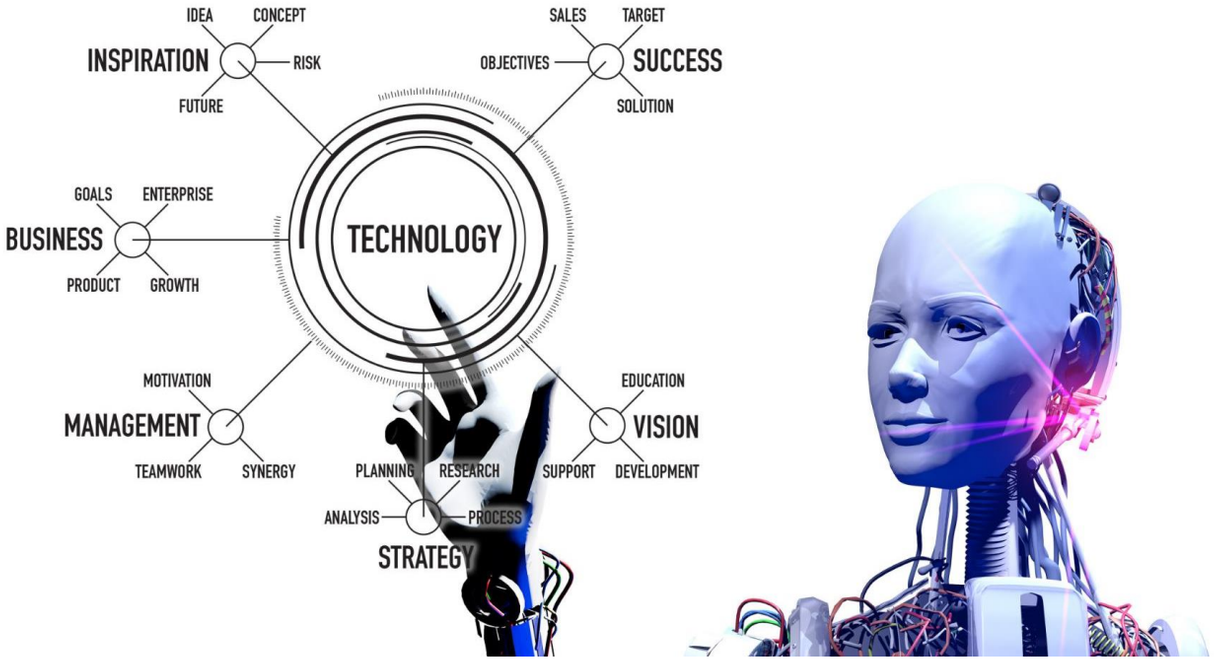
Microsoft Azure

Bot State Management

Radhika Jayaprakash
CSA

February , 2024



Purpose of Bots

Bots are programs designed to mimic human conversations and automate tasks

Bot Service provides a platform for building, deploying, and managing bots

User interacts with bot through a channel or a communicati on channel

Bot understands the user's intents and returns relevant response

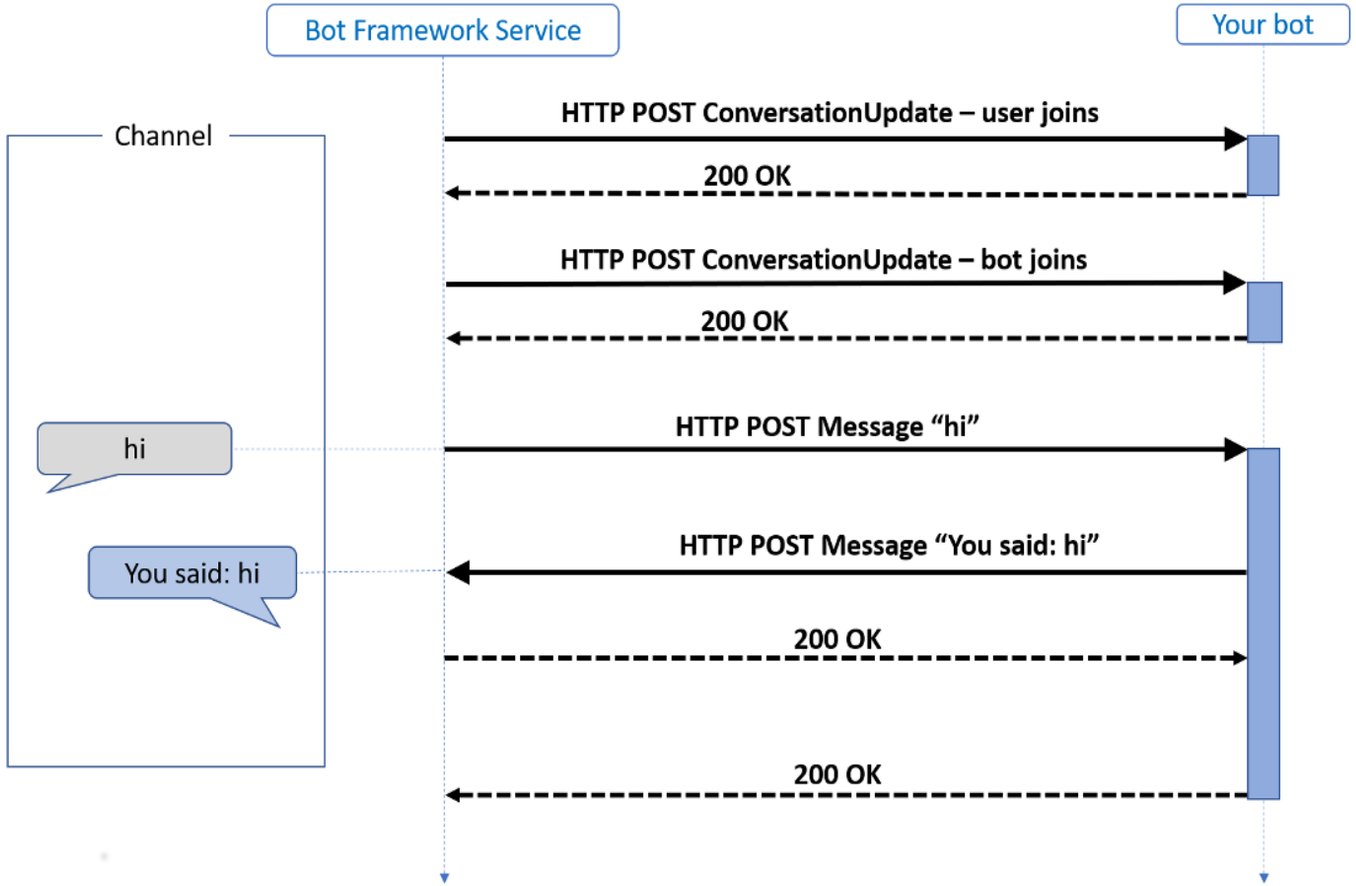
Bot's actions can be customized to integrate with other services

Diagram showing how Bot Service and user interact with each other

©Microsoft Corporation
Azure

Azure AI Bot

- Cloud platform
- Hosts bots and makes them available to channels, such as Microsoft Teams, Facebook, or Slack.
- Bot Framework Service : sends information between the user's bot-connected app and the bot
- Uses Activity object to communicate with its users



The diagram shows using conversation and message activity types

©Microsoft Corporation
Azure

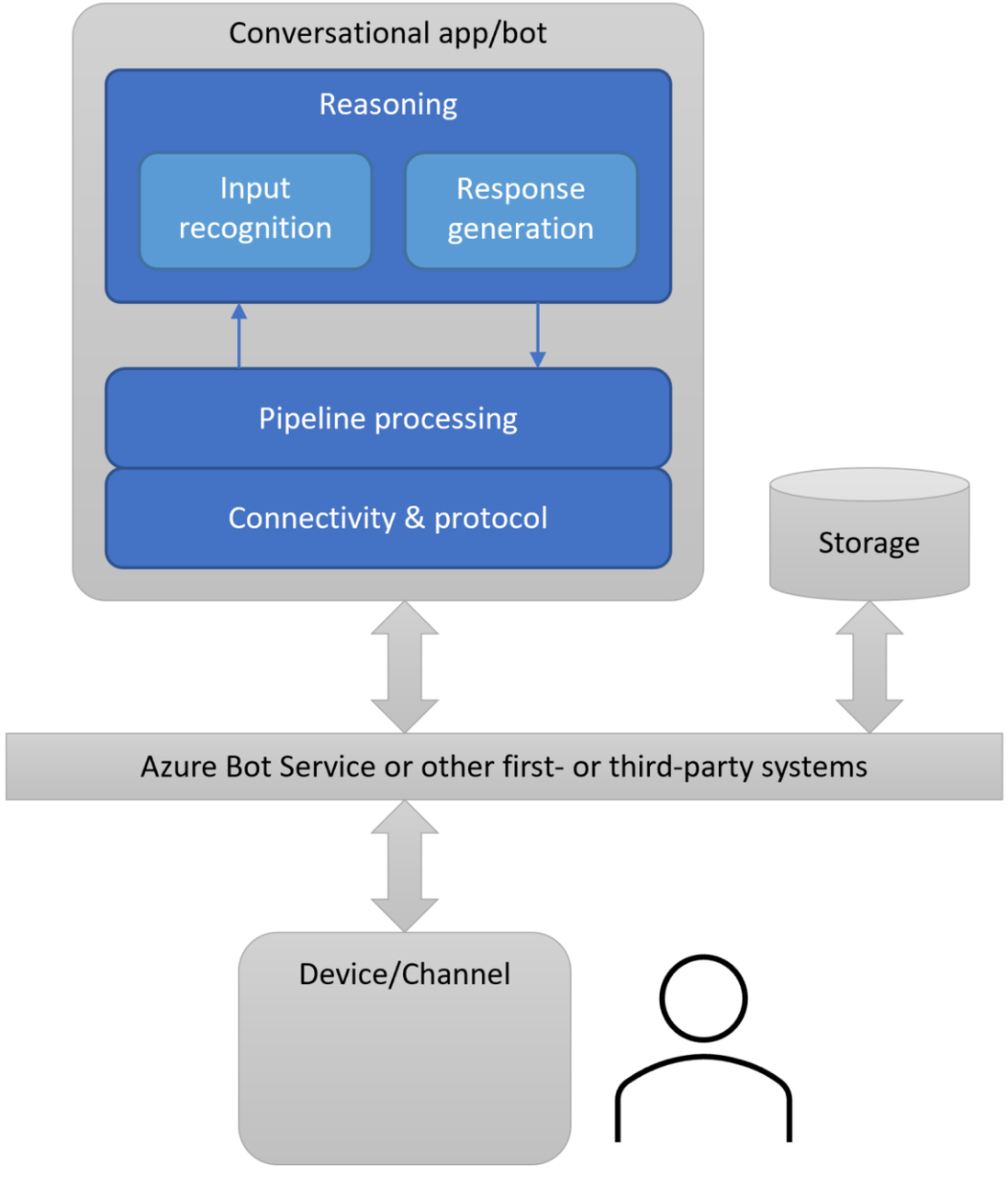
Bot Framework SDK

- Allows you to build bots that can be hosted on the Azure AI Bot Service
- Service defines a REST API and an activity protocol
- The SDK builds upon this REST API and provides an abstraction of the service
- Develop bots in C#, JavaScript, Python, or Java. (The Java SDK is retired with final long-term support ending in November 2023.)
- Additionally, bots may use other Azure services, such as:
  - Azure AI services to build intelligent applications
  - Azure Storage for cloud storage solution
- Bot interactions involve the exchange of activities, which are handled in turns.

