WSO2CON2024

ORANGE ROOM



Ballerina
Swan Lake

Utilizing Ballerina for Cloud native integrations

WSO2CON - 2024

Utilizing Ballerina for Cloud Native and Decentralized Integrations



Anupama Pathirage
Director of Engineering - Integration
WSO2



Maryam Ziyad
Senior Technical Lead
WSO2



Mohamad Shafreen
Associate Director/ Architect
WSO2



Hasitha Aravinda
Associate Director/ Architect
WSO2

In this tutorial

Session 1: Understanding Ballerina's role in integration

Session 2: Designing and implementing REST APIs for integration

Session 3: Designing and implementing GraphQL services with Ballerina

Session 4 : Introduction to data integration

https://github.com/wso2con2024/ballerina-tutorial/

Session 1: Understanding Ballerina's role in integration



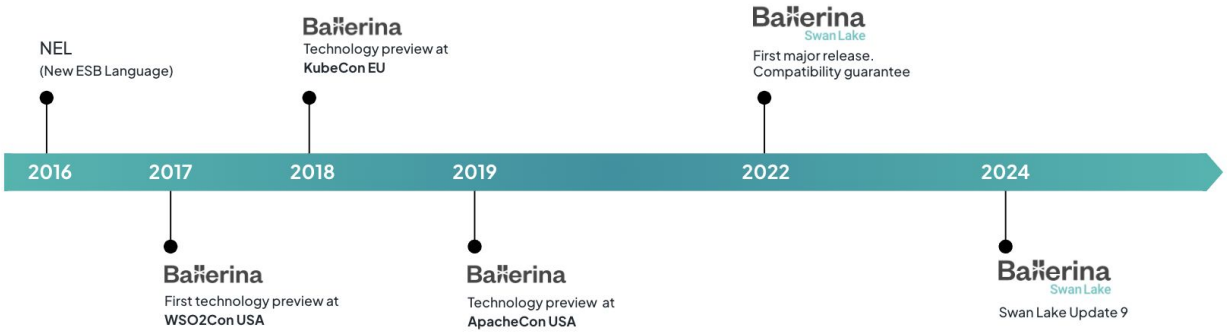
Ballerina
Swan Lake

What is Ballerina?

Ballerina is about *integration*.

What is Ballerina?

- Open source, cloud-native technology **optimized for integration**
- **Developed by WSO2** since 2016 and first released in February 2022
- **Rich ecosystem** of different services, data formats, and connectors
- Edit/view source code **textually or graphically** as sequence diagrams and flowcharts
- Built-in, easy and **efficient concurrency** with sequence diagrams and safety primitives



Ballerina provides right abstractions for …

Data

- Representing the data - shape of data, separation from code (behavior)
- Manipulating data
- Communicating data

Network

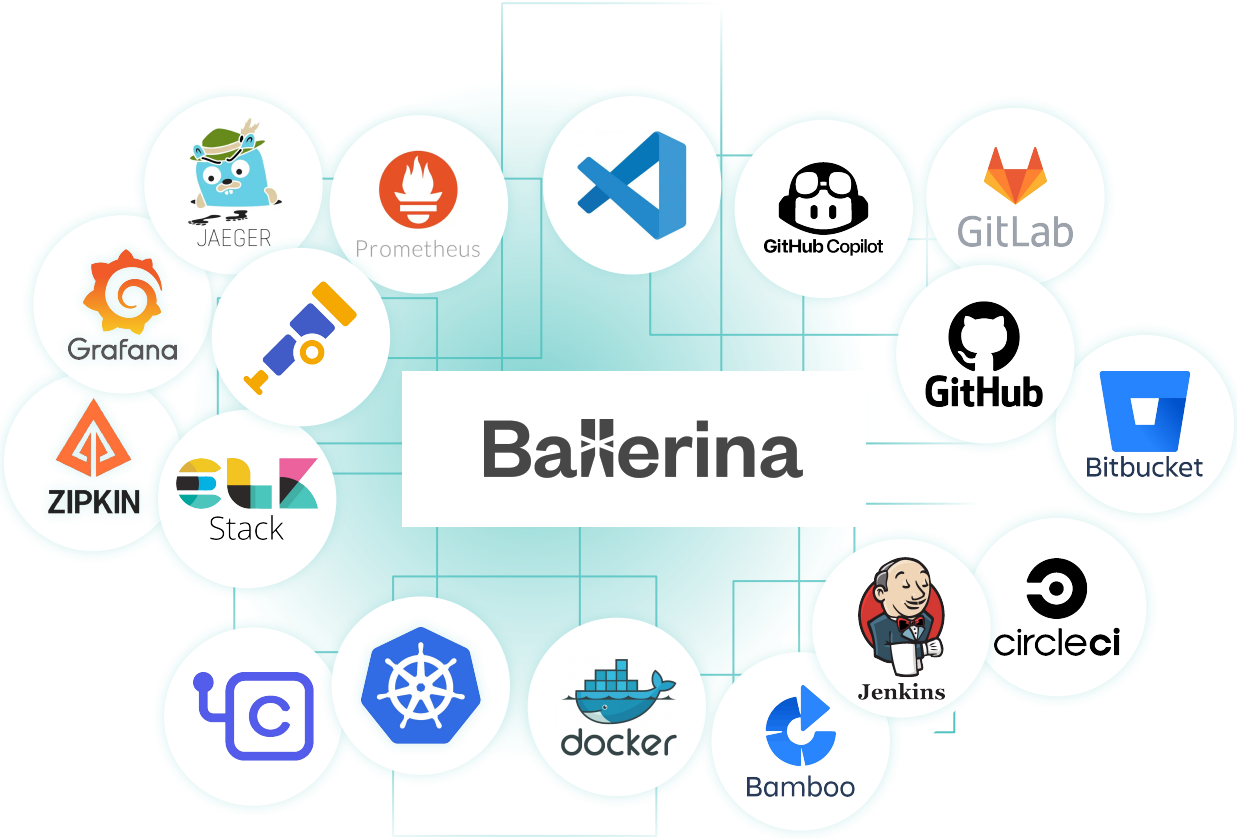
- Communicating data across different clients and services via different network protocols

Concurrency

- Concurrency safety during application scaling and inherently concurrent operations on data

Development

VS Code as tooling with familiar toolset



Let’s write the first Ballerina package

- Use `bal new` command

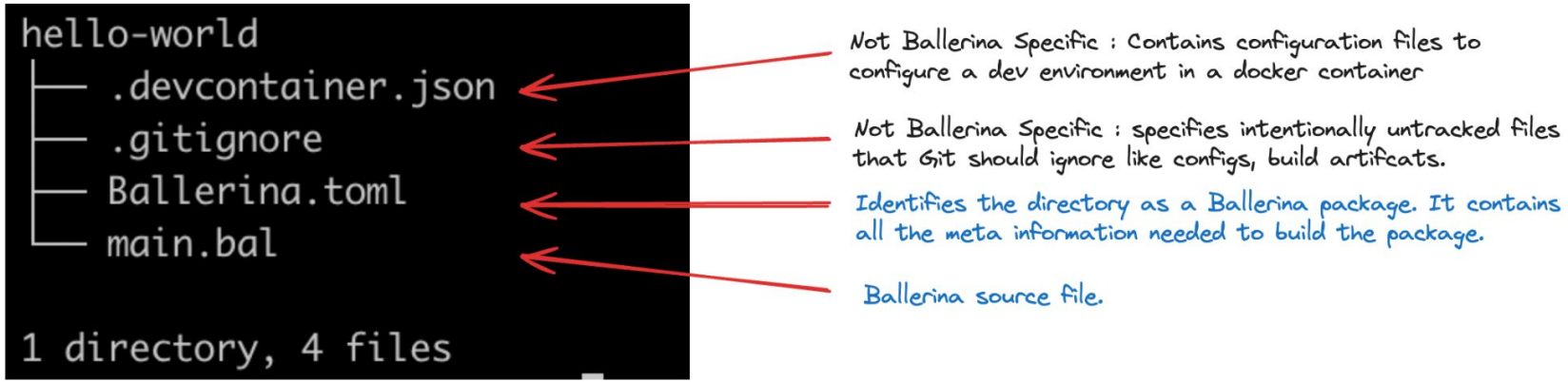
```
bal new <package-path> [-t |--template <template-name> ]
```

E.g.

- `bal new hello-world` – Creates a package with main function
- `bal new hello-world -t service` – Creates a package with service template
- `bal new hello-world -t lib` – Creates a package with library template

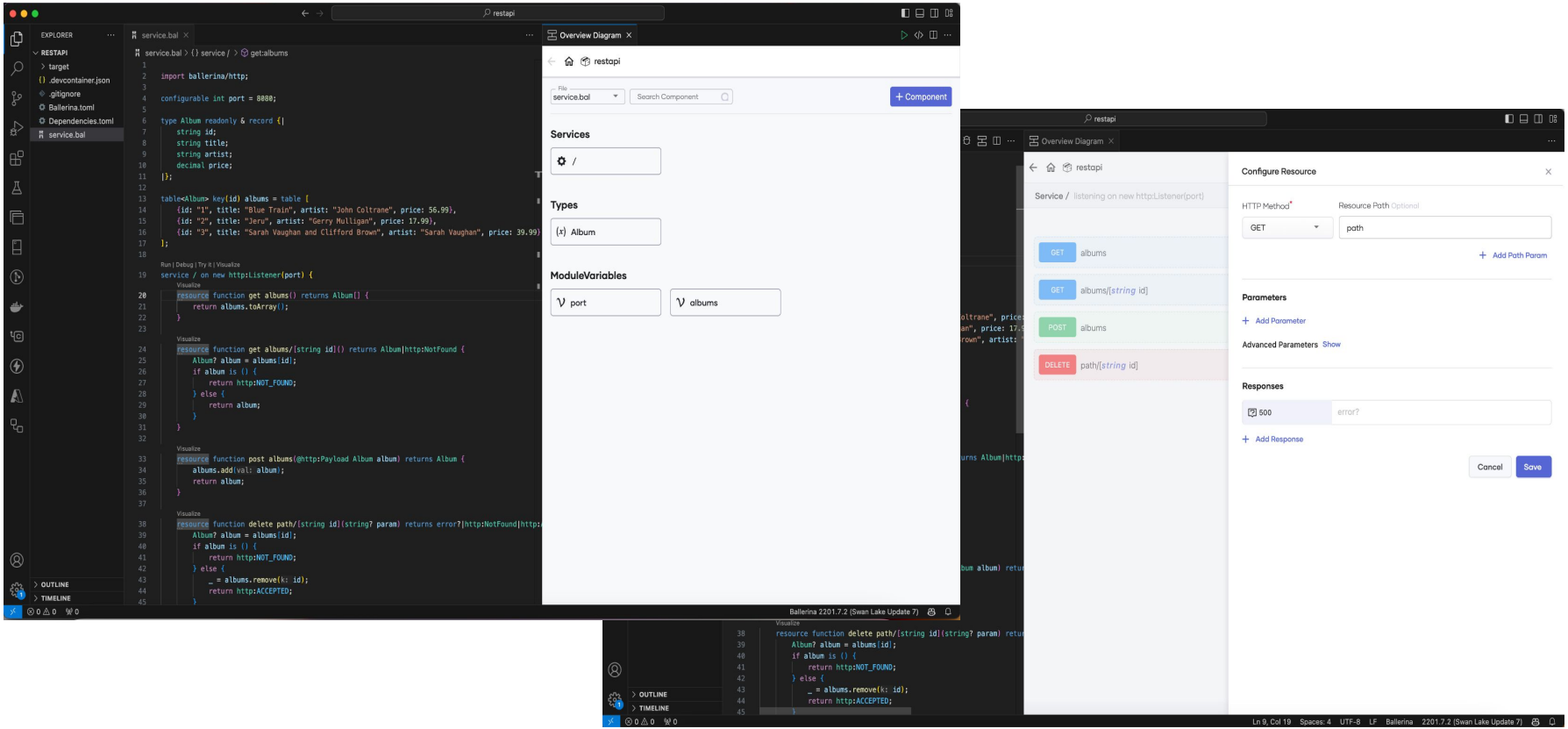
Reference: https://ballerina.io/learn/cli-commands/#package-commands

