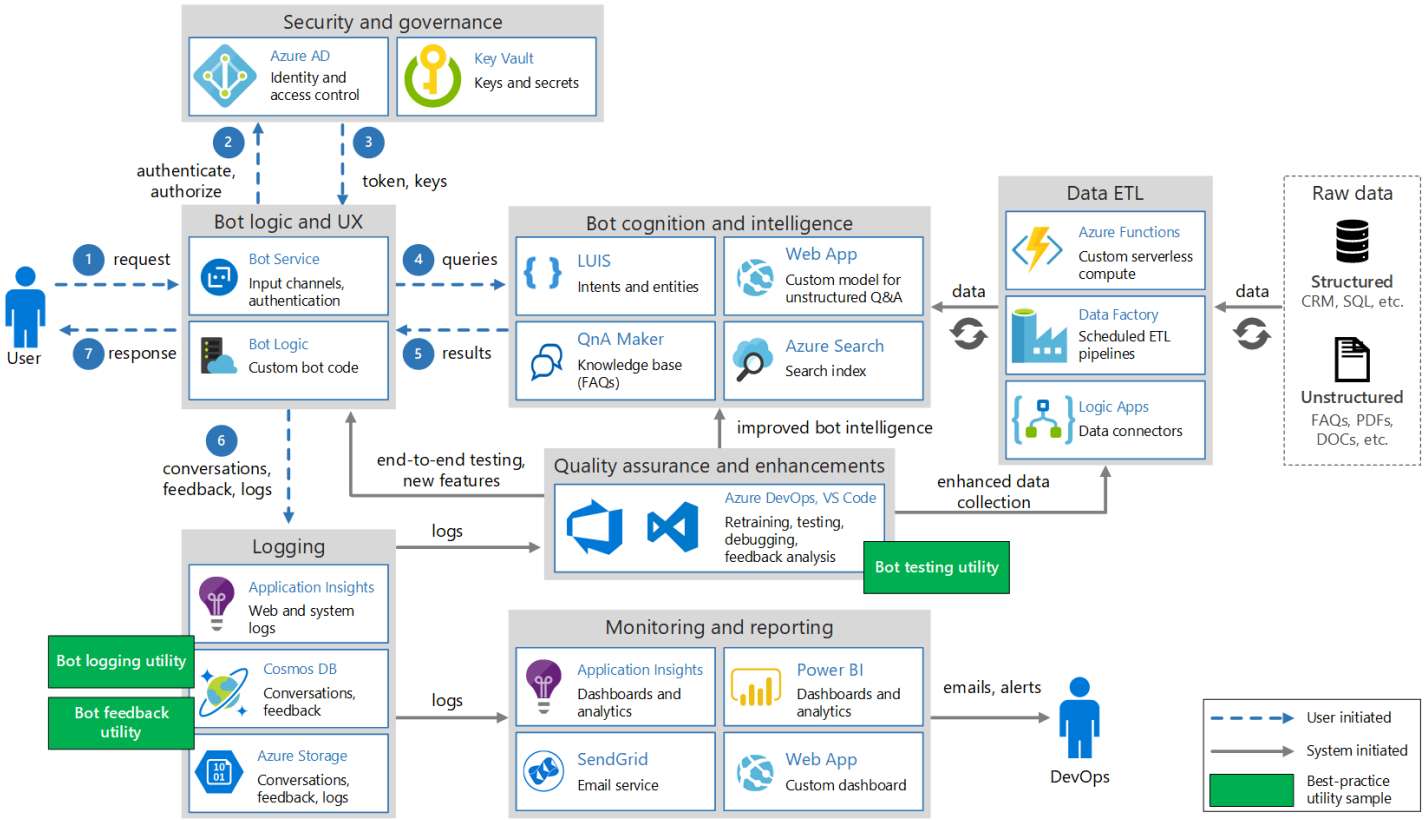
Enterprise-grade conversational bot

01/24/2019 • 13 minutes to read • Contributors

In this article

This reference architecture describes how to build an enterprise-grade conversational bot (chatbot) using the Azure Bot Framework. Each bot is different, but there are some common patterns, workflows, and technologies to be aware of. Especially for a bot to serve enterprise workloads, there are many design considerations beyond just the core functionality. This article covers the most essential design aspects, and introduces the tools needed to build a robust, secure, and actively learning bot.



The best practice utility samples used in this architecture are fully open-sourced and available on GitHub.

Architecture

The architecture shown here uses the following Azure services. Your own bot may not use all of these services, or may incorporate additional services.

Bot logic and user experience

- **Bot Framework Service** (BFS). This service connects your bot to a communication app such as Cortana, Facebook Messenger, or Slack. It facilitates communication between your bot and the user.