©Microsoft Corporation
Azure

Bot Application Structure

- The bot class handles the conversational reasoning for the bot app.
  - Recognizes and interprets the user's input.
  - Reasons about the input and performs relevant tasks.
  - Generates responses about what the bot is doing or has done.
- An adapter class that handles connectivity with the channels.
  - Provides a method for handling requests from and methods for generating requests to the user's channel.
  - Includes a middleware pipeline, which includes turn processing outside of your bot's turn handler.
  - Calls the bot's turn handler and catches errors not otherwise handled in the turn handler.
- Bots often need to retrieve and store state each turn. State is handled through storage, bot state, and property accessor classes.



©Microsoft Corporation
Azure

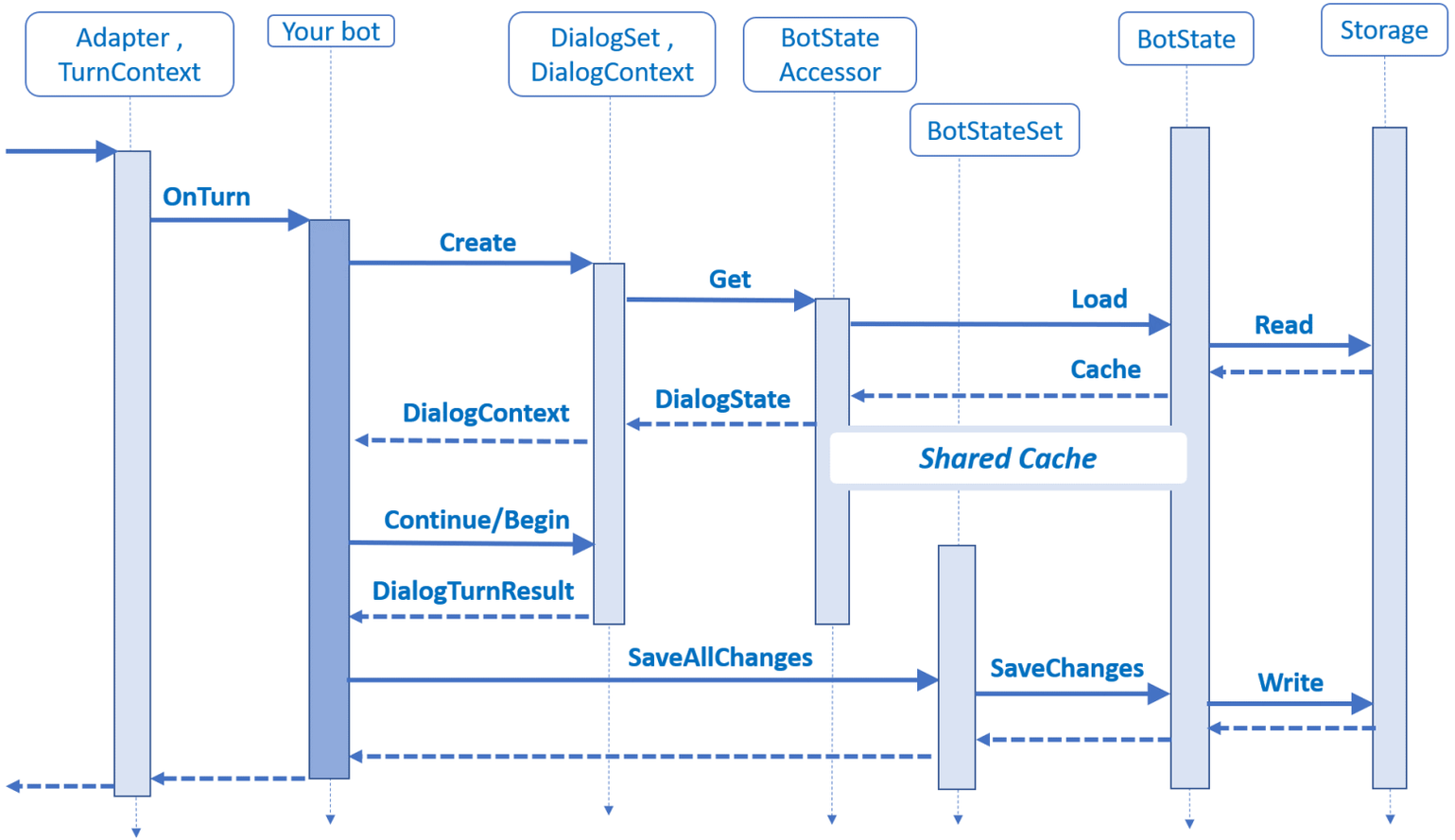
Bot - Recap

- **Bot Object** : *includes logic for a turn and exposes a **turn handler** that accept incoming activities from the bot adapter.*
  - Activity Handler
  - Dialogs Library
- **Bot Adapter:** *The adapter has a process activity method for starting a turn.*
- **Turn-Context:** *Provides info on sender and receiver, the channel, and other data needed to process the activity.*
- **Middleware:** *comprise a linear set of components that are each executed in order, giving each a chance to operate on the activity.*
- **Bot State and Storage:** *allows to have more meaningful conversation and store information for longer duration than the current turn*
- **Messaging Endpoints and provisioning:** *REST endpoint at which to receive messages , provision resource for bot based on platform of choise*

©Microsoft Corporation
Azure

Bot – Managing State

- State is a way to persist data across multiple user interactions with a bot.
- The state data is stored in a storage system using a key-value pair model.
- Bot framework provides built-in storage providers such as Memory Storage, Azure Blob Storage, Cosmos DB Storage, etc.
- Bot State Service is also an option for storing user and conversation state data.
- It is important to use state to maintain user context during a conversation.



©Microsoft Corporation
Azure

Layers to use State

- Storage Layer :
  - Memory
  - Blob
  - CosmosDB Partitioned Storage
- State management
  - State properties scoped to buckets , keys are constructed as
  - User state : *{Activity.ChannelId}/users/{Activity.From.Id}#YourPropertyName*
  - Conversation state : *{Activity.ChannelId}/conversations/{Activity.Conversation.Id}#YourPropertyName*
  - Private Conversation state: *{Activity.ChannelId}/conversations/{Activity.Conversation.Id}/users/{Activity.From.Id}#YourPropertyName*
- State property accessor
  - read or write one of your state properties, and provide get, set, and delete methods
  - The accessors allow the SDK to get state from the underlying storage, and update the bot's state cache

©Microsoft Corporation
Azure

Types of State and When to use

- **Conversation state is good for tracking the context of the conversation, such as:**
  - *Whether the bot asked the user a question, and which question that was*
  - *What the current topic of conversation is, or what the last one was*
- **User state is good for tracking information about the user, such as:**
  - *Non-critical user information, such as name and preferences, an alarm setting, or an alert preference*
  - *Information about the last conversation they had with the bot*
  - *For instance, a product-support bot might track which products the user has asked about.*
- **Private conversation state is good for channels that support group conversations, but where you want to track both user and conversation specific information.**
  - *For example, if you had a classroom clicker bot:*
    - *The bot could aggregate and display student responses for a given question.*
    - *The bot could aggregate each student's performance and privately relay that back to them at the end of the session.*

©Microsoft Corporation
Azure

State Property Accessor

To persist any changes you make to the state property you get from the accessor, the property in the state cache must be updated.

- **The accessor's *get* method:**
  - Accessor requests property from the state cache.
  - If the property is in the cache, return it. Otherwise, get it from the state management object. (If it's not yet in state, use the factory method provided in the accessors get call.)
- **The accessor's *set* method:**
  - Update the state cache with the new property value.
- **The state management object's *save changes* method:**
  - Check the changes to the property in the state cache.
  - Write that property to storage.

©Microsoft Corporation
Azure

Multiple Databases

- If bot needs to connect to multiple databases, create a storage layer for each database.
- Choose to use multiple databases if your bot collects information that has different security, concurrency, or data location needs.
- For each storage layer, create the state management objects you need to support your state properties.

