



Ballerina

Swan Lake

Code to Cloud: Mastering Kubernetes Deployments with Ballerina



Anupama Pathirage
Director/ Head of Engineering - Integration
@
WS02



anupama@wso2.com



[@anupama_pathira](https://twitter.com/anupama_pathira)



The Journey from Code to Cloud

- The rise of cloud-native applications has transformed how we build and deploy software.
- Modern applications require seamless integration, rapid scalability, and reliable deployment strategies.
- Kubernetes has emerged as the de facto standard for orchestrating these complex, cloud-native deployments.



Challenges in Cloud Integration

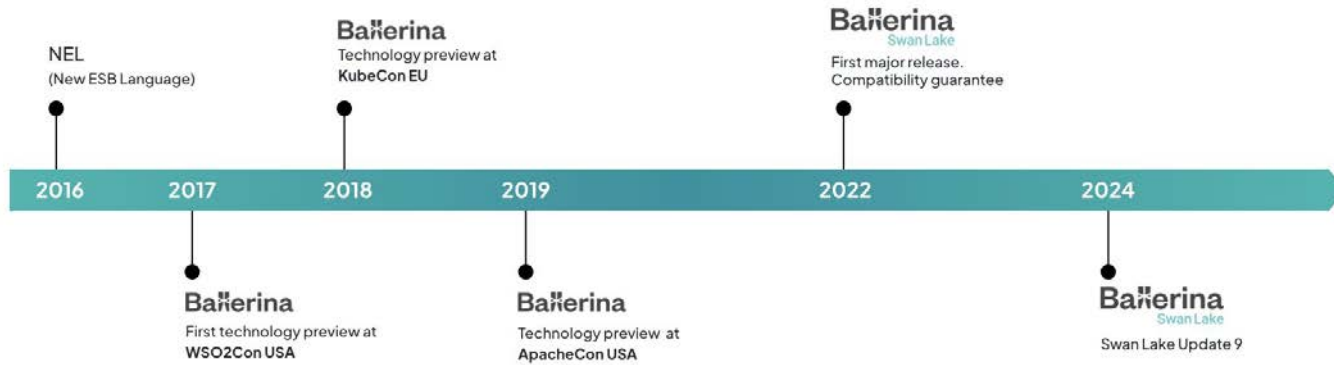
- **Multiple Services:** Cloud integration often involves numerous services and APIs that must work together.
- **Dynamic Environments:** Cloud environments are highly dynamic, with containers and microservices adding layers of complexity.
- **Configuration Overhead:** Managing deployment configurations, scaling, and monitoring can be intricate and time-consuming.
- **Security and Compliance:** Ensuring secure data flow and compliance across various cloud components is challenging.

A glowing lightbulb is the central focus of the image, set against a solid teal background. The lightbulb is illuminated from within, casting a bright glow that fades into the background. The filament is visible, and the base of the bulb is dark. The overall mood is one of innovation and ideas.

Ballerina : A technology for
simplifying cloud integrations....

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 is about **cloud-native integrations**

- Ballerina: More than just a programming language; it's a full framework.
- Other Languages:
 - Many languages exist (C, C++, C#, Python, Java, Go, etc.).
 - These are general-purpose and not specifically focused on integrations or API creation.
- Importance of Abstractions:
 - Right level of abstraction, natural to the problem, is crucial.
- Ballerina provides clear and clean abstractions/tools for integration.
- Built on existing technologies and not a research product.

Ballerina provides right abstractions for ...

Data

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

Network

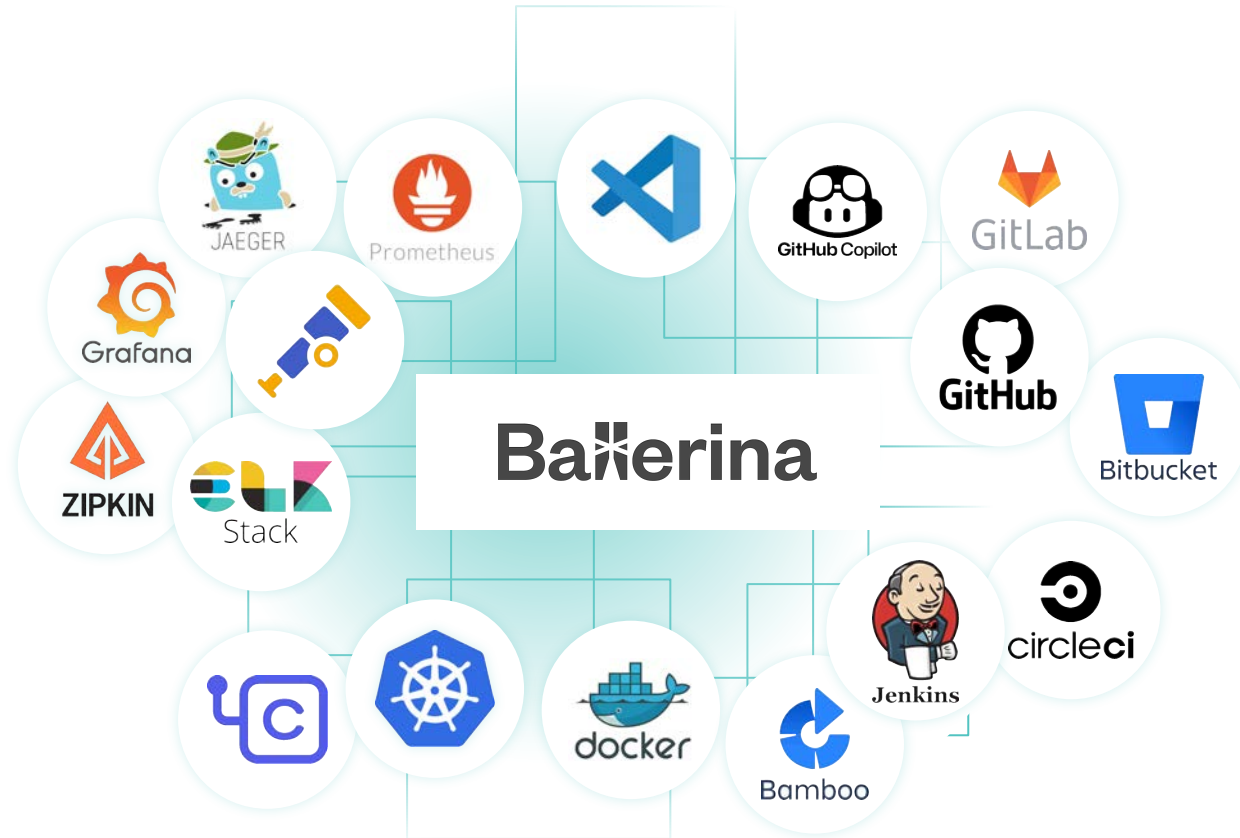
- Communicating data across different clients and services via different network protocols

Concurrency

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

Development / Maintenance

VS Code as tooling with familiar toolset



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

Structure of Ballerina package

```
hello-world
├── .devcontainer.json
├── .gitignore
├── Ballerina.toml
└── main.bal

1 directory, 4 files
```

Not Ballerina Specific : Contains configuration files to configure a dev environment in a docker container

Not Ballerina Specific : specifies intentionally untracked files that Git should ignore like configs, build artifacts.