- **Azure App Service**. The bot application logic is hosted in Azure App Service.

Bot cognition and intelligence

- **Language Understanding** (LUIS). Part of Azure Cognitive Services, LUIS enables your bot to understand natural language by identifying user intents and entities.
- **Azure Search**. Search is a managed service that provides a quick searchable document index.
- **QnA Maker**. QnA Maker is a cloud-based API service that creates a conversational, question-and-answer layer over your data. Typically, it's loaded with semi-structured content such as FAQs. Use it to create a knowledge base for answering natural-language questions.
- **Web app**. If your bot needs AI solutions not provided by an existing service, you can implement your own custom AI and host it as a web app. This provides a web endpoint for your bot to call.

Data ingestion

The bot will rely on raw data that must be ingested and prepared. Consider any of the following options to orchestrate this process:

- **Azure Data Factory**. Data Factory orchestrates and automates data movement and data transformation.
- **Logic Apps**. Logic Apps is a serverless platform for building workflows that integrate applications, data, and services. Logic Apps provides data connectors for many applications, including Office 365.
- **Azure Functions**. You can use Azure Functions to write custom serverless code that is invoked by a trigger — for example, whenever a document is added to blob storage or Cosmos DB.

Logging and monitoring

- **Application Insights**. Use Application Insights to log the bot's application metrics for monitoring, diagnostic, and analytical purposes.
- **Azure Blob Storage**. Blob storage is optimized for storing massive amounts of unstructured data.
- **Cosmos DB**. Cosmos DB is well-suited for storing semi-structured log data such as conversations.
- **Power BI**. Use Power BI to create monitoring dashboards for your bot.

Security and governance

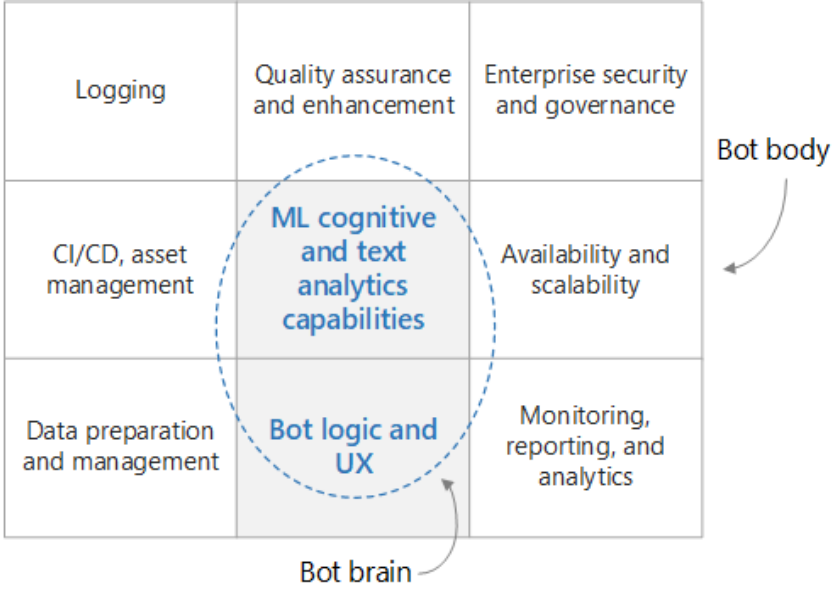
- **Azure Active Directory** (Azure AD). Users will authenticate through an identity provider such as Azure AD. The Bot Service handles the authentication flow and OAuth token management. See Add authentication to your bot via Azure Bot Service.
- **Azure Key Vault**. Store credentials and other secrets using Key Vault.

Quality assurance and enhancements

- **Azure DevOps**. Provides many services for app management, including source control, building, testing, deployment, and project tracking.
- **VS Code** A lightweight code editor for app development. You can use any other IDE with similar features.

Design considerations

At a high level, a conversational bot can be divided into the bot functionality (the "brain") and a set of surrounding requirements (the "body"). The brain includes the domain-aware components, including the bot logic and ML capabilities. Other components are domain agnostic and address non-functional requirements such as CI/CD, quality assurance, and security.



Before getting into the specifics of this architecture, let's start with the data flow through each subcomponent of the design. The data flow includes user-initiated and system-initiated data flows.

User message flow

Authentication. Users start by authenticating themselves using whatever mechanism is provided by their channel of communication with the bot. The bot framework supports many communication channels, including Cortana, Microsoft Teams, Facebook Messenger, Kik, and Slack. For a list of channels, see Connect a bot to channels. When you create a bot with Azure Bot Service, the Web Chat channel is automatically configured. This channel allows users to interact with your bot directly in a web page. You can also connect the bot to a custom app by using the Direct Line channel. The user's identity is used to provide role-based access control, as well as to serve personalized content.

