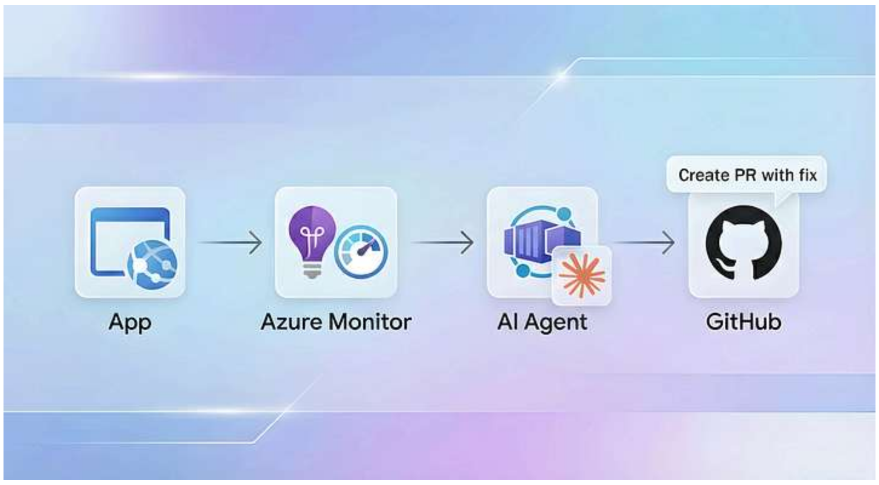
From Exceptions to Pull Requests: Building a Self Fixing .NET 10 App with AI Agents

****



What if your application could fix itself?

Not science fiction. Not a distant future. Today, with AI agents, we can build systems that:

- Monitor Application Insights 24/7
- Analyze exceptions and read your actual source code
- Diagnose root causes with LLM reasoning
- Propose fixes with clear explanations
- Create Pull Requests automatically

I built one. Here I used Azure services but you can do the same by using others.

In this article, we'll cover :

- The Infrastructure to manage a self-fixing production application
- How it works
- The cost

Try it yourself here (includes Bicep templates files to create new Azure Resources):

https://github.com/jBeyondstars/ai-diagnostics-agent

Every time an exceptions appears in my application's App Insights (Azure's service for logs **treatment, management, ...** my Agent API is called to analyze and process the issue.

What Makes an AI Agent Different?

Before we dive into code, let's clarify an important distinction: an AI agent is not a chatbot.

A chatbot answers questions. An agent takes actions.

Think of it this way:

Chatbot: "Here's how you might fix that NullReferenceException..."

Agent: reads your code, identifies the bug, creates a branch, commits a fix, opens a PR

The agent follows a loop: Perceive → Reason → Act → Observe.

It perceives the world through its tools. (**example of tools**)