Structure of Ballerina package



Reference: https://ballerina.io/learn/package-references/#package-layout

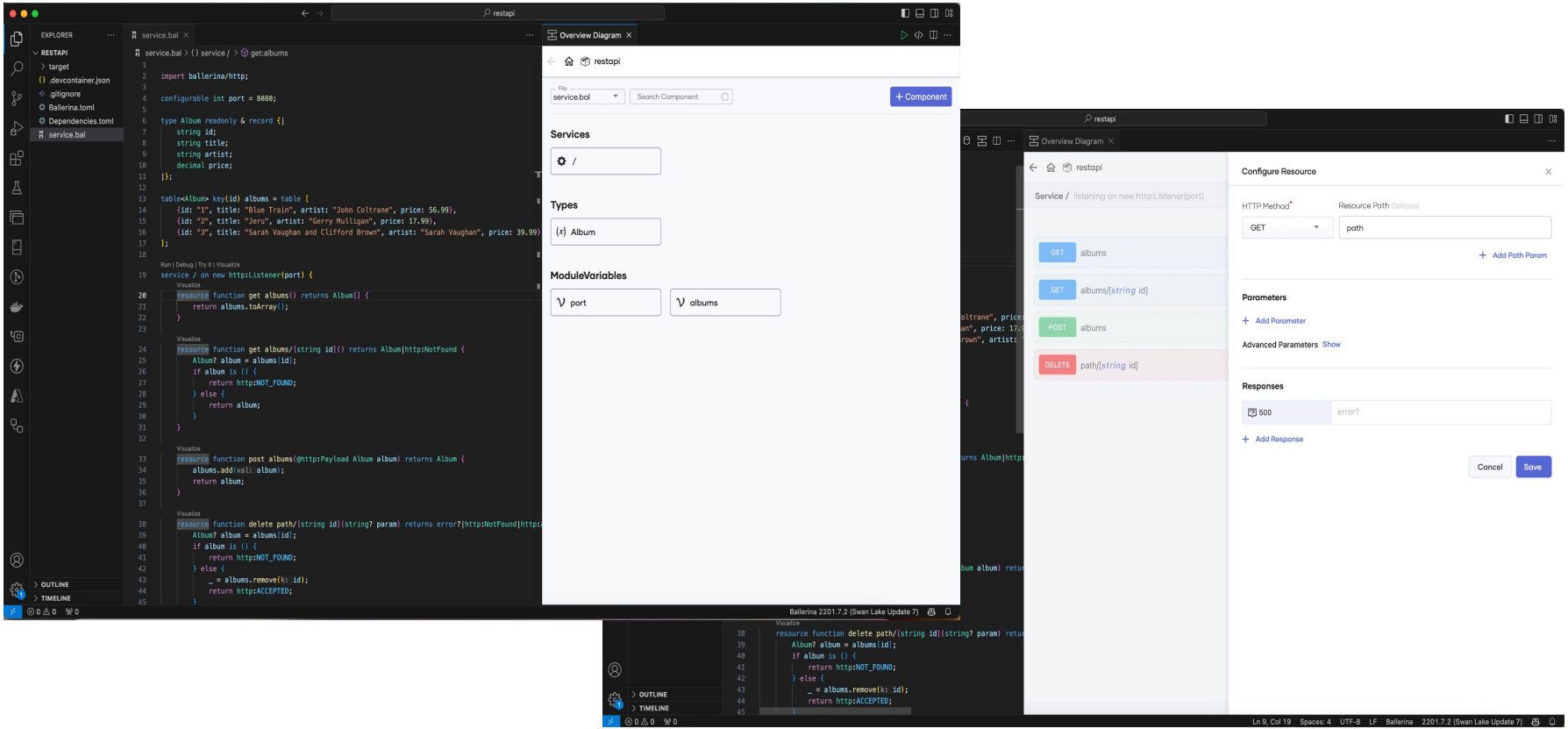
Service designing - REST



Ballerina Swan Lake

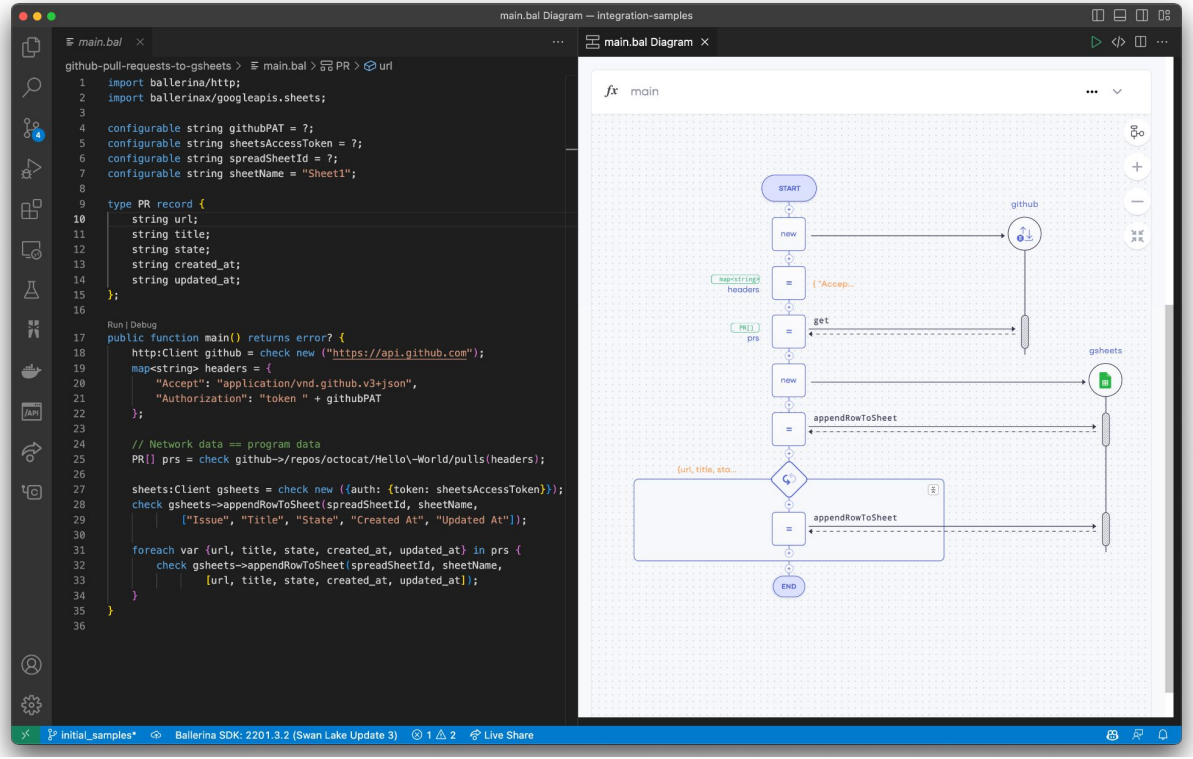
Sample: [REST service]

Service designing - GraphQL



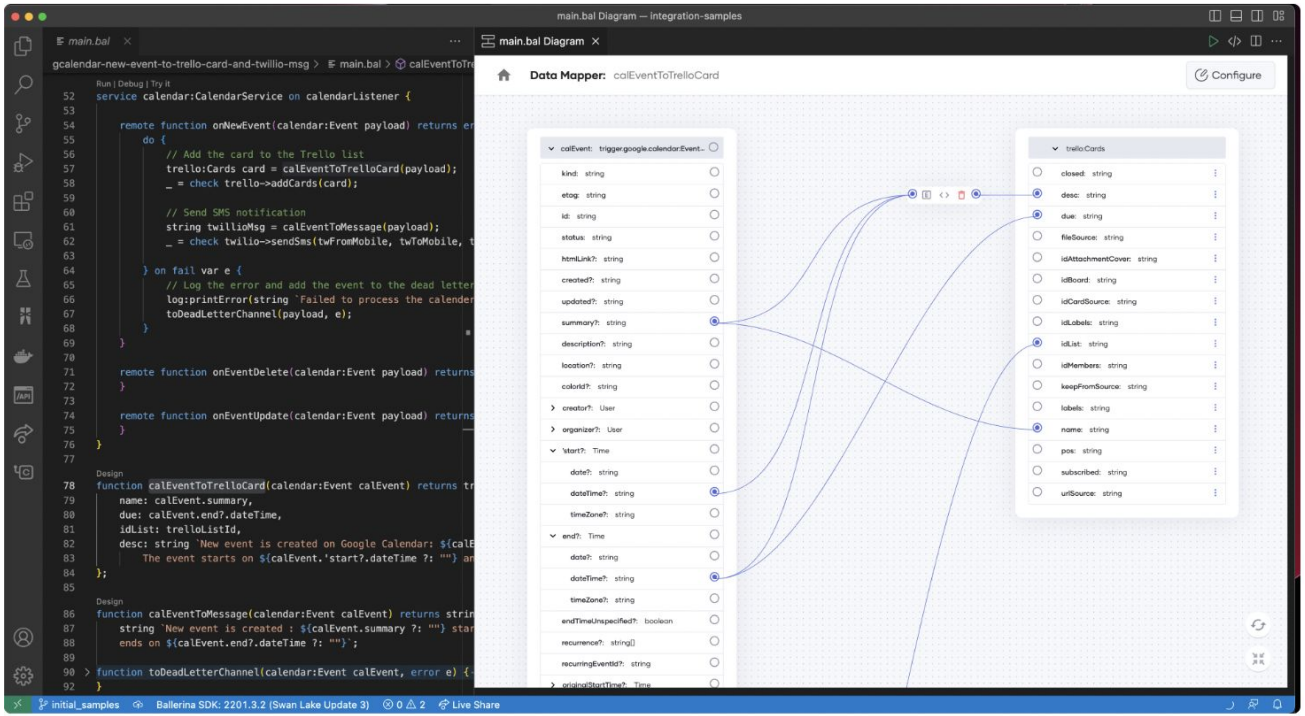
Sample: [GraphQL service]

Text and graphical syntax parity



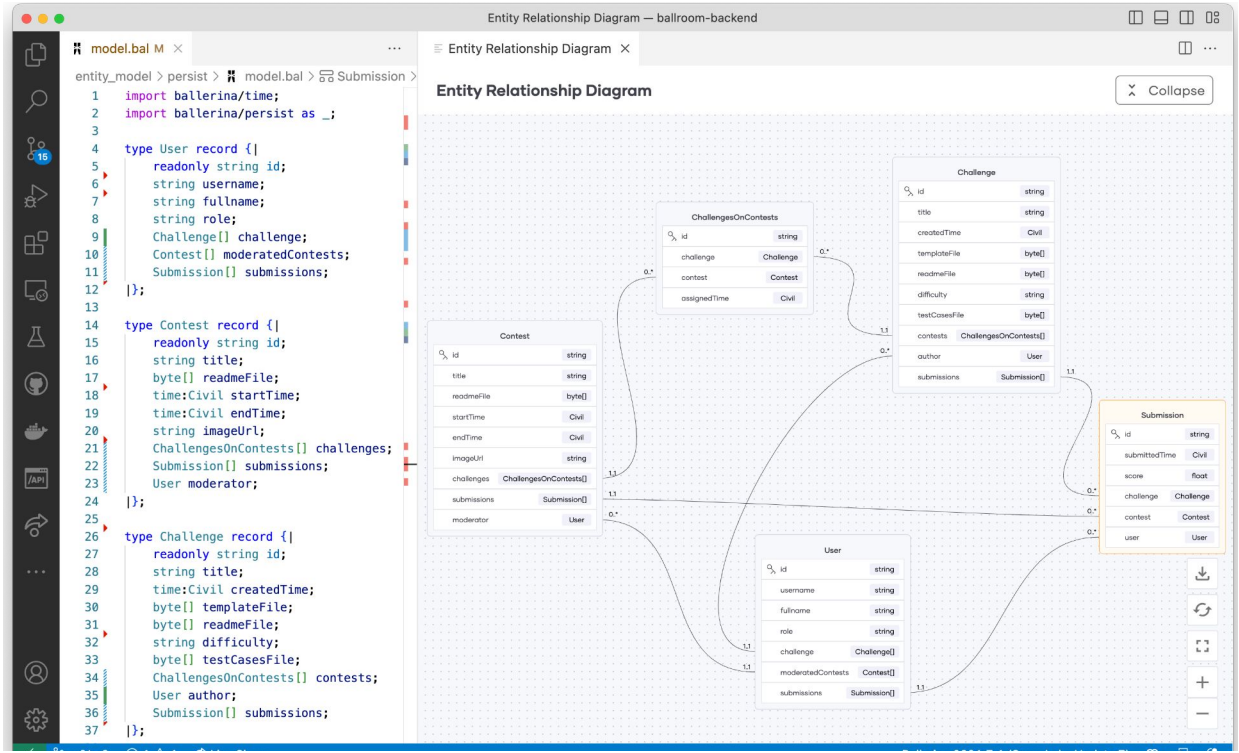
Sample: [Consuming services]

Data transformations



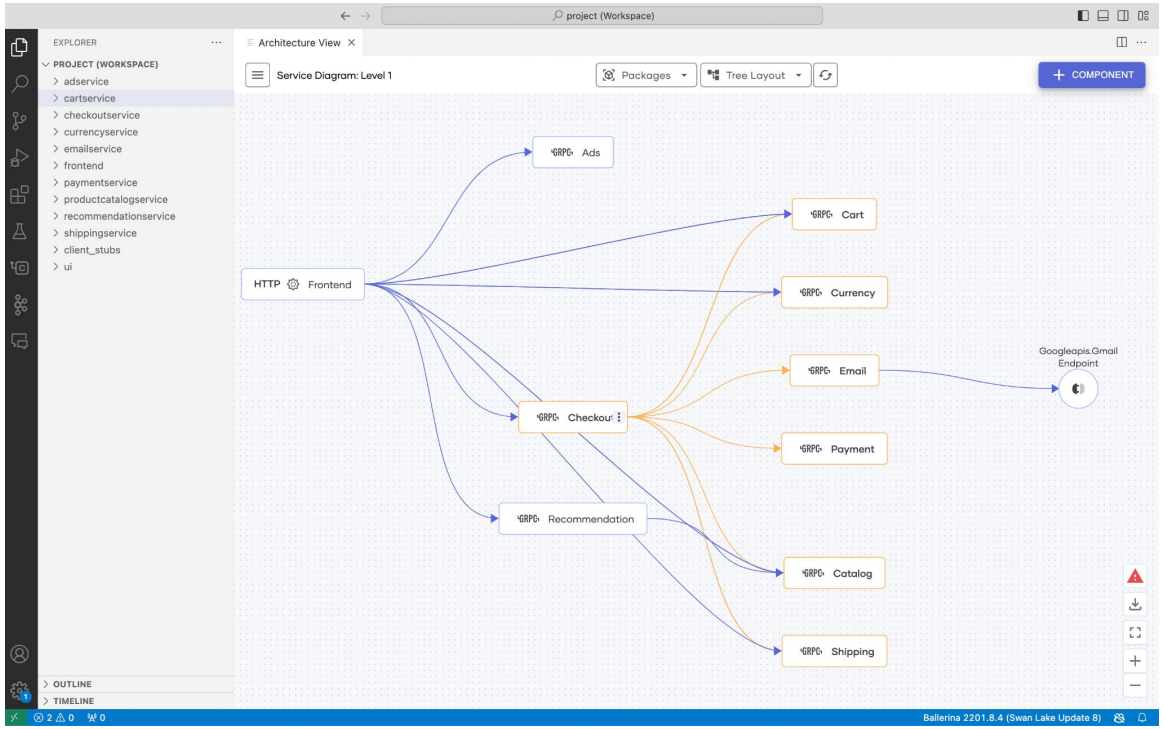
Sample: [Data mapper]