©Microsoft Corporation
Azure

Storage - Examples

- **Save Conversation and User state :** Save user and conversation data - Bot Service | Microsoft Learn
- **Write directly to storage :** Write directly to storage - Bot Service | Microsoft Learn
- **Custom storage implementation :** Implement custom storage for your bot - Bot Service | Microsoft Learn

©Microsoft Corporation
Azure

Microsoft Azure

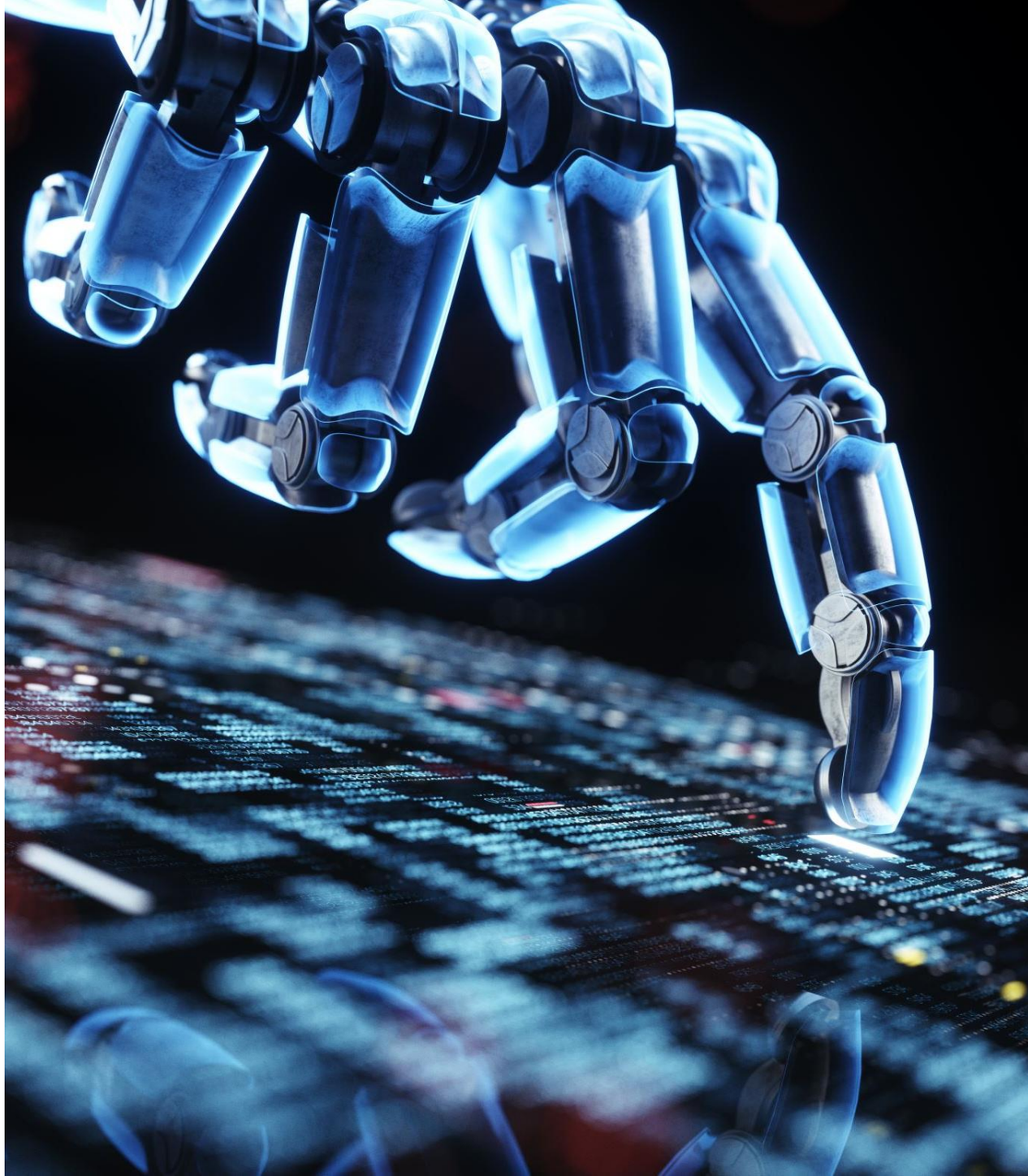
Labs

- **rjayapra/ai-state-management-bot: Manage Bot conversation state in external storage (github.com)**
- **rjayapra/ai-custom-storage-bot (github.com)**

Microsoft Azure

Bot Monitoring

March 2024



©Microsoft Corporation
Azure

Prerequisites

- The CoreBot sample code/CQA sample code
- The Application Insights sample code
- A subscription to Microsoft Azure
- An Application Insights key
- Familiarity with Application Insights
- git

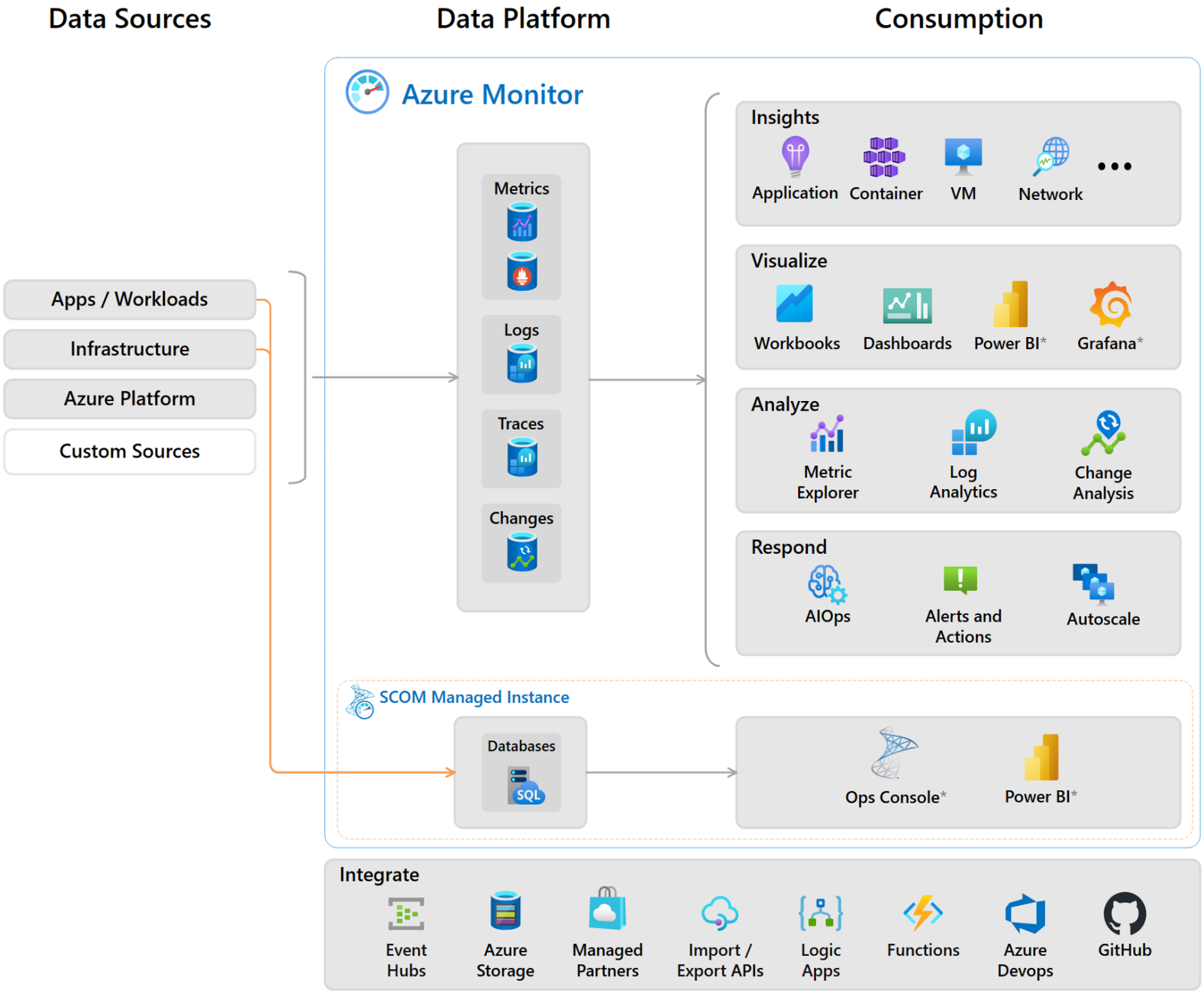
©Microsoft Corporation
Azure

Azure monitor

- Collects and aggregates metrics and logs
- View of availability, performance, and resilience
- Alerting/Notification on issues
- Use Azure portal, Powershell , Azure CLI, REST API or client libraries to set up and view data

©Microsoft Corporation
Azure

Metrics and Logs



Metrics and Logs

- Metrics are numerical values that describe some aspect of a system at a particular point in time.
- They are lightweight and capable of supporting near real-time scenarios.
- Logs contain different kinds of data organized into records with different sets of properties for each type. KQL (kusto query language) can be use to analyze the logs
- Telemetry such as events and traces are stored as logs in addition to performance data so that it can all be combined for analysis.

Logs - Kusto queries

- Logs of Clients to Direct line channel requests

```
ABSBotRequests
| where OperationName contains "ClientToDirectLine"
| sort by TimeGenerated desc
| limit 100
```