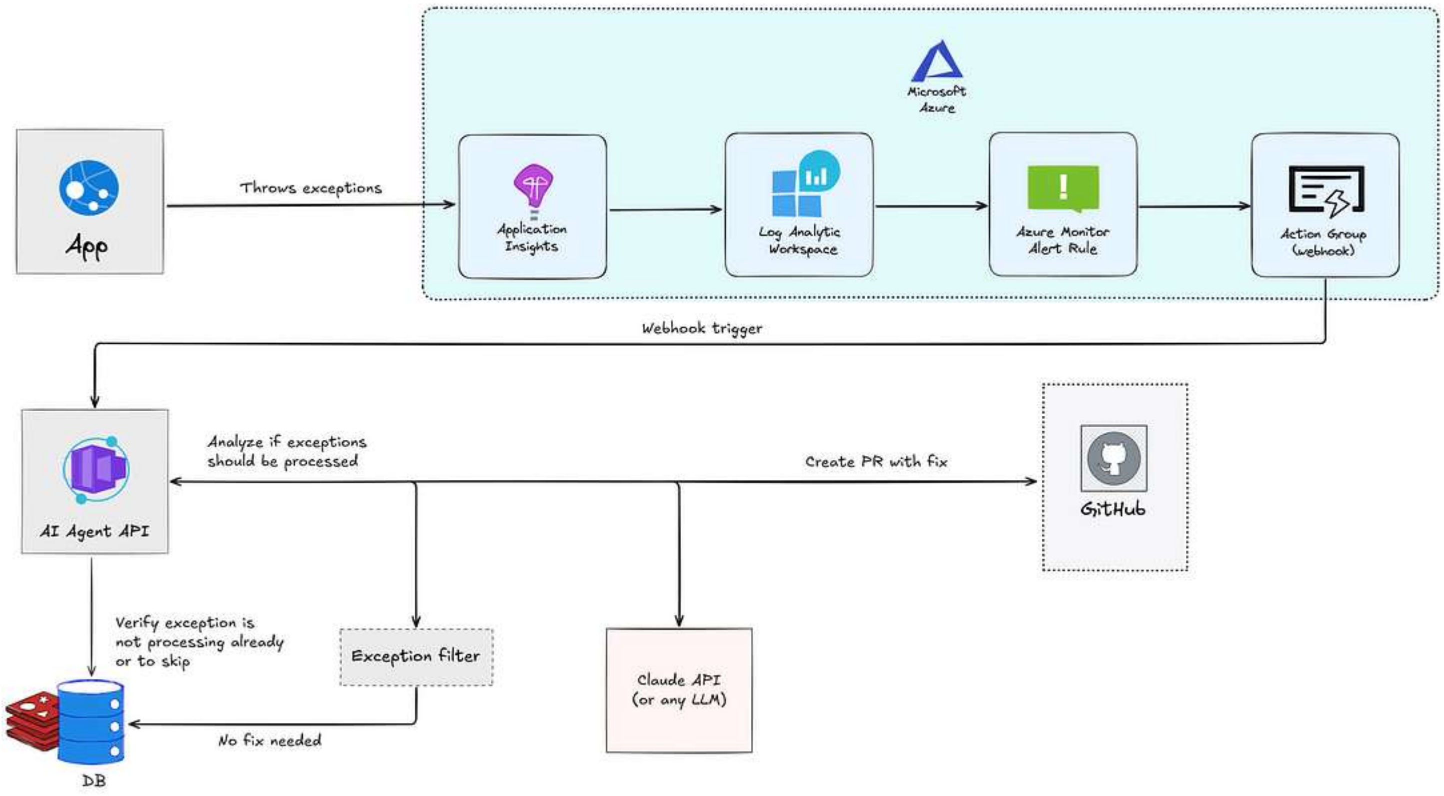
We will use Semantic Kernel (and explain what it is) to define "plugins" for the collections of functions the agent can call. The LLM decides which functions to call and in what order.

We don't write IF/ELSE logic; the agent will reason.

Our agent has two "senses":

- AppInsights Plugin: Sees exceptions, stack traces, -performance
- Metrics GitHub Plugin: Reads code, creates branches, opens PRs

The Architecture

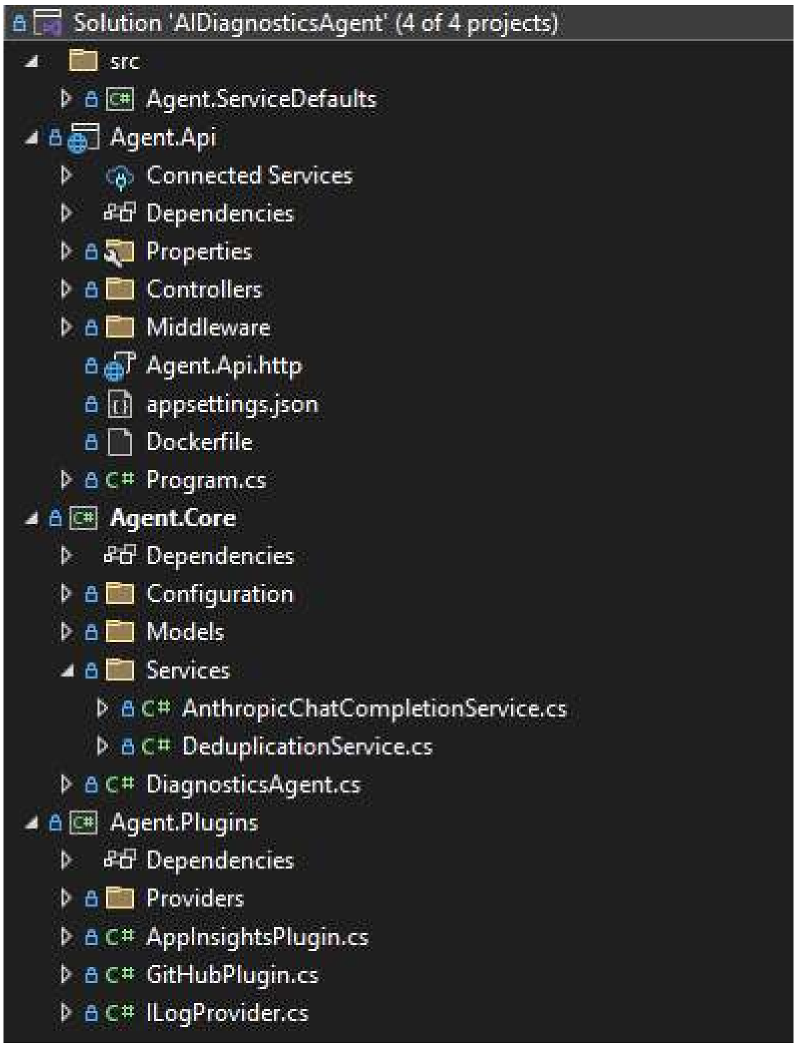


Simplified Architecture Diagram

As I said, for this project I used an Azure environment but you could apply the same process in your own setup.
Let's deep dive into the infrastructure :