Identifies the directory as a Ballerina package. It contains all the meta information needed to build the package.

Ballerina source file.

Integration Designer - REST

The screenshot displays the SAP Integration Designer interface for configuring a REST API. The main workspace is divided into several panels:

- EXPLORER:** Shows the project structure with 'service bal' selected.
- Code Editor:** Contains the REST API configuration code for the 'restapi' service. The code defines a 'getAlbums' endpoint and a 'delete path' endpoint. The 'getAlbums' endpoint is configured with a GET method and a resource path of '/albums'. The 'delete path' endpoint is configured with a DELETE method and a resource path of '/albums/{string id}'. The code includes logic for handling HTTP status codes like 'HTTP_NOT_FOUND' and 'HTTP_ACCEPTED'.
- Overview Diagram:** Shows the service 'restapi' and its components, including the 'getAlbums' endpoint and the 'delete path' endpoint. The diagram also shows the 'Album' type and the 'port' module variable.
- Configure Resource:** A dialog box for configuring the resource path and method. It shows the HTTP Method set to 'GET' and the Resource Path set to '/albums'. There are options to add path parameters and advanced parameters.

```
1 import ballerina/http;
2 configurable int port = 8080;
3
4 type Album readonly & record {
5     string id;
6     string title;
7     string artist;
8     decimal price;
9 };
10
11 table<Album> key(id) albums = table {
12     {id: "1", title: "Blue Train", artist: "John Coltrane", price: 56.99},
13     {id: "2", title: "Jazz", artist: "Henry Mulligan", price: 37.99},
14     {id: "3", title: "Sarah Vaughan and Clifford Brown", artist: "Sarah Vaughan", price: 39.99}
15 };
16
17 run | debug | py | visualize
18
19 service / on new httpListener(port) {
20     resource function getAlbums() returns Album[] {
21         return albums.toArray();
22     }
23
24     resource function getAlbums(string id) returns Album|httpNotFound {
25         Album? album = albums[id];
26         if album is () {
27             return http:NOT_FOUND;
28         } else {
29             return album;
30         }
31     }
32
33     resource function postAlbums(httpPayload Album album) returns Album {
34         albums.add(new album);
35         return album;
36     }
37
38     resource function delete path/{string id}(string? param) returns error?|httpNotFound|httpAccepted {
39         Album? album = albums[id];
40         if album is () {
41             return http:NOT_FOUND;
42         } else {
43             = albums.remove(k: id);
44             return http:ACCEPTED;
45         }
46     }
47 }
```


Integration Designer - GraphQL

The screenshot displays the Ballerina IDE's GraphQL Designer interface. On the left, a code editor shows the definition of a GraphQL schema for a movie rating system. The schema includes types for `Movie`, `Director`, `User`, and `Review`, along with several resource functions for interacting with the data.

```
1 import ballerina/graphql;
2 import ballerina/graphql/data_loader;
3 import ballerina/constraint;
4
5 > isolated service class Movie {
61 }
62
63 > public isolated service class Director {
64 }
65
66 > public isolated service class User {
67 }
68
69 # Description.
70 public isolated service class Review {
71   private final string id;
72   private final string userId;
73   private final string movieId;
74   private final int score;
75   private final string description;
76 }
77
78 isolated function init(ReviewRecord reviewRecord) {
79   self.id = reviewRecord.id;
80   self.userId = reviewRecord.userId;
81   self.movieId = reviewRecord.movieId;
82   self.score = reviewRecord.score;
83   self.description = reviewRecord.description;
84 }
85
86 # The ID of the review.
87 # + return - ID of the review.
88 Visualize
89 isolated resource function get id() returns @;
90
91 # The user who wrote the review.
92 # + return - User who wrote the review.
93 Visualize
94 isolated resource function get user(graphql:Context) returns @;
95   return getUserById(self.userId);
96 }
97
98 # The movie that was reviewed.
99 # + return - Movie that was reviewed.
100 Visualize
101 isolated resource function get movie(graphql:Context) returns @;
102   return getMovieById(self.movieId);
103 }
104
105 # The score given by the user.
106 # + return - Score given by the user.
107 Visualize
108 isolated resource function get score() returns @;
109
110 # The description of the review.
111 Visualize
112 isolated resource function get description() returns @;
```

The right side of the IDE shows the GraphQL Designer visual representation. It features a root node with several fields: `movies` (type `[Movie!]`), `users` (type `[User!]`), `directors` (type `[Director!]`), `reviews` (type `[Review!]`), and `addReview` (type `Review`). The `Movie` type is defined with fields: `id` (type `!ID!`), `title` (type `String!`), `year` (type `Int!`), `description` (type `String!`), `rating` (type `Float`), and `director` (type `Director`). The `Director` type has fields: `id` (type `!ID!`), `name` (type `String!`), and `bio` (type `String!`). The `User` type has fields: `id` (type `!ID!`), `name` (type `String`), and `email` (type `String`). The `Review` type has fields: `id` (type `!ID!`), `user` (type `User`), `movie` (type `Movie`), `score` (type `Int!`), and `description` (type `String!`). Arrows indicate the relationships between the root fields and their respective types.