- List of logs of unsuccessful requests

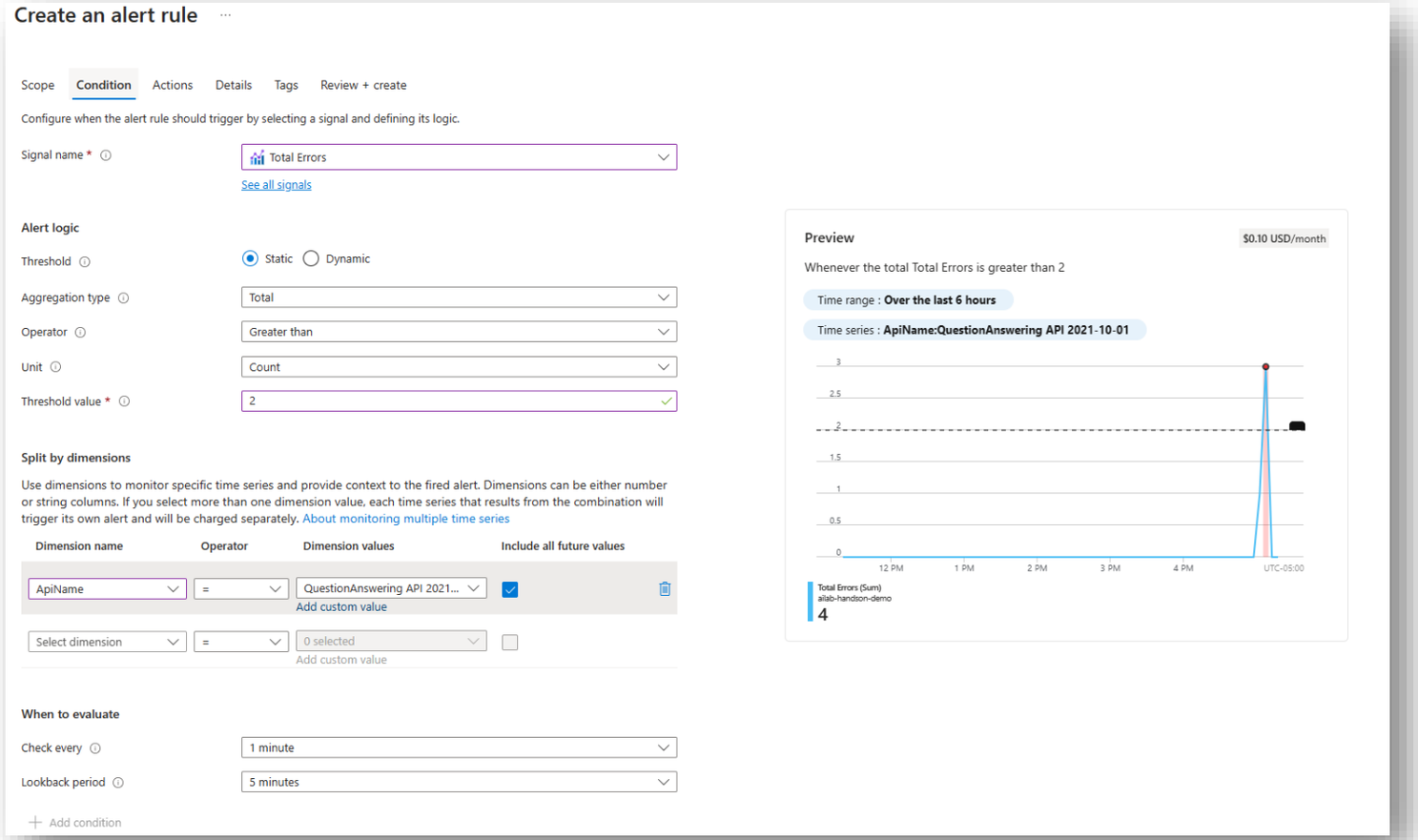
```
ABSBotRequests
| where ResultCode < 200 or ResultCode >= 300
| sort by TimeGenerated desc
```

- Line Chart showing requests response status codes

```
ABSBotRequests
| summarize Number_Of_Requests = count() by tostring(ResultCode), bin(TimeGenerated, 5m)
| render timechart
```

Alerts

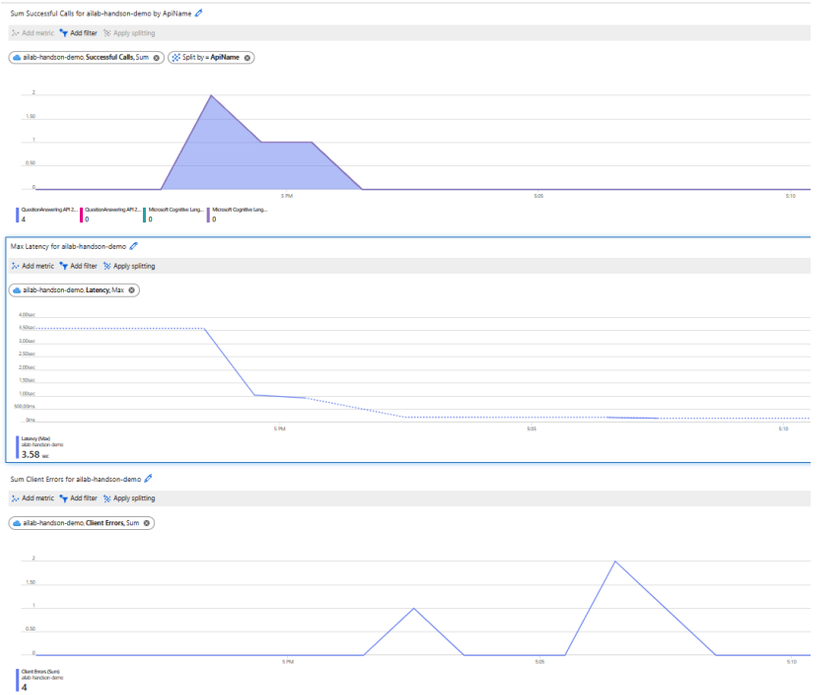
- Metric alerts evaluate resource metrics at regular intervals.
  - platform metrics
  - custom metrics
  - logs from Azure Monitor converted to metrics
  - Application Insights metrics.
  - Metric alerts can also apply multiple conditions and dynamic thresholds.
- Log alerts allow users to use a Log Analytics query to evaluate resource logs at a predefined frequency.
- Activity log alerts trigger when a new activity log event occurs that matches defined conditions.
  - Resource Health alerts
  - Service Health alerts



Alert type	Condition	Description
Metrics	Request Latency milliseconds exceed threshold	Signal source: Platform metrics
Metrics	Requests Traffic count is greater than threshold %	Signal source: Platform metrics
Activity log	A Bot Service is deleted	Signal source: Administrative

Metrics

Azure Monitor Metrics is a feature of Azure Monitor that collects numeric data from monitored resources into a time-series database. Metrics are numerical values that are collected at regular intervals and describe some aspect of a system at a particular time.



Category	Metric	Name in REST API	Unit	Aggregation	Dimensions	Time Grains	DS Export
Latency	**Request Latency** Time taken by the server to process the request	`RequestLatency`	Milliseconds	Total	`Operation`, `Authentication`, `Protocol`, `DataCenter`	PT1M	Yes
Traffic	**Requests Traffic** Number of Requests Made	`RequestsTraffic`	Percent	Count	`Operation`, `Authentication`, `Protocol`, `StatusCode`, `StatusCodeClass`, `DataCenter`	PT1M	Yes

Data Collection from following Tiers

- Application monitoring data: Data about the performance and functionality of the code you have written, regardless of its platform.
- Guest OS monitoring data: Data about the operating system on which your application is running. This could be running in Azure, another cloud, or on-premises.
- Azure resource monitoring data: Data about the operation of an Azure resource.
- Azure subscription monitoring data: Data about the operation and management of an Azure subscription, as well as data about the health and operation of Azure itself.
- Azure tenant monitoring data: Data about the operation of tenant-level Azure services, such as Azure Active Directory.

Application Insights

Add Telemetry to bot

Package : Microsoft.Bot.Builder.Integration.ApplicationInsights.Core

Add to the startup.cs

```
using Microsoft.ApplicationInsights.Extensibility;
using Microsoft.Bot.Builder.ApplicationInsights;
using Microsoft.Bot.Builder.Integration.ApplicationInsights.Core;
using Microsoft.Bot.Builder.Integration.AspNet.Core;
```

Make Telemetry services available to bot through Dependency Injection (DI):

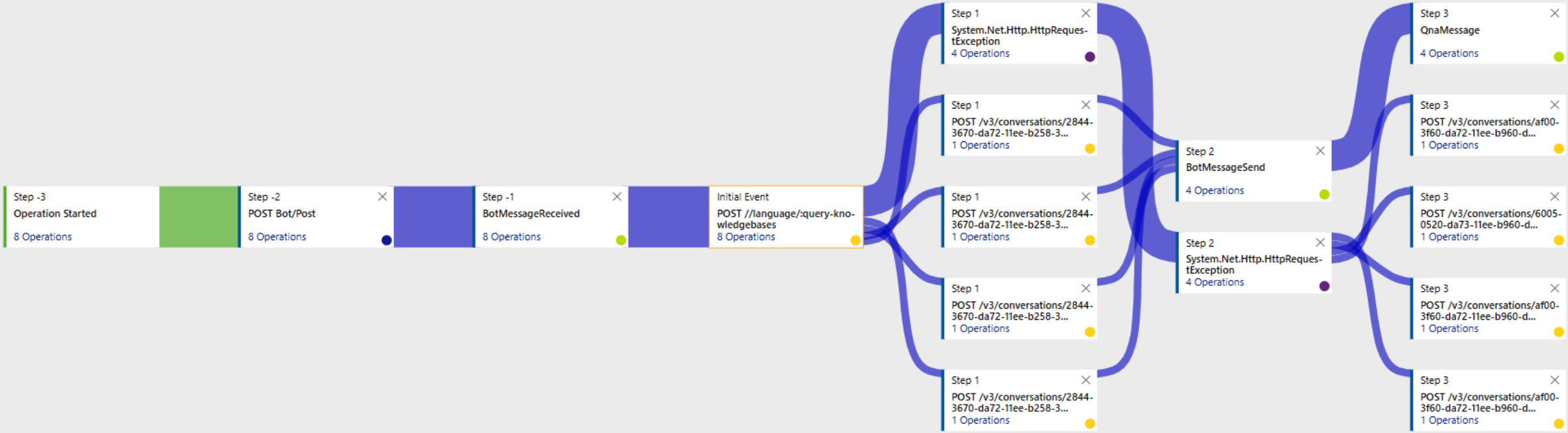
```
// Add telemetry initializer that will set the correlation context for all telemetry items.
services.AddSingleton<ITelemetryInitializer, OperationCorrelationTelemetryInitializer>();
// Add telemetry initializer that sets the user ID and session ID (in addition to other bot-specific properties such as activity ID)
services.AddSingleton<ITelemetryInitializer, TelemetryBotIdInitializer>();
// Create the telemetry middleware to initialize telemetry gathering
services.AddSingleton<TelemetryInitializerMiddleware>();
// Create the telemetry middleware (used by the telemetry initializer) to track conversation events
services.AddSingleton<TelemetryLoggerMiddleware>();
```

Add App Insights instrumentation key to appSettings.json

```
{ "MicrosoftAppId": "“
, "MicrosoftAppPassword": "“,
 "ApplicationInsights":
   { "InstrumentationKey": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx" }
}
```

©Microsoft Corporation
Azure

Application Insights – User flows



©Microsoft Corporation
Azure

Application Insights – End-End transaction

End-to-end transaction details

openai-rg

Search results | Learn more | Copy link | Feedback | Leave preview

Search results

Filtered on

- timestamp > 2024-03-04, 4:...
- timestamp < 2024-03-04, 5:...
- client_Type != ...
- with Dependency response code == 404
- target has == "ailab-handson...
- Success == false
- with Dependency response code 404

