

WSO2con2025

**End-to-End
Integration Simplified**



WSO2 Integrator – Overview

Solve any integration challenge

Out of the box support to connect anything—APIs, systems, databases, AI agents— across any environment and protocol with 100% enterprise integration pattern (EIP) compliance.



Automations

Automatically trigger your integrations based on a predefined schedule.

AI Agents

AI Agents that can reason and plan with GenAI models, leverage available knowledge, and call available tools.

Integrations as APIs

Use API requests to trigger your integrations in real-time.

Event-based Integrations

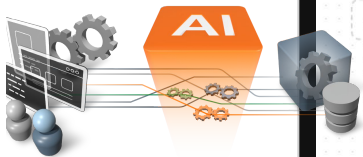
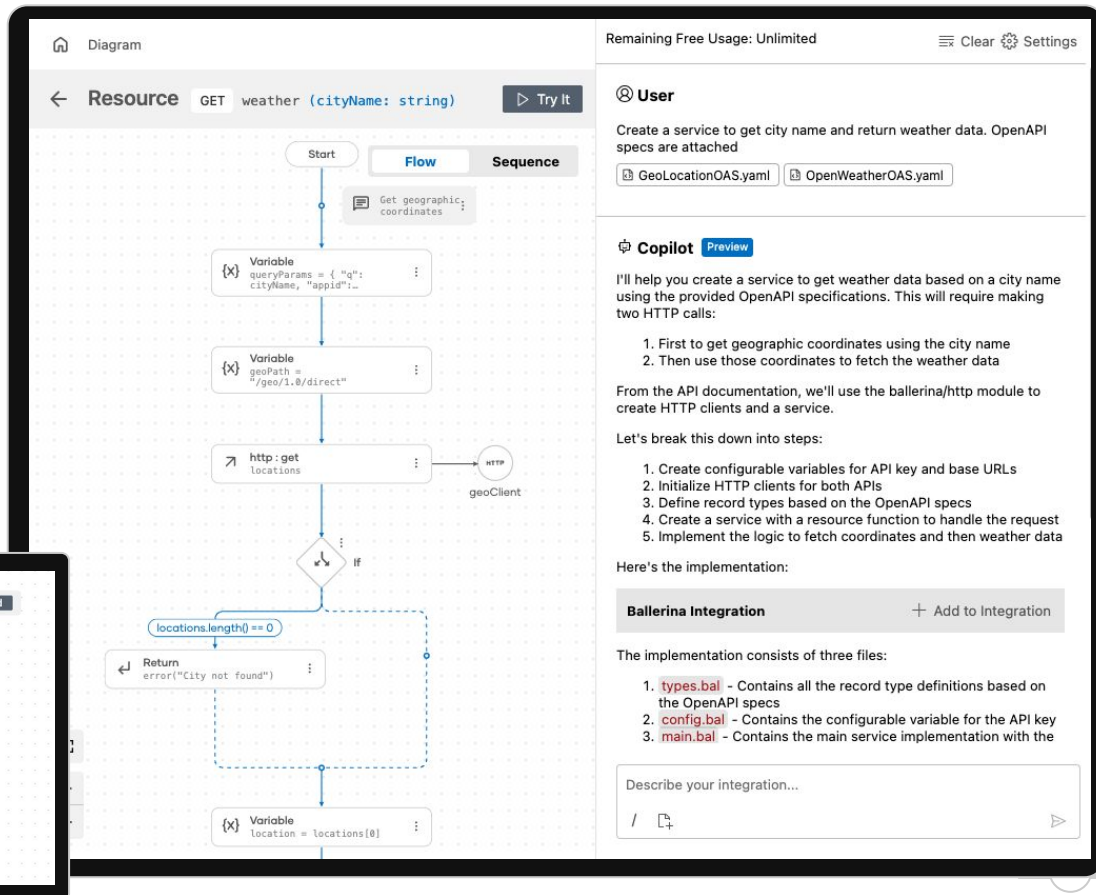
Trigger your integrations based on events.

File-based Integrations

Trigger your integrations based on the availability of files in a location.