Graphical View

The image shows a code editor window with two panes. The left pane displays the source code for a Ballerina program named `main.bal`. The right pane displays the graphical flow diagram for the same program, titled `main`.

```
1  import ballerina/http;
2  import ballerina/googleapis/sheets;
3
4  configurable string githubPAT = ?;
5  configurable string sheetsAccessToken = ?;
6  configurable string spreadsheetId = ?;
7  configurable string sheetName = "Sheet1";
8
9  type PR record {
10     string url;
11     string title;
12     string state;
13     string created_at;
14     string updated_at;
15 };
16
17 public function main() returns error? {
18     http:Client github = check new ("https://api.github.com");
19     map<string> headers = {
20         "Accept": "application/vnd.github.v3+json",
21         "Authorization": "token " + githubPAT
22     };
23
24     // Network data == program data
25     PR[] prs = check github->/repos/octocat/Hello\~World/pulls(headers);
26
27     sheets:Client gsheets = check new ({auth: {tokens: sheetsAccessToken}});
28     check gsheets->appendRowToSheet(spreadsheetId, sheetName,
29         ["Issue", "Title", "State", "Created At", "Updated At"]);
30
31     foreach var {url, title, state, created_at, updated_at} in prs {
32         check gsheets->appendRowToSheet(spreadsheetId, sheetName,
33             [url, title, state, created_at, updated_at]);
34     }
35 }
36
```

The graphical flow diagram on the right illustrates the execution flow of the program. It starts with a `START` node, followed by a `new` node that creates a `github` client. This is followed by an `=` node that sets the `headers` variable, with a note `map<string> headers` and a value `{ "Accept": "application/vnd.github.v3+json", "Authorization": "token " + githubPAT }`. The flow then goes to a `get` node, which is connected to the `github` client and returns a `PR[] prs` variable. Next is another `new` node that creates a `gsheets` client. This is followed by an `=` node that calls `appendRowToSheet` on the `gsheets` client. A decision diamond (diamond with a question mark) follows, with a note `loop, while, etc...`. Below the diamond is another `=` node that also calls `appendRowToSheet` on the `gsheets` client. The flow ends at an `END` node.

In-Sync Documentation

The image shows a side-by-side comparison of code and its generated documentation. On the left, an IDE window displays the source code for a service named `social-media`. The code includes an `init()` function, a `createInterceptors()` function, and a `get users()` resource function. On the right, the Swagger UI displays the corresponding API documentation for the `default` API. The documentation lists six endpoints with their respective HTTP methods and descriptions.

```
al_media_service.bal x
al_media_service.bal > {} service /social\media

final http:Client sentimentEndpoint = check new (sc

Run | Debug | Try it | Visualize
service SocialMedia /social\media on new http:Lis

Visualize
public function init() returns error? {
    log:printInfo("Social media service started
}

// Service-level error interceptors can handle
Visualize
public function createInterceptors() returns Re
    return new ResponseErrorInterceptor();
}

# Get all the users
#
# + return - The list of users or error message
Visualize
resource function get users() returns User[]|e
    stream<User, sql:Error?> userStream = soci
    return from User user in userStream
        select user;
}
```

Swagger UI - default

- GET** /users Get all the users
- POST** /users Create a new user
- GET** /users/{id} Get a specific user
- DELETE** /users/{id} Delete a user
- GET** /users/{id}/posts Get posts for a give user
- POST** /users/{id}/posts Create a post for a given user

Schemas

Easy Data Transformations

The image displays a development environment with two main panels. The left panel is a code editor showing Ballerina code for a REST API service. The right panel is a visual data mapper interface for the 'enrollPerson' data mapper.

```
rest-api-with-data-mapper > main.bal > service / > post:persons
30 configurable int port = 8080;
31 configurable string sheetsAccessToken = ?;
32 configurable string spreadsheetId = ?;
33 configurable string sheetName = "enrollments";
34
35 final sheets:Client gsheetsClient = check new ({auth: {token: sheetsA
36
37 const D_TIER_4_VISA = "D tier-4";
38
39 table<Person> key(id) persons = table {
40   {
41     id: 1001,
42     firstName: "John",
43     lastName: "Doe",
44     age: 25,
45     country: "LK"
46   },
47   {
48     id: 1002,
49     firstName: "Jane",
50     lastName: "Doe",
51     age: 23,
52     country: "US"
53   }
54 };
55
56 var totalCredits = function(int total, record {string id; string name
57   returns int => total + (course.id.startsWith("
58
59 Visualize
60 function enrollPerson(Person person, Course[] courses) returns Studen
61   id: person.id.toString() + (isForeign ? "F" : "N"),
62   age: person.age,
63   fullName: person.firstName + " " + person.lastName,
64   courses: from var coursesItem in courses
65     where coursesItem.id.startsWith("CS")
66     select {
67       title: coursesItem.id + " - " + coursesItem.name,
68       credits: coursesItem.credits
69     },
70   visaType: isForeign ? D_TIER_4_VISA : "n/a",
71   totalCredits: courses.reduce(totalCredits, 0)
72 };
73
74 Run | Debug | Try It | Visualize
75 service / on new http:Listener(port) {
```

The visual data mapper interface shows the following components:

- Data Mapper:** enrollPerson
- Filter Input:** filter input
- Auto Map:** Auto Map
- Configure:** Configure
- person: Person:** id: int, firstName: string, lastName: string, age: int, country: string
- Student:** id: string, fullName: string, age: int, courses: record[], totalCredits: int, visaType: string
- courses: Course[]**
- Local Variables:** isForeign: boolean
- Module Variables:** D_TIER_4_VISA: string, totalCredits: > course returns int

Blue lines indicate the mapping between the source fields (person and courses) and the target fields (Student).

Better Debugging Experience

The screenshot displays the Ballerina IDE interface during a debugging session. The main editor shows the following code:

```
geo > main.bal > getGeoData
30     return {body: {message: "error occured while getting the location from
31     }
32 }
33 }
34 }
35
36 Visualize
37 isolated function getGeoData(string ip) returns json:error {
38     http:Client ipAPIEndpoint = check new ("http://ip-api.com");
39     http:Response resp = check ipAPIEndpoint->get("/{json}/" + ip);
40     json:error responsePayload = resp.getJsonPayload();
41     if (responsePayload is json) {
42         json payload = responsePayload;
43         return payload;
44     }
45     return error("Error while getting the response from ip");
46 }
47
48 Visualize
49 isolated function getWeatherData(decimal lat, decimal lon) returns json:error {
50     io:println(...values: "lat:" + lat.toString() + " " + "lon:" + lon.toString());
51     http:Client weatherEp = check new ("http://api.weatherapi.com");
52     http:Response resp = check weatherEp->get("/{v1/current.json?key=" + USER_KEY);
53     json:error responsePayload = resp.getJsonPayload();
54     if (responsePayload is json) {
55         json payload = responsePayload;
56         return payload;
57     }
58     return error("Error while getting the response payload", responsePayload);
59 }
```

The left sidebar shows the Explorer (O'RE) with the following structure:

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

The WATCH panel shows:

- getGeoData
- location

The CALL STACK panel shows:

- Paused on breakpoint
- getGeoData main.bal 38
- location main.bal 26

The BREAKPOINTS panel shows:

- main.bal geo 38

The right sidebar shows the service overview for 'geo' listening on http:Listener(9090). It lists two endpoints:

- GET weather(string ip)
- GET location(string ip)

The Responses table shows:

Code	Description
200	json
500	error

The terminal at the bottom shows the following commands and output:

```
bal run --debug 5810 /Users/anuruddha/Workspace/conf42/geo
+ geo bal run --debug 5810 /Users/anuruddha/Workspace/conf42/geo
Compiling source
  anuruddha/geo:0.1.0

Running executable

Listening for transport dt_socket at address: 5810
```


Everything under one roof

FTP

 **gRPC**

 **kafka**

 **GraphQL**

Websub

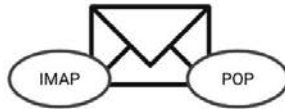
 **WebSocket**

Websubhub

 **MQTT**

TCP

 **FHIR**


IMAP POP

































































UDP

 **REST API**

 **NATS**

FILE

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	 top	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

Batteries included Ballerina connectors - 600+ Connectors

 aayu.mftg.as2	1.3.1	 ably	1.5.1	 activecampaign	1.3.1	 activemq.driver	1.0.1
 adobe.analytics	1.3.1	 adp.paystatements	1.5.1	 adp.workerpayrollinstructions	1.5.1	 alfresco	1.3.1
 amadeus.flightcreateorders	1.3.1	 amadeus.flightoffersprice	1.3.1	 amadeus.flightofferssearch	1.3.1	 Apache	0.1.0
 api2pdf	1.5.1	 apideck.accounting	1.5.1	 apideck.lead	1.5.1	 apideck.proxy	1.5.1
 apple.appstore	1.5.1	 asana	1.6.1	 asb	3.7.0	 asynclapi.native.handler	0.2.0
 atspoke	1.5.1	 automata	1.5.1	 avatax	1.3.1	 avaza	1.5.1
 aws.dynamodb	2.3.0	 aws.lambda	3.2.0	 aws.redshift	1.0.2	 aws.redshift.driver	0.1.0
 aws.s3	3.3.0	 aws.ses	2.1.0	 aws.simplifiedb	2.2.0	 aws.sns	2.1.0
 aws.sqs	3.1.0	 azure.cosmosdb	4.2.0	 azure.eventhub	3.1.0	 azure.storage.service	4.3.0
 azure.ad	2.5.0	 azure.analysis.services	1.3.1	 azure.anomalydetector	1.5.1	 azure.data.lake	1.5.1
 azure.functions	4.1.0	 azure.iotcentral	1.5.1	 azure.iothub	1.5.1	 azure.keyvault	1.6.0
 azure.openal.chat	3.0.1	 azure.openal.deployment	1.0.1	 azure.openal.embeddings	1.0.2	 azure.openal.finetunes	1.0.1
 azure.openal.text	1.0.3	 azure.qnamaker	1.5.1	 azure.sql.db	1.5.1	 azure.textanalytics	1.5.1
 azure.timeseries	1.5.1	 beezup.merchant	1.6.0	 bing.autosuggest	1.5.1	 bintable	1.5.1
 bisstats	1.5.1	 bitbucket	1.5.1	 bitly	1.5.1	 bmc.truesight.presentation.server	1.5.1
 botify	1.5.1	 box	1.5.1	 boxapi	0.1.1	 brex.onboarding	1.5.1
 brex.team	1.5.1	 browshot	1.5.1	 bulksms	1.5.1	 candid	0.1.1
 capsulecrm	1.5.1	 cdata.connect	1.2.0	 cdata.connect.driver	1.1.1	 chaingateway	1.5.1
 choreo	0.4.12	 clever.data	1.5.1	 client.config	1.0.1	 cloudmersive.barcode	1.5.1

Model it once, persist anywhere

The screenshot displays a development environment with two main components: a code editor on the left and an Entity Relationship Diagram (ERD) on the right.

Code Editor (model.bal):

```
entity_model > persist > model.bal > Submission >  
1 import ballerina/time;  
2 import ballerina/persist as _;  
3  
4 type User record {  
5     readonly string id;  
6     string username;  
7     string fullname;  
8     string role;  
9     Challenge[] challenge;  
10    Contest[] moderatedContests;  
11    Submission[] submissions;  
12 };  
13  
14 type Contest record {  
15     readonly string id;  
16     string title;  
17     byte[] readmefile;  
18     time:Civil startTime;  
19     time:Civil endTime;  
20     string imageUrl;  
21     ChallengesOnContests[] challenges;  
22     Submission[] submissions;  
23     User moderator;  
24 };  
25  
26 type Challenge record {  
27     readonly string id;  
28     string title;  
29     time:Civil createTime;  
30     byte[] templateFile;  
31     byte[] readmefile;  
32     string difficulty;  
33     byte[] testCasesFile;  
34     ChallengesOnContests[] contests;  
35     User author;  
36     Submission[] submissions;  
37 };
```

Entity Relationship Diagram (ballroom-backend):

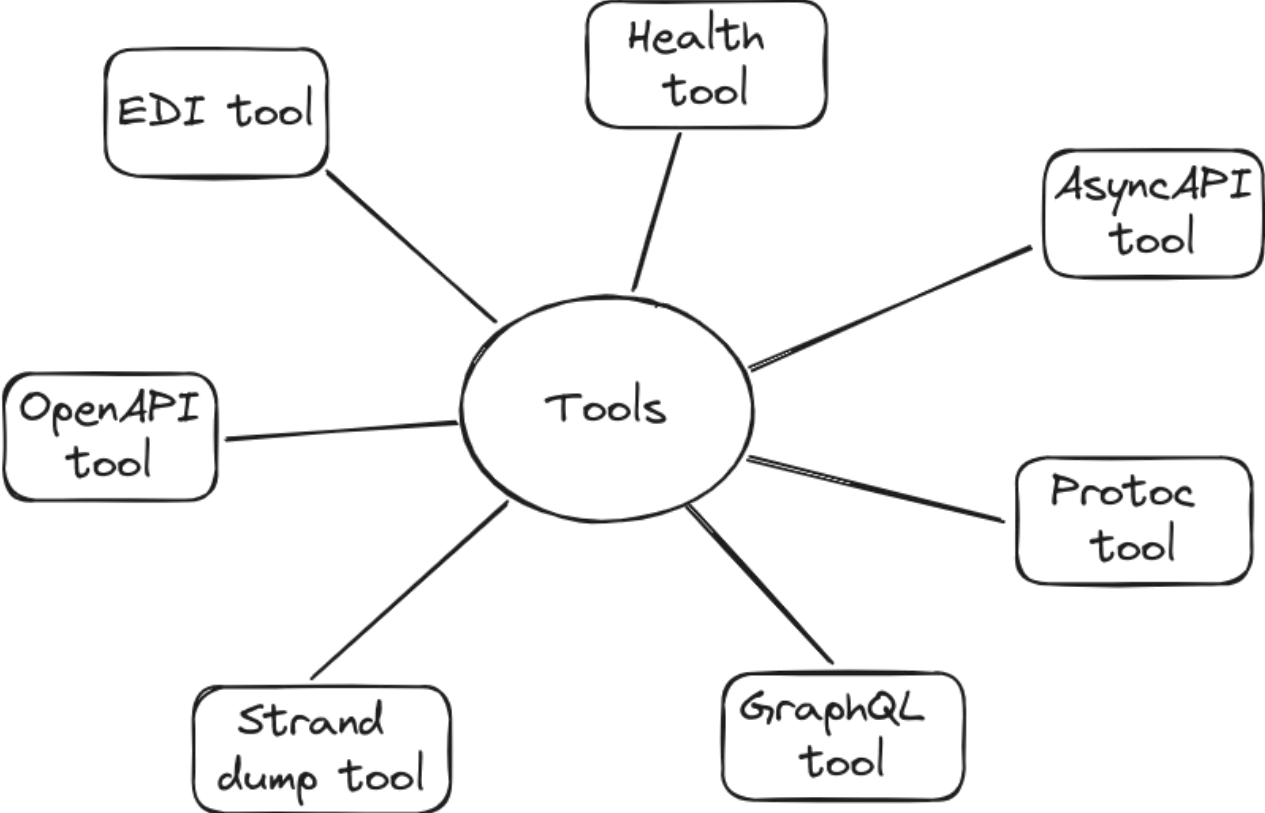
The ERD shows the following entities and their relationships:

- Challenge**: Attributes: id (string), title (string), createTime (Civil), templateFile (byte[]), readmefile (byte[]), difficulty (string), testCasesFile (byte[]), contests (ChallengesOnContests[]), author (User), submissions (Submission[]).
- Contest**: Attributes: id (string), title (string), readmefile (byte[]), urlTime (Civil), endTime (Civil), imageUrl (string), challenges (ChallengesOnContests[]), submissions (Submission[]), moderator (User).
- ChallengesOnContests**: Attributes: id (string), challenge (Challenge), contest (Contest), assignedTime (Civil).
- User**: Attributes: id (string), username (string), fullname (string), role (string), challenge (Challenge[]), moderatedContests (Contest[]), submissions (Submission[]).
- Submission**: Attributes: id (string), submittedTime (Civil), score (float), challenge (Challenge), contest (Contest), user (User).

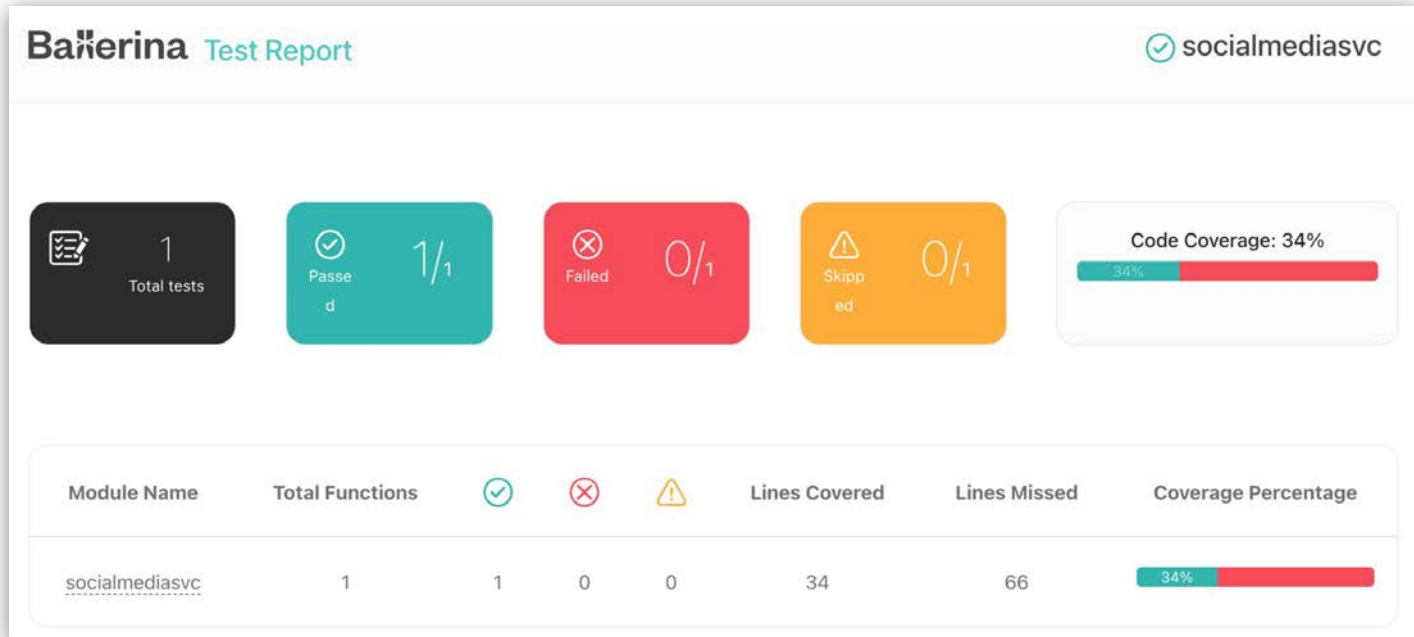
Relationships (Cardinality):

- Challenge (1) to Submission (1)
- Contest (1) to Submission (1)
- User (1) to Submission (1)
- Challenge (0..*) to ChallengesOnContests (1)
- Contest (0..*) to ChallengesOnContests (1)
- Challenge (1) to Contest (1)
- User (1) to Challenge (1)
- User (0..*) to Contest (0..*)
- User (0..*) to Challenge (0..*)
- User (0..*) to Submission (0..*)

Powerful Integration Tools



Better Tests



In-built AI Assistant

```
# Delete a user
#
# + id - The user ID of the user to be deleted
# + return - The success message or error message
Visualize
resource function delete users/[int id]() returns http:NoContent|error {
  _ = check socialMediaDb->execute(`DELETE FROM users WHERE id = ${id}`);
  return http:NO_CONTENT;
}
```



