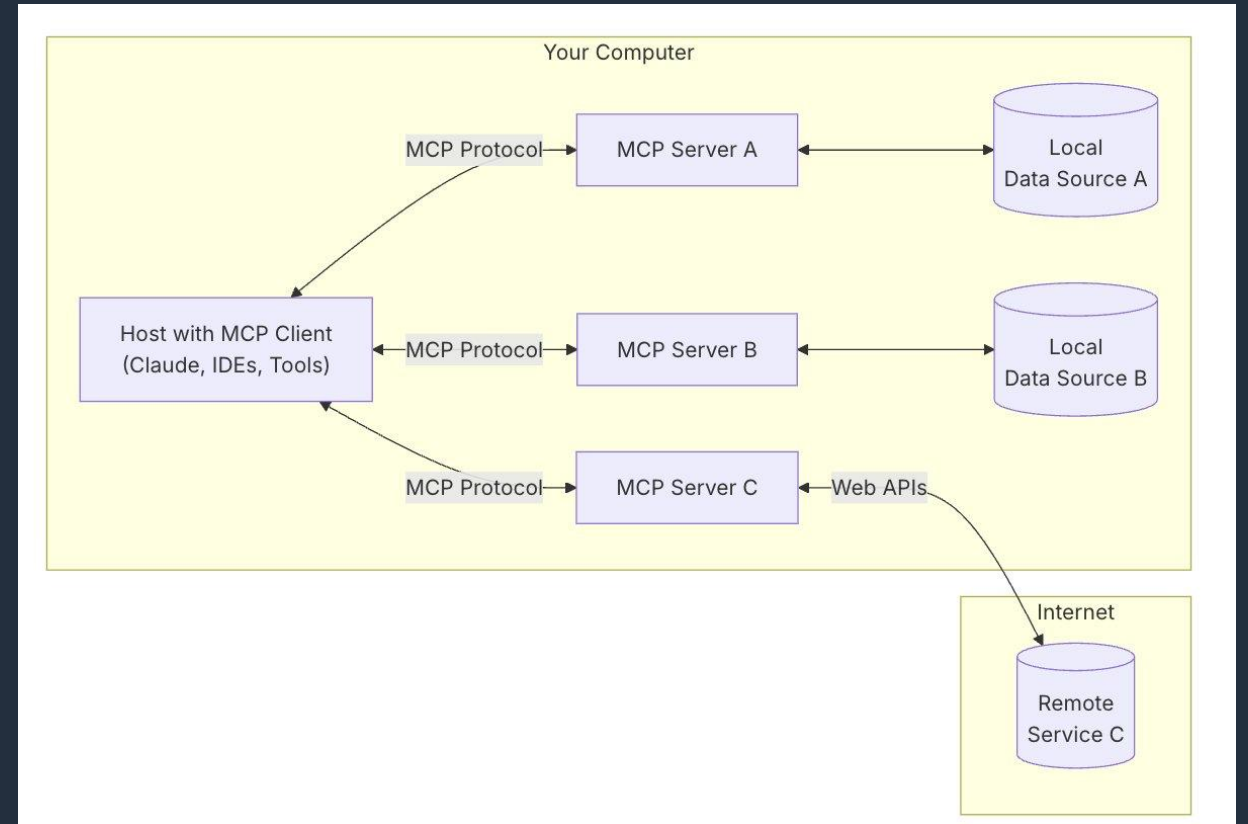
Amazon Q Developer

– Q for Command line (Q Developer CLI)



