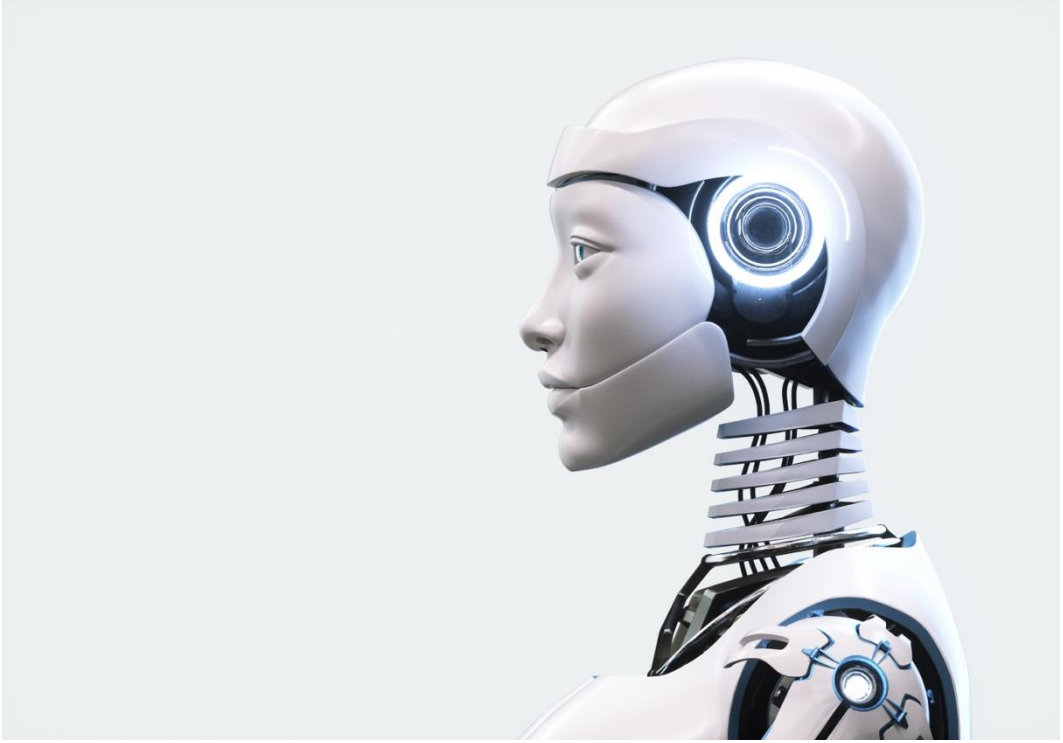
Patient Admission AI Challenge

Implemented by Brian Pulito



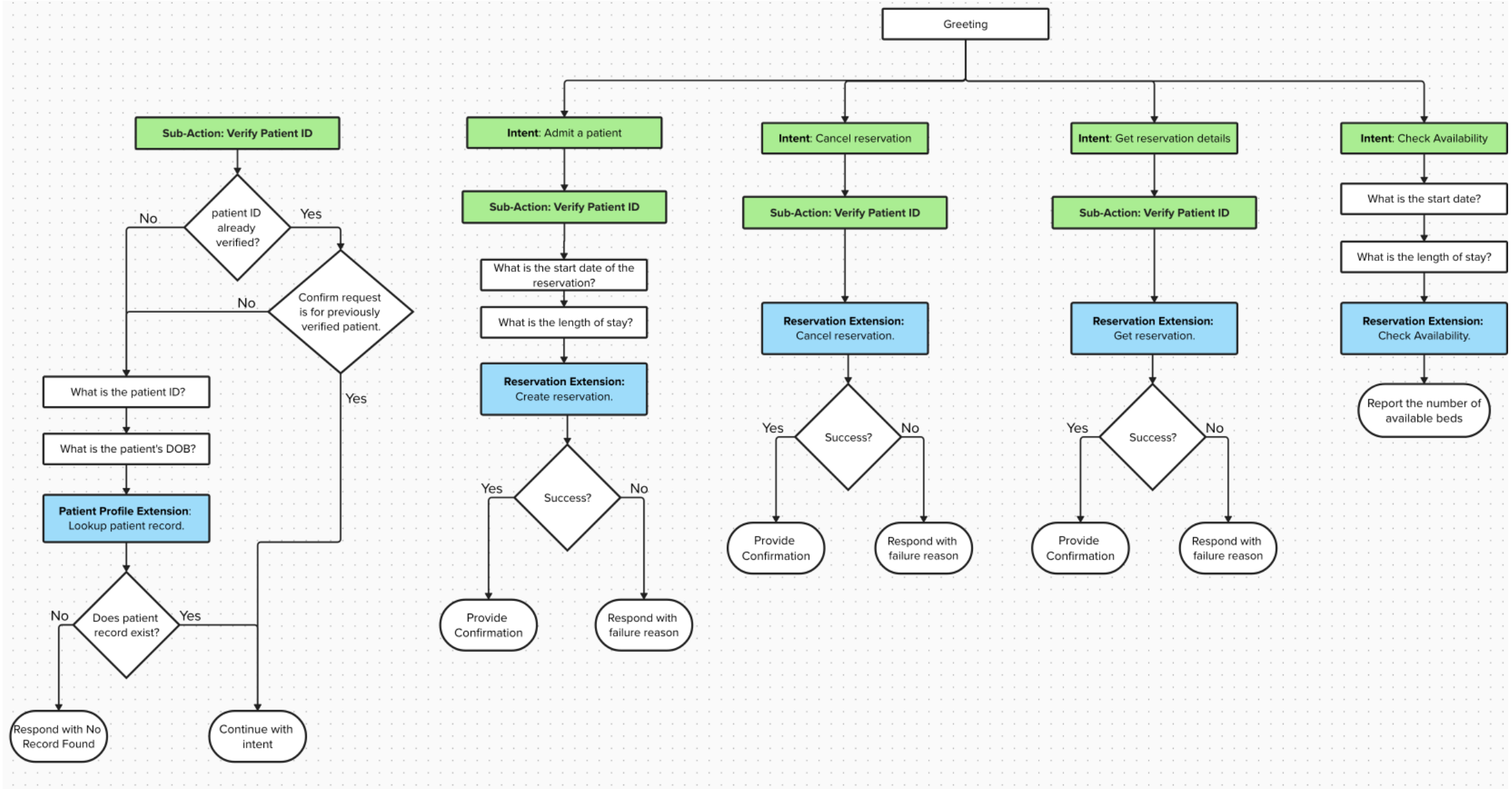
Project Requirements

- **Workflow**: Patient Admission & Bed Assignment implemented
- **Flow Design**: Link to slide
- **Prototype Implementation:** Link to git repo
- **Optimization, Scalability and Enhancements:** Link to slide
- **Logging, Monitoring and Testing:** Link to slide

High-level Implementation Overview

- Created both a Conversational AI Assistant and LLM based Agent to show two different approaches to solving the challenge.
- Used an Elasticsearch index to store reservation documents to allow for horizontal scaling of the solution to multiple users.
- Developed a python-based reservation-tool as a front-end to the Elastic index that runs in a Docker container on IBM Code Engine to enable remote access.
- Developed a python-based patient-profile serverless function running on IBM Code Engine to serve up static patient profiles for patient verification.
- Assistant and Agent were both developed on watsonx Orchestrate.
- UI elements used to demo solution came from watsonx Orchestrate.
- Twilio used to provide phone number and SIP trunk into watsonx Orchestrate to demo the phone channel. Watson Speech To Text and Text To Speech used for voice models.
- Uploaded an AI generated admissions guideline document to the vector store built into the watsonx Orchestrate Agent Builder to show semantic search using RAG.