User message. Once authenticated, the user sends a message to the bot. The bot reads the message and routes it to a natural language understanding service such as LUIS. This step gets the **intents** (what the user wants to do) and **entities** (what things the user is interested in). The bot then builds a query that it passes to a service that serves information, such as Azure Search for document retrieval, QnA Maker for FAQs, or a custom knowledge base. The bot uses these results to construct a response. To give the best result for a given query, the bot might make several back-and-forth calls to these remote services.

Response. At this point, the bot has determined the best response and sends it to the user. If the confidence score of the best-matched answer is low, the response might be a disambiguation question or an acknowledgement that the bot could not reply adequately.

Logging. When a user request is received or a response is sent, all conversation actions should be logged to a logging store, along with performance metrics and general errors from external services. These logs will be useful later when diagnosing issues and improving the system.

Feedback. Another good practice is to collect user feedback and satisfaction scores. As a follow up to the bot's final response, the bot should ask the user to rate their satisfaction with the reply. Feedback can help you to solve the cold start problem of natural language understanding, and continually improve the accuracy of responses.

System Data Flow

ETL. The bot relies on information and knowledge extracted from the raw data by an ETL process in the backend. This data might be structured (SQL database), semi-structured (CRM system, FAQs), or unstructured (Word documents, PDFs, web logs). An ETL subsystem extracts the data on a fixed schedule. The content is transformed and enriched, then loaded into an intermediary data store, such as Cosmos DB or Azure Blob Storage.

Data in the intermediary store is then indexed into Azure Search for document retrieval, loaded into QnA Maker to create question and answer pairs, or loaded into a custom web app for unstructured text processing. The data is also used to train a LUIS model for intent and entity extraction.

Quality assurance. The conversation logs are used to diagnose and fix bugs, provide insight into how the bot is being used, and track overall performance. Feedback data is useful for retraining the AI models to improve bot performance.

Building a bot

Before you even write a single line of code, it's important to write a functional specification so the development team has a clear idea of what the bot is expected to do. The specification should include a reasonably comprehensive list of user inputs and expected bot responses in various knowledge domains. This living document will be an invaluable guide for developing and testing your bot.

Ingest data

Next, identify the data sources that will enable the bot to interact intelligently with users. As mentioned earlier, these data sources could contain structured, semi-structured, or unstructured data sets. When you're getting started, a good approach is to make a one-off copy of the data to a central store, such as Cosmos DB or Azure Storage. As you progress, you should create an automated data ingestion pipeline to keep this data current. Options for an automated ingestion pipeline include Data Factory, Functions, and Logic Apps. Depending on the data stores and the schemas, you might use a combination of these approaches.

As you get started, it's reasonable to use the Azure portal to manually create Azure resources. Later on, you should put more thought into automating the deployment of these resources.

Core bot logic and UX

Once you have a specification and some data, it's time to start making your bot into reality. Let's focus on the core bot logic. This is the code that handles the conversation with the user, including the routing logic, disambiguation logic, and logging. Start by familiarizing yourself with the Bot Framework, including:

- Basic concepts and terminology used in the framework, especially conversations, turns, and activities.
- The Bot Connector service, which handles the networking between the bot and your channels.
- How conversation state is maintained, either in memory or better yet in a store such as Azure Blob Storage or Azure Cosmos DB.
- Middleware, and how it can be used to hook up your bot with external services, such as Cognitive Services.

For a rich user experience, there are many options.

- You can use cards to include buttons, images, carousels, and menus.
- A bot can support speech.
- You can even embed your bot in an app or website and use the capabilities of the app hosting it.

To get started, you can build your bot online using the Azure Bot Service, selecting from the available C# and Node.js templates. As your bot gets more sophisticated, however, you will need to create your bot locally then deploy it to the web. Choose an IDE, such as Visual Studio or Visual Studio Code, and a programming language. SDKs are available for the following languages:

- C#
- JavaScript
- Java (preview)
- Python (preview)

As a starting point, you can download the source code for the bot you created using the Azure Bot Service. You can also find sample code, from simple echo bots to more sophisticated bots that integrate with various AI services.

Add smarts to your bot

For a simple bot with a well-defined list of commands, you might be able to use a rules-based approach to parse the user input via regex. This has the advantage of being deterministic and understandable. However, when your bot needs to understand the intents and entities of a more natural-language message, there are AI services that can help.