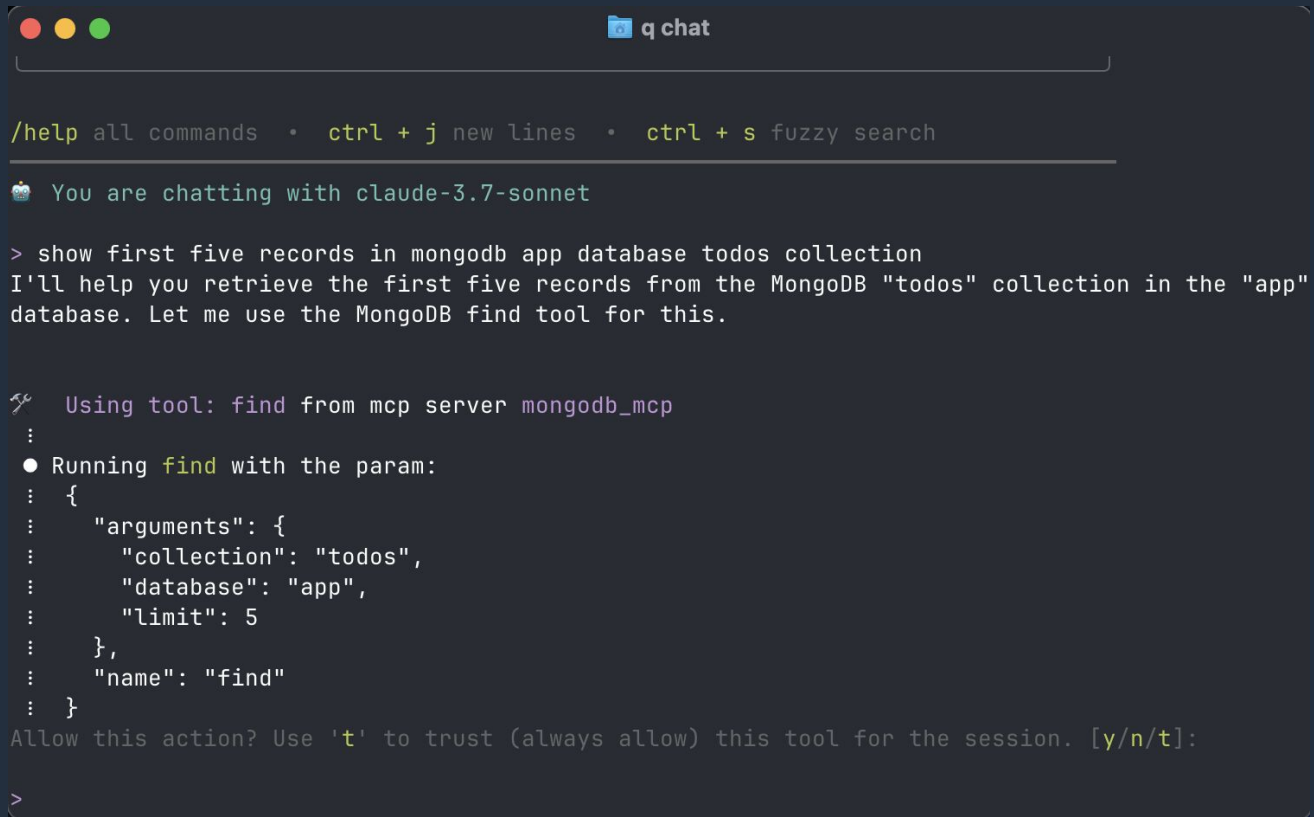
MCP Architecture

- **MCP Host:** LLM Applications that initiate connection. Example: Cursor, Windsurf, Claude Desktop etc.
- **MCP Client:** Maintain 1:1 connections with servers, inside the host application
- **MCP Server:** provide context, tools, and prompt to clients