In-built AI Assistant

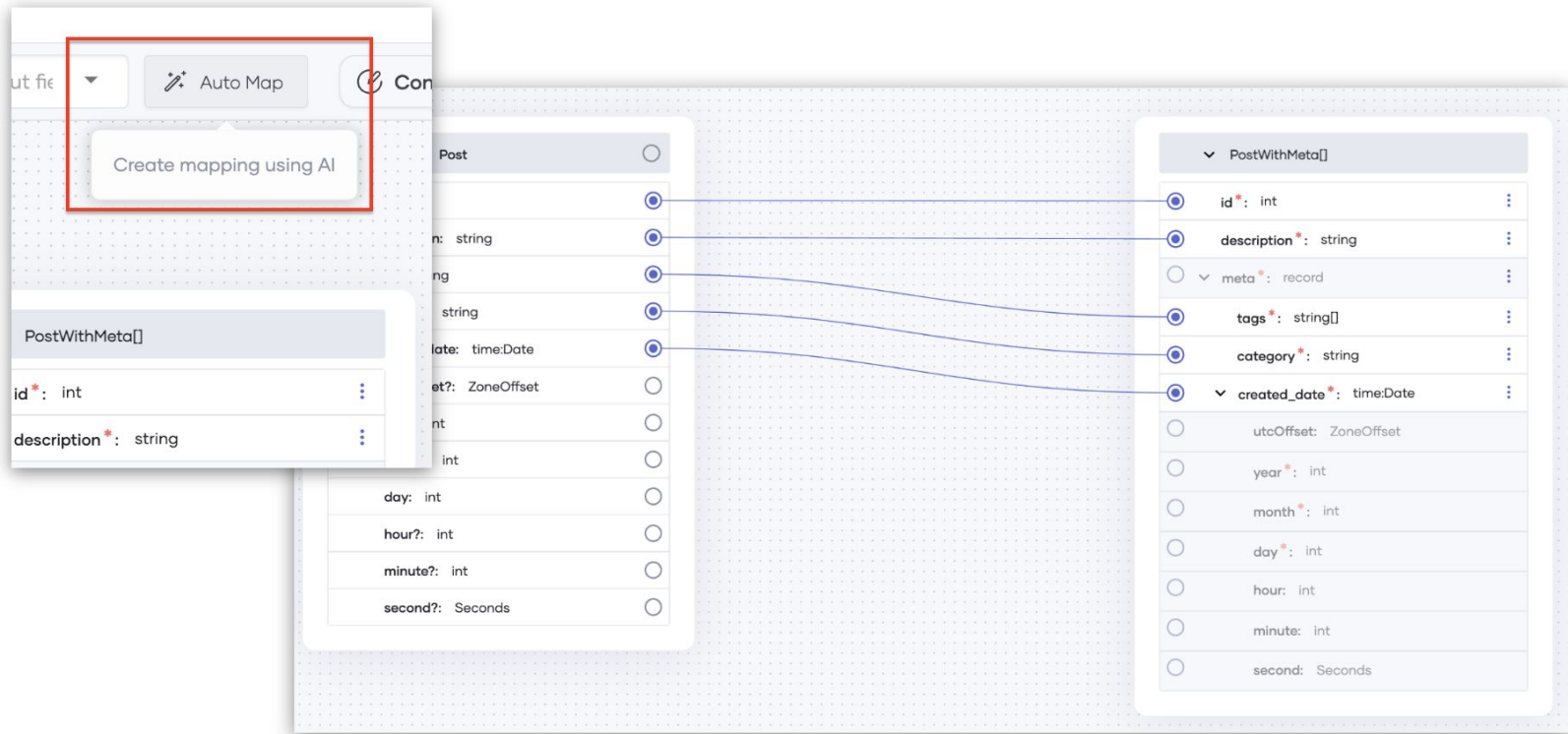
```
final http:Client sentimentEndpoint
  ⚡
Run | Debug | Try it | Visualize
More Actions...
  ⚡ Generate unit tests with copilot
  ⚡ Document this
  ⚡ Visualize
// Service-level error intercept
Visualize
public function createIntercepto
```

```
document this
test:Config{}
visualize
public function testSentimentAnalysis() returns error? {
  User userExpected = { id: 999, name: "foo", birthDate: {year: 0
  test:prepare(socialMediaDb).when("queryRow").thenReturn(userExp

  ⚡ http:Client socialMediaEndpoint = check new("localhost:9095/soc
  User userActual = check socialMediaEndpoint->/users/[userExpect

  test:assertEquals(userActual, userExpected);
}
```

In-built AI Assistant



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
```



Ballerina Code to Cloud is designed to allow developers to write code without thinking about the deployment platform.

This greatly simplifies the experience of developing and deploying Ballerina code in the cloud. It also enables using cloud-native technologies easily without in-depth knowledge.

Build a Docker image

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

```
(.) devcontainer.json 2 import ballerina.io;
.gitignore 3
Ballarina.toml 4 type Country record {
data.bal 5 string country;
Dependencies.toml

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS AZURE
datahandle bal build --cloud=docker
Compiling source
  anuruddha/datahandle:0.1.0