- LUIS is specifically designed to understand user intents and entities. You train it with a moderately sized collection of relevant user input and desired responses, and it returns the intents and entities for a user's given message.
- Azure Search can work alongside LUIS. Using Search, you create searchable indexes over all relevant data. The bot queries these indexes for the entities extracted by LUIS. Azure Search also supports synonyms, which can widen the net of correct word mappings.
- QnA Maker is another service that is designed to return answers for given questions. It's typically trained over semi-structured data such as FAQs.

Your bot can use other AI services to further enrich the user experience. The Cognitive Services suite of pre-built AI services (which includes LUIS and QnA Maker) has services for vision, speech, language, search, and location. You can quickly add functionality such as language translation, spell checking, sentiment analysis, OCR, location awareness, and content moderation. These services can be wired up as middleware modules in your bot to interact more naturally and intelligently with the user.

Another option is to integrate your own custom AI service. This approach is more complex, but gives you complete flexibility in terms of the machine learning algorithm, training, and model. For example, you could implement your own topic modeling and use algorithm such as LDA to find similar or relevant documents. A good approach is to expose your custom AI solution as a web service endpoint, and call the endpoint from the core bot logic. The web service could be hosted in App Service or in a cluster of VMs. Azure Machine Learning provides a number of services and libraries to assist you in training and deploying your models.

Quality assurance and enhancement

Logging. Log user conversations with the bot, including the underlying performance metrics and any errors. These logs will prove invaluable for debugging issues, understanding user interactions, and improving the system. Different data stores might be appropriate for different types of logs. For example, consider Application Insights for web logs, Cosmos DB for conversations, and Azure Storage for large payloads. See Write directly to Azure Storage.

Feedback. It's also important to understand how satisfied users are with their bot interactions. If you have a record of user feedback, you can use this data to focus your efforts on improving certain interactions and retraining the AI models for improved performance. Use the feedback to retrain the models, such as LUIS, in your system.

Testing. Testing a bot involves unit tests, integration tests, regression tests, and functional tests. For testing, we recommend recording real HTTP responses from external services, such as Azure Search or QnA Maker, so they can be played back during unit testing without needing to make real network calls to external services.

> **Note**
>
> To jump-start your development in these areas, look at the **Botbuilder Utils for JavaScript**. This repo contains sample utility code for bots built with **Microsoft Bot Framework v4** and running Node.js. It includes the following packages:
>
> - **Cosmos DB Logging Store**. Shows how to store and query bot logs in Cosmos DB.
> - **Application Insights Logging Store**. Shows how to store and query bot logs in Application Insights.

- **Feedback Collection Middleware**. Sample middleware that provides a bot user feedback-request mechanism.
- **Http Test Recorder**. Records HTTP traffic from services external to the bot. It comes pre-built with support for LUIS, Azure Search, and QnAMaker, but extensions are available to support any service. This helps you automate bot testing.

These packages are provided as utility sample code, and come with no guarantee of support or updates.

Availability considerations

As you roll out new features or bug fixes to your bot, it's best to use multiple deployment environments, such as staging and production. Using deployment slots from Azure DevOps allows you to do this with zero downtime. You can test your latest upgrades in the staging environment before swapping them to the production environment. In terms of handling load, App Service is designed to scale up or out manually or automatically. Because your bot is hosted in Microsoft's global datacenter infrastructure, the App Service SLA promises high availability.

Security considerations

As with any other application, the bot can be designed to handle sensitive data. Therefore, restrict who can sign in and use the bot. Also limit which data can be accessed, based on the user's identity or role. Use Azure AD for identity and access control and Key Vault to manage keys and secrets.

Manageability considerations

Monitoring and reporting

Once your bot is running in production, you will need a DevOps team to keep it that way. Continually monitor the system to ensure the bot operates at peak performance. Use the logs sent to Application Insights or Cosmos DB to create monitoring dashboards, either using Application Insights itself, Power BI, or a custom web app dashboard. Send alerts to the DevOps team if critical errors occur or performance falls below an acceptable threshold.

Automated resource deployment

The bot itself is only part of a larger system that provides it with the latest data and ensures its proper operation. All of these other Azure resources — data orchestration services such as Data Factory, storage services such as Cosmos DB, and so forth — must be deployed. Azure Resource Manager provides a consistent management layer that you can access through the Azure portal, PowerShell, or the Azure CLI. For speed and consistency, it's best to automate your deployment using one of these approaches.

Continuous bot deployment

You can deploy the bot logic directly from your IDE or from a command line, such as the Azure CLI. As your bot matures, however, it's best to use a continual deployment process using a CI/CD solution such as Azure DevOps, as described in the article Set up continuous deployment. This is a good way to ease the friction in testing new features and fixes in your bot in a near-production environment. It's also a good idea to have multiple deployment environments, typically at least staging and production. Azure DevOps supports this approach.