Suggested

3/4/2024, 5:05:44 PM
ailab-handson-demo.cognitiveservices.azure.com
Duration: 502.1 ms
Response code: 404
Dependency name:
POST //language/:query-knowledgebases

All — Sort by: Relevance

3/4/2024, 5:05:44 PM
ailab-handson-demo.cognitiveservices.azure.com
Duration: 502.1 ms
Response code: 404
Dependency name:
POST //language/:query-knowledgebases

3/4/2024, 5:01:09 PM
ailab-handson-demo.cognitiveservices.azure.com
Duration: 558.0 ms
Response code: 404
Dependency name:
POST //language/:query-knowledgebases

3/4/2024, 5:06:51 PM
ailab-handson-demo.cognitiveservices.azure.com

End-to-end transaction

Operation ID: b03b23f3505d43b018b54d4a85f61484

= Dependency (outgoing) | = Exception | = Request (incoming) | = Traces & events occurrences | = Traces available

Search

EVENT		RES.	DURATION
localhost:3978	POST Bot/Post	200	1.4 s
ailab-handson-demo.cognitiveservices.azure.com	POST //language/:c	404	502.1 ms
EXCEPTION	System.Net.Http.HttpRequestException		
EXCEPTION	System.Net.Http.HttpRequestException		
localhost:60747	POST /v3/conversations/af003f60-da72-11ee-b960-d15	200	112.6 ms
localhost:60747	POST /v3/conversations/af003f60-da72-11ee-b960-d15	200	54.8 ms
localhost:60747	POST /v3/conversations/af003f60-da72-11ee-b960-d15	200	69.6 ms

Create work item

ailab-handson-demo.cognitiveservices.azure.com
POST //language/:query-knowledgebases

Dependency Properties

Show all

Event time	3/4/2024, 5:05:44.88972 PM (Local time)
Type	HTTP
Result code	404
Call status	false
Duration	502.1 ms
Name	POST //language/:query-knowledgebases

Custom Properties

AspNetCoreEnvironment	Development
DeveloperMode	true
channelId	emulator
activityType	message
activityId	593b8070-da73-11ee-b258-39541147055a
conversationId	af003f60-da72-11ee-b960-d15599ef2838|livechat

Path

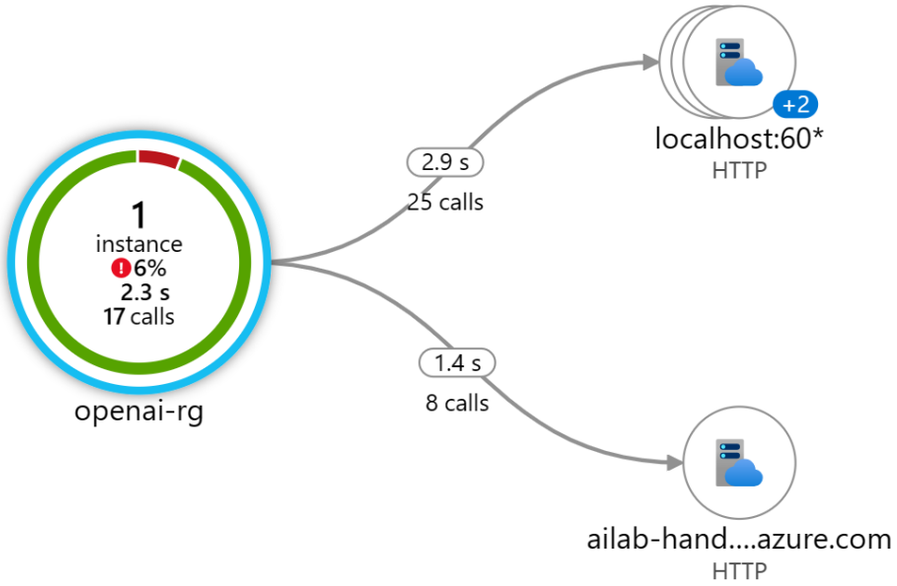
Copy

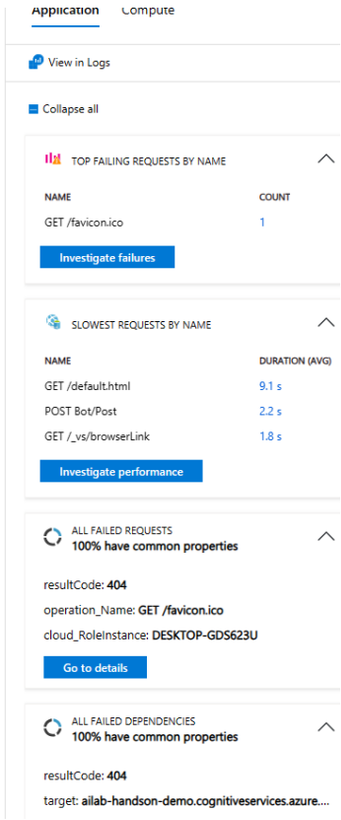
```
https://ailab-handson-demo.cognitiveservices.azure.com//language/:query-knowledgebases?projectName=SurfaceQnA123&deploymentName=production&api-version=2021-10-01
```

Related Items

©Microsoft Corporation
Azure

Application Insights – Application Map

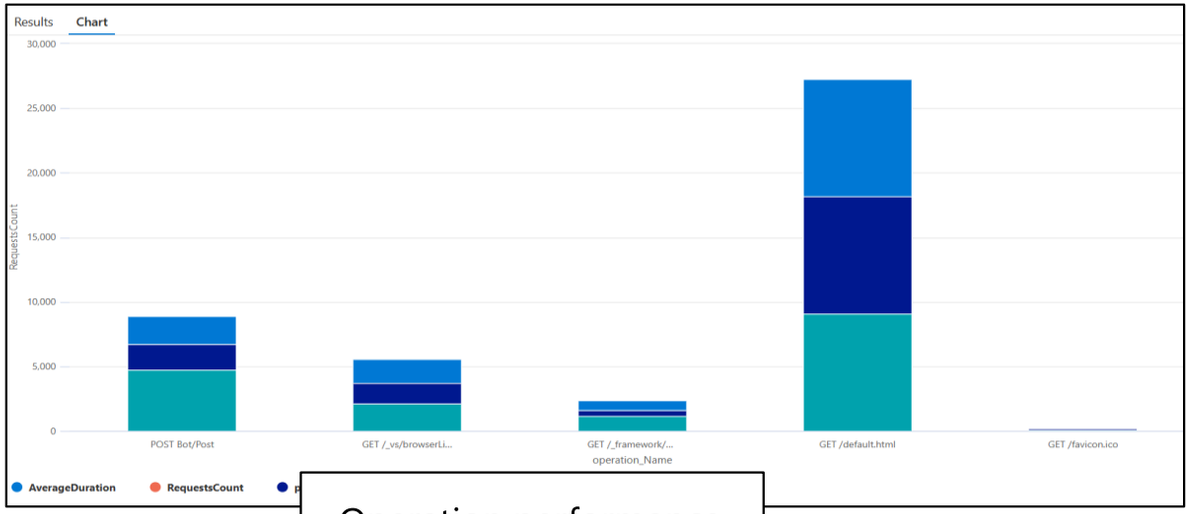




©Microsoft Corporation
Azure

Analyze telemetry data

- [Analyze the telemetry data from your bot - Bot Service | Microsoft Learn]



Operation performance

```
// Failing dependencies
// Which 5 dependencies failed the most today?
dependencies
| where success == false
| summarize totalCount=sum(itemCount) by type
| top 5 by totalCount desc
```

Results | Chart

type	totalCount
HTTP	4
type	HTTP
totalCount	4

Dependency failures

Results | Chart

operation_Name	RequestsCount	AverageDuration	percentile_duration_50	percentile_duration_95	percentile_duration_99
POST Bot/Post	11	2168.229990909091	2023.0982000000001	4709.1846000000005	4709.1846000000005
GET /_vs/browserLink	2	1845.58165	1579.2435	2111.9198	2111.9198
GET /_framework/aspnetcore-browser-refresh.js	2	783.7981	391.6227	1175.9735	1175.9735
GET /default.html	1	9085.3091	9085.3091	9085.3091	9085.3091
GET /favicon.ico	1	89.355	89.355	89.355	89.355

