Ballerina

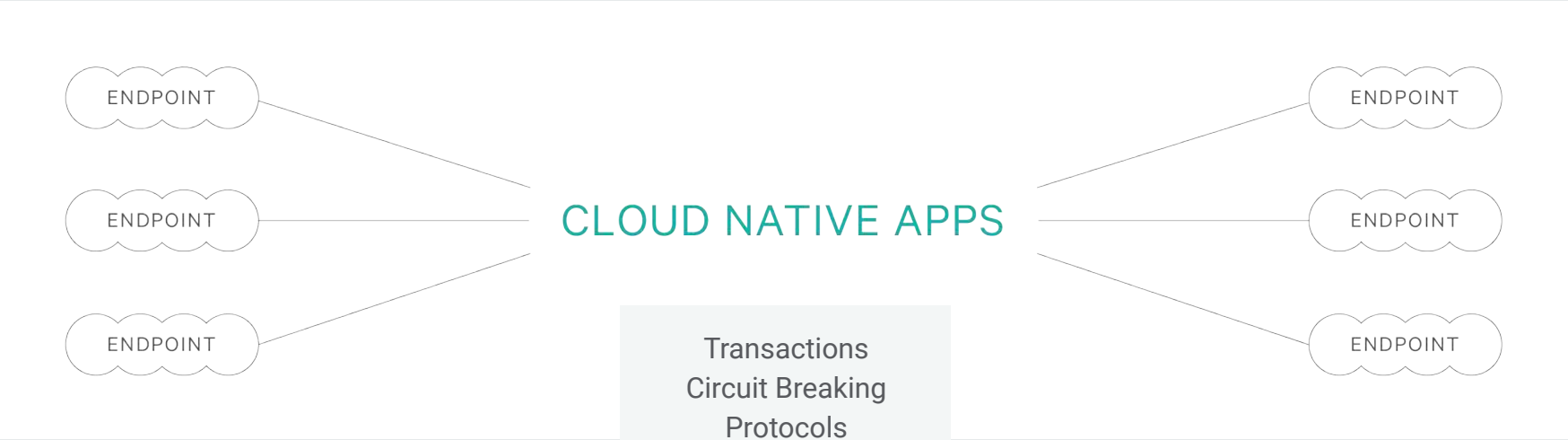
Simplify Network Services for Real-World, Cloud Native Applications with Ballerina

Anuruddha Liyanarachchi
Conf42 2023

Introduction to Ballerina

- Open source, cloud-native programming language optimized for integration
- Developed by WSO2 since 2016 and first released in February 2022
- Rich ecosystem of network protocols, 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

Integration in an increasingly disaggregated world



Transactions
Circuit Breaking
Protocols
Payloads
Events
Security
Workflow
Streams
Compensation

INTEGRATION
PRODUCTS

ESB, BPMN, EAI

NOT AGILE

GENERAL PURPOSE
PROGRAMING LANGUAGES

Java / Spring
JavaScript / Node

NOT INTEGRATION SIMPLE

Ballerina

INTEGRATION PRODUCTS

ESB, BPMN, EAI

NOT AGILE

The Integration Gap

GENERAL PURPOSE PROGRAMING LANGUAGES

Java / Spring
JavaScript / Node

NOT INTEGRATION SIMPLE

Ballerina

Why Ballerina

Cloud native

Network primitives in the language make it simpler to write services and run them in the cloud.

Flexibly typed

Structural types with support for openness are used both for static typing within a program and for describing service interfaces.

Data oriented

Type-safe, declarative processing of JSON, XML, and tabular data with language-integrated queries.

Graphical

Programs have both a textual syntax and an equivalent graphical form based on sequence diagrams.

Concurrent

Easy and efficient concurrency with sequence diagrams and language-managed threads without the complexity of asynchronous functions.

Reliable, maintainable

Explicit error handling, static types, and concurrency safety, combined with a familiar, readable syntax make programs reliable and maintainable.