- A .NET 10 App that is being monitored
- Azure Monitor / AppInsights / Log Analytics Workspace / Alert Rule & Action Group
- A .NET 10 Agent App
- A Redis In memory cache but I highly recommend the use of an external DB
- Claude API (we could go with Azure OpenAI SDK for fully Azure integrated or any modern LLM API such as Gemini or ChatGPT). I personnaly prefer Claude Opus 4.5 model for coding task.
- A GitHub Repo and Personal Access Token with permissions to allow our agent to create the PRs.

For the demo, I used a single solution for both the App and the agent. It is structured as below:



```
Solution 'AIDiagnosticsAgent' (4 of 4 projects)
  src
    Agent.ServiceDefaults
  Agent.Api
    Connected Services
    Dependencies
    Properties
    Controllers
    Middleware
    Agent.Api.http
    appsettings.json
    Dockerfile
    Program.cs
  Agent.Core
    Dependencies
    Configuration
    Models
    Services
      AnthropicChatCompletionService.cs
      DeduplicationService.cs
    DiagnosticsAgent.cs
  Agent.Plugins
    Dependencies
    Providers
    AppInsightsPlugin.cs
    GitHubPlugin.cs
    ILogProvider.cs
```

Agent.Api

The REST API layer that exposes our diagnostic endpoints.

When Azure Monitor detects exceptions, it calls the /analyze webhook endpoint here.

Agent.Core

The brain of the operation. This is where the orchestration happens, among other things:

- Creates and manages the Semantic Kernel agent
- Prevents analyzing the same exception twice

Agent.Plugins

The custom tools that Claude can invoke. I set up a list of tools for communicating with Azure AppInsights and GitHub. Find the list below.

AppInsights Plugin:

Function Name	Description
`get_exceptions`	Retrieves exceptions grouped by type with occurrence counts
`get_exception_details`	Gets detailed samples with full stack traces

GitHub Plugin:

Tool	Purpose
`get_file_content`	Retrieves source file content from the repository
`search_code`	Searches for code snippets in the repository
`get_multiple_files`	Retrieves multiple files efficiently in one call

….

maybe add some interesting details here or explanations

The Flow

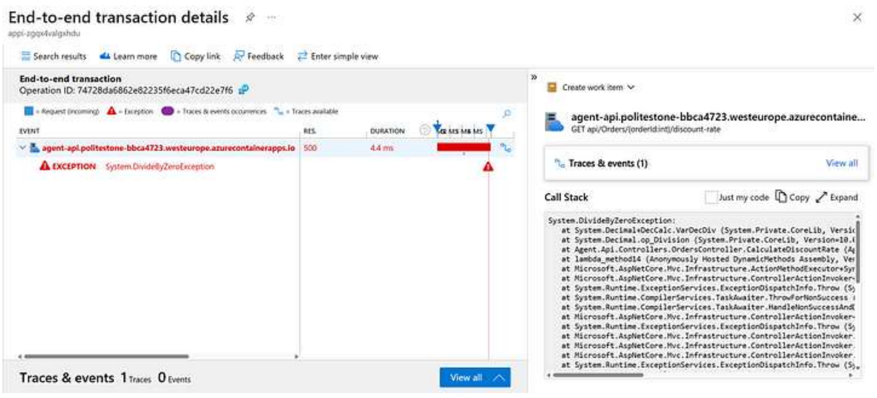
Let’s consider a real case scenario to understand what it exactly does. Mr. Doe is a user of the app. He is having a bit of a rough day interacting with the app store:

1. He applies a coupon to an empty cart. The app tries to divide the discount by zero items, triggering a DivideByZeroException.

2. He clicks “View Order” on that empty cart. The code calls .First() on an empty collection, throwing an InvalidOperationException.

3. Then later, he tries to pay without a profile. The system looks for a shipping address on a null user object, resulting in a NullReferenceException

A few seconds later, App Insights will show the exceptions created by the App when the errors occured. App Insights is the place you can interact with logs and exception that are ingested from the Log Analytic Workspace (LAW).