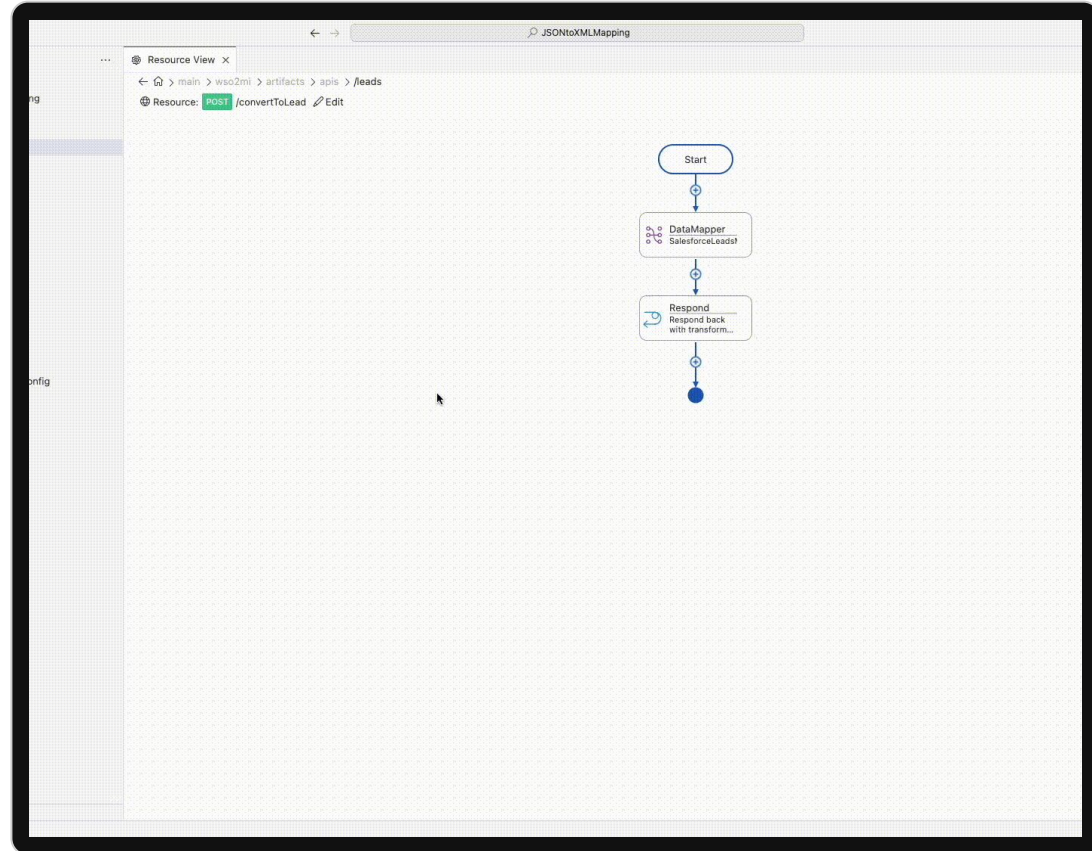
AI-assisted development

- Describe your integration requirements in natural language, and **have AI generate the code for you**
- **In-line code suggestions** when adding constructs



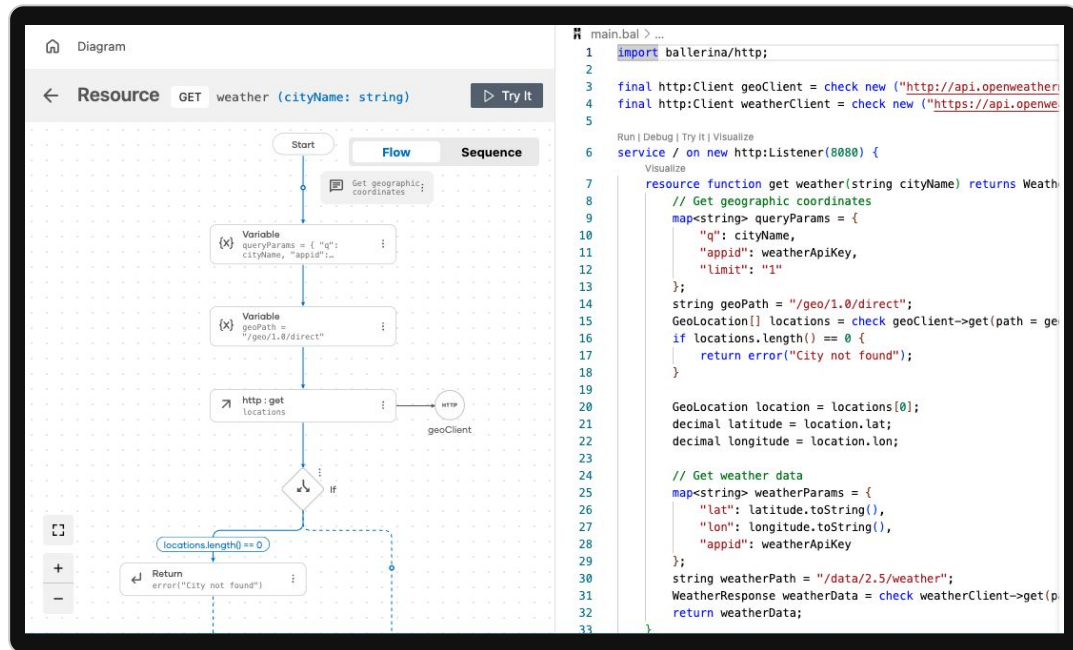
AI-assisted data mapping

- **Automatically map data fields** between the source and target schemas using AI
- Transform data from any format using a graphical interface or scripting