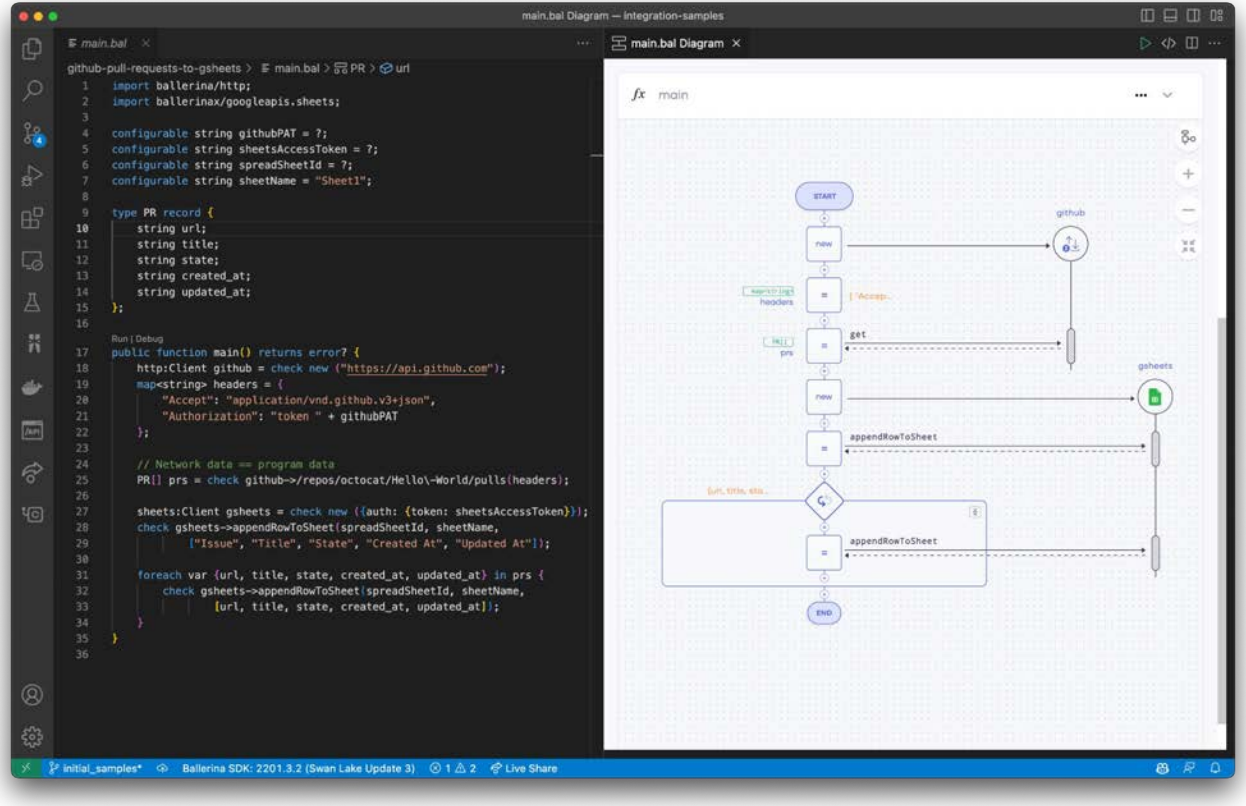
Ballerina In Action

Edit, debug, and run in VSCode

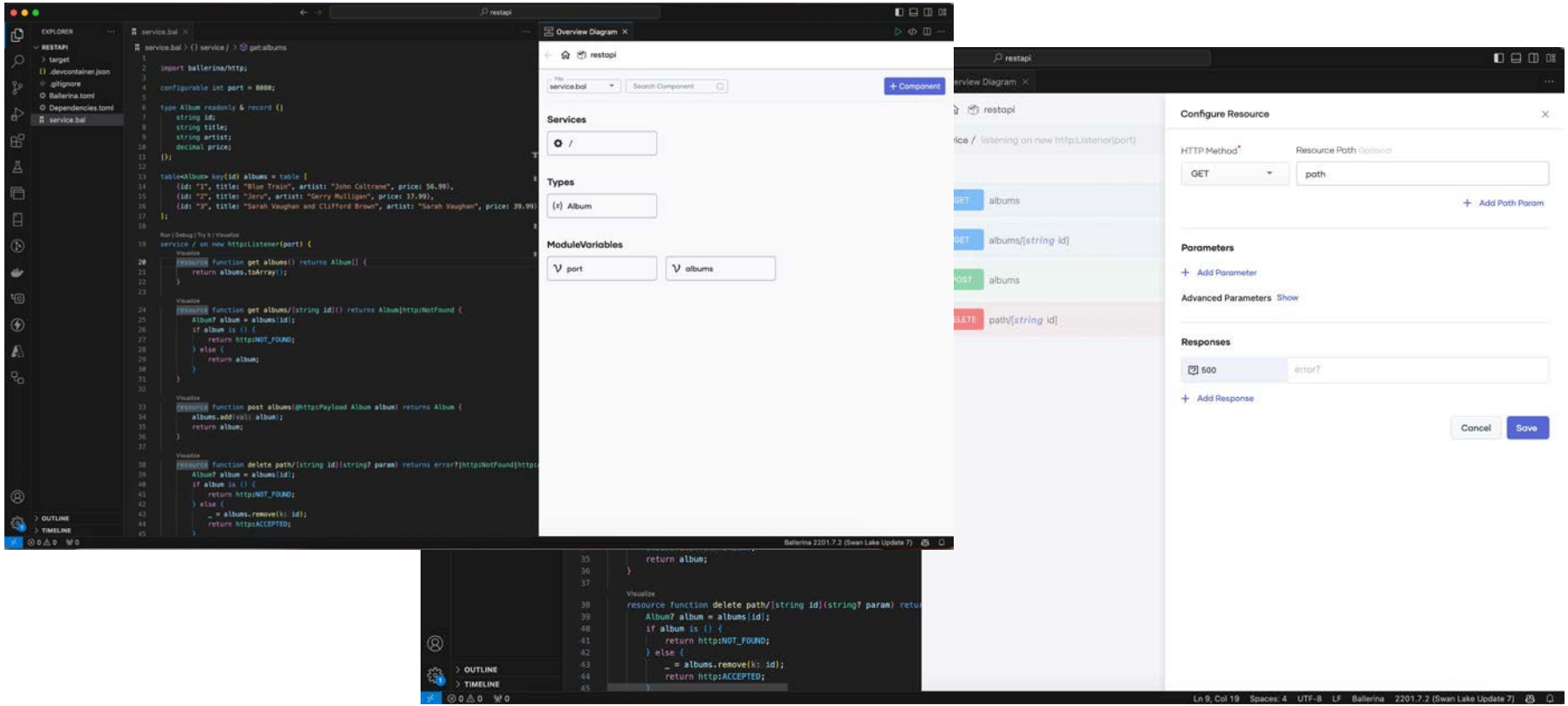
Tired of disjointed toolchains disrupting your workflow? Take control of your integration development with Ballerina. Realize your ideas in VSCode, use your favorite tools, and store them in Git.