End-to-end transaction
Operation ID: 74728da6862e82235f6eca47cd22e7f6

EVENT	RES.	DURATION
agent-api.politestone-bbca4723.westeurope.azurecontainerapps.io	500	4.4 ms
EXCEPTION System.DivideByZeroException		

Traces & events 1 Traces 0 Events

agent-api.politestone-bbca4723.westeurope.azurecontaine...
GET api/Orders/{orderId:int}/discount-rate

Call Stack

```
System.DivideByZeroException:
   at System.Decimal+DecCalc.VarDecDiv (System.Private.CoreLib, Versic
   at System.Decimal.op_Division (System.Private.CoreLib, Version=10.(
   at Agent.Api.Controllers.OrdersController.CalculateDiscountRate (Ag
   at lambda_method14 (Anonymously Hosted DynamicMethods Assembly, Ver
   at Microsoft.AspNetCore.Mvc.Infrastructure.ActionMethodExecutor+Syr
   at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker-
   at System.Runtime.ExceptionServices.ExceptionDispatchInfo.Throw (Sy
   at System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess (
   at System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndI
   at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker-
   at System.Runtime.ExceptionServices.ExceptionDispatchInfo.Throw (Sy
   at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.
   at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.
   at Microsoft.AspNetCore.Mvc.Infrastructure.ControllerActionInvoker.
   at System.Runtime.ExceptionServices.ExceptionDispatchInfo.Throw (Sy
```

App Insights interface on exception details (DivideByZeroException).

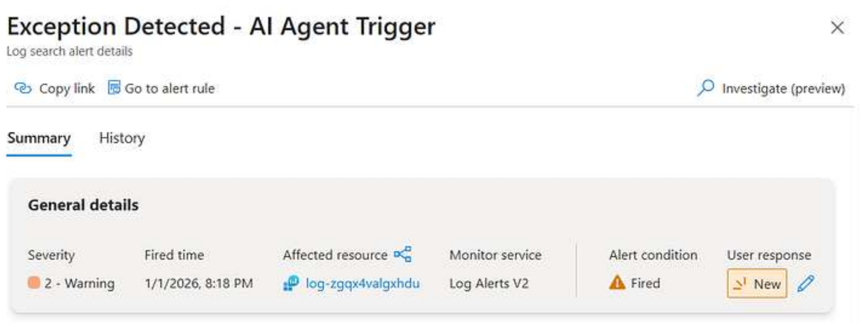
Aside, our Azure Monitor Alert Rule is configured to query the LAW using KQL (Kusto Query Language) every 5 minutes.

The query we are using is pretty simple:

```
AppExceptions
| where TimeGenerated > ago(5m)
| summarize Count = count() by ExceptionType, Pr
| where Count >= 1
```

Ding, after a few seconds when the query is triggered, the alert **verb**.

We can verify on the Azure Portal in our Alert Rule history:


Exception Detected - AI Agent Trigger
Log search alert details

Copy link | Go to alert rule | Investigate (preview)

Summary | History

General details

Severity	Fired time	Affected resource	Monitor service	Alert condition	User response
2 - Warning	1/1/2026, 8:18 PM	log-zgqx4valgxhdu	Log Alerts V2	Fired	New

This Alert Rule has one job. Calling the /analyze endpoint of our Agent API (webhook).
The endpoint will run the analyze of the exceptions that occured the last 5 minutes. If many occured, it will handle the remediation of all of them separatly.
If an exception is already being treated or as been marked as skipped (Exception filter) in the DB, we just pass.

The Exception Filter now takes place. Its role, as **en anglais : comme son nom le fait penser** is to filter the exception that should not be treated. We will deep dive in that in the next section.

Once we know what exception to fix, we prepare a fully detailed prompt for the LLM. I spent a long time trying to find the best way to do this.
At first I wasn't providing enough details in the first call to the LLM and it was inducing too many tool calls.
The idea here is to pre-fetch code and exception detail so the LLM only ask for more if needed (for exemple more code context if required).
The prompt we will be using now look like this:

multiple and/or complex changes for PR we could use github PR tool from Claude directly maybe

```
"You are an expert .NET developer. Analyze this

{{contextBuilder}}

## Instructions

I have PRE-FETCHED the source code above. Analy

**IMPORTANT**: Only use tools if absolutely nec
```