Data persistence



Architectural design view

Services

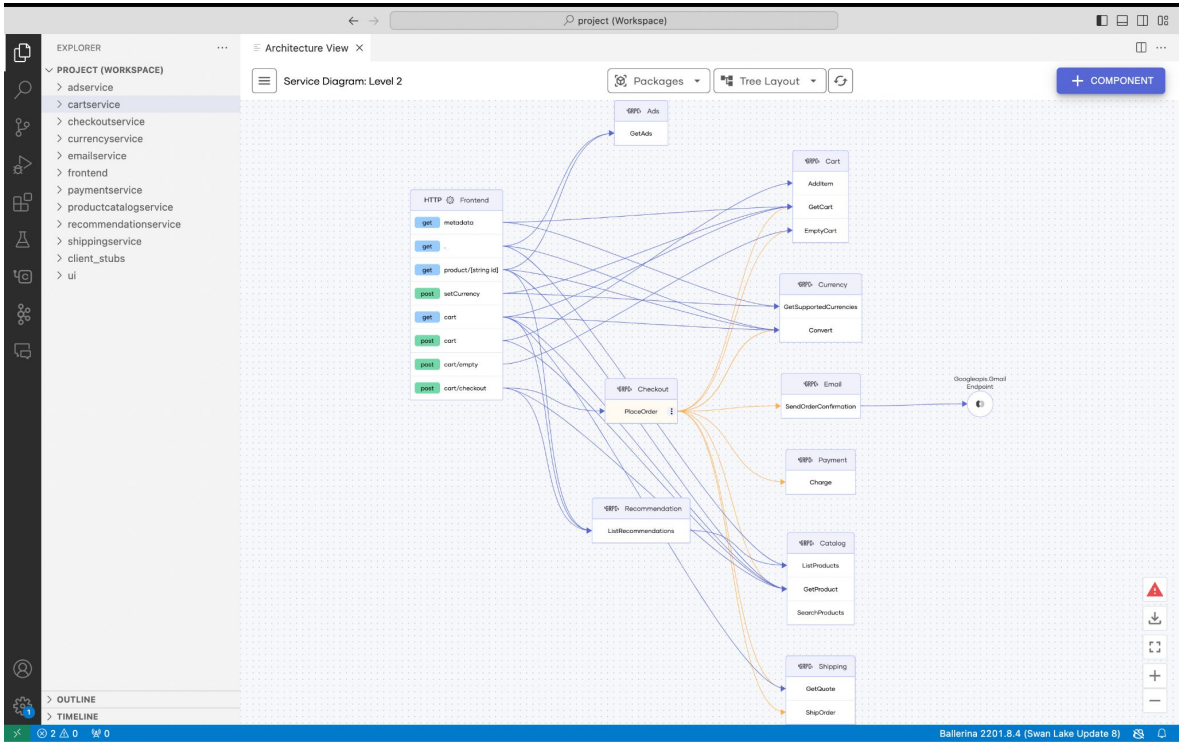


Ballerina Swan Lake

Sample: [gcp-microservices-demo]

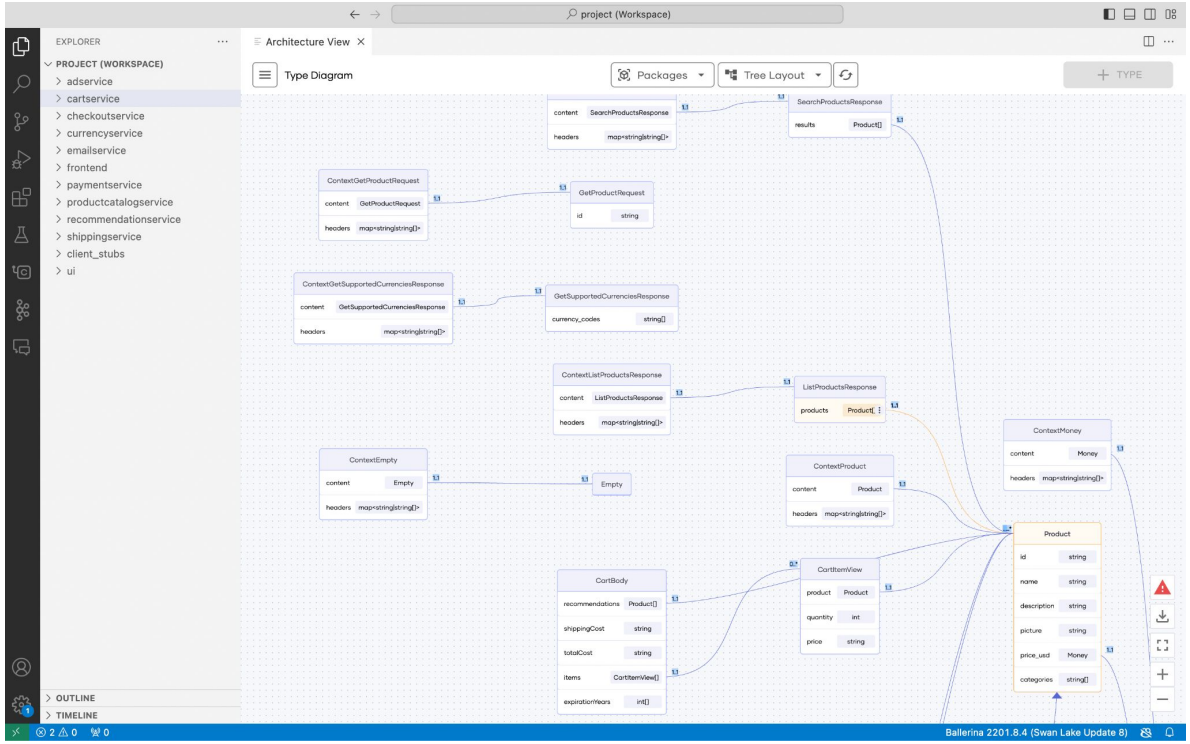
Architectural design view (cont..)

Resources

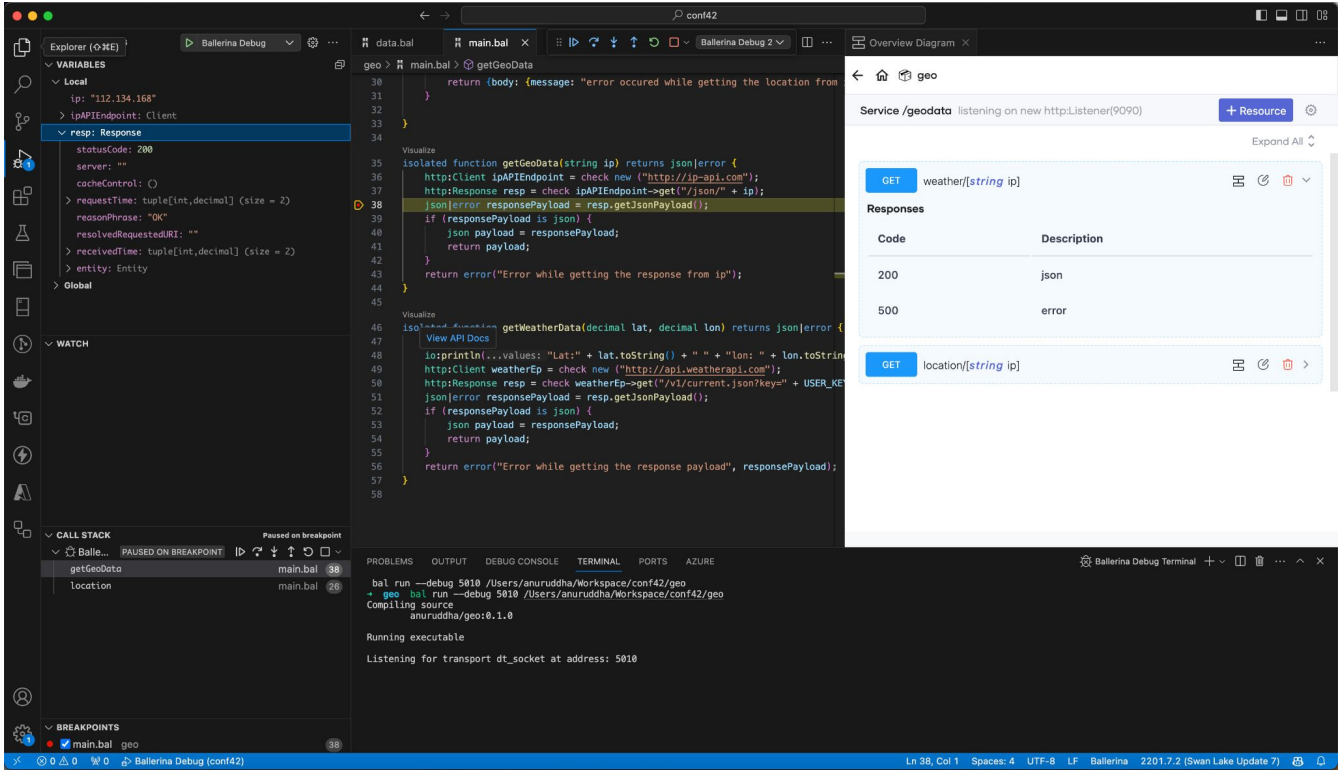


Architectural design view (cont..)