Code is the picture / Picture is the code

Instead of deciphering lines of code, Ballerina programs can be viewed and edited as sequence diagrams with flow charts. This makes maintaining and understanding integration applications a breeze. Code never goes out of sync with the picture and vice versa.



Integration designer



Ballerina Library

foo.bal 2, U

hyatt > walk-in-reservations-use-case > integration_service > foo.bal > main

```
1  import ballerina/io;
2
3  ser
4      service                                  Snippet
       service on email:ImapListener
5  pub service on email:PopListener
6      service on file:Listener
7  }   service on ftp:Listener
8      service on graphql:Listener
       service on grpc:Listener
       service on http:Listener
       service on kafka:Listener
       service on tcp:Listener
       service on udp:Listener
       service on websocket:Listener
```

Discover Ballerina packages of reusable code, and assemble them in powerful ways

Search

Ballerina Central is a globally hosted package management system that is used to discover, download, and publish packages.

Popular packages

How to push a package

ballerinax/choreo

ballerinax/client.config

ballerinax/asyncapi.native.handler

ballerinax/googleapis.sheets

ballerinax/twitter

ballerinax/mysql

ballerinax/postgresql

ballerinax/mssql.driver

Ballerina library (API) documentation

Search

Language library
The lang library provides fundamental operations on the data types defined by the language. All the lang libraries have the lang. prefix before their module names and are shipped with the distribution.

Standard library
The standard library includes low-level, general-purpose functionality support for a wide variety of network protocols, interface standards, and data formats. These libraries are published under the ballerina organization.

Extended library
The extended library has connectors to a wide variety of third-party systems including databases and SaaS APIs. These libraries are maintained by the Ballerina community and are published under the ballerinax organization.

Third-party library
The third-party libraries are published in Ballerina Central. Navigate to Ballerina Central to access the API documentation of those packages and to discover more about them.

Language library Distribution (2201.7.2)

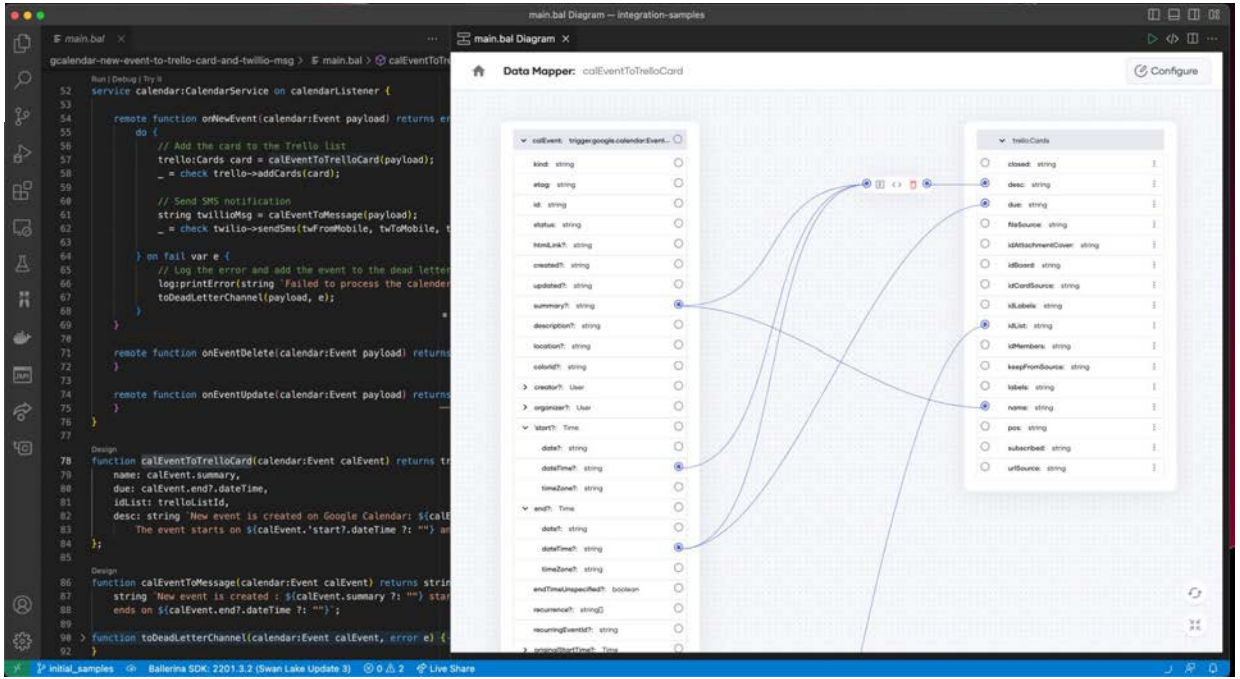
lang.array	lang.boolean	lang.decimal	lang.error
lang.float	lang.function	lang.future	lang.int
lang.map	lang.object	lang.query	lang.regexp
lang.runtime	lang.stream	lang.string	lang.table
lang.transaction	lang.typedesc	lang.value	lang.xml

Standard library

graphql	1.9.0	http	2.9.2	openapi	1.7.2	cloud	2.10.0
grpc	1.9.0	sql	1.10.0	websubhub	1.9.0	websocket	2.9.0
xmldata	2.6.2	email	2.8.0	log	2.8.1	oauth2	2.9.0
tcp	1.8.0	file	1.8.1	ftp	2.8.0	udp	1.8.0

Data transformations

Ballerina has cracked the challenge of mapping one kind of data value to another kind of data value, simultaneously as code and picture, so that both are simple, powerful, and boundless.