Bot performance

©Microsoft Corporation
Azure

Microsoft Azure

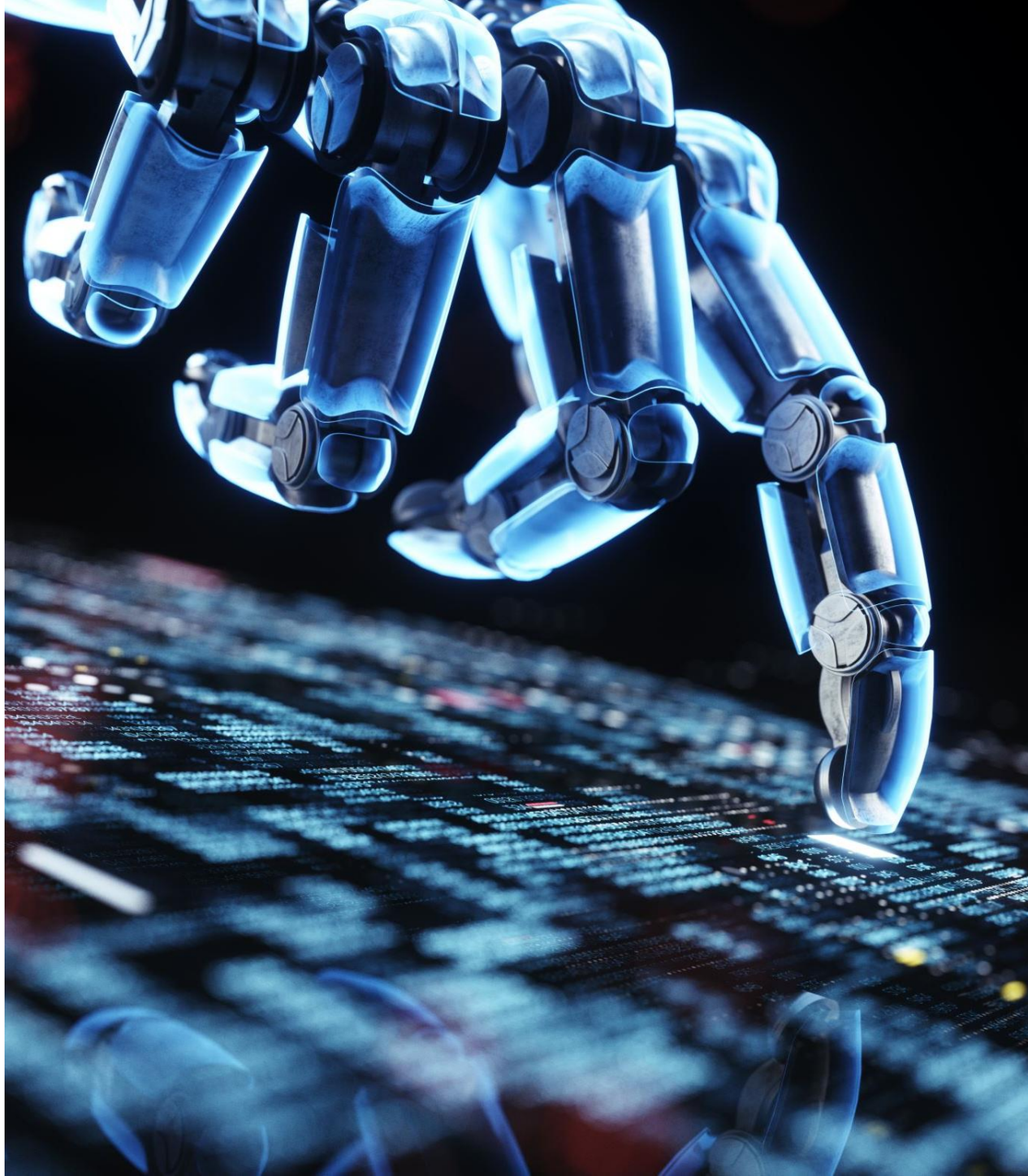
Labs

[ai-services-hackathon/BotMonitoring at master · rjayapra/ai-services-hackathon (github.com)]

Microsoft Azure

Bot Debugging and Testing

March 2024



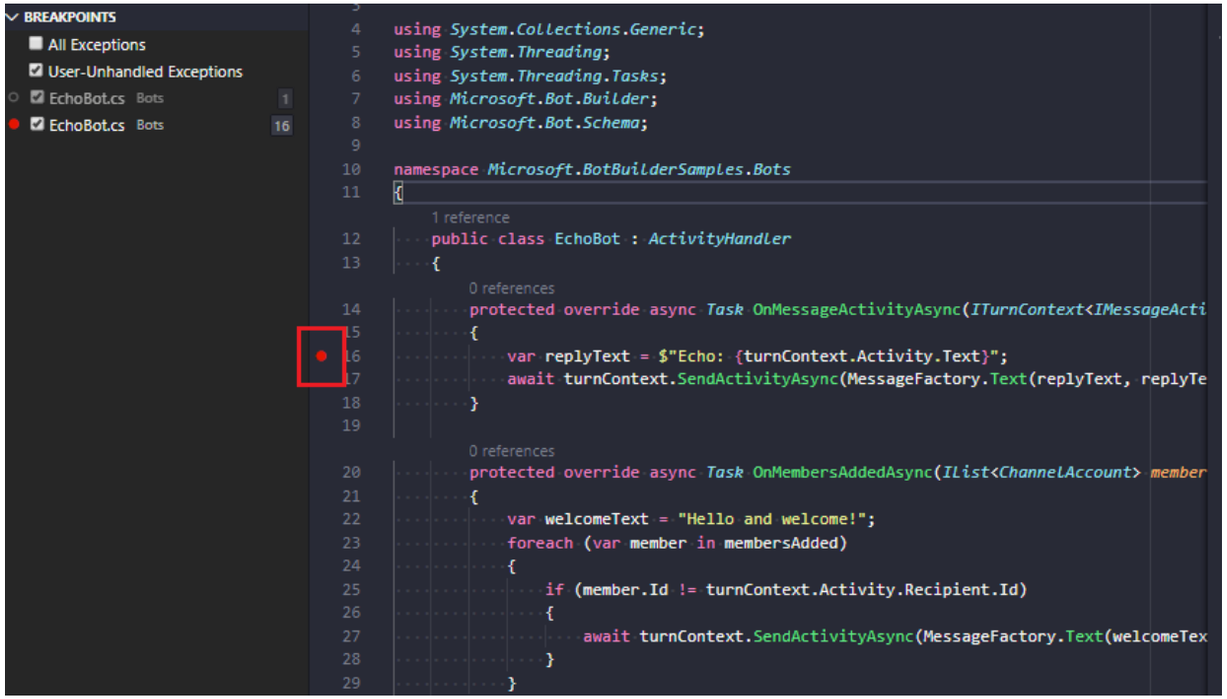
©Microsoft Corporation
Azure

Debug with IDE

- Use breakpoints in VS Code or VS

Prerequisites:

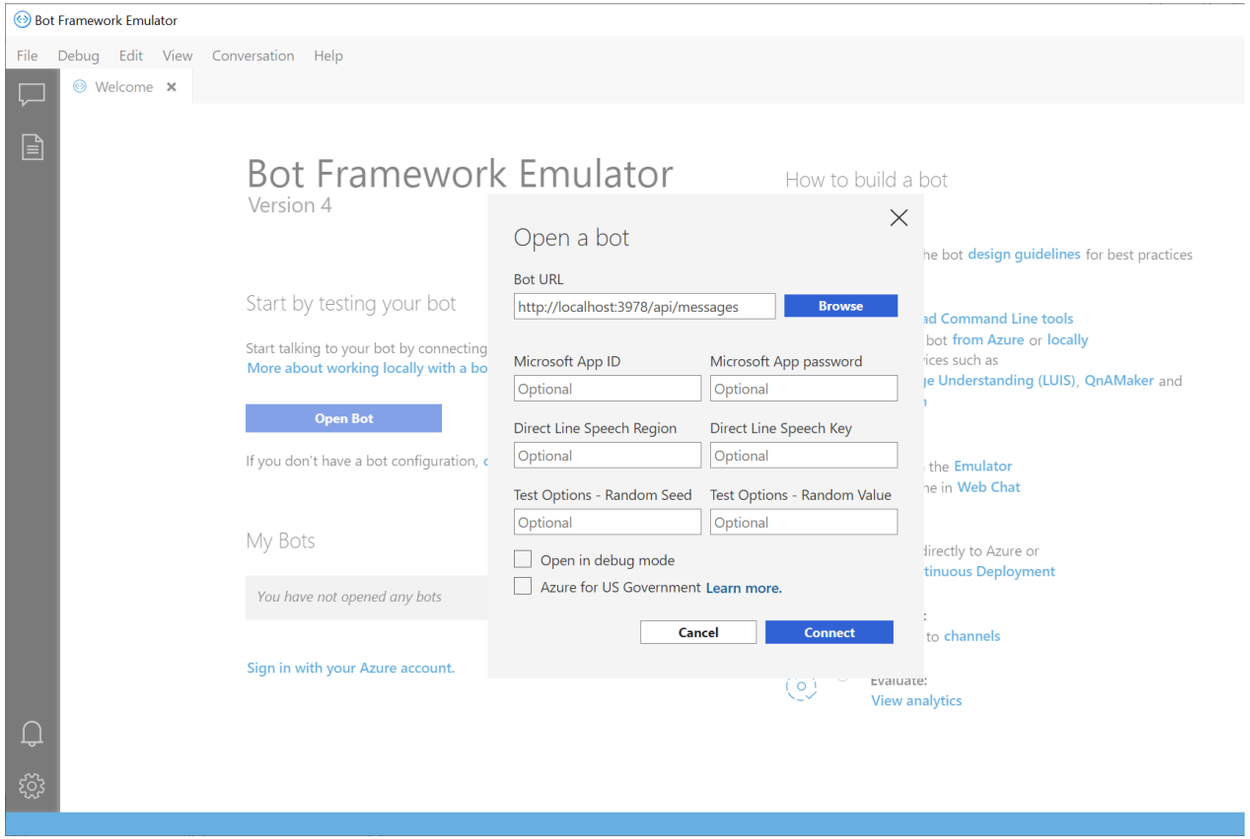
- Download and install the Bot Framework Emulator.
- Download and install Visual Studio Code or Visual Studio.



©Microsoft Corporation
Azure

Debug with Emulator

- Run a bot locally
- Connect to bot running locally
- Configure proxy settings
- Configure for authentication
- Connect from Bot framework emulator
- **http://localhost:<port number>/api/messages**



©Microsoft Corporation
Azure

Debug with Inspection Middleware

- Use Inspection middleware
- First Emulator acts as user interaction channel
- Second emulator will work as debugger
- When opened in debug mode a unique identifier (/INSPECT attach <identifier>) command is displayed
- Copy to second emulator
- Now send message from first emulator and debug in second

Prerequisites

- Knowledge of bot Middleware and Managing state
- Knowledge of how to Debug an SDK-first bot and Test and debug with the Emulator
- An install of the Bot Framework Emulator
- An install ngrok (if you want to debug a bot configured in Azure to use other channels)
- A copy of the inspection bot sample for C#, JavaScript, Java, or Python

©Microsoft Corporation
Azure

Debug with Transcript files

- It is a specialized JSON file that preserves the interactions between a user and your bot.
- It holds contents of a message, interaction details such as the user ID, channel ID, channel type, channel capabilities, time of the interaction, etc.,
- This can be used to help find and resolve issues when testing or debugging bot
- Create transcript with bot framework emulator

©Microsoft Corporation
Azure

Testing

- Create unit tests for bots
- Use assert to check for activities returned by a dialog turn against expected values.
- Use assert to check the results returned by a dialog.
- Create different types of data driven tests.
- Create mock objects for the different dependencies of a dialog, such as language recognizers, and so on.

Reference the Microsoft.Bot.Builder.Testing package, XUnit, and Moq to create unit tests.