Types

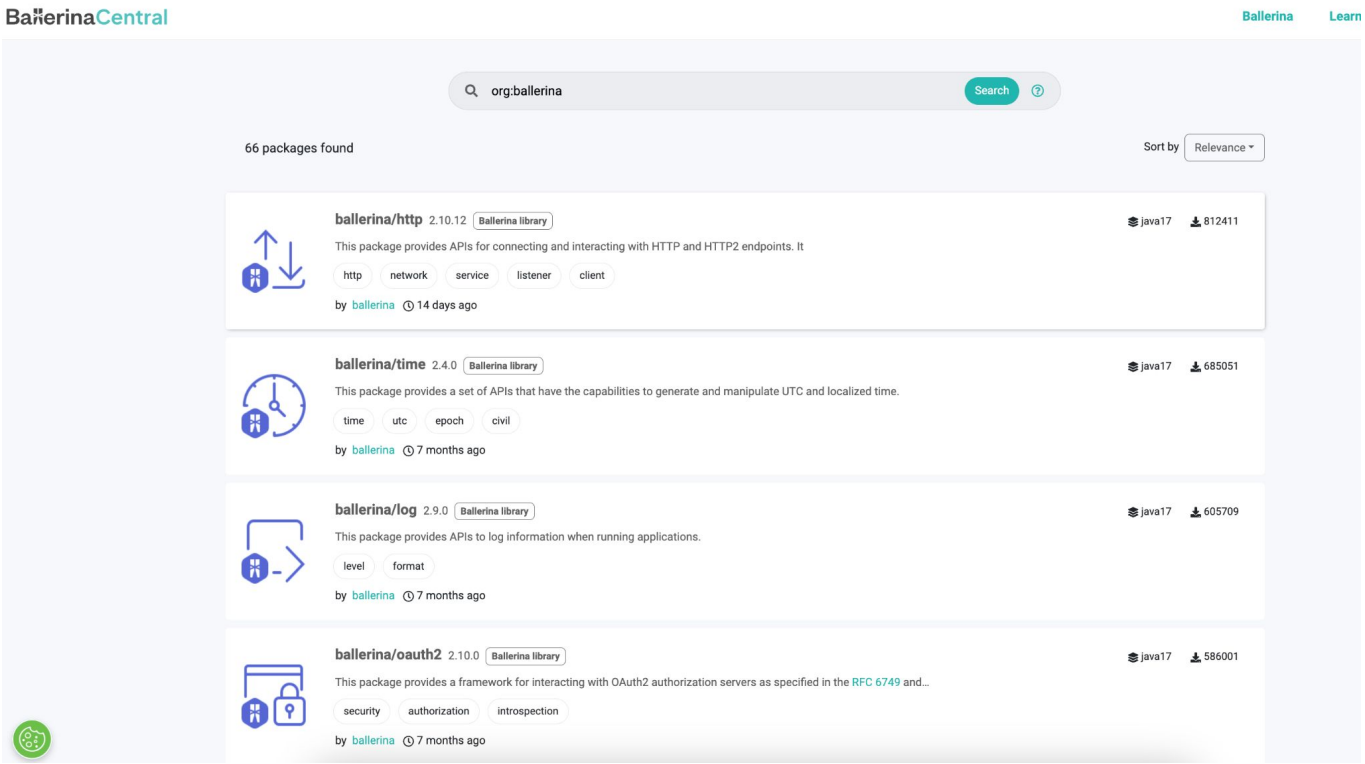


Debug your integration



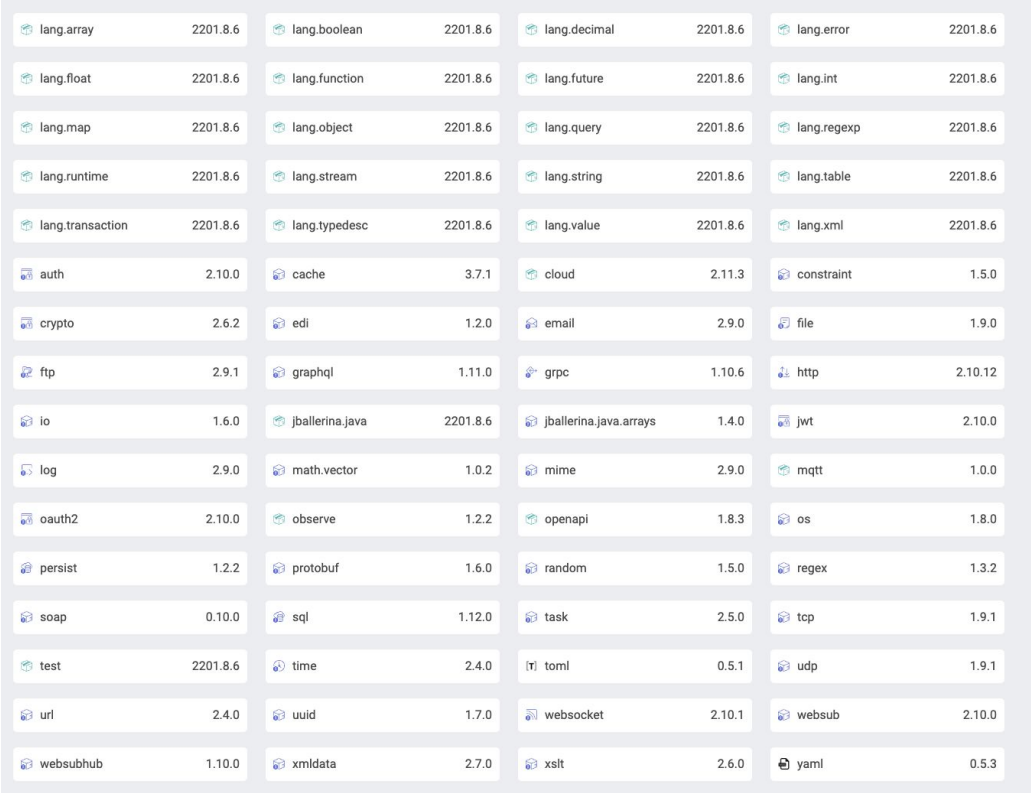
Ballerina
Swan Lake

Ballerina packages in Central



Visit : https://central.ballerina.io/

Ballerina library



lang.array 2201.8.6	lang.boolean 2201.8.6	lang.decimal 2201.8.6	lang.error 2201.8.6
lang.float 2201.8.6	lang.function 2201.8.6	lang.future 2201.8.6	lang.int 2201.8.6
lang.map 2201.8.6	lang.object 2201.8.6	lang.query 2201.8.6	lang.regexp 2201.8.6
lang.runtime 2201.8.6	lang.stream 2201.8.6	lang.string 2201.8.6	lang.table 2201.8.6
lang.transaction 2201.8.6	lang.typedesc 2201.8.6	lang.value 2201.8.6	lang.xml 2201.8.6
auth 2.10.0	cache 3.7.1	cloud 2.11.3	constraint 1.5.0
crypto 2.6.2	edi 1.2.0	email 2.9.0	file 1.9.0
ftp 2.9.1	graphql 1.11.0	grpc 1.10.6	http 2.10.12
io 1.6.0	jballerina.java 2201.8.6	jballerina.java.arrays 1.4.0	jwt 2.10.0
log 2.9.0	math.vector 1.0.2	mime 2.9.0	mqtt 1.0.0
oauth2 2.10.0	observe 1.2.2	openapi 1.8.3	os 1.8.0
persist 1.2.2	protobuf 1.6.0	random 1.5.0	regex 1.3.2
soap 0.10.0	sql 1.12.0	task 2.5.0	tcp 1.9.1
test 2201.8.6	time 2.4.0	toml 0.5.1	udp 1.9.1
url 2.4.0	uuid 1.7.0	websocket 2.10.1	websub 2.10.0
websubhub 1.10.0	xmldata 2.7.0	xslt 2.6.0	yaml 0.5.3

Visit : https://central.ballerina.io/ballerina-library

Ballerina official connectors - 600+ Connectors



Ballerina Swan Lake

Visit : https://central.ballerina.io/ballerina-library

Support for service types

FTP

gRPC

kafka

GraphQL

Websub

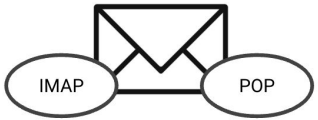
WebSocket

Websubhub

MQTT

TCP

FHIR



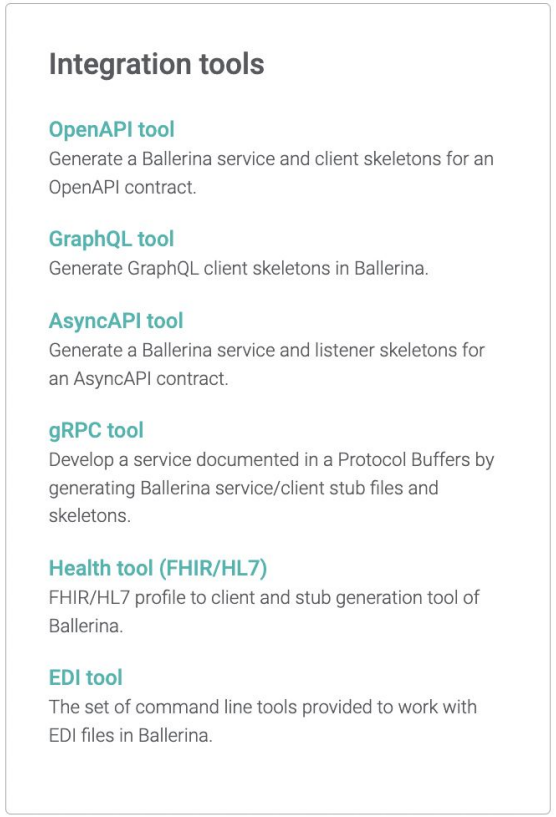
UDP

REST API

FILE

NATS

Integration tools



Support for all Enterprise Integration Patterns

Messaging Systems



Message Channel

Connects one application to another

Message Channel | Point-to-Point Channel | Message



Message

Data transmitted through a message channel

Message | Command Message | Document Message | Message Channel



Pipes And Filters

Divide the processing task into a sequence of smaller, independent processing steps

Pipes and Filters | Message Filter | Message Channel | Message



Message Router

Route messages to different output channels

Message Router | Message Channel | Message Endpoint | Message



Message Translator

Transform one message format to another

Message Translator | Message Endpoint



Message Endpoint

Connects an application to a messaging channel

Message Endpoint | Message Channel

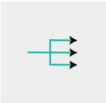
Messaging Channels



Point-to-Point Channel

Ensures that only one receiver will receive a particular message

Point-to-Point Channel | Message Channel | Command Message



Publish-Subscribe Channel

Delivers a copy of a particular event to each receiver

Publish-Subscribe Channel | Message Channel | Message Endpoint | Message Router

Deployment

Build a self-contained executable (.jar)

```
→  datahandle bal build
Compiling source
        anuruddha/datahandle:0.1.0

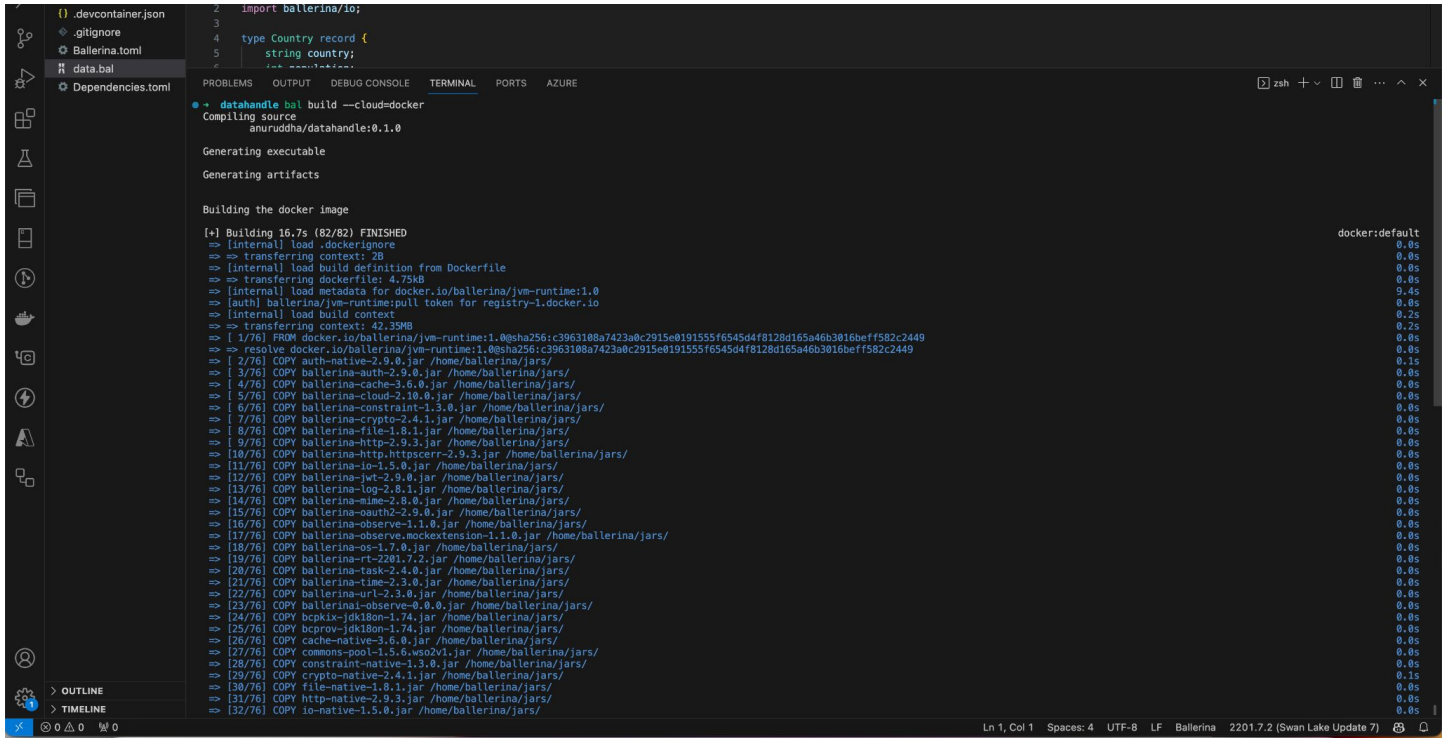
Generating executable
        target/bin/datahandle.jar
```

```
→  geo bal run
Compiling source
        anuruddha/geo:0.1.0

Running executable
```

Build a Docker container

```
bal build --cloud=docker
```



Build Kubernetes artifacts

```
import ballerina/http;

service / on new http:Listener(9090) {

    // This function responds with `string` value `Hello, World!` to HTTP GET requests.
    resource function get greeting() returns string {
        return "Hello, World!";
    }
}
```

```
$ bal build --cloud=k8s
Compiling source
        ballerina/helloworld:0.1.0

Generating executable

Generating artifacts...

        @kubernetes:Service                     - complete 1/1
        @kubernetes:Deployment                  - complete 1/1
        @kubernetes:HPA                         - complete 1/1
        @kubernetes:Docker                      - complete 2/2

        Execute the below command to deploy the Kubernetes artifacts:
        kubectl apply -f /Volumes/data/ballerina/code/testBalProject/target/kubernetes/helloworld

        Execute the below command to access service via NodePort:
        kubectl expose deployment helloworld-deployment --type=NodePort --name=helloworld-svc-local

        target/bin/helloworld.jar
```

```
---
apiVersion: "v1"
kind: "Service"
metadata:
  labels:
    app: "testhello"
  name: "testhello-svc"
spec:
  ports:
  - name: "port-1-testhell"
    port: 9090
    protocol: "TCP"
    targetPort: 9090
  selector:
    app: "testhello"
  type: "ClusterIP"
---
apiVersion: "apps/v1"
kind: "Deployment"
metadata:
  labels:
    app: "testhello"
  name: "testhello-deployment"
spec:
  replicas: 1
  selector:
    matchLabels:
      app: "testhello"
  template:
    metadata:
      labels:
        app: "testhello"
    spec:
      containers:
      - image: "testhello:latest"
        lifecycle:
          preStop:
            exec:
              command:
              - "sleep"
              - "15"
        name: "testhello-deployment"
        ports:
        - containerPort: 9090
          name: "port-1-testhell"
          protocol: "TCP"
```

Build a GraalVM Native Image

```
$ bal build --graalvm

Compiling source
     user/hello_world:0.1.0

================================================================================================
GraalVM Native Image: Generating 'hello_world' (executable)...
================================================================================================
[1/7] Initializing...                                                          (7.3s @ 0.47GB)
 Version info: 'GraalVM 22.3.1 Java 11 CE'
 Java version info: '11.0.18+10-jvmci-22.3-b13'
 C compiler: cc (apple, arm64, 14.0.3)
 Garbage collector: Serial GC
 2 user-specific feature(s)
 - com.oracle.svm.thirdparty.gson.GsonFeature
 - io.ballerina.stdlib.crypto.svm.BouncyCastleFeature
[2/7] Performing analysis...  [***********]                                (116.0s @ 2.63GB)
  24,926 (93.71%) of 26,599 classes reachable
  81,454 (81.08%) of 100,467 fields reachable
 134,363 (72.76%) of 184,660 methods reachable
   1,477 classes,    15 fields, and 2,740 methods registered for reflection
      91 classes,    94 fields, and    66 methods registered for JNI access
       6 native libraries: -framework CoreServices,-framework Foundation,dl,pthread,stdc++,z
[3/7] Building universe...                                                     (12.4s @ 4.55GB)
[4/7] Parsing methods...      [*****]                                          (21.1s @ 3.22GB)
[5/7] Inlining methods...     [***]                                             (7.3s @ 4.51GB)
[6/7] Compiling methods...    [********]                                       (75.3s @ 4.54GB)
[7/7] Creating image...                                                        (9.9s @ 5.48GB)
  87.22MB (58.51%) for code area:    97,270 compilation units
  60.14MB (40.34%) for image heap:  472,434 objects and 32 resources
   1.72MB ( 1.15%) for other data
 149.07MB in total
------------------------------------------------------------------------------------------------
Top 10 packages in code area:         Top 10 object types in image heap:
  15.91MB ballerina.http/2              14.81MB byte[] for code metadata
```

- GraalVM is a high-performance, cloud-native, and polyglot JDK designed to accelerate the execution of applications.
- GraalVM provides the following advantages over the standard JIT compilation making it ideal for container runtimes.
    - Use a fraction of the resources required by the JVM.
    - Applications start in milliseconds.
    - Deliver peak performance immediately with no warmup.
    - Can be packaged into lightweight container images for faster and more efficient deployments.