Low-code and pro-code development

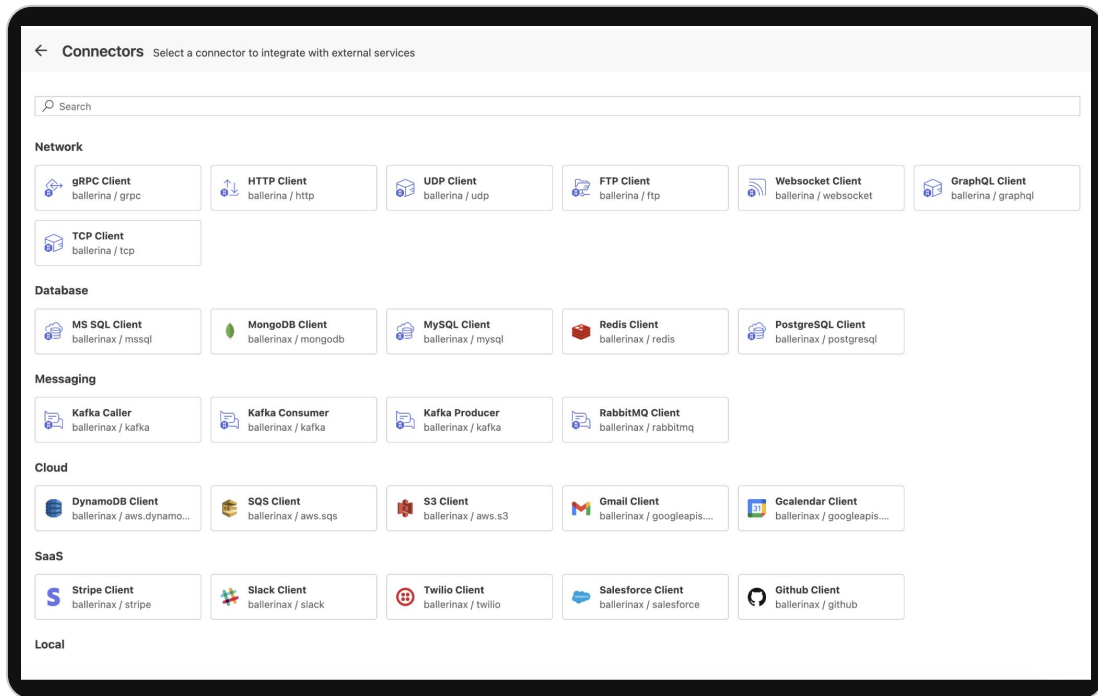
- **Unique low-code approach** to develop any complex integration without hitting roadblocks
- **Low-code and pro-code on a single technology** with fluid switching between the two



“Low-code is pro-code, and pro-code is low-code”

Pre-built connectors and templates

- Rapidly develop integrations between systems, cloud services, and AI Agents
- 200+ pre-built connectors
- Distributed through Ballerina Central and WSO2 Store for easy reuse
- Ability to generate connectors using OpenAPI specification, WSDL



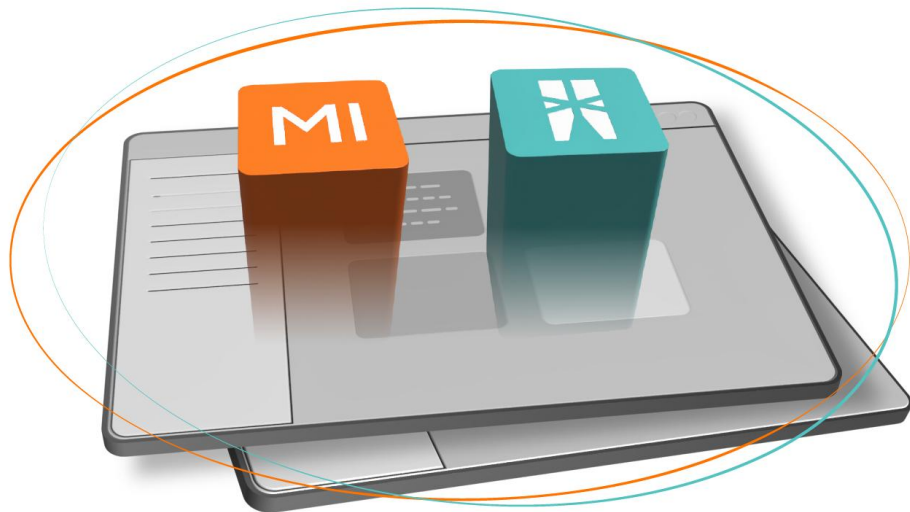
Build and manage AI Agents

- **Build and manage your own AI Agents** with built-in capabilities
- Includes comprehensive connectivity to GenAI models, knowledge bases, and tools (systems, APIs, functions)
- Seamlessly integrate AI Agents into apps to create **intelligent digital experiences**

The screenshot displays the 'AI Agent' configuration interface, titled 'Create a new AI agent for your integration'. It features a progress bar with three steps: 1. Basic Agent Setup, 2. Model Configuration, and 3. Tool Integration. The 'Add New Tool' section allows users to enhance their AI Agent's capabilities by adding tools. It includes a 'Tool Type' dropdown menu set to 'Connector', radio buttons for 'Create New Connection' (selected) and 'Use Existing Connection', a 'Search Connector' input field, and a list of connectors: Gmail and Google Calendar. An 'Import from Open API Specification' section with an 'Import' button is also present. A workflow diagram on the right shows a sequence: 'Start' leads to a 'chatAgent' block (containing placeholder text), which then leads to a 'sendMessage' block (labeled 'msgRes'), and finally to a blue circle representing the end of the process. The 'chatAgent' block has three output arrows pointing to icons for OpenAI, Gmail, and a third service. The 'sendMessage' block has one output arrow pointing to a Telegram icon.