©Microsoft Corporation
Azure

Testing Dialogs

```
var sut = new BookingDialog();
var testClient = new DialogTestClient(Channels.Msteams, sut);

var reply = await testClient.SendActivityAsync<IMessageActivity>("hi");
Assert.Equal("Where would you like to travel to?", reply.Text);
reply = await testClient.SendActivityAsync<IMessageActivity>("Seattle");
Assert.Equal("Where are you traveling from?", reply.Text);
reply = await testClient.SendActivityAsync<IMessageActivity>("New York");
Assert.Equal("When would you like to travel?", reply.Text);
reply = await testClient.SendActivityAsync<IMessageActivity>("tomorrow");
Assert.Equal("OK, I will book a flight from Seattle to New York for tomorrow, Is this Correct?", reply.Text);
reply = await testClient.SendActivityAsync<IMessageActivity>("yes");
Assert.Equal("Sure thing, wait while I finalize your reservation...", reply.Text);
reply = testClient.GetNextReply<IMessageActivity>();
Assert.Equal("All set, I have booked your flight to Seattle for tomorrow", reply.Text);
```

©Microsoft Corporation
Azure

Analyze test results

```
public class BookingDialogTests
{
    private readonly IMiddleware[] _middlewares;

    public BookingDialogTests(ITestOutputHelper output)
        : base(output)
    {
        _middlewares = new[] { new XUnitDialogTestLogger(output) };
    }

    [Fact]
    public async Task SomeBookingDialogTest()
    {
        // Arrange
        var sut = new BookingDialog();
        var testClient = new DialogTestClient(Channels.Msteams, sut,
middlewares: _middlewares);

        ...
    }
}
```

Test Name: CoreBot.Tests.Dialogs.BookingDialogTests.DialogFlowUseCases(testData: TestDataObject { TestObject = "{\"Name\":\"Full flow\

Test Outcome: Passed

Standard Output

```
Test Case: Full flow

User:  hi
    -> ts: 01:34:10

Bot:   Text=Where would you like to travel to?
    Speak=Where would you like to travel to?
    InputHint=expectingInput
    -> ts: 01:34:10 elapsed: 0 ms

User:  Seattle
    -> ts: 01:34:10

Bot:   Text=Where are you traveling from?
    Speak=Where are you traveling from?
    InputHint=expectingInput
    -> ts: 01:34:10 elapsed: 2 ms

User:  New York
    -> ts: 01:34:10

Bot:   Text=When would you like to travel?
    Speak=When would you like to travel?
    InputHint=expectingInput
    -> ts: 01:34:10 elapsed: 0 ms

User:  tomorrow
    -> ts: 01:34:10

Bot:   Text=Please confirm, I have you traveling to: Seattle from: New York on: 2019-06-22. Is this correct? (1) Yes or (2) No
    Speak=Please confirm, I have you traveling to: Seattle from: New York on: 2019-06-22. Is this correct?
    InputHint=expectingInput
    -> ts: 01:34:10 elapsed: 3 ms

User:  yes
    -> ts: 01:34:10
```

©Microsoft Corporation
Azure

Microsoft Azure

Labs

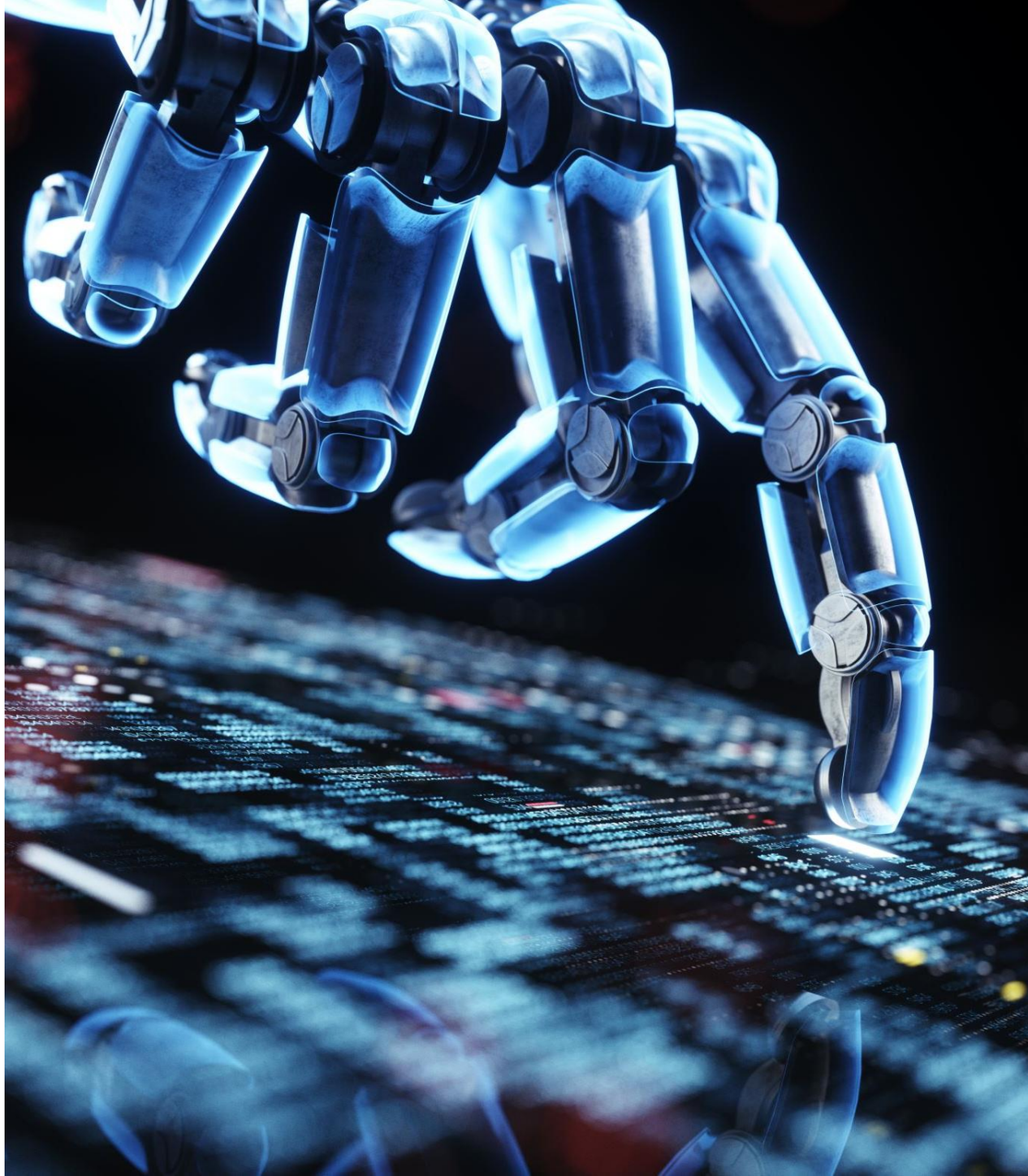
[ai-services-hackathon/inspection-bot at master · rjayapra/ai-services-hackathon (github.com)]

[ai-services-hackathon/BotTests at master · rjayapra/ai-services-hackathon (github.com)]

Microsoft Azure

Middleware

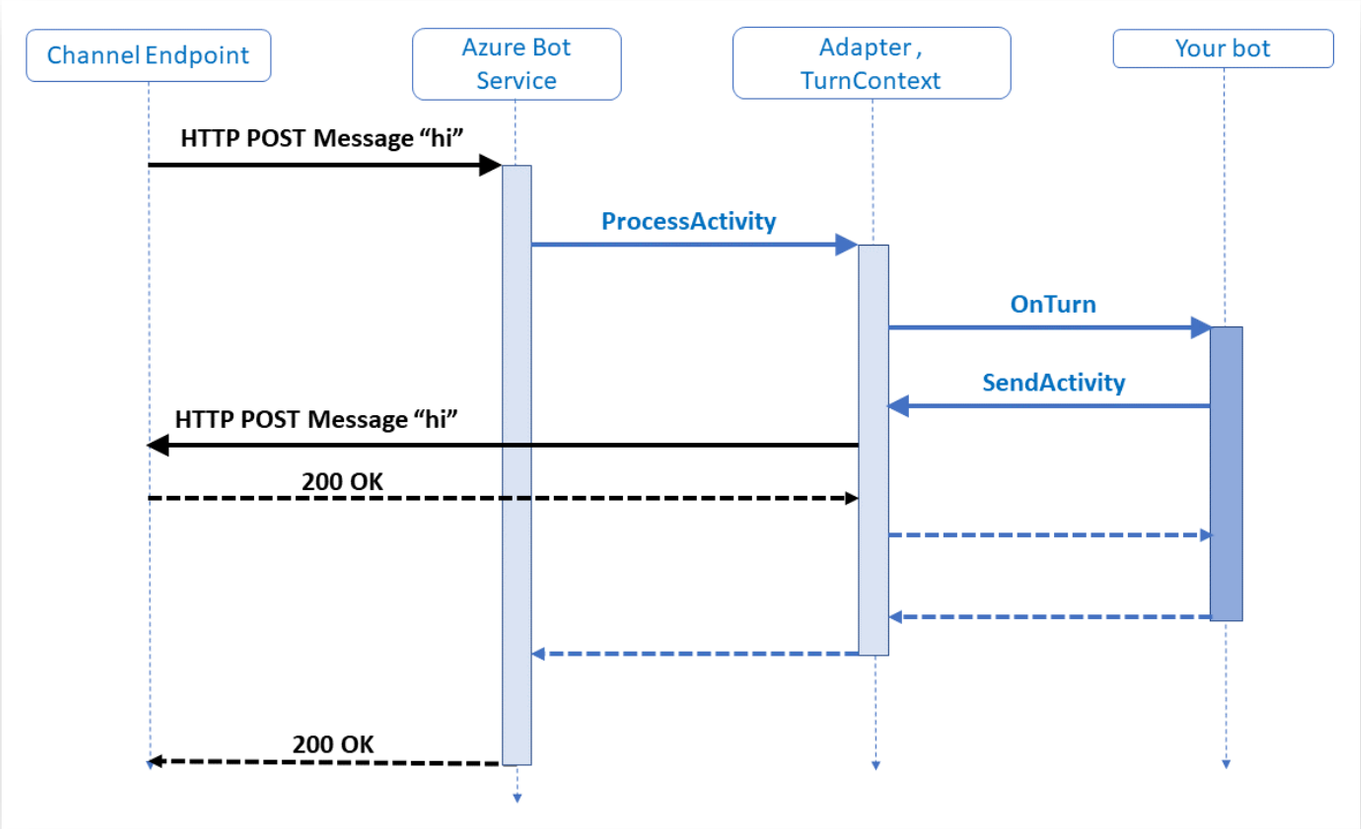
March 2024



©Microsoft Corporation
Azure

Middleware

- A class that sits between adapter and the bot logic
- Gets added to adapter's middleware collection during initialization
- SDK helps to write your own middleware or add one created by others
- Every activity in the bot flows through middleware
- When the activity flows in and out of bot, each piece of middleware can inspect or act upon the activity, either before or/and after bot logic