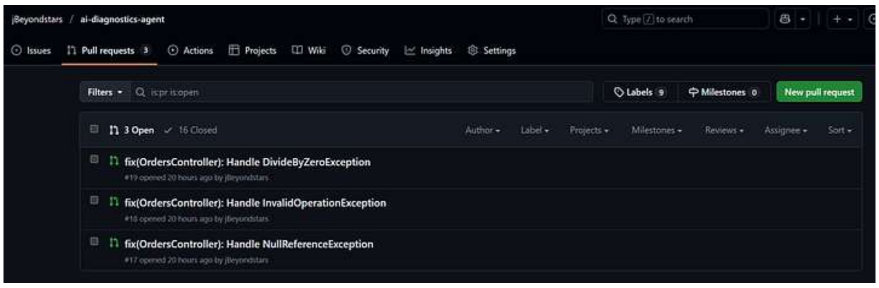
~~~
    - Use `GitHub_get_file_content` if you need to
    - Use `GitHub_search_code` if you need to find
    - Use `AppInsights_get_exception_details` if yo
    - Do NOT call tools if the pre-fetched code is

    ## Analysis Steps

    1. First, analyze the pre-fetched code to ident
    2. If you need more context, use the tools
    3. Determine if this is an INPUT VALIDATION err
    4. If it's a bug, propose a fix

    ## Response Format

    Respond with JSON:
    ```json
    {
        "action": "fix" or "no_fix_needed",
        "rootCause": "explanation",
        "severity": "High|Medium|Low",
        "toolsUsed": ["list of tools you called, or
        "fix": {
            "filePath": "path/to/file.cs",
            "originalCode": "EXACT code to replace"
            "fixedCode": "corrected code",
            "explanation": "what was wrong and how
            "confidence": "High|Medium|Low"
        }
    }
    ```

    If action is "no_fix_needed", omit the "fix" ob
~~~

The variable `contextBuilder` contains the details from the exception.

Here we explain to Claude what he tools he can use and how we want him to proceed with the exceptions.
We'll deep dive in the Tools section to understand how Claude recognize how to use the tools.

Next, Claude analyzes and return the detail for the PR to be created.

The remediation is now prepared and ready to ship. The PRs (one for each exception) are created by our agent.



3 Open ✓ 16 Closed

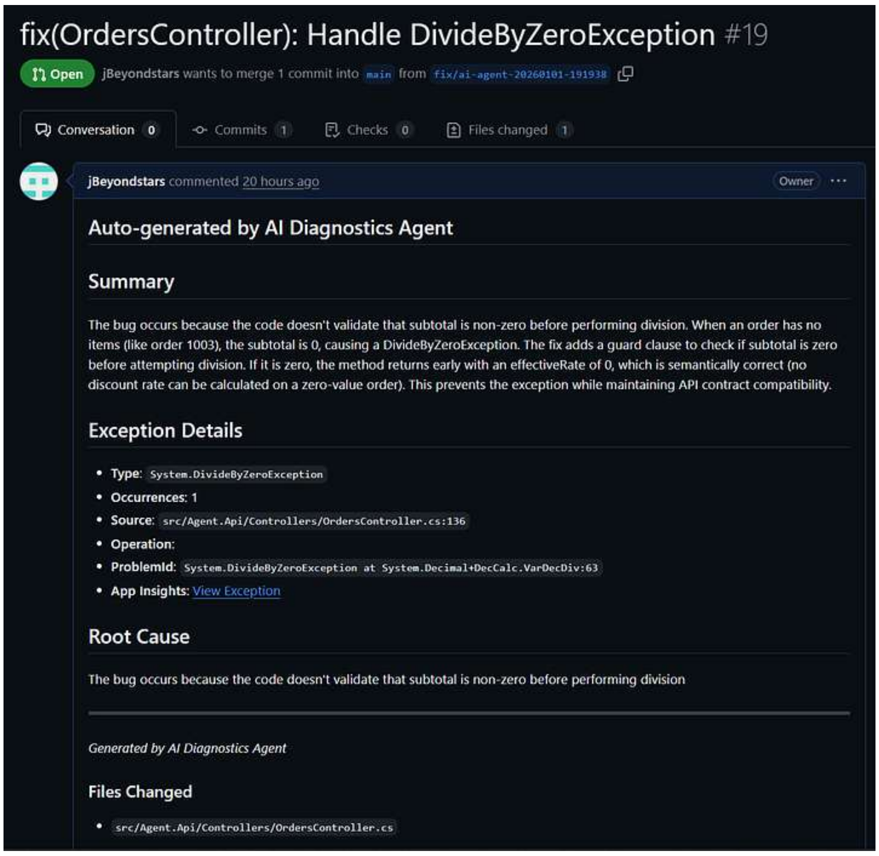
fix(OrdersController): Handle DivideByZeroException
#19 opened 20 hours ago by jBeyondstars

fix(OrdersController): Handle InvalidOperationException
#18 opened 20 hours ago by jBeyondstars

fix(OrdersController): Handle NullReferenceException
#17 opened 20 hours ago by jBeyondstars