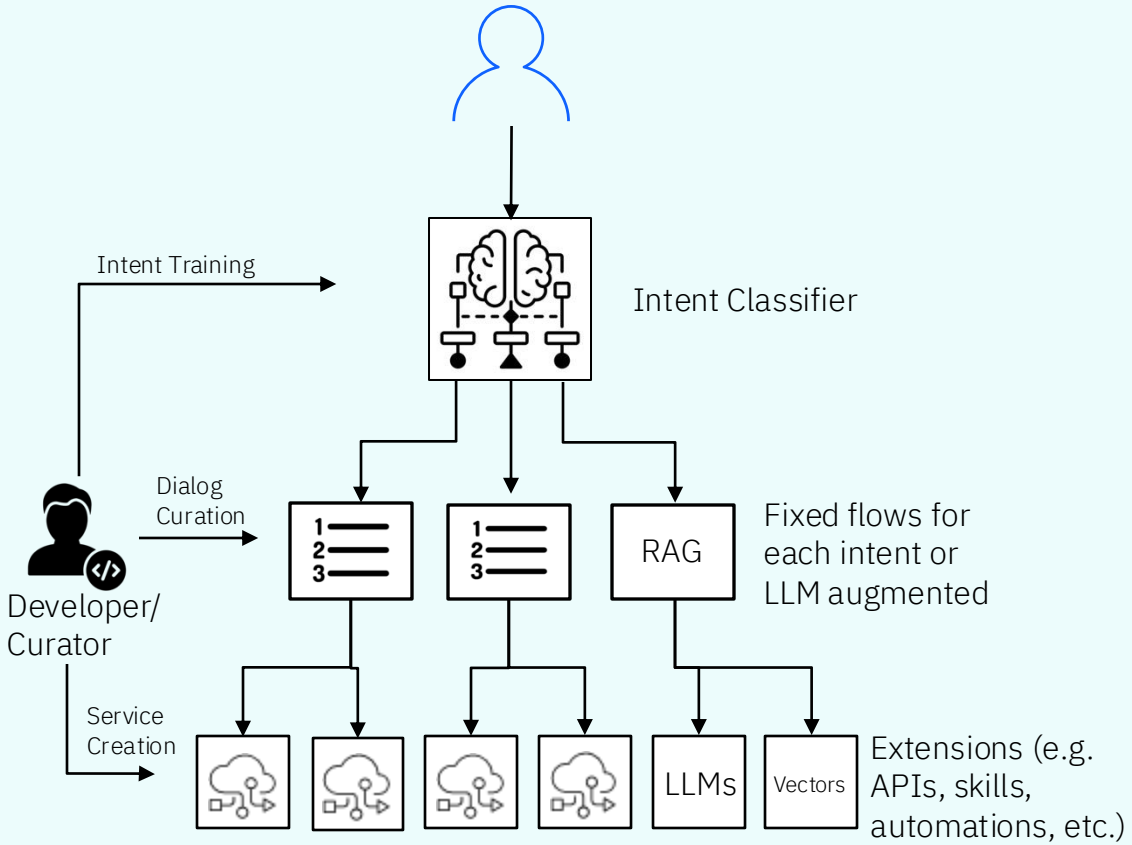
Assistant Intents and Flows



Two approaches to building AI solutions