Amazon Q CLI + MongoDB MCP Server

Amazon Q Developer on the command line



```
q chat

/help all commands • ctrl + j new lines • ctrl + s fuzzy search

👤 You are chatting with claude-3.7-sonnet

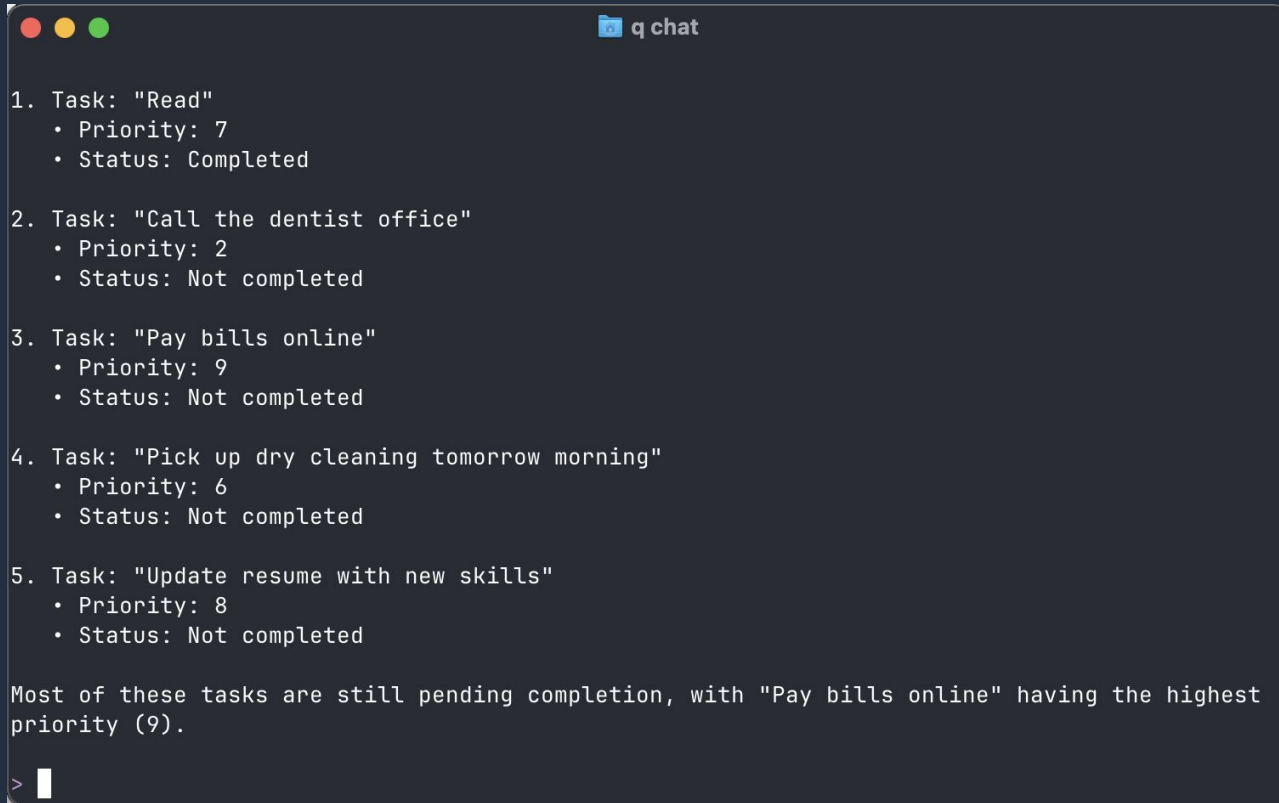
> show first five records in mongodb app database todos collection
I'll help you retrieve the first five records from the MongoDB "todos" collection in the "app" database. Let me use the MongoDB find tool for this.

🔧 Using tool: find from mcp server mongodb_mcp
:
● Running find with the param:
: {
:   "arguments": {
:     "collection": "todos",
:     "database": "app",
:     "limit": 5
:   },
:   "name": "find"
: }
```

- Command line interface to chat through natural language conversations, questions, and responses within your terminal environment.
- Translate natural language instructions to executable shell code snippets.
- Inline suggestions as you type in your command prompt, with completions to popular CLIs.

Amazon Q CLI + MongoDB MCP Server

Amazon Q Developer on the command line



```
1. Task: "Read"
  • Priority: 7
  • Status: Completed

2. Task: "Call the dentist office"
  • Priority: 2
  • Status: Not completed

3. Task: "Pay bills online"
  • Priority: 9
  • Status: Not completed

4. Task: "Pick up dry cleaning tomorrow morning"
  • Priority: 6
  • Status: Not completed

5. Task: "Update resume with new skills"
  • Priority: 8
  • Status: Not completed

Most of these tasks are still pending completion, with "Pay bills online" having the highest priority (9).

> 
```

- Command line interface to chat through natural language conversations, questions, and responses within your terminal environment.
- Translate natural language instructions to executable shell code snippets.
- Inline suggestions as you type in your command prompt, with completions to popular CLIs.

Kiro



Starting



Let's build

Plan, search, or build anything



Vibe

Chat first, then build. Explore ideas and iterate as you discover needs.



Spec

Plan first, then build. Create requirements and design before coding starts.

Great for:

- Thinking through features in-depth
- Projects needing upfront planning
- Building features in a structured way

Create CRUD React Application for todos collection in MongoDB.



Claude Sonnet 3.7  Autopilot 

[Report a bug / Suggest an idea](#) [Kiro Public Preview](#) [Autocomplete](#)

ces, Inc. or its affiliates. All rights reserved.

Spec: todo-crud-app

1 Requirements2 Design3 Task list

Refine

1# Requirements Document

2

3## Introduction

4

5This feature involves creating a full-stack CRUD (Create, Read, Update, Delete) React application that manages a todos collection stored in MongoDB. The application will provide users with a modern, responsive interface to manage their todo items with full database persistence. Users will be able to add new todos, view their existing todos, edit todo details, mark todos as complete/incomplete, and delete todos they no longer need.

6

7## Requirements

8

9### Requirement 1

10

11**User Story:** As a user, I want to create new todo items, so that I can track tasks I need to complete.

12

13#### Acceptance Criteria

14

151. WHEN the user enters a todo title and clicks "Add Todo" THEN the system SHALL create a new todo item in the MongoDB collection

162. WHEN a new todo is created THEN the system SHALL display

Create CRUD React Application for todos collection in MongoDB.