The Agent created a PR for each fixeable exception

We can detail the PR description and title as much as we want. Our agent currently provide a brief summary, some exception details with a link to the exception on App Insights.



fix(OrdersController): Handle DivideByZeroException #19

jBeyondstars wants to merge 1 commit into `main` from `fix/ai-agent-20260101-191938`

jBeyondstars commented 20 hours ago

Auto-generated by AI Diagnostics Agent

Summary

The bug occurs because the code doesn't validate that subtotal is non-zero before performing division. When an order has no items (like order 1003), the subtotal is 0, causing a DivideByZeroException. The fix adds a guard clause to check if subtotal is zero before attempting division. If it is zero, the method returns early with an effectiveRate of 0, which is semantically correct (no discount rate can be calculated on a zero-value order). This prevents the exception while maintaining API contract compatibility.

Exception Details

- **Type:** `System.DivideByZeroException`
- **Occurrences:** 1
- **Source:** `src/Agent.Api/Controllers/OrdersController.cs:136`
- **Operation:**
- **ProblemId:** `System.DivideByZeroException at System.Decimal+DecCalc.VarDecDiv:63`
- **App Insights:** View Exception

Root Cause

The bug occurs because the code doesn't validate that subtotal is non-zero before performing division

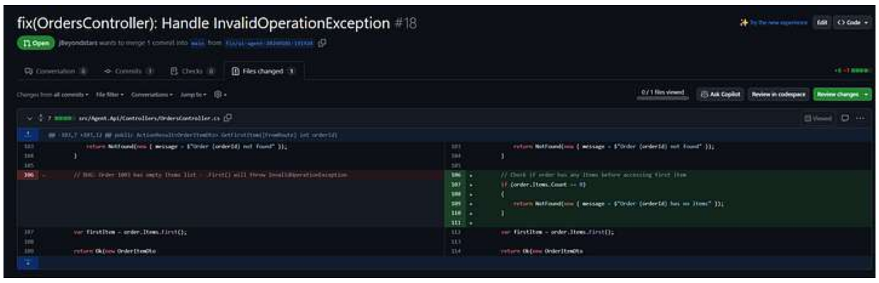
Generated by AI Diagnostics Agent

Files Changed

- `src/Agent.Api/Controllers/OrdersController.cs`

Fully detailed description with App Insights link of the exception

You may wonder know, is the fix viable? Well, if you ever worked with Claude (especially Opus 4.5) you already know how reliable it is in code generation.
So yes, every fixes suggested so far in my tests were consistent and pertinent.



fix(OrdersController): Handle InvalidOperationException #18

src/Agent.Api/Controllers/OrdersController.cs

```
            return NotFound(new { message = $"Order {orderId} not found" });
        }

        // Check if order has any items before accessing first item
        if (order.Items.Count == 0)
        {
            return NotFound(new { message = $"Order {orderId} has no items" });
        }

        var firstItem = order.Items.First();

        return Ok(new OrderItemDto
```

The changes suggested by the Agent for a piece of code trying to call .First() on a collection without verifying in contains elements.

Exception Filtering

As you know, all exceptions don’t require a remediation. HttpRequestException when an external API is down? No code fix. TimeoutException on a slow database? Infrastructure problem.

The agent uses a 3-level filtering system.

Level 1: Exclusion List Infrastructure and transient exceptions are automatically skipped

```
public string[] ExcludedTypes { get; set; } =
[
"System.Net.Http.HttpRequestException",
"System.TimeoutException",
"System.Threading.Tasks.TaskCanceledException",
"StackExchange.Redis.RedisConnectionException",
"Azure.RequestFailedException",
"Microsoft.Data.SqlClient.SqlException",
    // ...
];
```

Level 2: Pattern Matching Partial string matches catch variants

```
public string[] ExcludedPatterns { get; set; } =
[
"ConnectionException",
"Timeout",
"Unreachable",
"NetworkError",
"HostNotFound"
  // ...
];
```

Level 3: Claude Evaluation

For ambiguous exceptions (containing “api”, “service”, “50*”, …), Claude evaluates.

```
private async Task<bool> EvaluateIfFixableAsync(
{
var prompt =
"You are evaluating if a production exception ca

Exception Type: {exception.ExceptionType}
Message: {exception.Message}
Stack Trace (first 500 chars): {exception.StackT
Operation: {exception.OperationName}

Question: Can this exception be prevented or han

Consider:
- If this is a transient infrastructure error (n
- If the code is missing proper error handling,
- If the exception reveals a bug in the business
- If the exception is caused by external systems

Respond with ONLY one word: YES or NO";
// ..
}
```

…

How Tool Use Works with Claude API

When building an AI agent that can interact with external systems, you might think that simply telling Claude about available tools in the system prompt is enough. It’s not.