Data persistence

Entity Relationship Diagram — ballroom-backend

model.bal M

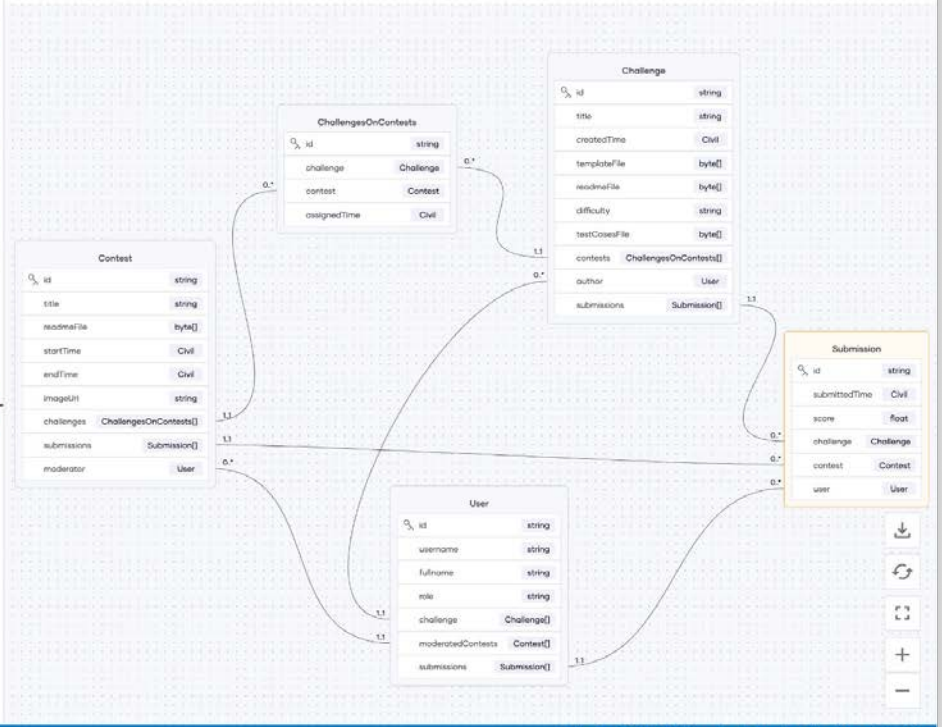
entity_model > persist > model.bal > Submission

```
import ballerina/time;
import ballerina/persist as _;

type User record {|
    readonly string id;
    string username;
    string fullname;
    string role;
    Challenge[] challenge;
    Contest[] moderatedContests;
    Submission[] submissions;
|};

type Contest record {|
    readonly string id;
    string title;
    byte[] readmeFile;
    time:Civil startTime;
    time:Civil endTime;
    string imageUrl;
    ChallengesOnContests[] challenges;
    Submission[] submissions;
    User moderator;
|};

type Challenge record {|
    readonly string id;
    string title;
    time:Civil createdTime;
    byte[] templateFile;
    byte[] readmeFile;
    string difficulty;
    byte[] testCasesFile;
    ChallengesOnContests[] contests;
    User author;
    Submission[] submissions;
|};
```

Entity Relationship Diagram

Entity Relationship Diagram

Collapse



m2* 1 1 Live Share

Ballerina 2201.7.1 (Swan Lake Update 7)

Debug your integration

```
VARIABLES
Local
  ip: "112.134.168"
  ipAPIEndpoint: Client
  resp: Response
    statusCode: 200
    server: ""
    cacheControl: ()
    requestTime: tuple[int,decimal] (size = 2)
    reasonPhrase: "OK"
    resolvedRequestedURI: ""
    receivedTime: tuple[int,decimal] (size = 2)
    entity: Entity
Global

WATCH

CALL STACK                                  Paused on breakpoint
Balle... PAUSED ON BREAKPOINT
  getGeoData                    main.bal 38
  location                      main.bal 26

BREAKPOINTS
  main.bal geo 38
```

```
geo > main.bal > getGeoData
30          return {body: {message: "error occured while getting the location from
31      }
32
33  }
34
    Visualize
35  isolated function getGeoData(string ip) returns json|error {
36      http:Client ipAPIEndpoint = check new ("http://ip-api.com");
37      http:Response resp = check ipAPIEndpoint->get("/json/" + ip);
38      json|error responsePayload = resp.getJsonPayload();
39      if (responsePayload is json) {
40          json payload = responsePayload;
41          return payload;
42      }
43      return error("Error while getting the response from ip");
44  }
45
    Visualize
46  isolated function getWeatherData(decimal lat, decimal lon) returns json|error {
47      View API Docs
48      io:println(...values: "Lat:" + lat.toString() + " " + "lon: " + lon.toStrin
49      http:Client weatherEp = check new ("http://api.weatherapi.com");
50      http:Response resp = check weatherEp->get("/v1/current.json?key=" + USER_KE
51      json|error responsePayload = resp.getJsonPayload();
52      if (responsePayload is json) {
53          json payload = responsePayload;
54          return payload;
55      }
56      return error("Error while getting the response payload", responsePayload);
57  }
58
```