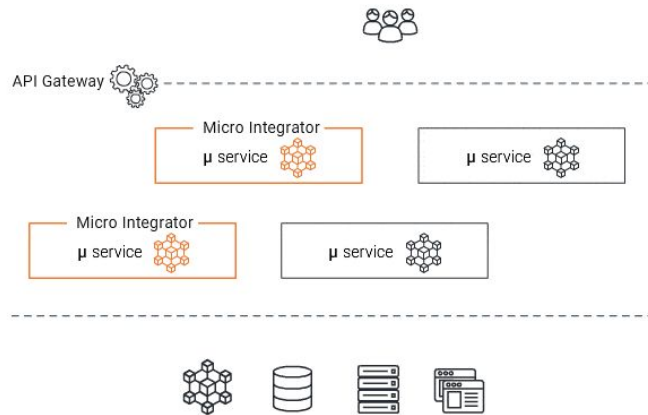
Centralized monitoring and management

- Monitor and manage all your integration projects from a single control plane
- Easily switch between nodes and groups, maintaining control and flexibility across your integration landscape

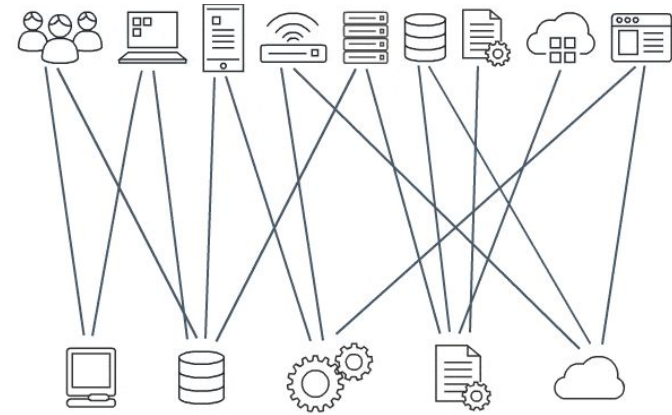


Multiple architectural styles

- Supports both centralized ESB-style, or decentralized Microservices-style integration architectures



Microservices



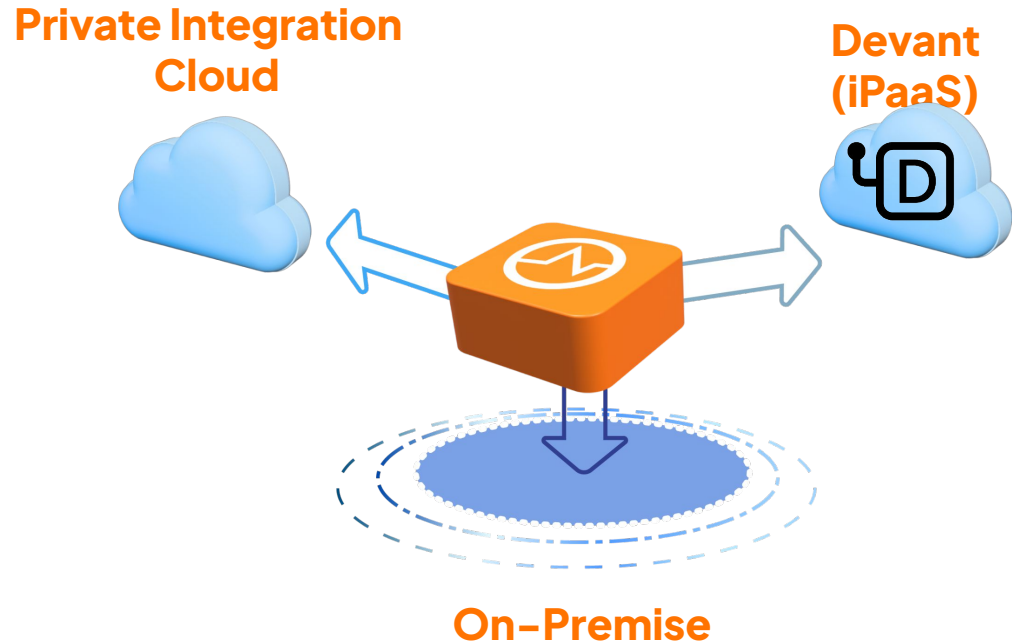
ESB (Centralized)