Here’s what actually happens behind the scenes.

1. Tool Definitions

When calling the Claude API, we must pass a tools array containing JSON schemas for each tool. This tells Claude what tools exist and how to call them:

```
{
  "name": "AppInsights_get_exceptions",
  "description": "Retrieves exceptions from th
  "input_schema": {
    "type": "object",
    "properties": {
      "hours": { "type": "integer", "default":
    }
  }
}
```

2. Tool Execution

Our backend must contain the actual code that performs the work. In our case, we use Semantic Kernel plugins:

```
[KernelFunction("get_exceptions")]
  [Description("Retrieves exceptions from the la
  public async Task<string> GetExceptionsAsync(i
  {
      // Actually query Application Insights her
      // Code available in the Github repo
  }
```

What is Kernel?

Semantic Kernel is Microsoft's SDK that acts as a bridge between your code and LLMs. Think of it as the middleware that connects Claude (or any LLM) to your application's capabilities.

In this project we use the Kernel for three purposes:

- LLM Abstraction: lets us swap between Claude, OpenAI, or Azure OpenAI with a simple config change.
- Plugin Registry
- Agent Orchestration: The Kernel is passed to the Agent, which uses it to discover and execute tools.

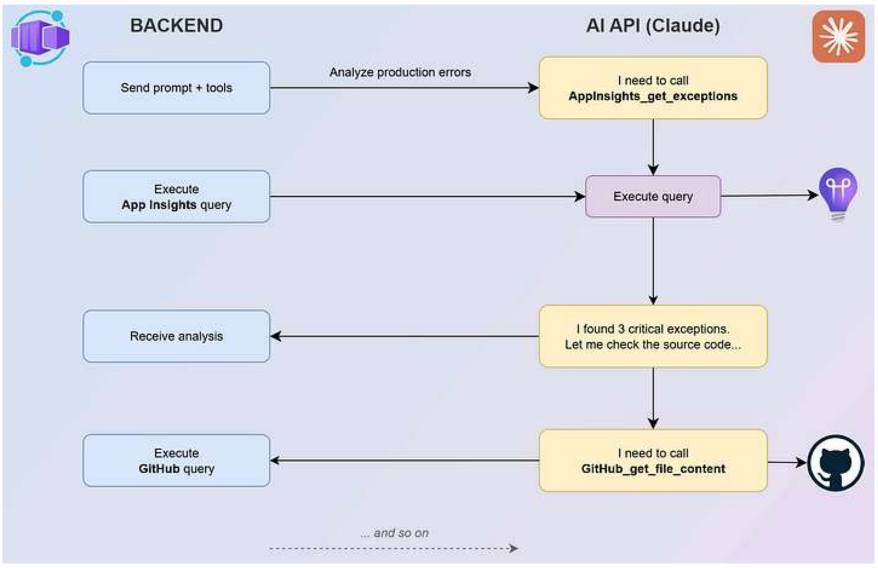
```
var agent = new ChatCompletionAgent
 {
   Name = "DiagnosticsAgent",
   Instructions = SystemPrompt,
   Kernel = kernel, // Contains all registered p
   Arguments = new KernelArguments(new OpenAIPro
   {
   FunctionChoiceBehavior = FunctionChoiceBehavi
   })
 };
```

The FunctionChoiceBehavior.Auto() is the key here. The LLM autonomously decides which tools to invoke based on the conversation context.

3. The Tool Loop

When Claude decides to use a tool, it doesn't execute anything. It simply requests a tool call. Our code must:

1. Detect that Claude wants to use a tool (StopReason = "tool_use")
2. Parse the tool name and arguments
3. Execute the corresponding function
4. Send the result back to Claude
5. Let Claude continue reasoning with the new data


BACKEND
AI API (Claude)
Send prompt + tools
Analyze production errors
I need to call AppInsights_get_exceptions
Execute App Insights query
Execute query
Receive analysis
I found 3 critical exceptions. Let me check the source code...
Execute GitHub query
I need to call GitHub_get_file_content
... and so on

Agent flow diagram

Deduplication Handling

Without deduplication, the same NullReferenceException thrown 50 times would trigger 50 Claude API calls and 50 identical PRs.

Not ideal, right?

We'll use two layers of protection.

1. GitHub PR Check
2. History storage in DB.

No open PR doesn't mean we should re-analyze immediately. The exception might reoccur while the fix is being deployed, or the developer is still reviewing. We enforce a cooldown before allowing re-analysis of the same problem.