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   AZURE

bal run --debug 5010 /Users/anuruddha/Workspace/conf42/geo
+ geo  bal run --debug 5010 /Users/anuruddha/Workspace/conf42/geo
Compiling source
        anuruddha/geo:0.1.0

Running executable

Listening for transport dt_socket at address: 5010
```

Overview Diagram

geo

Service /geodata listening on new http:Listener(9090) + Resource

Expand All

GET weather/[*string* ip]

Responses

Code	Description
200	json
500	error

GET location/[*string* ip]

Ln 38, Col 1 Spaces: 4 UTF-8 LF Ballerina 2201.7.2 (Swan Lake Update 7)

Application Architecture View

EXPLORER

OPEN EDITORS

shipping_service.ba...

GCP-DEMO (WORKSPACE)

- ads
- cart
- checkout
- currency
- email
- frontend
- payment
- productcatalog
- recommendation
- shipping
  - tests
  - .devcontainer.json
  - .gitignore
  - Ballerina.toml
  - Cloud.toml
  - demo_pb.bal
  - Dependencies.toml
  - shipping_service.bal
  - utils.bal
- ui

OUTLINE

TIMELINE

>

Ballerina: Architecture View — recently used

Developer: Reload Window

Preferences: Color Theme — ⌘ K ⌘ T

Developer: Reload Webviews

Developer: Open Webview Developer Tools

Choreo: Create New Component

Ballerina: Show Examples

Workspaces: Remove Folder from Workspace...

Preferences: File Icon Theme

Workspaces: Save Workspace As...

View: Focus Problems (Errors, Warnings, Infos)

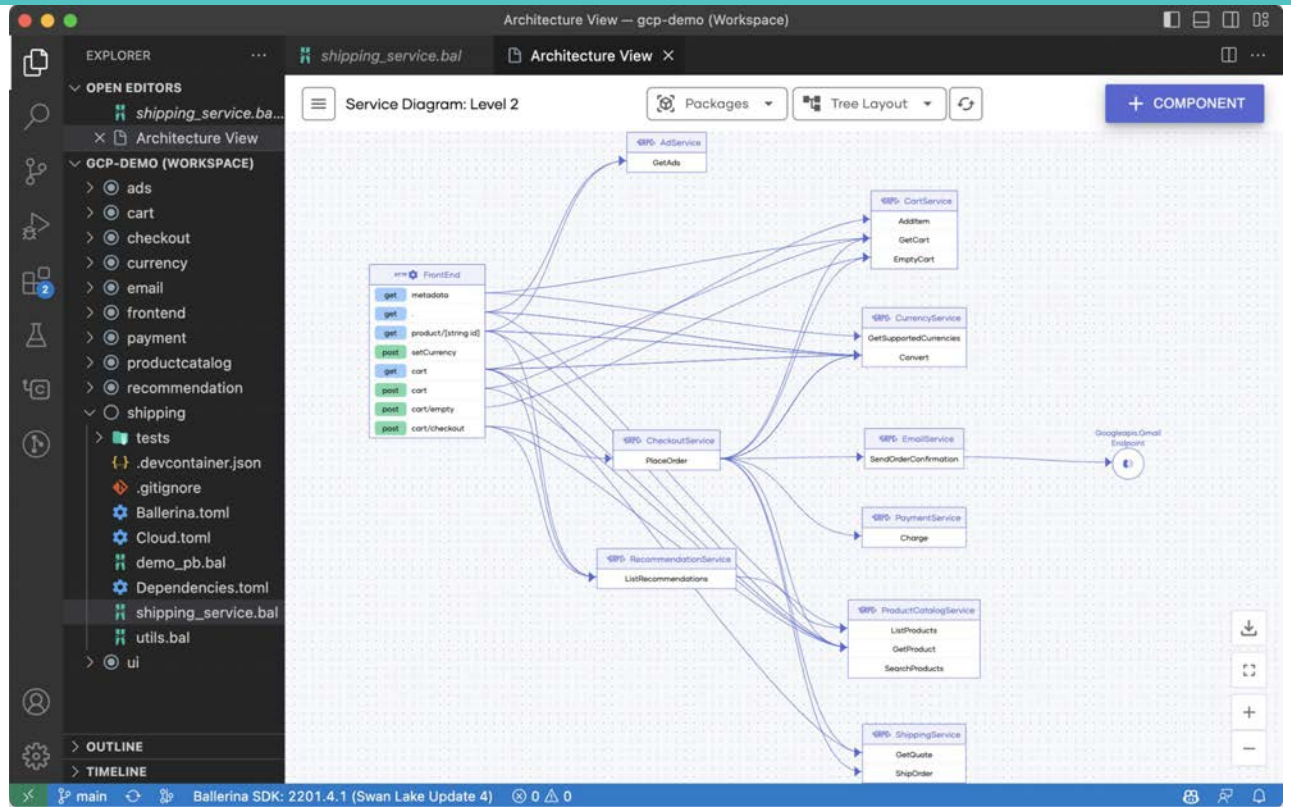
WSO2 Choreo: Focus on Account View