Kiro

I'll help you create a spec for a CRUD React Application for a todos collection in MongoDB. Let me start by creating the requirements document for this feature.

Created requirements.md

I've created the initial requirements document for your CRUD React Application for todos collection in MongoDB. The requirements cover all the essential CRUD operations (Create, Read, Update, Delete) along with user experience considerations like responsive design and error handling.

Do the requirements look good? If so, we can move on to the design.

Move to design phase

Ask a question or describe a task...

#🔍

Claude Sonnet 3.7Autopilot

Ln 15, Col 130Spaces: 4UTF-8LFMarkdownReport a bug / Suggest an ideaKiro Public PreviewAutocomplete



Design

Spec: todo-crud-app 1 Requirements 2 Design 3 Task list ↻ Refine

1 # Design Document

2

3 ## Overview

4

5 The Todo CRUD application will be built as a full-stack web application using React for the frontend and Node.js with Express for the backend API. The application will use MongoDB as the database with Mongoose as the ODM (Object Document Mapper). The architecture follows a client-server pattern with RESTful API endpoints for data operations.

6

7 The frontend will be a single-page application (SPA) built with React, featuring a clean and responsive user interface. The backend will provide a REST API that handles all CRUD operations and communicates with the MongoDB database. This separation allows for scalability and maintainability.

8

9 ## Architecture

10

11 ```mermaid

12 graph TB

Do the requirements look good? If so, we can move on to the design.

Great! Now I'll create the design document based on the approved requirements. Let me research the best practices for building a CRUD React application with MongoDB and then create a comprehensive design.

Created design.md

I've created a comprehensive design document that covers the full-stack architecture for your Todo CRUD application. The design includes:

- A three-tier architecture with React frontend, Express.js backend, and MongoDB database
- Detailed component structure and API interfaces
- Data models and schemas
- Error handling strategies
- Testing approach
- Technology stack recommendations

Does the design look good? If so, we can move on to the implementation plan.

Move to implementation plan

Tasks

Spec: todo-crud-app 1 Requirements 2 Design 3 Task list Update tasks

```
1 # Implementation Plan
2
3 ⚡ Start task
4 [ ] 1. Set up project structure and development environment
5   - Create React frontend project with Vite
6   - Create Node.js backend project with Express
7   - Set up package.json files with required dependencies
8   - Configure development scripts and environment variables
9   - Requirements: 6.5_
10
11 ⚡ Start task
12 [ ] 2. Implement MongoDB connection and Todo model
13   - Set up MongoDB connection with Mongoose
14   - Create Todo schema with validation rules
15   - Implement database connection utilities and error handling
16   - Write unit tests for Todo model validation
17   - Requirements: 1.1, 3.2, 4.2, 5.2_
18
19 ⚡ Start task
20 [ ] 3. Create backend API endpoints for CRUD operations
21   ⚡ Start task
22   [ ] 3.1 Implement GET /api/todos endpoint
23     - Create route handler to fetch all todos from database
24     - Implement proper error handling and response formatting
25     - Write unit tests for GET endpoint
26     - Requirements: 2.1, 2.4, 2.5_
```

Does the design look good? If so, we can move on to the implementation plan.

Perfect! Now I'll create the implementation plan with actionable coding tasks based on the approved design and requirements.

Created tasks.md

I've created a comprehensive implementation plan that breaks down the Todo CRUD application development into 11 main tasks with detailed sub-tasks. The plan follows a logical progression:

1. **Project Setup** - Environment and structure
2. **Database Layer** - MongoDB connection and models
3. **Backend API** - All CRUD endpoints with tests
4. **Frontend Setup** - React project structure
5. **Core Components** - Main React components
6. **Styling** - Responsive UI implementation
7. **Error Handling** - Comprehensive error management
8. **Status Toggle** - Completion functionality
9. **Integration Tests** - End-to-end testing
10. **Build Scripts** - Development workflow
11. **Final Polish** - Optimization and accessibility

Each task includes specific coding objectives and references the requirements they fulfill.

Do the tasks look good?

Ask a question or describe a task...

#

Claude Sonnet 3.7 Autopilot

Execution

 Task in progress |  View changes

- [] 1. Set up project structure and development environment
 - Create React frontend project with Vite
 - Create Node.js backend project with Express
 - Set up package.json files with required dependencies
 - Configure development scripts and environment variables
 - Requirements: 6.5_

 Start task

- [] 2. Implement MongoDB connection and Todo model
 - Set up MongoDB connection with Mongoose
 - Create Todo schema with validation rules
 - Implement database connection utilities and error handling
 - Write unit tests for Todo model validation
 - Requirements: 1.1, 3.2, 4.2, 5.2_

 Start task

- [] 3. Create backend API endpoints for CRUD operations

 Start task

- [] 3.1 Implement GET /api/todos endpoint
 - Create route handler to fetch all todos from database
 - Implement proper error handling and response formatting
 - Write unit tests for GET endpoint
 - Requirements: 2.1, 2.4, 2.5_

and development environment."