Redis vs Database

I used Redis for this demo (simple and fast to implement). For production, I'd recommend adding a database to store fix history: which exceptions were fixed, which PRs were merged vs rejected. This data enables a learning loop for improving the agent over time.

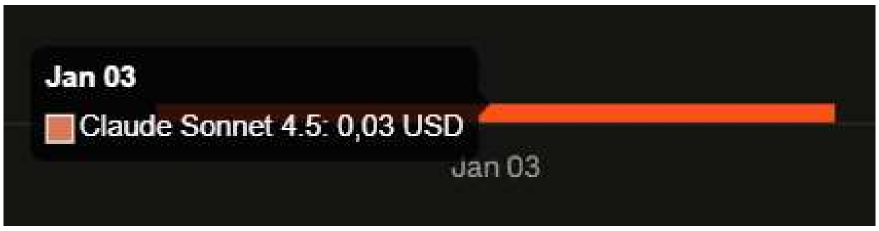
Staying in Sync with GitHub

Since we track state locally, we need to know when PRs are closed. A simple GitHub webhook calls our agent when a PR is merged or rejected, clearing the cooldown cache so re-analysis can happen if needed.

Cost Breakdown

I was actually agreeably surprised by how low the cost for the whole process is.

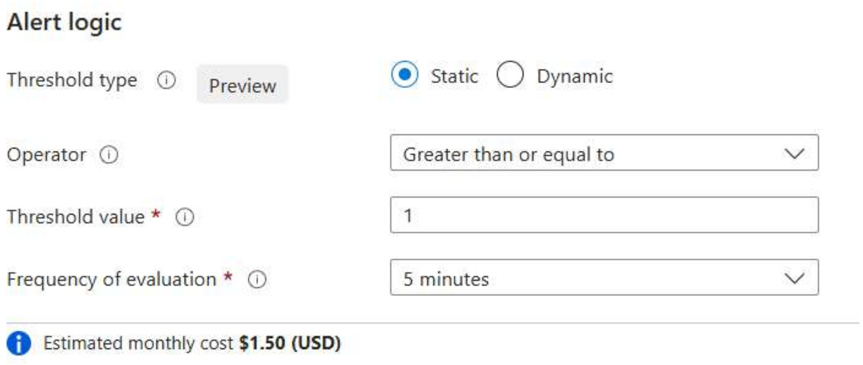
Claude API cost



Jan 03
🟧 Claude Sonnet 4.5: 0,03 USD

Approximately 0,03$ per simple exception (first prompt context is enough) using Claude Sonnet 4.5.

Alert rule cost:



Alert logic

Threshold type ⓘ Preview	[✓ Static] Dynamic
Operator ⓘ	Greater than or equal to
Threshold value * ⓘ	1
Frequency of evaluation * ⓘ	5 minutes

Estimated monthly cost **$1.50 (USD)**

The estimated monthly cost if you run the alert rule 24h/7 is approximately $1.50.

Agent Hosting

If you plan using Azure to deploy your agent, most of the time using a Container App would cost nothing.

If you use a free container registry (such as Docker Hub or GitHub Container Registry) you would have no other fees.
If you stay on Azure then using the Azure Container Registry would cost $5.00/month for Basic tier.

As you see, that's less than $10/month for an always-on automated diagnostics system.

What's Next?

The Agent is a prototype still, there is few enhancements that could and should be added to be production ready.

1. Multi-Agent (Coder Agent, Reviewer Agent..)
2. Rework on Pull Request comments (Agent would listen to your comments and suggest new fix)
3. Slack/Teams Integration to notify when PRs are created Allow manual approval before creation
4. Learning Loop Track which fixes were merged vs. rejected Fine-tune the prompt based on feedback

5.

Conclusion

We built a complete self-healing system that:

- Monitors 24/7 via Azure Monitor
- Alert Rules Triggers automatically via webhooks
- Filters non-actionable exceptions (3-level smart filtering)
- Deduplicates at multiple levels (Redis + existing PR check)
- Analyzes with Claude in optimized LLM calls
- Creates Pull Requests automatically
- Costs ~$0.04 per analysis

Resources

- https://github.com/jBeyondstars/ai-diagnostics-agent—Full source code with Bicep templates
- https://learn.microsoft.com/en-us/semantic-kernel/—Microsoft's SDK for LLM integration
- https://learn.microsoft.com/en-us/azure/azure-monitor/alerts/alerts-overview—Configuring alerts and webhooks

. . .

Did you enjoy this deep dive? Follow me here or on LinkedIn for more content.