Function as a service - Azure Functions

Azure Functions

The Azure Functions extension provides the functionality to expose a Ballerina function as a serverless function in the Azure Functions platform.

> **Info:** Azure Functions can be written in Ballerina using the listeners and services provided by the Azure Functions package.

Supported triggers and bindings

An Azure Function consists of a trigger and optional bindings. A trigger defines how a function is invoked. A binding is an approach in which you can declaratively connect other resources to the function.

There are *input* and *output* bindings. An input binding is a source of data that flows into the function. An output binding allows outputting data from the function to an external resource. For more information, go to Azure Functions triggers and bindings concepts.

The following Azure Functions triggers and bindings are currently supported in Ballerina.

Supported triggers	Supported output bindings	Supported input bindings
HTTP	HTTP	-
Queue	Queue	-
Blob	Blob	Blob
-	Twilio SMS	-
Cosmos DB	Cosmos DB	Cosmos DB
Timer	-	-

Function as a service - AWS Lambda

AWS Lambda



The AWS Lambda extension provides the functionality to write AWS Lambda-compatible packages by exposing a Ballerina function as an AWS Lambda function.

> **Info:** Ballerina functions can be deployed in AWS Lambda by annotating a Ballerina function with `@awslambda:Function` adhering to the following function signature:
> `function (awslambda:Context, json|EventType) returns json|error`

Supported triggers

An AWS Lambda function can be triggered by various AWS services. You can find the list of supported notification types below.

- Direct invocation
- Simple Queue Service (SQS)
- Simple Storage Service (S3)
- DynamoDB
- Simple Email Service (SES)
- API Gateway

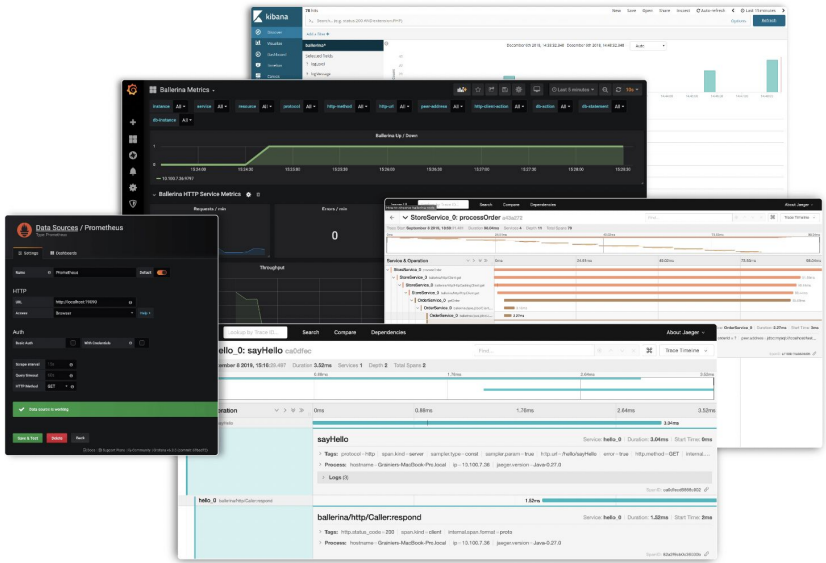
Deploy on Choreo

Choreo is an internal developer platform as a service that helps you harness the power of architectural best practices of CI/CD, DevOps, serverless, Kubernetes, eBPF, API management, marketplaces, domain-driven design, microservices architecture, and more.

Visit : https://wso2.com/choreo/

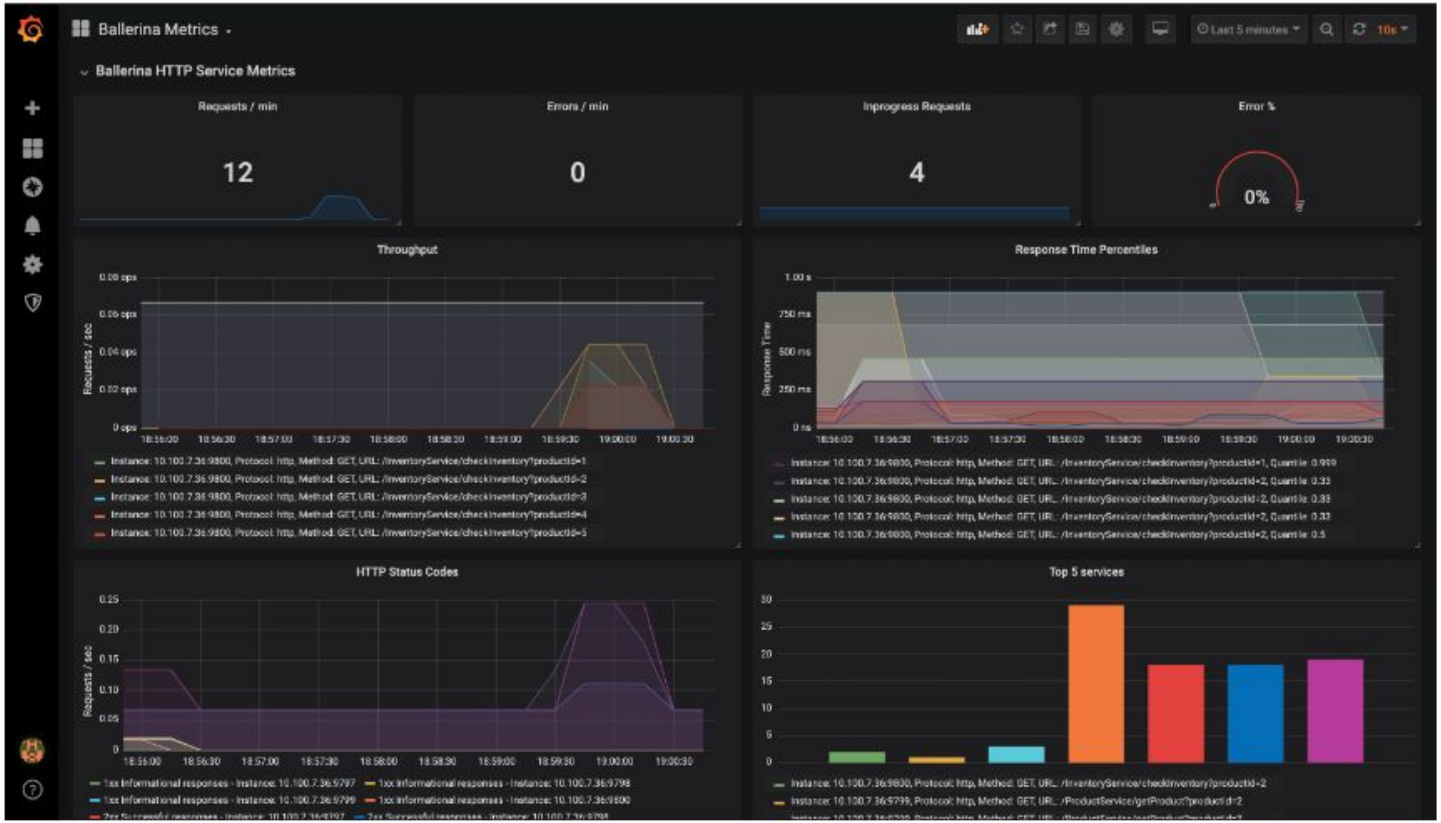
Observability

Observability in Ballerina

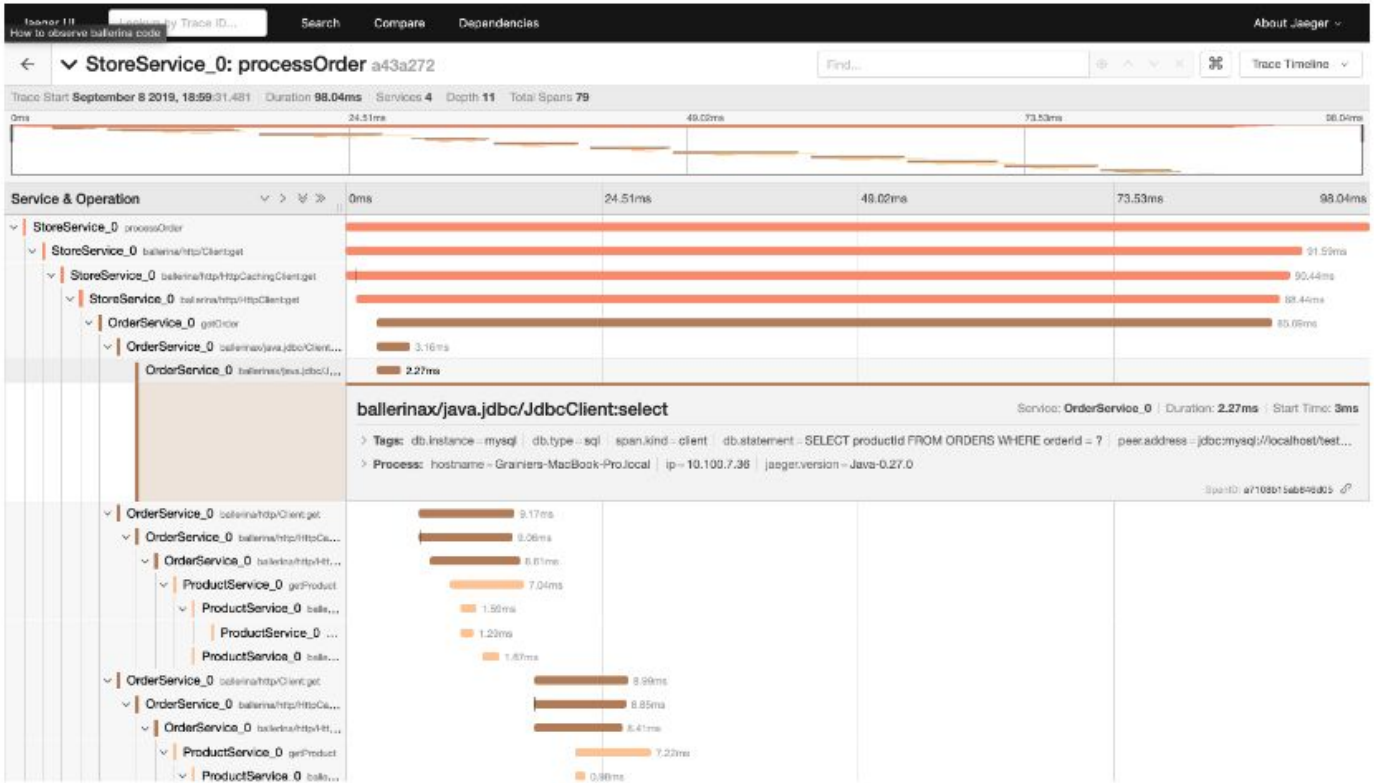


- Every Ballerina program is automatically observable by any Open Telemetry tool.
- Gives the complete control and visibility into the code's behavior and performance.
- It has 3 main pillars:
  - Metrics - Prometheus, Grafana
  - Tracing - Jaeger, Zipkin
  - Logging - Elastic Stack

Ballerina Metrics with Grafana



Ballerina Tracing with Jaeger



Distributed Logging

Distributed logging

In Ballerina, distributed logging and analysis are supported by the Elastic Stack. Ballerina has a log module for logging into the console. To monitor the logs, the Ballerina standard output needs to be redirected to a file.

This can be done by running the Ballerina service as below.

```
$ nohup bal run hello_world_service.bal > ballerina.log &
```

You can view the logs with the command below.

```
$ tail -f ~/wso2-ballerina/workspace/ballerina.log
```

Set up the external systems for log analytics

Set up Elastic Stack

The Elastic Stack comprises the following components.

1. Beats - Multiple agents that ship data to Logstash or Elasticsearch. In our context, Filebeat will ship the Ballerina logs to Logstash. Filebeat should be a container running on the same host as the Ballerina service. This is so that the log file (ballerina.log) can be mounted to the Filebeat container.
2. Logstash - Used to process and structure the log files received from Filebeat and send them to Elasticsearch.
3. Elasticsearch - Storage and indexing of the logs sent by Logstash.
4. Kibana - Visualizes the data stored in Elasticsearch.

Ballerina Advantage

Ballerina offers not just the language, but the full platform

- VSCode plugin
  - Source and graphical editing
  - Debugging
- Tools for working with OpenAPI, GraphQL, gRPC schemas, etc.
- Generate API Documentation & test framework
- Ballerina standard library and extended library
- Ballerina Central (https://central.ballerina.io/)
  - Module sharing platform
- Integration to Choreo by WSO2 for observability, CI/CD and DevOps

Ballerina Advantage

- One language for the entire integration with intuitive syntax.
  - E.g., Ballerina knows JSON and XML
- Better error handling
- AI powered code suggestions (E.g., CoPilot)
- Better debugging support
- Better concurrency
- Better testing
- Better code organization and reusability
- UI support for data mapping, REST API designing, record generation, etc.
- Better visualization with sequence diagrams, architecture diagrams, etc.