 Task: 1. Set up project structure and development environment from `tasks.md`
Status: In Progress

Let me start by creating the project structure. I'll create both the frontend and backend directories and set up the initial configuration.

 Searched workspace

Checking current directory structure to understand the workspace

Now I'll create the project structure with separate frontend and backend directories:

 Created `package.json`  

 Created `package.json`  

> Working..

Cancel Follow 

Ask a question or describe a task...





Claude Sonnet 3.7  Autopilot 

Execute

Spec: todo-crud-app

1 Requirements

2 Design

3 Task list

Update tasks

1 # Implementation Plan

9

Start task

10 [] 2. Implement MongoDB connection and Todo model

11 - Set up MongoDB connection with Mongoose

12 - Create Todo schema with validation rules

13 - Implement database connection utilities and error handling

14 - Write unit tests for Todo model validation

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

node

+

⌵

🗑

⋮

⌵

[0] > cd backend && npm run dev

[0]

[0]

[0] > todo-backend@1.0.0 dev

[0] > nodemon server.js

[0]

[1]

[1] > todo-frontend@0.0.0 dev

[1] > vite

[1]

[0] [nodemon] 3.1.10

[0] [nodemon] to restart at any time, enter `rs`

[0] [nodemon] watching path(s): *.*

[0] [nodemon] watching extensions: js,mjs,cjs,json

[0] [nodemon] starting `node server.js`

[0] Server is running on port 5000

[1]

[1] VITE v5.4.19 ready in 1271 ms

[1]

[1] → Local: http://localhost:3000/

[1] → Network: use --host to expose

Proxy Configuration to Backend API

2. Backend (Express.js):

- Express server with CORS enabled
- Environment variable configuration
- Organized directory structure for scalability
- Development with nodemon for auto-restart

3. Development Scripts:

- `npm run dev` - Runs both frontend and backend concurrently
- `npm run install:all` - Installs all dependencies
- `npm run test` - Runs tests for both projects

Next Steps:

To get started, you would run:

```
npm run install:all # Install all dependencies
npm run dev         # Start both servers
```

The frontend will be available at `http://localhost:3000` and the backend at `http://localhost:5000`.

Task 1 is now complete! Would you like me to proceed with task 2: "Implement MongoDB connection and Todo model"?

> 23 changes accepted (🔗 [View all](#))

Revert



Ask a question or describe a task...



Claude Sonnet 3.7



Autopilot



Tasks Completed

Implementation Plan

✓ Task completed

- [x] 6. Implement responsive styling and user interface
 - Create CSS styles for mobile-first responsive design
 - Implement visual feedback for user actions
 - Style loading states and error messages
 - Ensure accessibility compliance with proper ARIA labels
 - Requirements: 6.1, 6.2, 6.3, 6.5

✓ Task completed

- [x] 7. Implement status toggle functionality
 - Add checkbox/toggle UI for completion status
 - Implement optimistic updates with error rollback
 - Add visual distinction for completed todos
 - Write tests for status toggle operations
 - Requirements: 4.1, 4.2, 4.3, 4.4, 4.5

✓ Task completed

- [x] 8. Set up development and build scripts
 - Configure concurrent development servers for frontend and backend
 - Set up production build processes
 - Create environment configuration for different deployment stages
 - Add linting and formatting scripts
 - Requirements: 6.5

Todo CRUD App

React + Express + MongoDB Todo Application

Add Todo

☐

second todo

Edit

Delete

☐

my first todo

Edit

Delete

Getting started

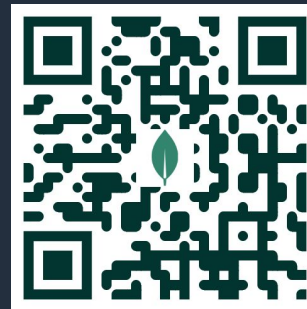


**Get started with
Amazon Bedrock AgentCore**

AI Agents with MongoDB:
<https://mdb.link/ai-agent-localnyc>



Discover Kiro



**MongoDB Atlas and
Amazon Bedrock hands-on
Workshop**

Thank you!

Igor Alekseev

ialek@amazon.com