©Microsoft Corporation
Azure

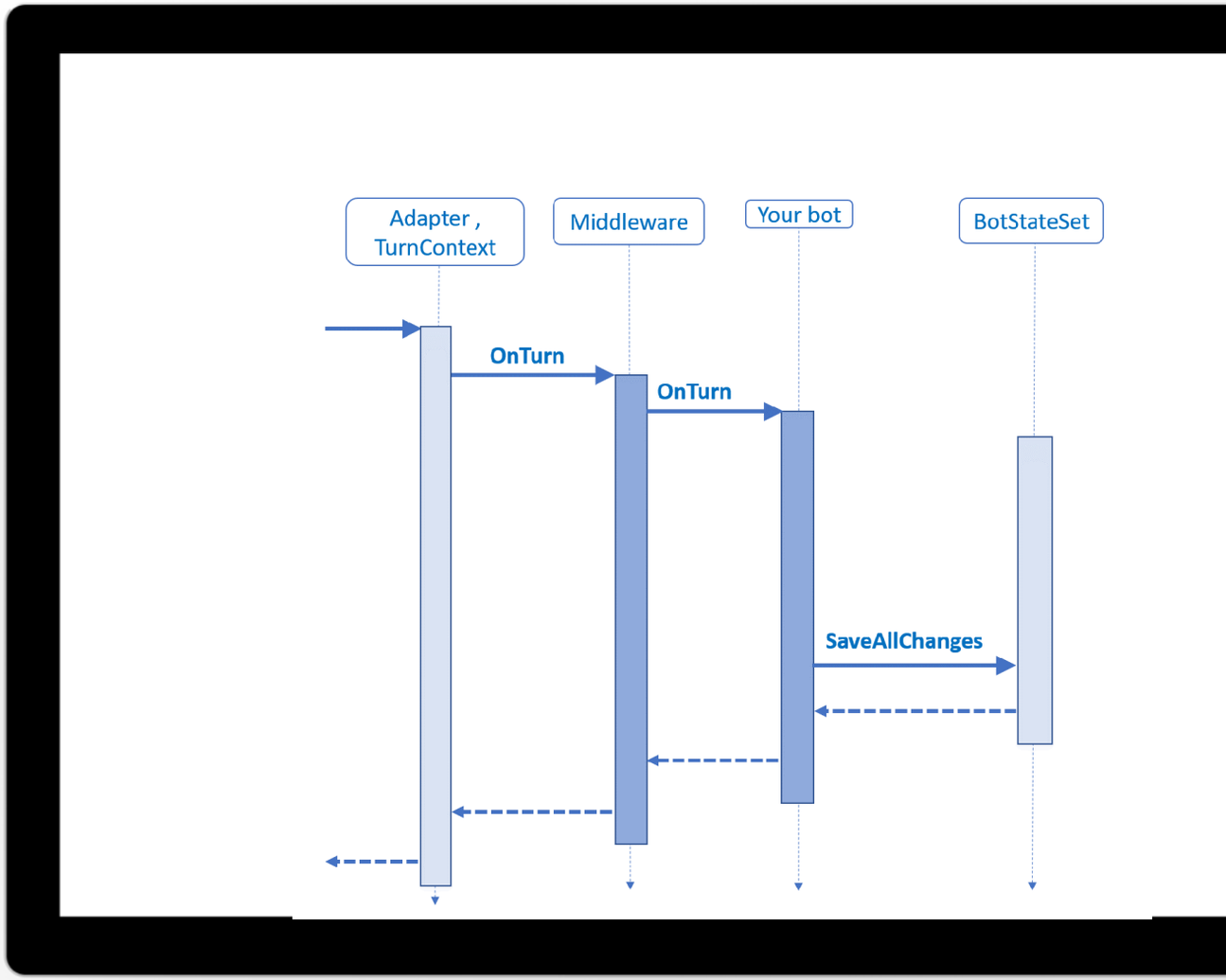
Middleware - When to implement ?

- Middleware provides extra opportunities to interact with user conversation flow both before and after each turn of conversation , also allows to store and retrieve information concerning the conversation and call additional logic as needed
- Looking at or acting on every activity
- Modifying or enhancing the turn context

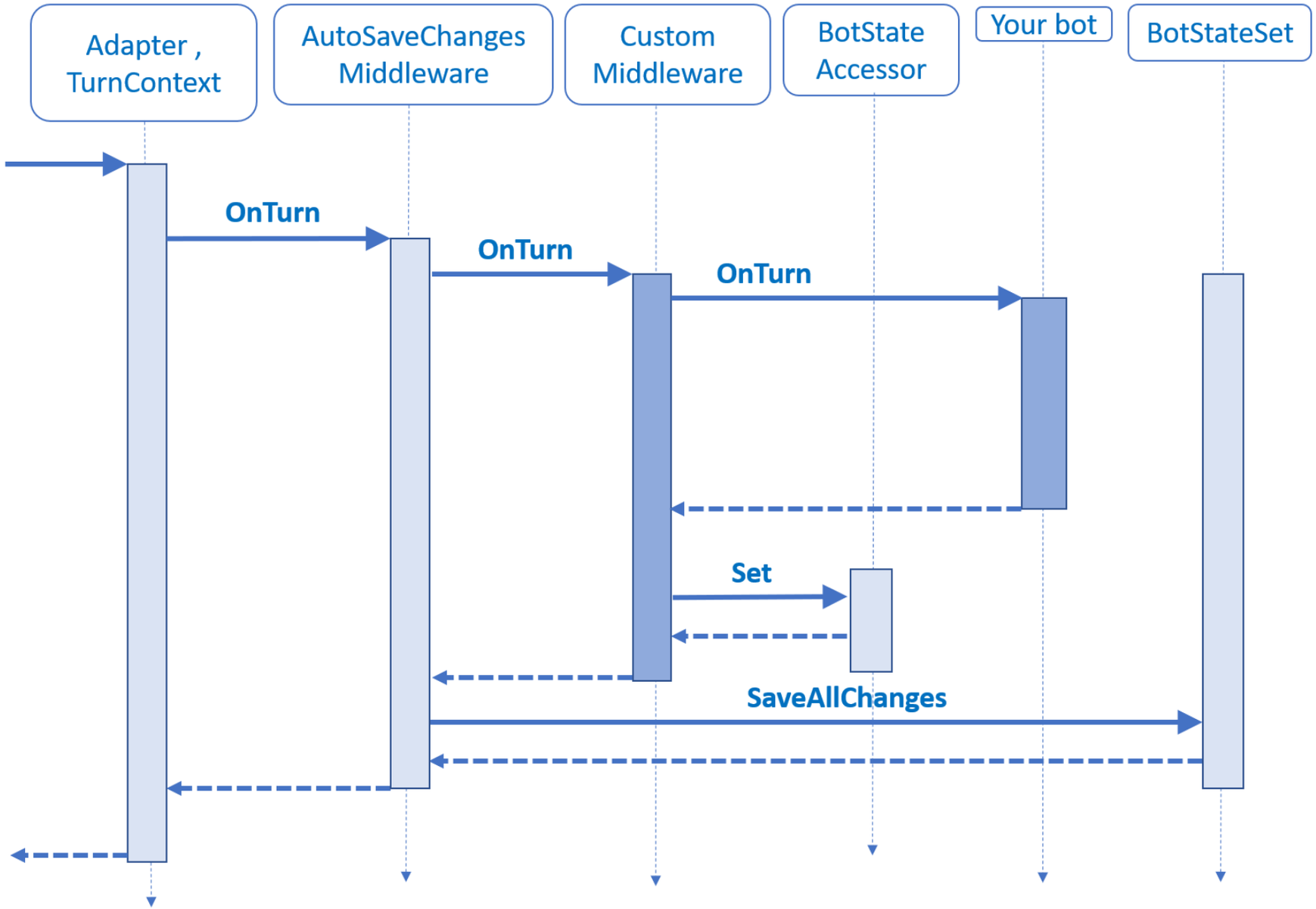
©Microsoft Corporation
Azure

Middleware Pipeline

- Adapter calls middleware in the order it gets added
- Adapter passes context object for turn and a next delegate
- Middleware calls delegate to pass control to next middleware
- If middleware doesn't call the next delegate , the adapter doesn't call any of the subsequent middleware/bot turn handler and it short circuits.
- Middleware takes care of low level *tasks* *Logging, Exception Handling, Transactions*
- Short Circuiting: When the delegate isn't called within the middleware the pipeline should short circuit and further layers are not executed



©Microsoft Corporation
Azure



©Microsoft Corporation
Azure

Microsoft Azure

Labs

[ai-services-hackathon/middleware-multilingual-bot at master · rjayapra/ai-services-hackathon (github.com)]

Microsoft Azure

Thank you.

© Copyright Microsoft Corporation. All rights reserved.

Microsoft Azure

Invent with purpose.

© Copyright Microsoft Corporation. All rights reserved.

Always check accessibility

Run Accessibility Checker before sharing this presentation

Just as you check spelling and grammar, it's also important to ensure accessibility is addressed so that all recipients can understand your presentation.

To improve accessibility, use the PowerPoint Accessibility Checker tool under the 'Review' tab.

For additional accessibility resources or to ask a question, visit https://aka.ms/eDADAnswerDesk (internal only).