Deployment Options

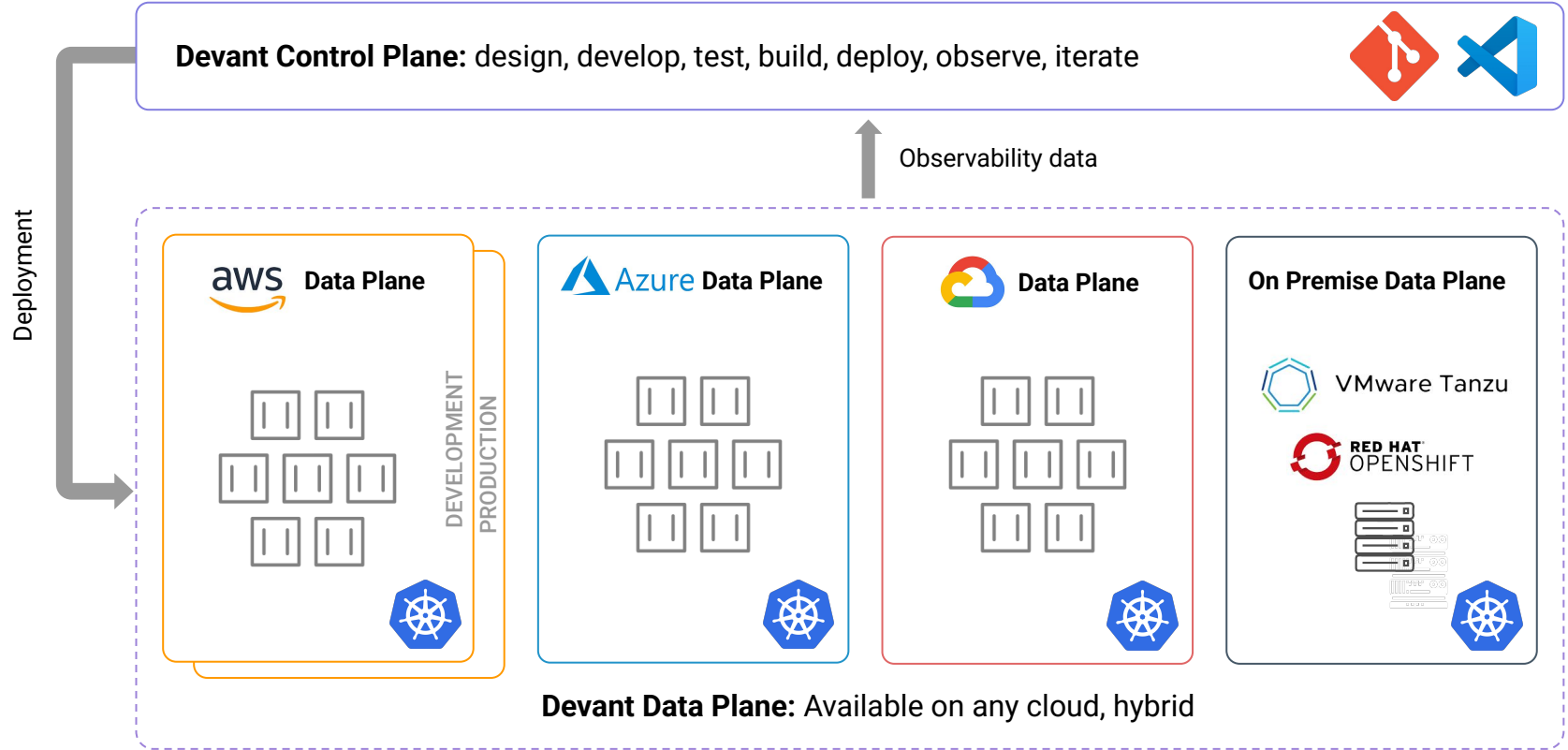
The background is a vibrant space scene. It features a gradient from deep red on the left to dark blue on the right. Numerous stars of varying sizes and colors (white, yellow, orange) are scattered across the field. In the lower-left corner, there is a large, translucent blue planet with a thin, light blue ring. In the lower-right corner, there is a smaller, translucent purple planet with a thin, light purple ring.

Flexible deployment options

- Supports on-premise, cloud and hybrid deployments
- Customers can manage deployments themselves
- WSO2 can manage deployments, so that customers don't have to worry about overheads
 - In a private cloud
 - In Devant (iPaaS)



iPaaS deployment



WSO2 Integration products



WSO2 Integration Product Portfolio

Software Product: WSO2 Integrator

WSO2 Integration Control Plane

WSO2 Micro Integrator

Low-code business integration.

WSO2 Streaming Integrator

Low-code streaming integration.

WSO2 Ballerina Integrator

Low-code business integration with pro-code parity.

“AI iPaaS”

WSO2 Devant

Low-code “AI iPaaS”

In ballet, “Devant” means “to the front”