```
15  // under the License.
16
17  import ballerina/grpc;
18
19  listener grpc:Listener ep = new (9095);
20
21  @display {
22      label: "ShippingService",
23      id: "shipping"
24  }
25  @grpc:ServiceDescriptor {descriptor: ROOT_DESCRIPTOR_DEMO, descMap: getDescriptorMapDemo()}
    Run | Debug | Try it | Visualize
26  service "ShippingService" on ep {
27      final float SHIPPING_COST = 8.99;
28
        Visualize
29      isolated remote function GetQuote(GetQuoteRequest value) returns GetQuoteResponse|error {
30          CartItem[] items = value.items;
31          int count = 0;
```

Anjana Supun, 14 months ago • Add initial implemenation of microserv

main | Ballerina SDK: 2201.4.1 (Swan Lake Update 4) | ⊗ 0 ⚠ 0 | Anjana Supun, 14 months ago | Ln 17, Col 23 | Spaces: 4 | UTF-8 | LF | Ballerina

Application Architecture View



Integration tools

OpenAPI tool
Generate a Ballerina service and client skeletons for an OpenAPI contract.

GraphQL tool
Generate GraphQL client skeletons in Ballerina.

AsyncAPI tool
Generate a Ballerina service and listener skeletons for an AsyncAPI contract.

Strand dump tool
Dump and inspect the currently available strands of a Ballerina program.

Health tool (FHIR/HL7)
FHIR/HL7 profile to client and stub generation tool of Ballerina.

EDI tool
The set of command line tools provided to work with EDI files in Ballerina.

Deployment

Build a self-contained executable (.jar)

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

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

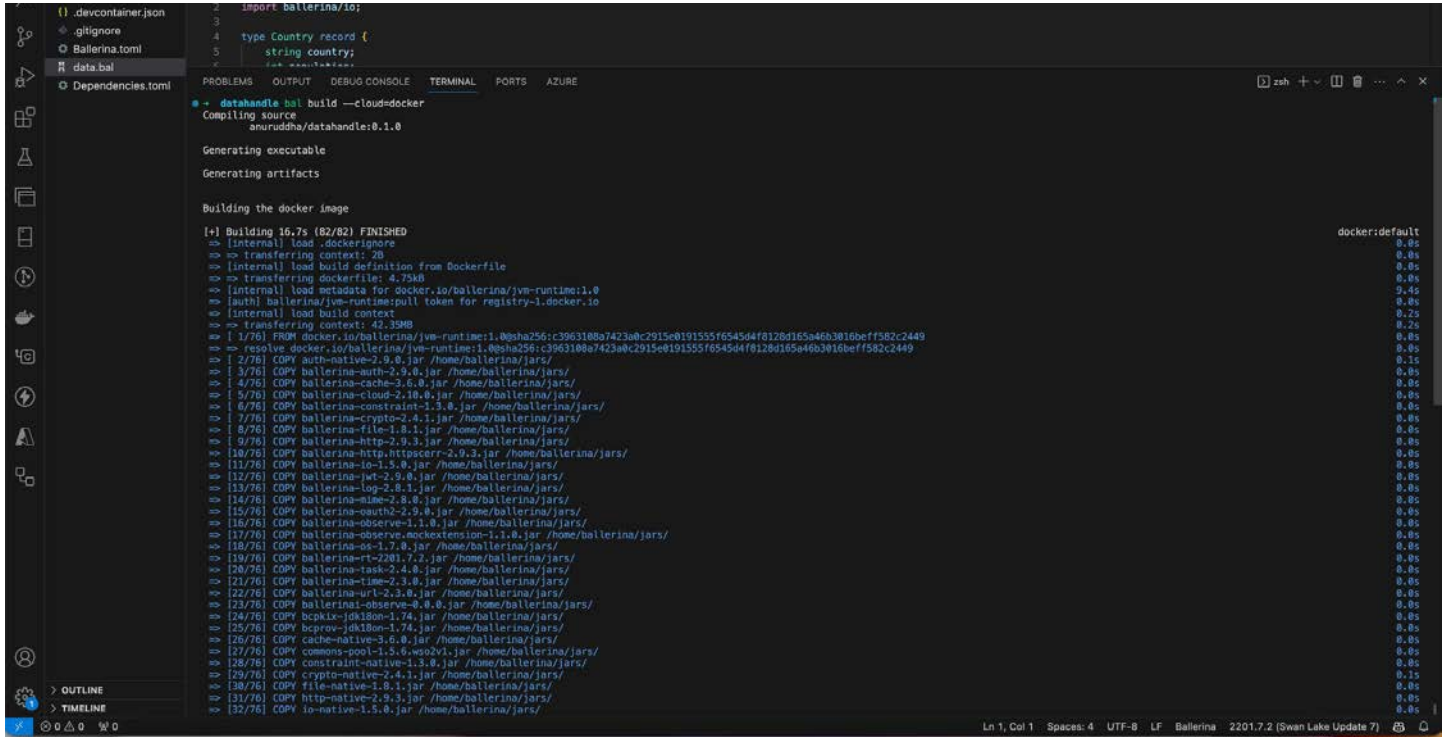
```
→  geo bal run
Compiling source
        anuruddha/geo:0.1.0