All of these result in

- Shorter development time
- Better readability and maintainability

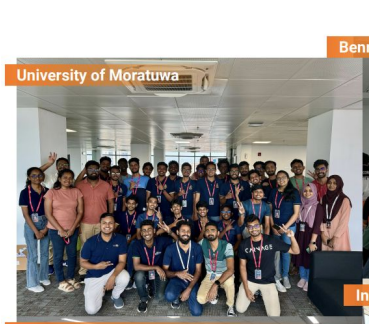
Ballerina Community





Ballerina
Swan Lake

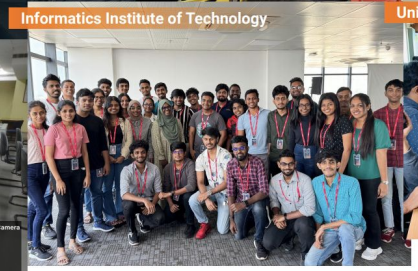
Visit : https://ballerina.io/community/





















Ballerina
Swan Lake

Visit : https://ballerina.io/community/

Find out more…

- Learn Ballerina
  - Learn pages
  - Ballerina by example
  - Ballerina VS Code extension
  - Ballerina training video series
  - Ballerina certification

- Join the Ballerina community



Discord
ballerinalang

stackoverflow
Tag : ballerina

@ballerinalang

GitHub
ballerina-lang

Provide Feedback



Session 2: Designing and implementing REST APIs for integration



Preaching and teaching integration with graphical languages

Bakerina

Session 2 - Objective

- The goal is to teach integration centric Ballerina features. Once completed, you will be able to get a look and feel of Ballerina and performing integration with Ballerina.

Session 2 - Overview

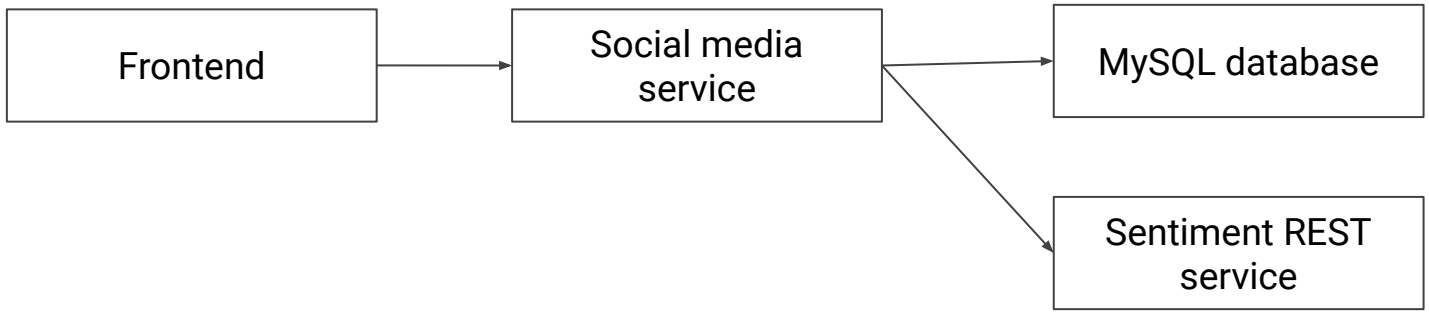
- Level - 1
  - Writing a simple REST API
- Level - 2
  - Accessing database
  - Debugging in VS Code
  - Externalizing configurations
- Level - 3
  - Using HTTP client
- Level - 4
  - Writing tests

Session 2 - Overview

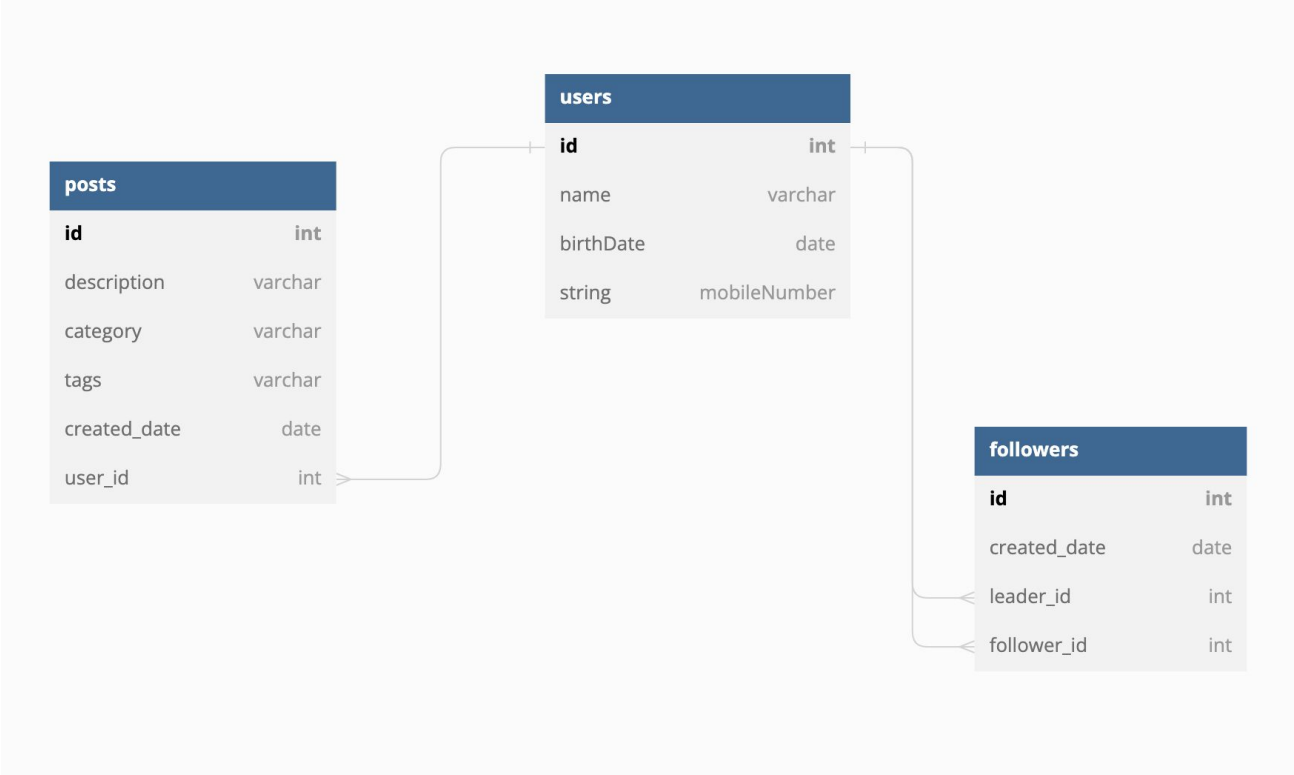
- Level - 5
  - Generating OpenAPI and client stubs
- Level - 6
  - Deploying in docker and kubernetes
  - Generating GraalVM images
  - Observability

Social media REST service

Social media REST service



Entity relationship diagram



A bit about Ballerina type system

Structural type system

```
string|int foo = 1;
```

```
string|int|error foo = 1;
```

```
string|int foo = check someFunction();
```

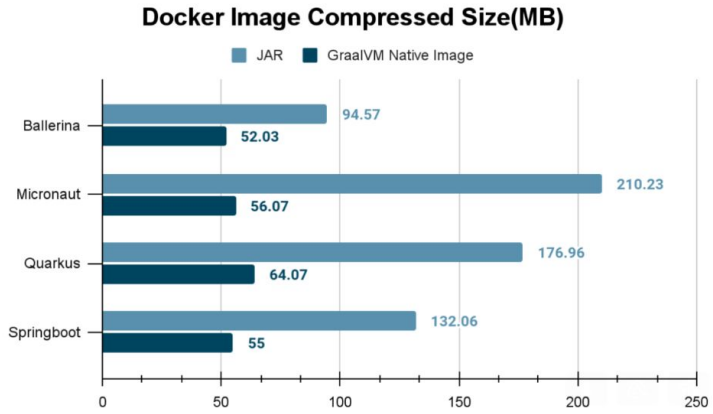
Build GraalVM Native Image

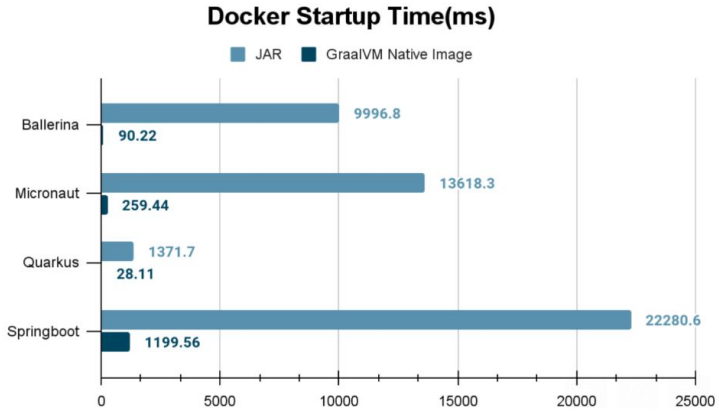
```
$ bal build --graalvm

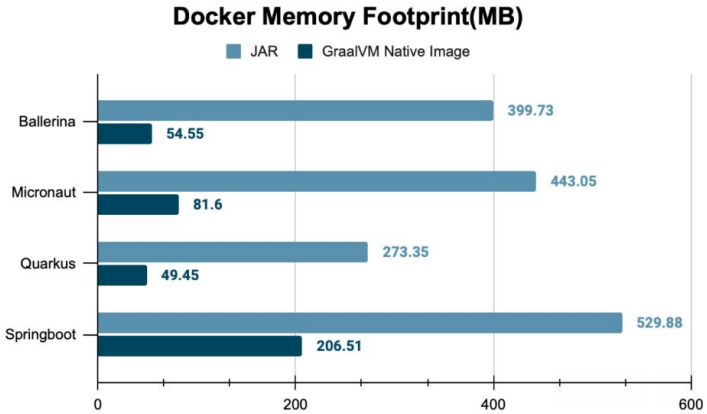
Compiling source
     user/hello_world:0.1.0

================================================================================================
GraalVM Native Image: Generating 'hello_world' (executable)...
================================================================================================
[1/7] Initializing...                                                          (7.3s @ 0.47GB)
 Version info: 'GraalVM 22.3.1 Java 11 CE'
 Java version info: '11.0.18+10-jvmci-22.3-b13'
 C compiler: cc (apple, arm64, 14.0.3)
 Garbage collector: Serial GC
 2 user-specific feature(s)
 - com.oracle.svm.thirdparty.gson.GsonFeature
 - io.ballerina.stdlib.crypto.svm.BouncyCastleFeature
[2/7] Performing analysis...  [***********]                                  (116.0s @ 2.63GB)
  24,926 (93.71%) of 26,599 classes reachable
  81,454 (81.08%) of 100,467 fields reachable
 134,363 (72.76%) of 184,660 methods reachable
   1,477 classes,    15 fields, and 2,740 methods registered for reflection
      91 classes,    94 fields, and    66 methods registered for JNI access
       6 native libraries: -framework CoreServices,-framework Foundation,dl,pthread,stdc++,z
[3/7] Building universe...                                                     (12.4s @ 4.55GB)
[4/7] Parsing methods...      [*****]                                          (21.1s @ 3.22GB)
[5/7] Inlining methods...     [***]                                             (7.3s @ 4.51GB)
[6/7] Compiling methods...    [********]                                       (75.3s @ 4.54GB)
[7/7] Creating image...                                                        (9.9s @ 5.48GB)
  87.22MB (58.51%) for code area:    97,270 compilation units
  60.14MB (40.34%) for image heap:  472,434 objects and 32 resources
   1.72MB ( 1.15%) for other data
 149.07MB in total
------------------------------------------------------------------------------------------------
Top 10 packages in code area:         Top 10 object types in image heap:
  15.91MB ballerina.http/2              14.81MB byte[] for code metadata
```

Docker + GraalVM







Session 2 - Wrap up!