Development Tools



VSCode Extensions



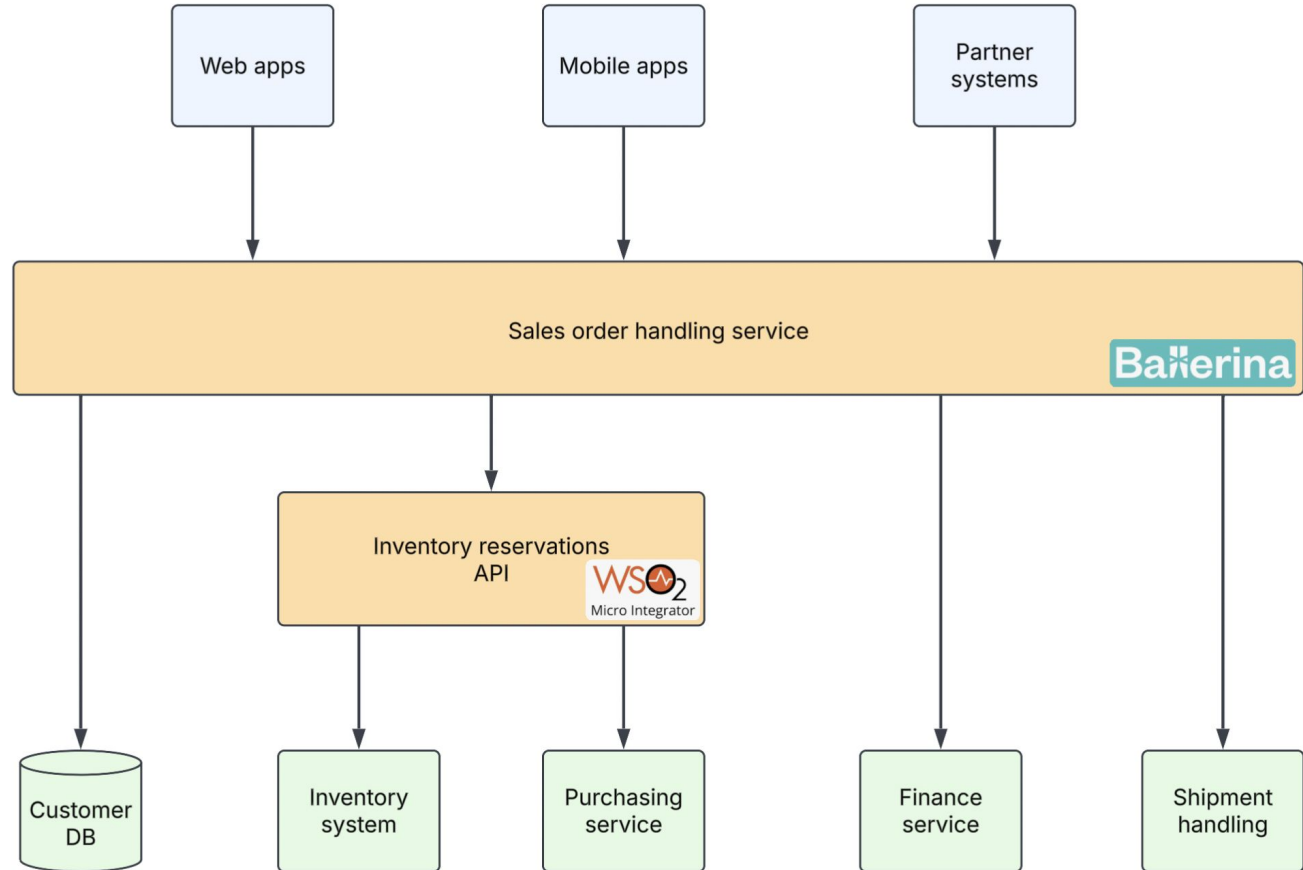
Streaming Integrator Tooling



Integration Lab

Use case – Sales order processing

Sales order processing scenario



Inventory reservations API – WSO2 MI

API path: /inventory/reservations

Request payload:

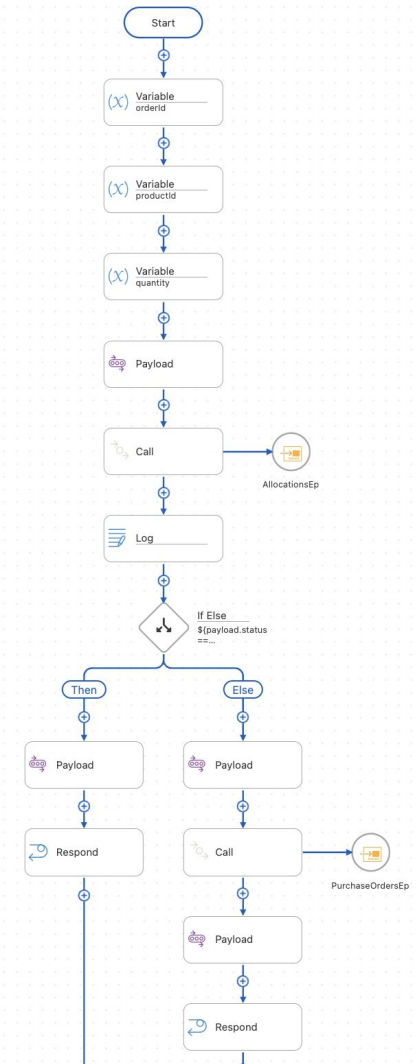
```
{  
  "order": "A100",  
  "product": "p1",  
  "amount": 100  
}
```

Flow:

- Allocate items from the **inventory system**
- If the allocation successful
 - Respond with success result
- If the product is out of stock
 - Call **purchasing service** to order items
 - Respond with the order status

```
{  
  "orderId": "A100",  
  "status": "success"  
}
```

```
{  
  "orderId": "A100",  
  "status": "ordered"  
}
```