Generating executable
Generating artifacts

Building the docker image
[+] Building 16.7s (82/82) FINISHED
=> [internal] load .dockerignore 0.0s
=> [internal] transferring context: 2B 0.0s
=> [internal] load build definition from Dockerfile 0.0s
=> [internal] transferring dockerfile: 4.75kB 0.0s
=> [internal] load metadata for docker.io/ballerina/jvm-runtime:1.0 9.4s
=> [auth] ballerina/jvm-runtime:pull token for registry-1.docker.io 3.0s
=> [internal] load build context 0.7s
=> [internal] transferring context: 42.35MB 0.2s
=> [1/76] FROM docker.io/ballerina/jvm-runtime:1.0@sha256:c3963180a7423ab0c7915e0191555f6545046178d165b463106ff582c2449 0.0s
=> resolve docker.io/ballerina/jvm-runtime:1.0@sha256:c3963180a7423ab0c7915e0191555f6545046178d165b463106ff582c2449 3.9s
=> [2/76] COPY auth-native-2.9.0.jar /home/ballerina/jars/ 0.11s
=> [3/76] COPY ballerina-auth-2.9.0.jar /home/ballerina/jars/ 0.05s
=> [4/76] COPY ballerina-cache-3.6.0.jar /home/ballerina/jars/ 0.05s
=> [5/76] COPY ballerina-cloud-2.10.0.jar /home/ballerina/jars/ 0.05s
=> [6/76] COPY ballerina-constraint-1.3.0.jar /home/ballerina/jars/ 0.05s
=> [7/76] COPY ballerina-crypto-2.4.1.jar /home/ballerina/jars/ 0.05s
=> [8/76] COPY ballerina-file-1.8.1.jar /home/ballerina/jars/ 0.05s
=> [9/76] COPY ballerina-http-2.9.3.jar /home/ballerina/jars/ 0.05s
=> [10/76] COPY ballerina-http.httpsclient-2.9.3.jar /home/ballerina/jars/ 0.05s
=> [11/76] COPY ballerina-io-1.5.0.jar /home/ballerina/jars/ 0.05s
=> [12/76] COPY ballerina-jet-2.9.0.jar /home/ballerina/jars/ 0.05s
=> [13/76] COPY ballerina-log-2.8.1.jar /home/ballerina/jars/ 0.05s
=> [14/76] COPY ballerina-mime-2.8.0.jar /home/ballerina/jars/ 0.05s
=> [15/76] COPY ballerina-observe-2.9.0.jar /home/ballerina/jars/ 0.05s
=> [16/76] COPY ballerina-observe-1.1.0.jar /home/ballerina/jars/ 0.05s
=> [17/76] COPY ballerina-observe.mockextension-1.1.0.jar /home/ballerina/jars/ 0.05s
=> [18/76] COPY ballerina-os-1.7.0.jar /home/ballerina/jars/ 0.05s
=> [19/76] COPY ballerina-rt-2001.7.2.jar /home/ballerina/jars/ 0.05s
=> [20/76] COPY ballerina-task-2.4.0.jar /home/ballerina/jars/ 0.05s
=> [21/76] COPY ballerina-time-2.3.0.jar /home/ballerina/jars/ 0.05s
=> [22/76] COPY ballerina-xml-2.3.0.jar /home/ballerina/jars/ 0.05s
=> [23/76] COPY ballerini-observe-0.8.0.jar /home/ballerina/jars/ 0.05s
=> [24/76] COPY bcprov-jdk18on-1.74.jar /home/ballerina/jars/ 0.05s
=> [25/76] COPY bcprov-jdk18on-1.74.jar /home/ballerina/jars/ 0.05s
=> [26/76] COPY cache-native-3.6.0.jar /home/ballerina/jars/ 0.05s
=> [27/76] COPY commons-pool-1.5.0-rc2.jar /home/ballerina/jars/ 0.05s
=> [28/76] COPY constraint-native-1.3.0.jar /home/ballerina/jars/ 0.05s
=> [29/76] COPY crypto-native-2.4.1.jar /home/ballerina/jars/ 0.1s
=> [30/76] COPY file-native-1.8.1.jar /home/ballerina/jars/ 0.05s
=> [31/76] COPY http-native-2.9.3.jar /home/ballerina/jars/ 0.05s
=> [32/76] COPY io-native-1.5.0.jar /home/ballerina/jars/ 0.05s

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


Work with Kubernetes

```
bal build --cloud=k8s
```

```
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:

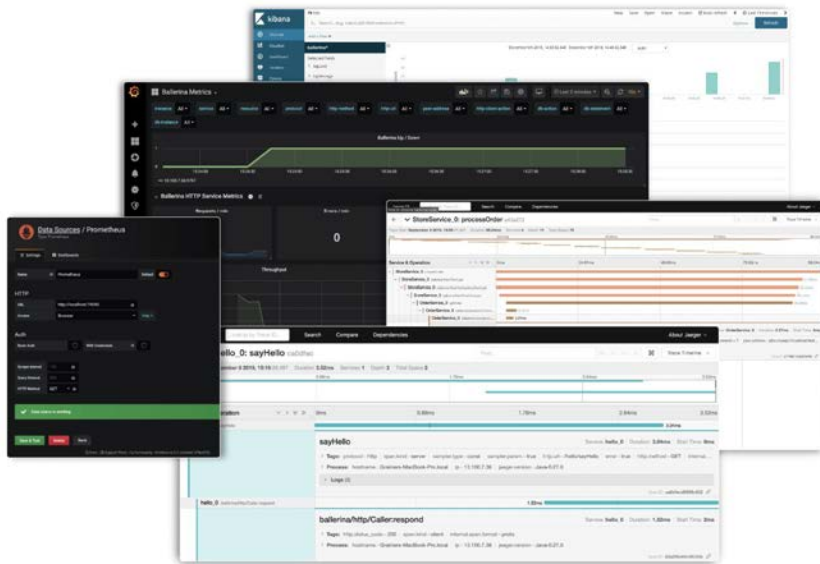
```
kubectl expose deployment helloworld-deployment --type=NodePort --name=helloworld-svc-local
```