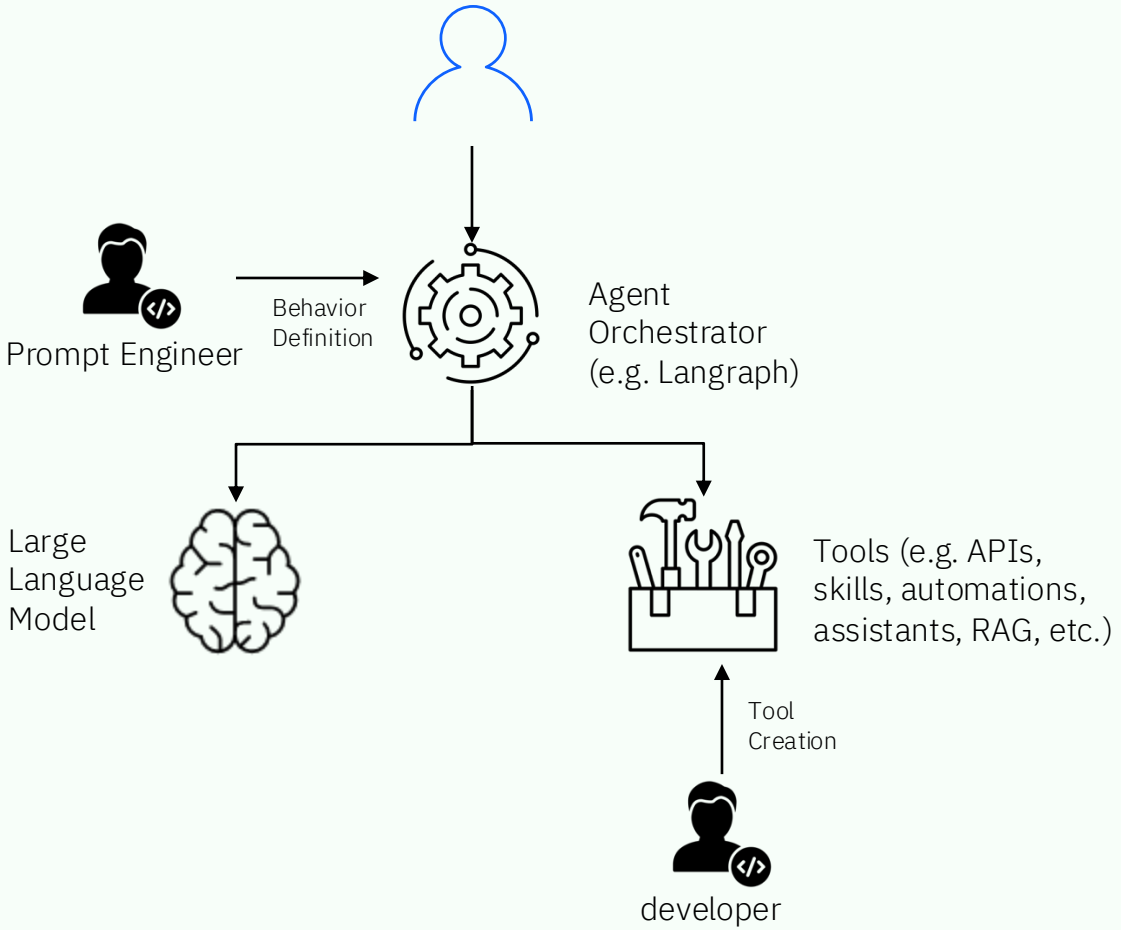
Assistant Based Solution

(e.g. Conversational AI)