Sales order handling API – Ballerina Integrator

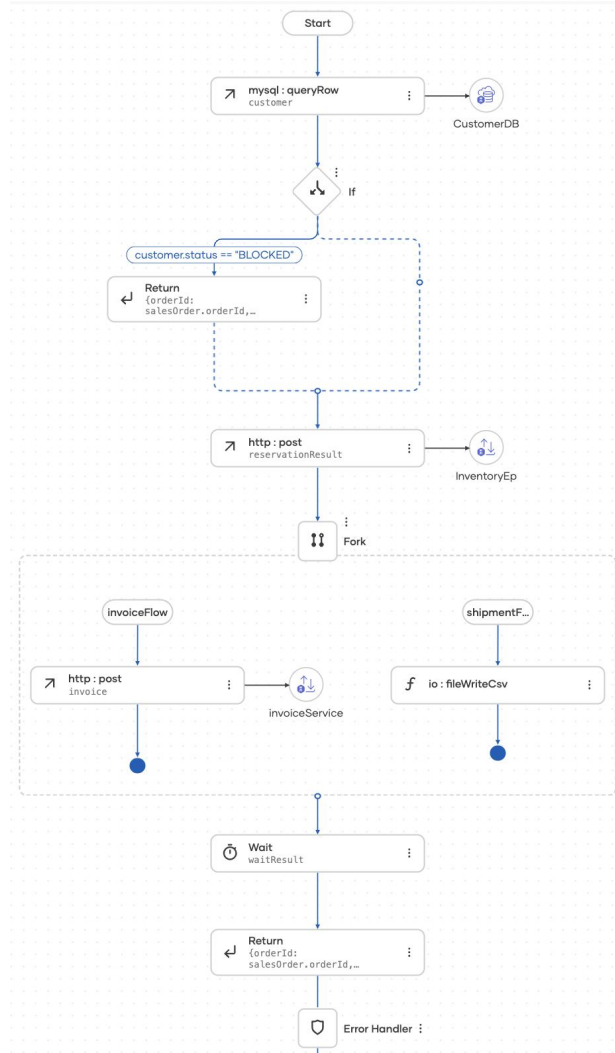
API path: /sales/orders

Request payload:

```
{
  "orderId": "A102",
  "customerId": "c1",
  "productId": "p1",
  "quantity": 10
}
```

Flow:

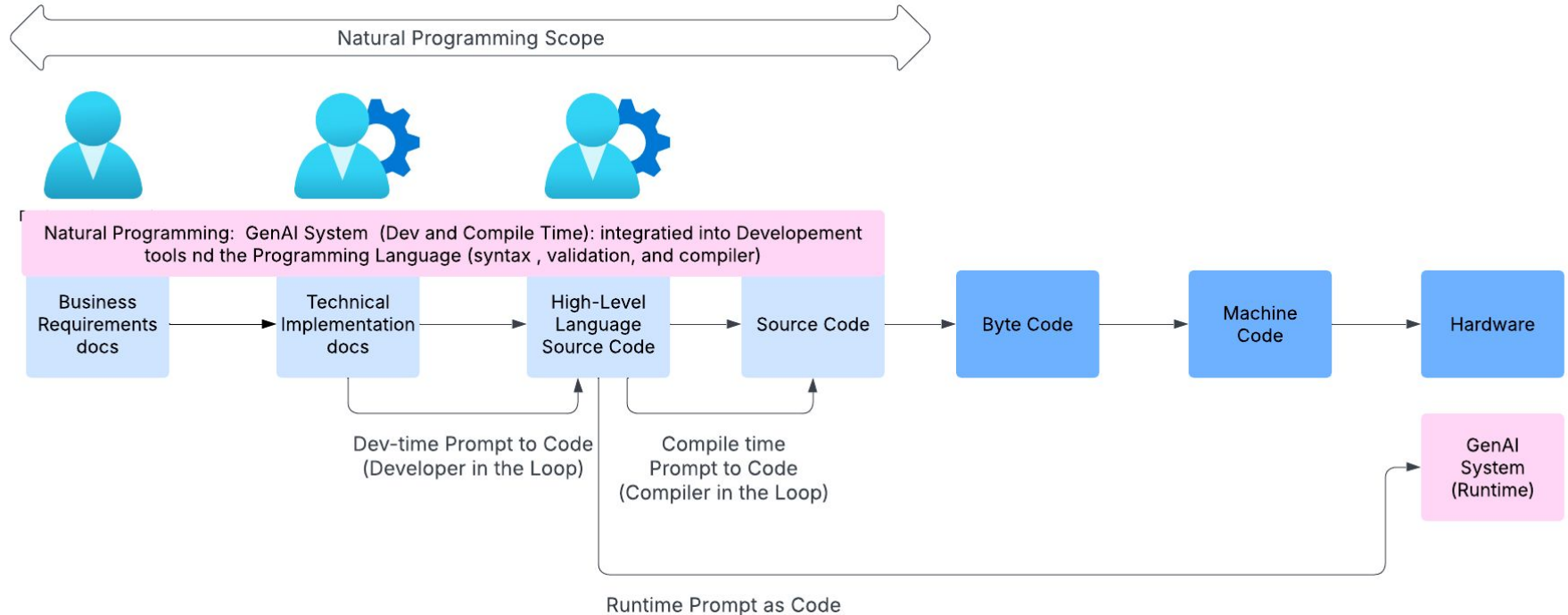
- Get customer data from the **Customer DB**
- If the customer is blocked
 - Respond with a failure message
- Call **Inventory API**
- Start **invoice processing** and **shipment handling** in parallel
 - Invoice is a REST service
 - Shipment handling is a CSV file based service
- Once all tasks are completed respond to the customer



Use case – Using AI for development



Re-imagining the development process with native Natural Language integration



“Natural Language as Code” at runtime

- Simplified LLM calls with support for data binding – the required response structure is inferred from the return type, simplifying the prompt.
- Compile-time validated prompts.
- Function signature enforces required variables.
- These functions become shareable versions of

```
import ballerina/np;

public isolated function rateBlog(
    Blog blog,
    np:Prompt prompt = `You are an expert content reviewer for a blog site that
    categorizes posts under the following categories: ${categories}

    Your tasks are:
    1. Suggest a suitable category for the blog from exactly the specified categories.
       If there is no match, use null.

    2. Rate the blog post on a scale of 1 to 10 based on the following criteria:
       - **Relevance**: How well the content aligns with the chosen category.
       - **Depth**: The level of detail and insight in the content.
       - **Clarity**: How easy it is to read and understand.
       - **Originality**: Whether the content introduces fresh perspectives or ideas.
       - **Language Quality**: Grammar, spelling, and overall writing quality.

    Here is the blog post content and submitted category:

    **Blog Post Content:**
    ${blog.title}
    ${blog.content}`) returns Review|error = @np:LlmCall external;
```



```
service on new http:Listener(8080) {
```