- Introduction to Integration with Ballerina

The limits of language are the limits of one’s world - Ludwig Wittgenstein

Provide Feedback



Session 3: Designing and implementing GraphQL services, connectors and package management, testing and observability

Session 3 - Overview

- GraphQL with Ballerina
- Testing Ballerina code
- Ballerina connectors and package management
- Observability with Ballerina

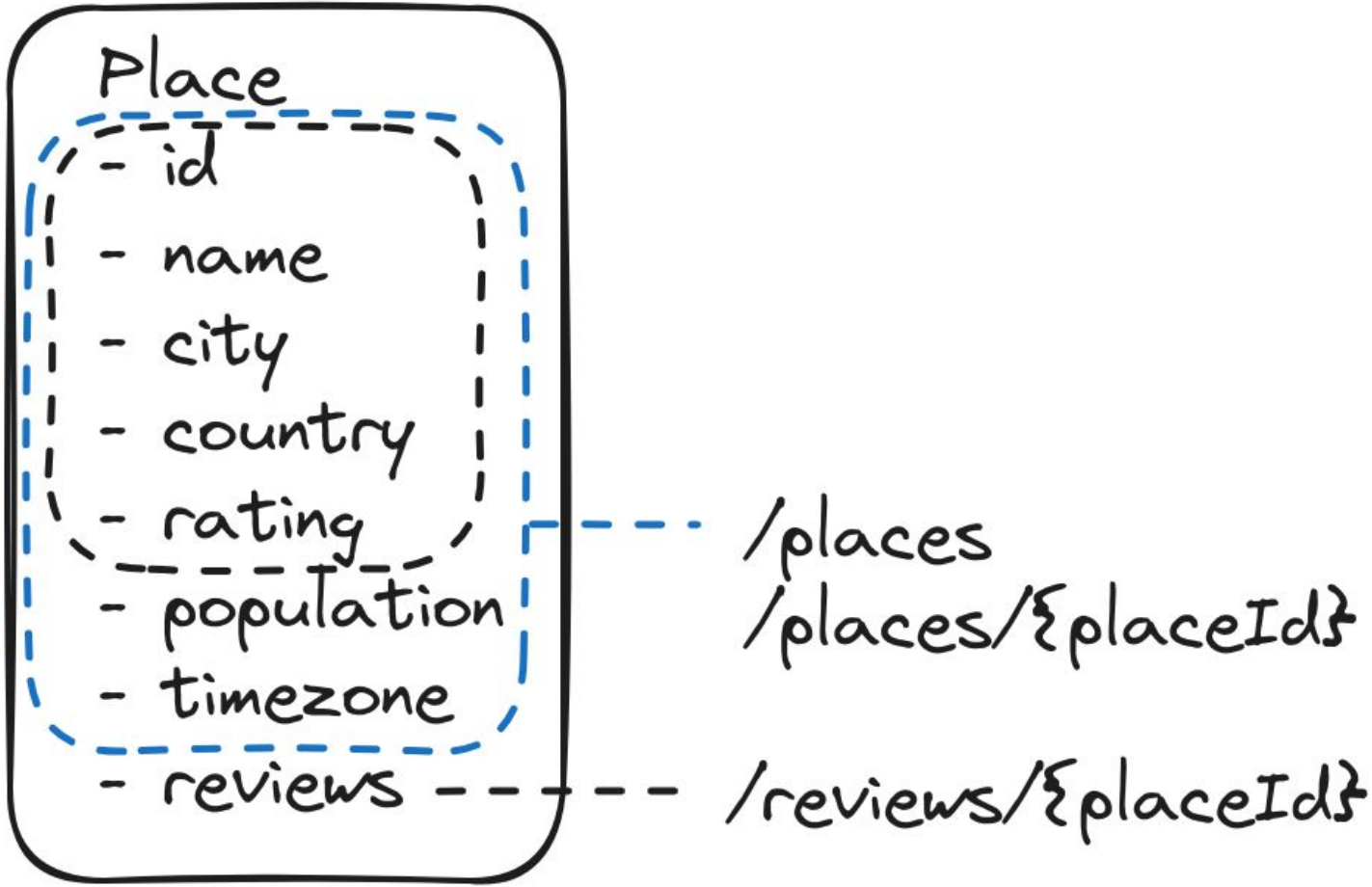
Using Ballerina for GraphQL

- One of the most commonly used features of Ballerina.
- Including
  - Avinya Foundation
  - WSO2 Internal Apps
  - Ballerina Central

    and more customers…

GraphQL

- What is GraphQL?
  - A query language for APIs and
  - A runtime engine to resolve data requested by the queries
- Comprehensive description of data in terms of types and fields defined in a schema
  - Uses the GraphQL schema definition language (SDL)
- Solves two main problems with REST APIs
  - Overfetching
  - Underfetching
- Ask for exactly (and all) what you need, and get exactly what you asked for.


Place
- id
- name
- city
- country
- rating
- population
- timezone
- reviews
/places
/places/{placeId}
/reviews/{placeId}

GraphQL is Strongly Typed

- The schema is like a contract between the server and the client.
  - Defined as object types with fields
  - Each field’s type can be a scalar, enum, or another object type
- Defines what data can be requested and how data can be requested.
- Client has more control over what data is returned.
- More flexibility on the server side to add new fields, deprecate fields, etc.
- Nullability, interfaces, unions, etc. are part of the schema.
- Better developer tools with client-side validation.
- A good match with Ballerina’s type system!

GraphQL Operations

- Queries - (read-only operations) to fetch data
- Mutations - to update data
- Subscriptions - to get real time updates (like queries, but continuous)

GraphQL Schema and Query Sample

```
type Query {
   place(placeId: ID!): Place!
}

type Place {
   id: ID!
   name: String!
   city: String!
   country: String!
   population: Int
   timezone: String
   reviews: [Review!]
   rating: Float
}

type Review {
   id: ID!
   rating: Int!
   content: String!
   place: Place
   author: Author
}

type Author {
   id: ID!
   username: String!
   reviews: [Review!]
}
```

```
{
   place(placeId: 8000) {
       name
       city
       timezone
       reviews {
           rating
       }
   }
}
```

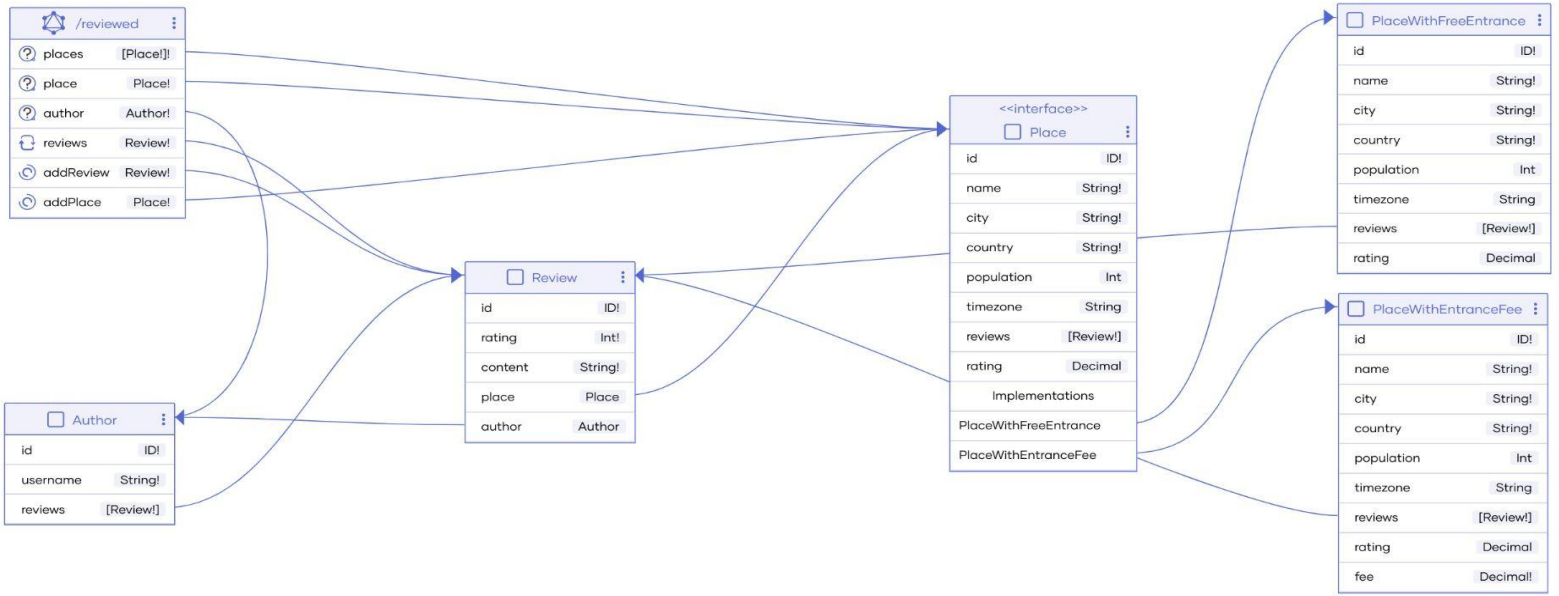
```
{
   "data": {
       "place": {
           "name": "Tower Vista",
           "city": "Colombo",
           "timezone": "Asia/Colombo",
           "reviews": [
               {
                   "rating": 5
               },
               {
                   "rating": 4
               },
               {
                   "rating": 5
               }
           ]
       }
   }
}
```

Ballerina
Swan Lake

Approaches to Development

- Two different approaches.
  - Schema-first
  - Code-first - the approach we will be taking today
- Both have their pros and cons.
- Ballerina supports both approaches.

Implementing `Reviewed!`



Ballerina

Testing Ballerina Code

Testing Ballerina Code

- Tests can be introduced with Ballerina's built-in test framework.
- Support for writing tests for and with clients and services.
  - https://ballerina.io/learn/test-ballerina-code/test-services-and-clients/
- Extensive support for mocking that also enables mocking remote calls.
  - https://ballerina.io/learn/test-ballerina-code/mocking/
- Built-in support for code coverage and test reports.
  - https://ballerina.io/learn/test-ballerina-code/code-coverage-and-reporting/
  - `bal test --test-report --code-coverage`
- And more as detailed on Ballerina learn pages

Connectors

Developing connectors and sharing packages via Ballerina Central

- A connector is a Ballerina package that contains a client.
- Connectors, and other packages, that need to be shared can be published to Ballerina central for use by others.
  - https://ballerina.io/learn/publish-packages-to-ballerina-central/
  - Build the package in distribution format.
    - `bal pack`
  - Push to
    - The local repository for testing - `bal push --repository local`
    - Ballerina Central - `bal push`
      - Obtaining an access token - https://ballerina.io/learn/publish-packages-to-ballerina-central/#obtain-an-access-token

Private Packages

- Also possible to make a package visible to only those belonging to a particular organization.
  - https://ballerina.io/learn/publish-packages-to-ballerina-central/#define-the-organization
  - Set the visibility to private
    - https://ballerina.io/learn/package-references/#the-visibility-field

Observability

Observing Ballerina Code

- Observability is especially important with distributed systems comprised of many microservices.
- Ballerina has built-in support for observability via any Open Telemetry tool.
- Ballerina supports all three pillars of observability:
  - Metrics - Prometheus, Grafana
  - Tracing - Jaeger, Zipkin
  - Logging - Elastic Stack

Provide Feedback



Session 4: Data Integration With Ballerina

Session 4 - Overview

- Universe of Ballerina Values and Plain Data
- Tools for Working with Data
  - Data Modules
  - Bal Persist Tool
  - Data Mapping
  - EDI Tool
- Data-Oriented Programming Principles
- Bindgen Tool

Languages and Programing Paradigms

Programing Paradigms

Object-Oriented Programming (OOP): Objects encapsulate data and behaviors.

Functional Programming: Treats everything as functions; emphasizes immutability.

Data-Oriented Programming: Focuses on efficient data manipulation.

Principles in Data-Oriented Programming

- Separating code (behavior) from data.
- Representing data with generic data structures.
- Treating data as immutable.
- Separating data schema from data representation.

Programing Paradigms (cont.)

Object-Oriented Programming (OOP):

- Java
- C++

Functional Programming:

- Haskell
- Erlang

Data-Oriented Programming (DOP):

- Clojure
- Ballerina

Most modern languages support many, many paradigms. Java21, JS and …

Programing Paradigms (cont.)

Object-Oriented Programming (OOP):

- ✅ Classes, Methods, Getters, and Setters
- ❌ Not Preferred: Heavy reliance on static methods and global state

Functional Programming:

- ✅ Functions, Immutability, Pure functions, Higher-order functions
- ❌ Not Preferred: Mutable state, Classes and traditional OOP methods, Closures with mutable data

Data-Oriented Programming (DOP):

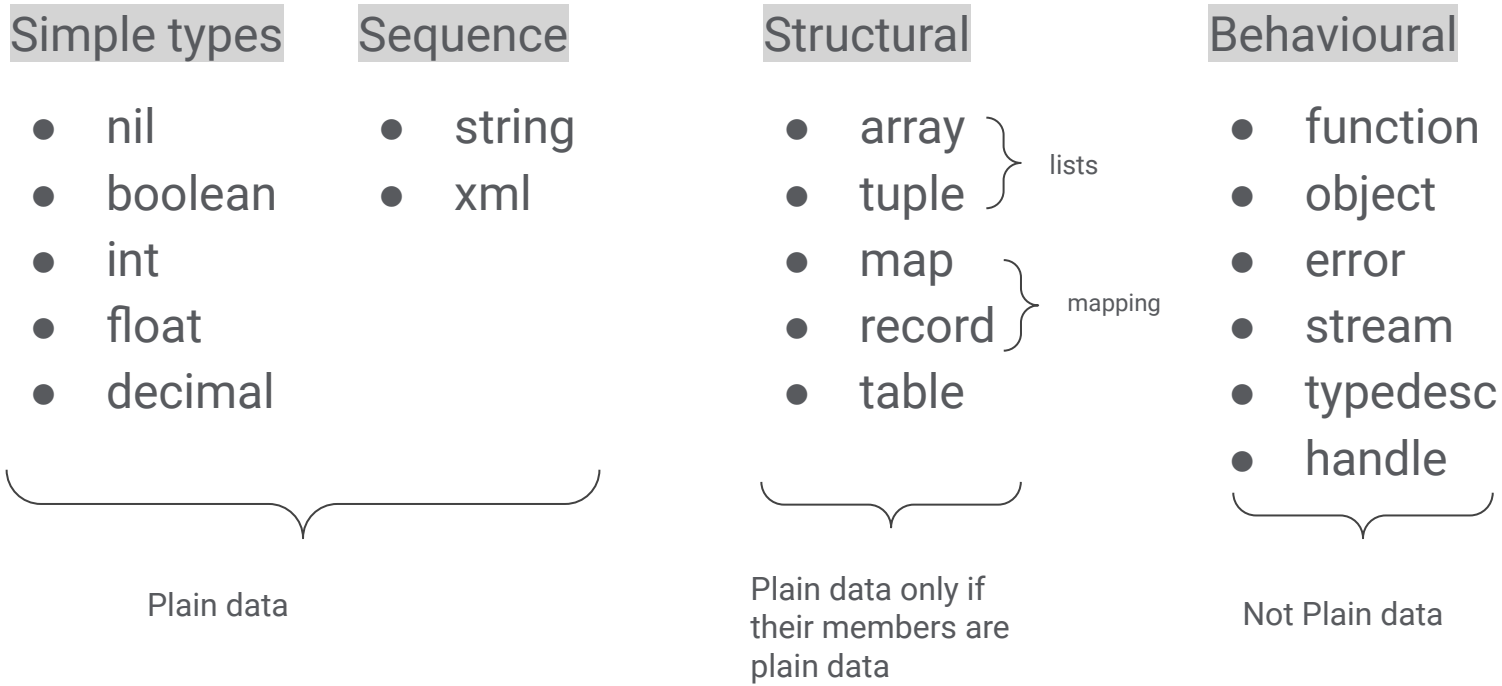
- ✅ Generic Data Structures: Records, Maps (simple objects without behavior) and List, Immutability, Pure functions
- ❌ Not Preferred: Classes and traditional OOP methods

Plain Data

Network Data

- Serializable and deserializable
  - Exchanged through network interfaces between programs.
- Representable in shapes.
- No behaviour (Pure Data).
- In Ballerina, this is known as plain data.