Running executable
```

Build a GraalVM Native Image

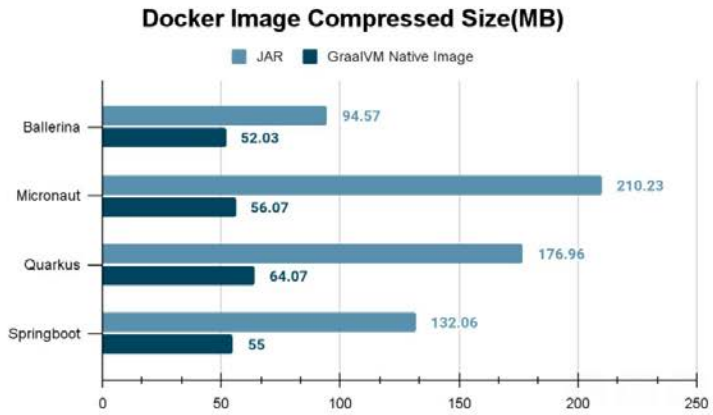
```
$ bal build --graalvm

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
```

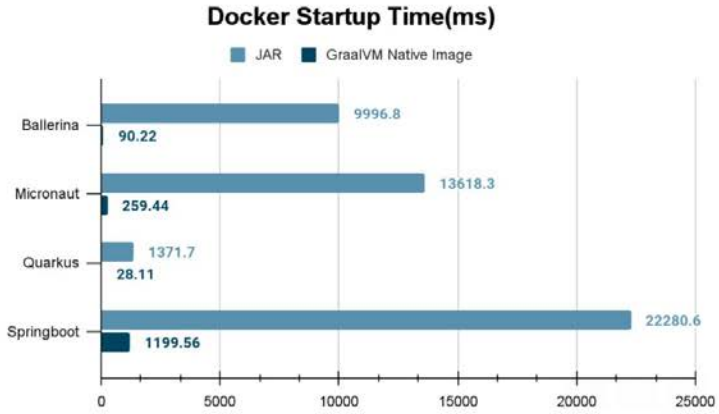
Build a Docker container

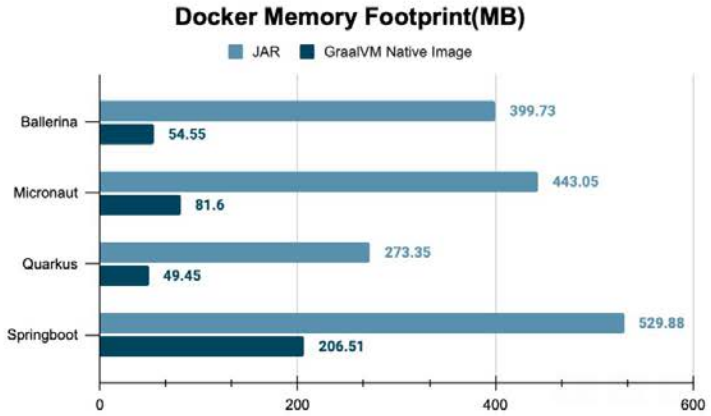
```
bal build -cloud=docker
```



Docker + GraalVM



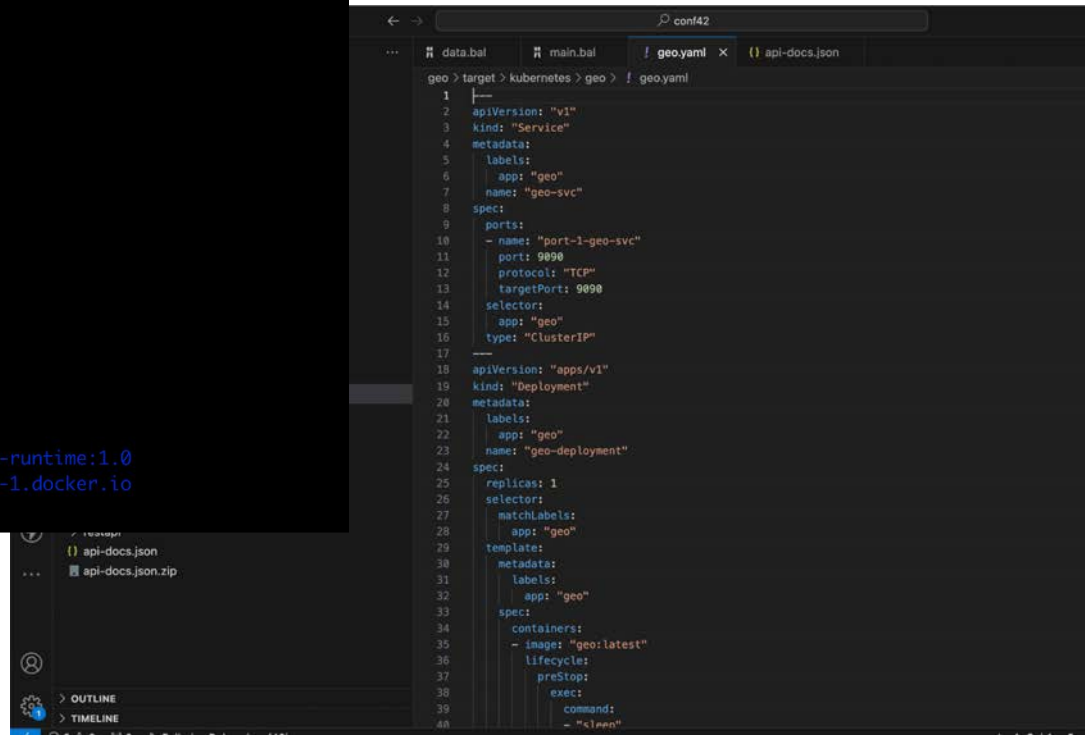




Build Kubernetes artifacts