Visualize

```
resource function post blog(Blog blog) returns http:Created|http:BadRequest|http:InternalServerError {
  do {
    Blog {title, content} = blog;

    Review {suggestedCategory, rating} = check rateBlog(blog);

    if suggestedCategory is () || rating < 4 {
      return <http:BadRequest>{
        body: "Blog rejected due to low rating or no matching category";
      }
    }

    _ = check db->execute(`INSERT INTO Blog (title, content, rating, category) VALUES (${
      title}, ${content}, ${rating}, ${suggestedCategory})`);
    return <http:Created> {body: "Blog accepted"};
  } on fail {
    return <http:InternalServerError>{body: "Blog submission failed"};
  }
}
```

Visualize

```
resource function get blogs/[string category]() returns Blog[]|http:InternalServerError {
  do {
    stream<BlogRecord, error?> result =
      db->query(
        `SELECT title, content, rating, category FROM Blog WHERE category = ${
          category} ORDER BY rating DESC LIMIT 10`);
    return check from BlogRecord blog in result select {title: blog.title, content: blog.content};
  } on fail {
    return <http:InternalServerError>{body: "Failed to retrieve blogs"};
  }
}
```

}



“Natural Language as Code” at compile time

- Using a Copilot at compile-time to generate code based on a natural language prompt.
- The generated code can change with each build.
 - ⦿ The code generated with the last build will be persisted in the generated folder, just to refer to if necessary.
- Enforces some restrictions compared to a traditional Copilot to avoid unintentional

```
import ballerina/http;
import ballerina/io;
import ballerina/np;

configurable string populationDataUrl = ?;

type Country record {
    string country;
    string continent;
    int population;
    decimal gdp;
};

type GDPPerCapita record {
    string country;
    string continent;
    decimal gdpPerCapita;
};

function getCountriesWithHighestGdpPerCapitaInContinent(Country[] countries, string continent)
    returns GDPPerCapita[]|error = @np:GenerateCode {
    | prompt: string `Filter the 10 countries with a population of at least 10,000,000 with
    | | | | | the highest GDP per capita in the given continent and return the data.`
    } external;

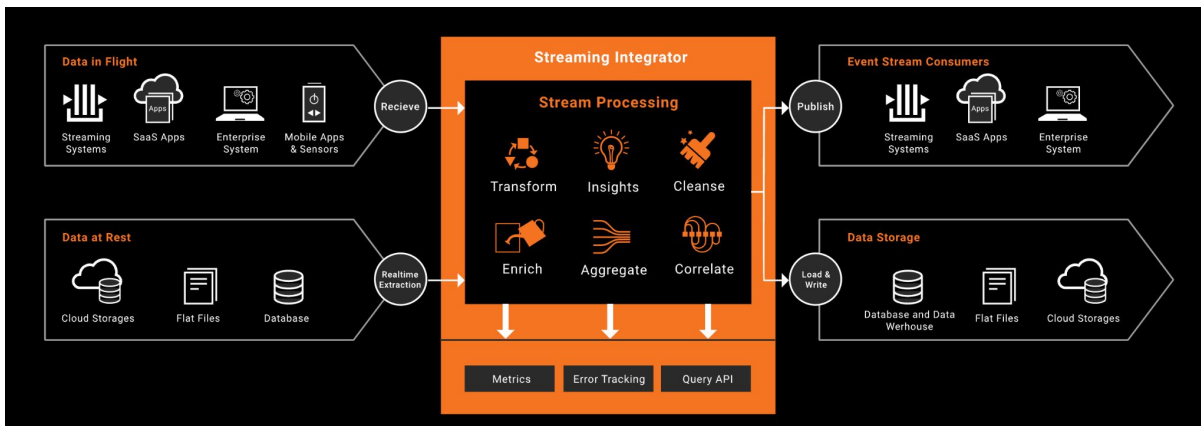
public function main(string continent) returns error? {
    http:Client populationEp = check new (populationDataUrl);
    Country[] countries = check populationEp->/countries;

    GDPPerCapita[] summary =
    | check getCountriesWithHighestGdpPerCapitaInContinent(countries, continent);
    io:println(summary);
}
```



WSO₂ Streaming Integrator

- WSO2 Streaming Integrator is a **real-time data processing** and integration solution that enables **event-driven architectures** by seamlessly ingesting, analyzing, and acting on **streaming data**.
- Execution happens in real time, **as and when** the data is made available
- Low resource usage** since only a fraction of data is processed at once
- Can work with any **database, file, data source, or destination**



Question Time!



The background is a dark, deep blue space filled with a vibrant, multi-colored nebula or galaxy. The nebula's colors range from deep reds and oranges to bright yellows and blues, creating a sense of depth and movement. Scattered throughout the scene are numerous 3D geometric shapes, primarily cubes and rectangular prisms, in various sizes and orientations. These shapes are rendered with a blue-to-red gradient, giving them a three-dimensional appearance. Some shapes have small, glowing square or triangular cutouts. A large, dark blue sphere is partially visible on the left edge. The overall composition is dynamic and futuristic.

Thank you!

WSO2con2025