Agent Based Solution

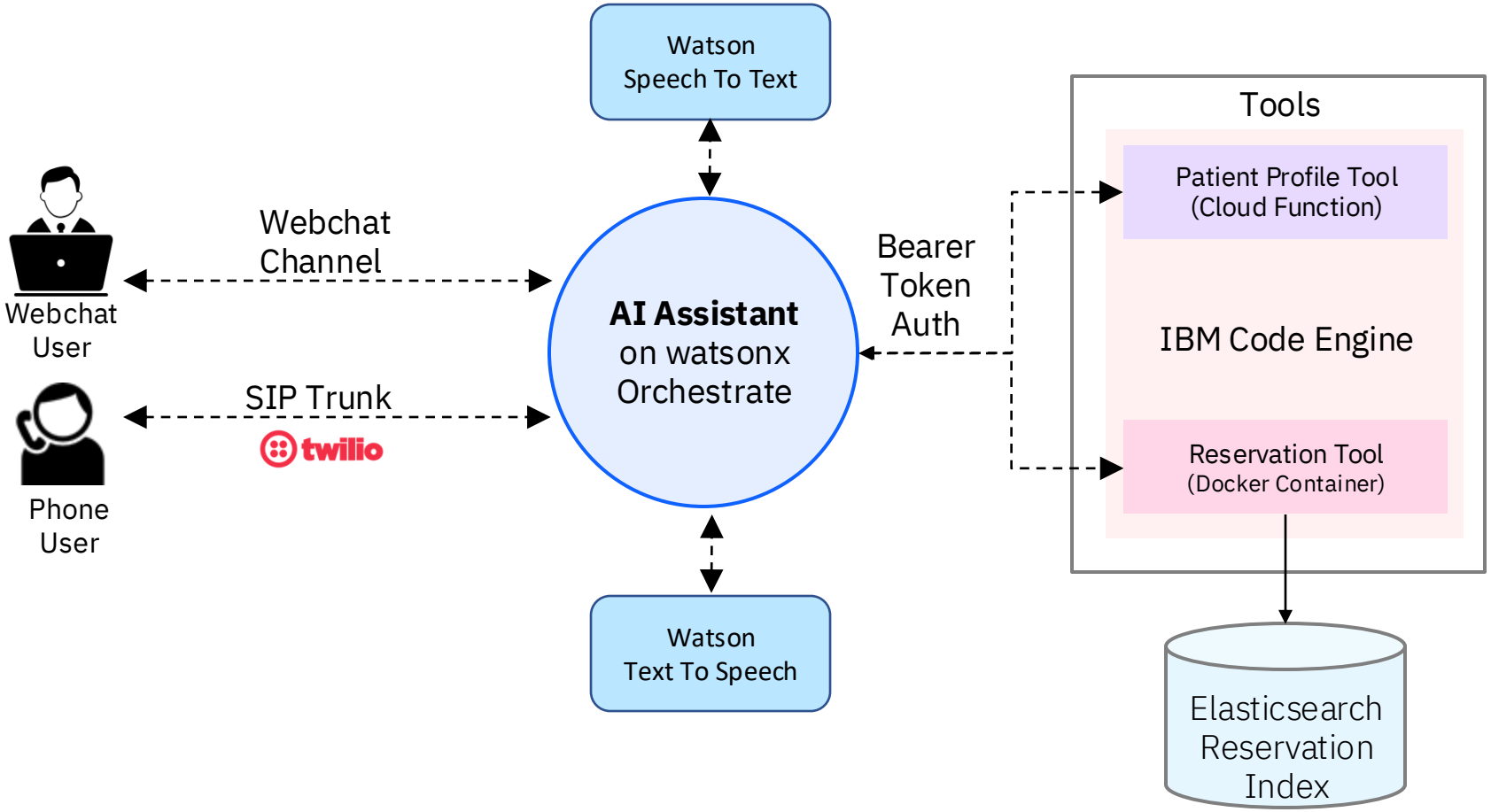
(e.g ReAct)



Patient Admissions Assistant

Built on watsonx Orchestrate, Watson Speech, Elasticsearch and IBM Code Engine.