```
➜  geo bal build --cloud=k8s
Compiling source
        anuruddha/geo:0.1.0

Generating executable

Generating artifacts

        @kubernetes:Service
        @kubernetes:ConfigMap
        @kubernetes:Secret
        @kubernetes:Deployment
        @kubernetes:HPA

Building the docker image

[+] Building 10.0s (82/82) FINISHED
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 4.73kB
 => [internal] load metadata for docker.io/ballerina/jvm-runtime:1.0
 => [auth] ballerina/jvm-runtime:pull token for registry-1.docker.io
 => [internal] load build context
```

bal build –cloud=k8s



Function as a service

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 servic

Supported triggers and bindings

An Azure Function consists of a trigger and optional bindings. A trigger defines h
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 flov
outputting data from the function to an external resource. For more information,
concepts.

The following Azure Functions triggers and bindings are currently supported in B

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

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

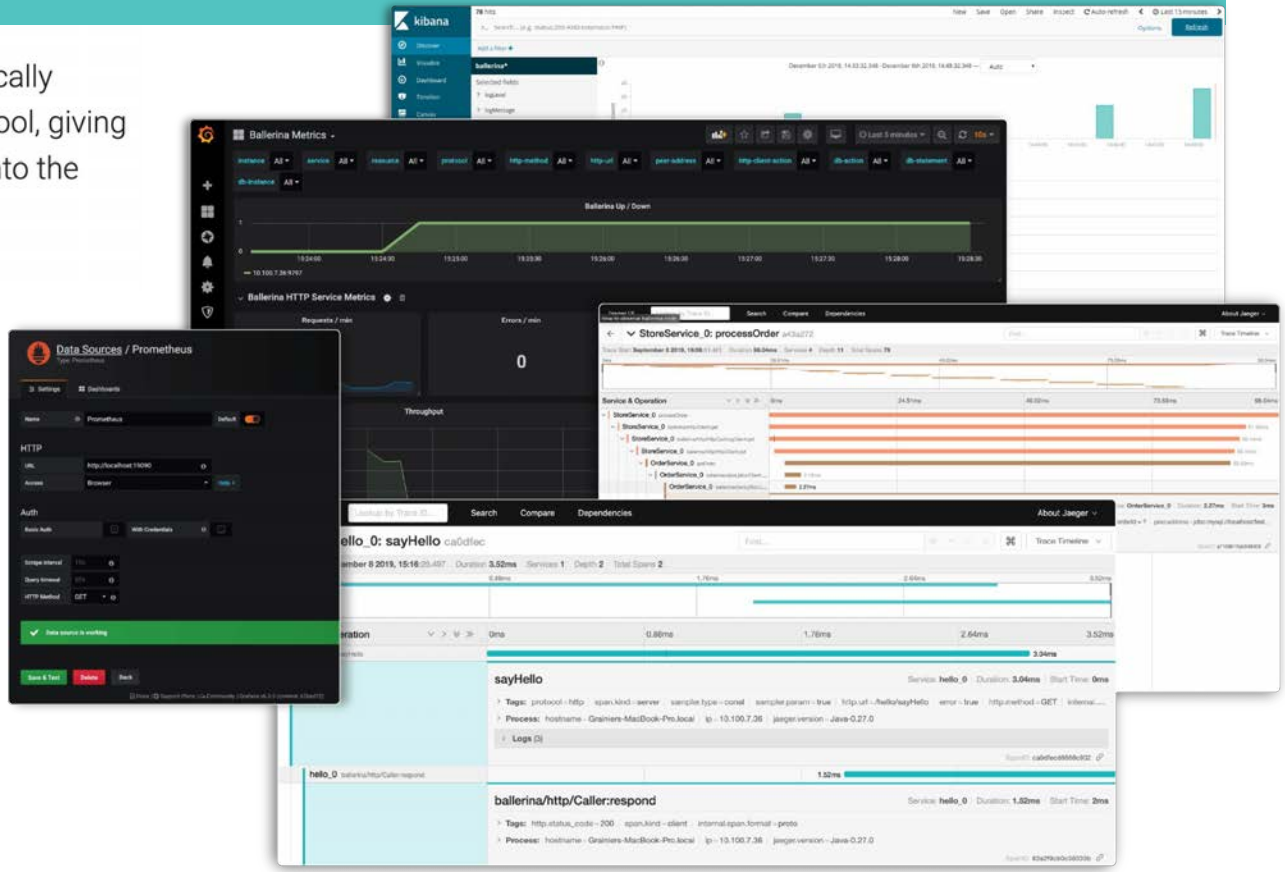
An AWS Lambda function can be triggered by various AWS services. You can find the list of supported notification types below.

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

Observability

Built-in observability

Every Ballerina program is automatically observable by any Open Telemetry tool, giving you complete control and visibility into the code's behavior and performance.



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.

Demo

Community

- Website: https://ballerina.io/
- Github: https://github.com/ballerina-platform/
- StackOverflow: https://stackoverflow.com/questions/tagged/ballerina
- Discord: https://discord.com/invite/ballerinalang
- Twitter: https://twitter.com/ballerinalang

THANK YOU