Ballerina Values



anydata - Type of Plain data
any - any value except for error

json - JSON compatible values

Plain Data

Example:

Git Repository :

Bal Persist

Bal Persist Tool

- ORM - (Object-Relational Mapping) Solution.
- Generates a Connector
  - Don't requires SQL queries to perform CRUD operations
  - Provides restful API to interact with data store
- Data Store
  - Inmemory
  - mysql
  - mssql
  - postgresql
  - googlesheets
  - redis

Data Formats

Data Formats

- JSON - Built-into the language
- XML - Built-into the language
- CSV - Library support
- TOML - Library support
- YAML - Library support
- EDI - Library and Tooling support
- Copybook - Library support
- HealthCare (HL7) - Library support
- Avro - Library support

Many more...

Data Libraries

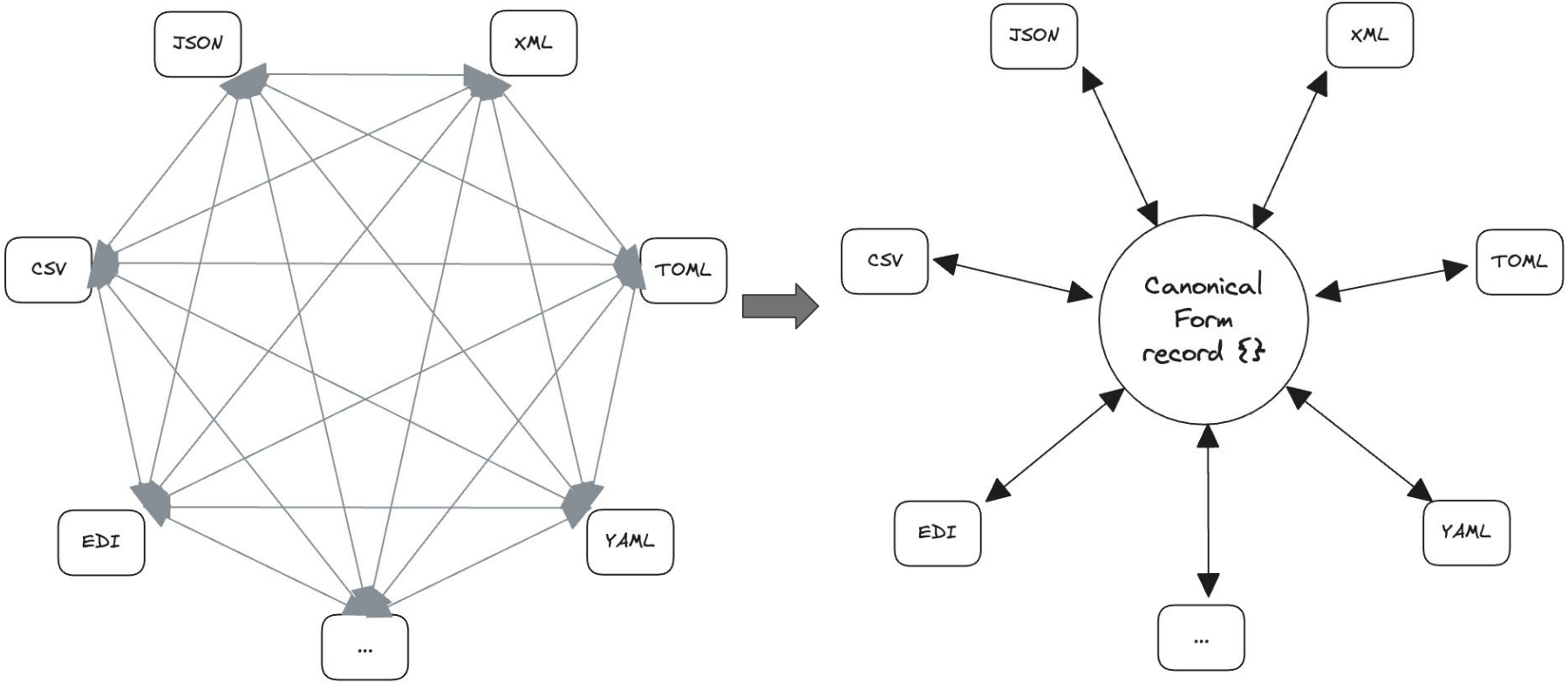
- Simplifies how you can bring data into your system.
- Data sources
  - String
  - Byte Array
  - Byte Stream
  - ballerina value
- Canonical data representation - `record{}` / `record{}[]`
- `ballerina/data.*`
  - `ballerina/data.xmldata`
  - `ballerina/data.jsondata`
  - `ballerina/data.csv` - (Ballerina SwanLake Update 10)
  - `ballerina/data.yaml` - (Ballerina SwanLake Update 10)

Data Libraries - Features

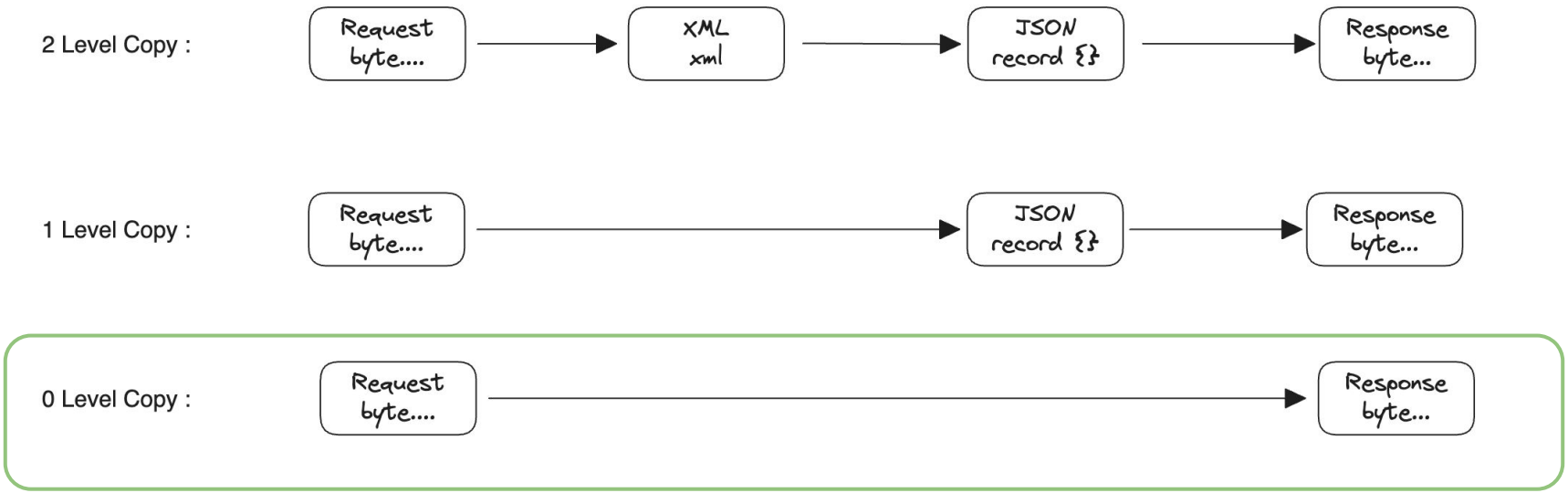
- Data projection
- JSON-Path Support
- JSON pretty-print
- Convenient XML conversion
    - Handling attributes, text nodes etc.

- Future
    - XML Path
    - CSV support
    - YAML support

Data Conversion



XML → JSON Conversion



Future

EDI Tool

EDI Tool

- Allows B2B Communication
- Electronic Data Interchange formats
  - X12
  - EDIFACT
- Wide Range of Applications and Domain
- EDI Tool generates APIs
  - To Convert EDI messages to Ballerina Values and vice versa
  - Represent EDI messages in Plain Data

Modernizing Legacy Systems Using Ballerina



Data-Oriented Programing

Principles in Data-Oriented Programming

- Separating code (behavior) from data.
- Representing data with generic data structures.
- Treating data as immutable.
- Separating data schema from data representation.

DOP and Ballerina

- Separating code (behavior) from data.
  - ✅ Mapping and List Values for Data
  - ✅ Functions for Code (Behaviour)
- Representing data with generic data structures.
  - ✅ Records, Maps, Arrays and Tuples are generic data structures
- Treating data as immutable.
  - ✅ `readonly` helps you to create and immutable values.
- Separating data schema from data representation.
  - ✅ Ballerina's structural type system

DOP and Ballerina

- ✅ Using algebraic data type
  - Product Types: Mappings, List
  - Sum Types: Union
- ✅ Data validation at boundary
- ✅ Pattern matching
- ✅ Data destructuring

Bindgen Tool

Find out more…

- Learn Ballerina
  - Learn pages
  - Ballerina by example
  - Ballerina VS Code extension
  - Ballerina training video series
  - Ballerina certification

- Join the Ballerina community



ballerinalang



Tag : ballerina



@ballerinalang

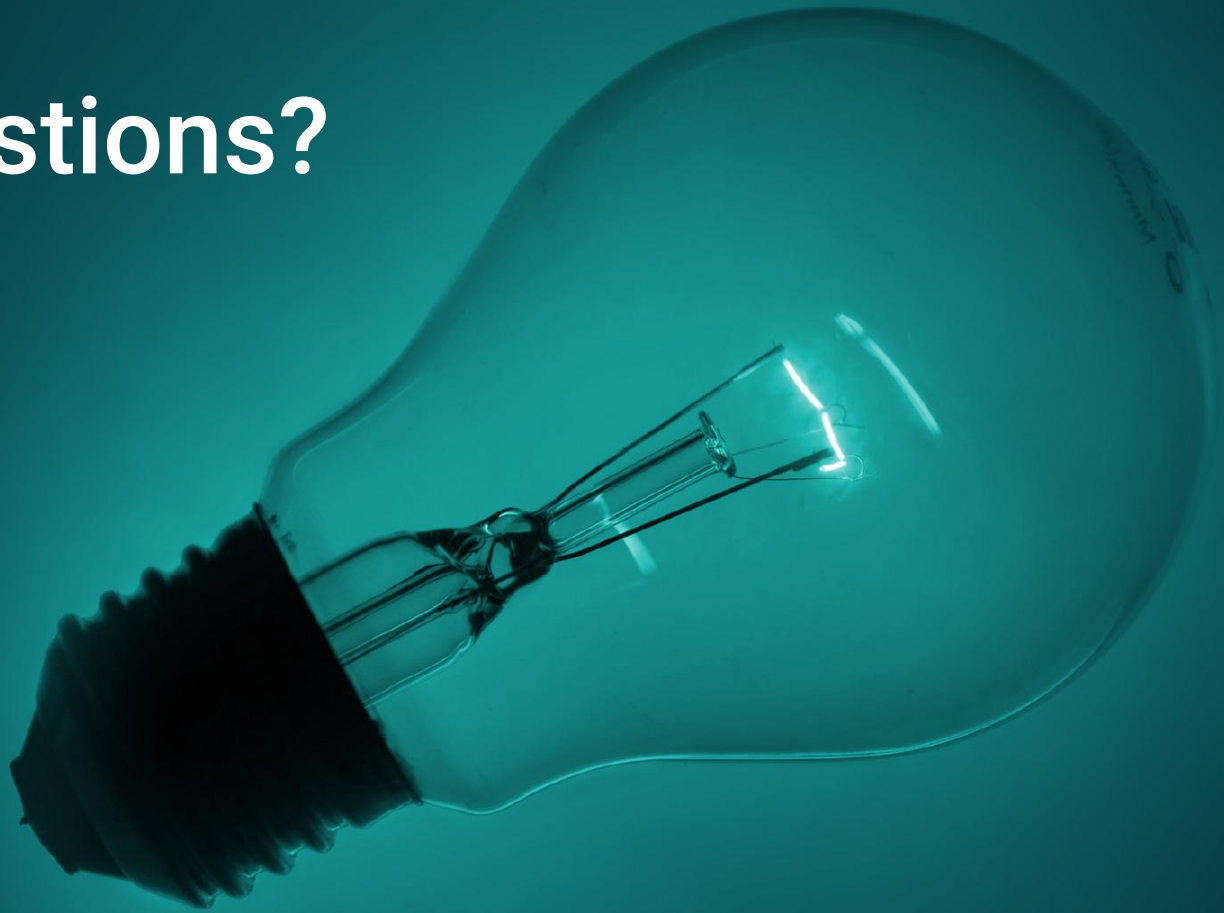


ballerina-lang

Provide Feedback



Questions?



Ballerina
Swan Lake