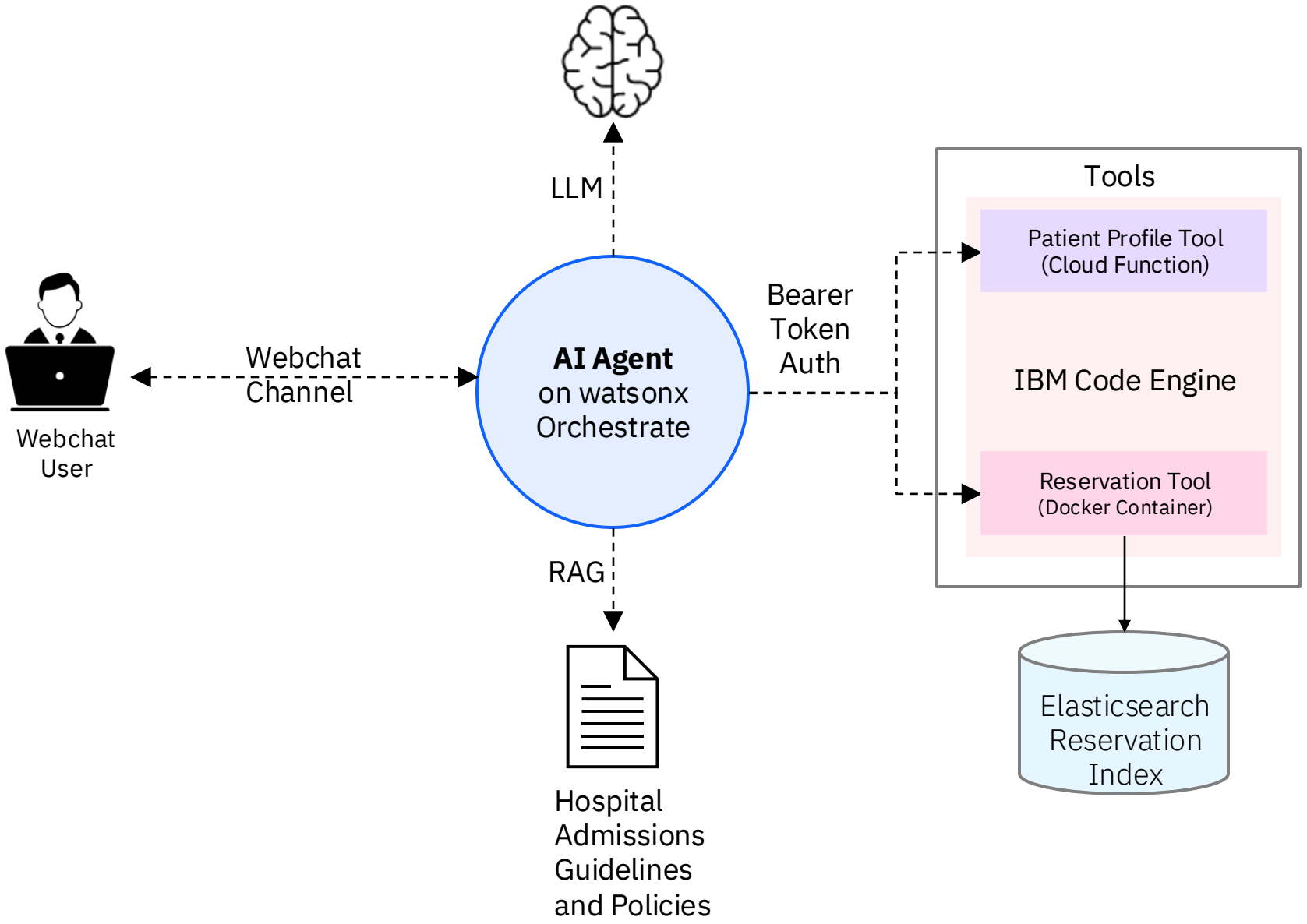
Exposes a webchat and phone channel.



Patient Admissions Agent

Built on watsonx Orchestrate. Elasticsearch and IBM Code Engine.

Exposes a webchat client.



Notes on Optimization, Scaling and Enhancements

- Optimizations
  - Direct user feedback (e.g., thumbs up or down) can be used to evaluate accuracy and appropriateness of responses and to improve prompts and dialog flows based on user feedback.
  - For anything that can be done deterministically (e.g., static responses, tool results, etc.) avoid using an LLM if possible. This will both reduce latency and avoid hallucinations. Use an LLM in a fallback capacity if possible.
  - Use RAG to ground both your assistants and agents in real knowledge to avoid hallucinations.
- Scaling
  - Assistants: add new Actions/Intents to increase the scope of the assistant. Fallback to RAG if possible.
  - Agents: Consider adding collaborator agents that a supervisor agent can use to increase the agent's capabilities.
  - Tooling should be stateless and should rely on persistent data stores for state.
  - Should implement method for locking the reservation index when making changes to allow multiple threads to interact with it at the same time.
- Enhancements
  - Reservation tool should include the ability to update an existing reservation.
  - Service should periodically clean up old reservations.

Notes on Logging, Monitoring and Testing

- Logging
  - Use message logging webhook or built in log store. Can be used to hand off history to a live agent or used for post session analytics.
  - Analytics based on conversation history logs for things like unrecognized intents, uncompleted intents, inaccurate RAG.
  - ELK (Elasticsearch, Logstash and Kibana) for logging events at scale.
- Monitoring
  - Log telemetry data for performance monitoring using tools such as LangFuse.
  - Small, purpose-built models can be used to monitor for things like prompt injections, inappropriate input, etc.
  - Prometheus for monitoring and alerting by collecting and storing time-series data.
- Testing
  - Use predefined list of inputs and expected outputs to drive regression testing. This capability is built into watsonx Orchestrate.
  - Use tools like Botium to test chat. Use tools like Hammer and sipp to test phone channel.