```
target/bin/helloworld.jar
```

```
1 |---
2 | apiVersion: "v1"
3 | kind: "Service"
4 | metadata:
5 |   labels:
6 |     app: "testhello"
7 |     name: "testhello-svc"
8 | spec:
9 |   ports:
10 |     - name: "port-1-testhell"
11 |       port: 9090
12 |       protocol: "TCP"
13 |       targetPort: 9090
14 |   selector:
15 |     app: "testhello"
16 |     type: "ClusterIP"
17 | ---
18 | apiVersion: "apps/v1"
19 | kind: "Deployment"
20 | metadata:
21 |   labels:
22 |     app: "testhello"
23 |     name: "testhello-deployment"
24 | spec:
25 |   replicas: 1
26 |   selector:
27 |     matchLabels:
28 |       app: "testhello"
29 |   template:
30 |     metadata:
31 |       labels:
32 |         app: "testhello"
33 |     spec:
34 |       containers:
35 |         - image: "testhello:latest"
36 |           lifecycle:
37 |             preStop:
38 |               exec:
39 |                 command:
40 |                   - "sleep"
41 |                   - "15"
42 |           name: "testhello-deployment"
43 |       ports:
44 |         - containerPort: 9090
45 |           name: "port-1-testhell"
46 |           protocol: "TCP"
```

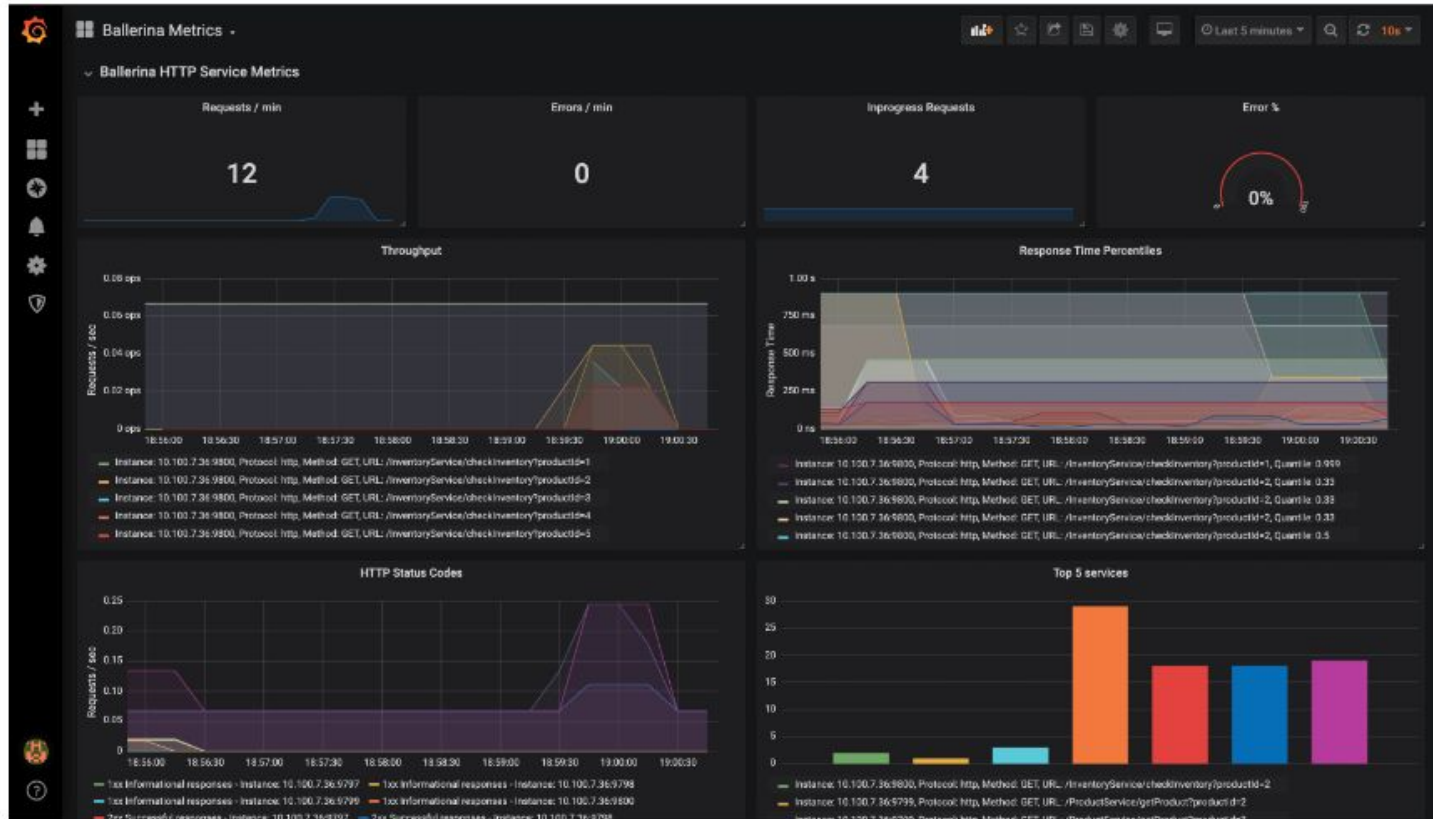
Operation

Observability in Ballerina

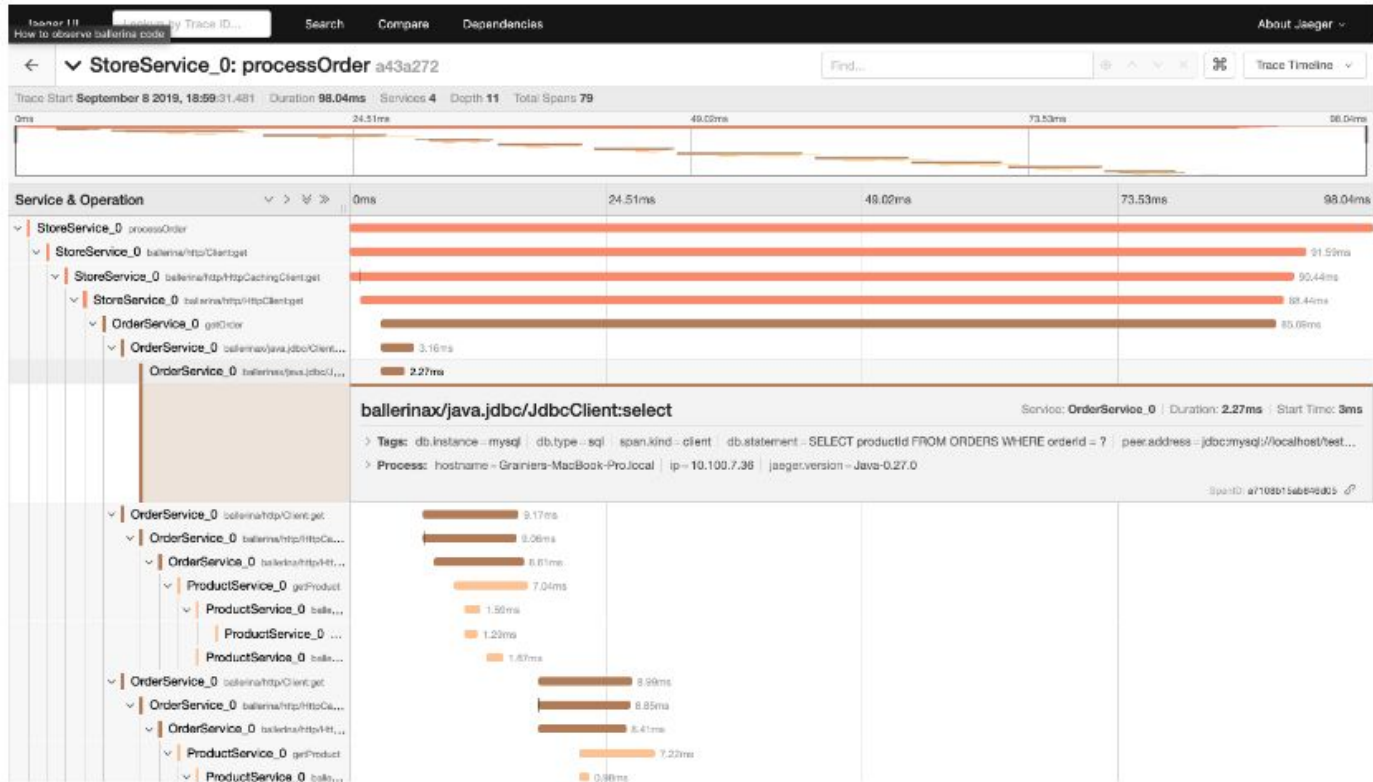


- Every Ballerina program is automatically observable by any Open Telemetry tool.
- Gives the complete control and visibility into the code's behavior and performance.
- It has 3 main pillars:
 - Metrics - Prometheus, Grafana
 - Tracing - Jaeger, Zipkin
 - Logging - Elastic Stack

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

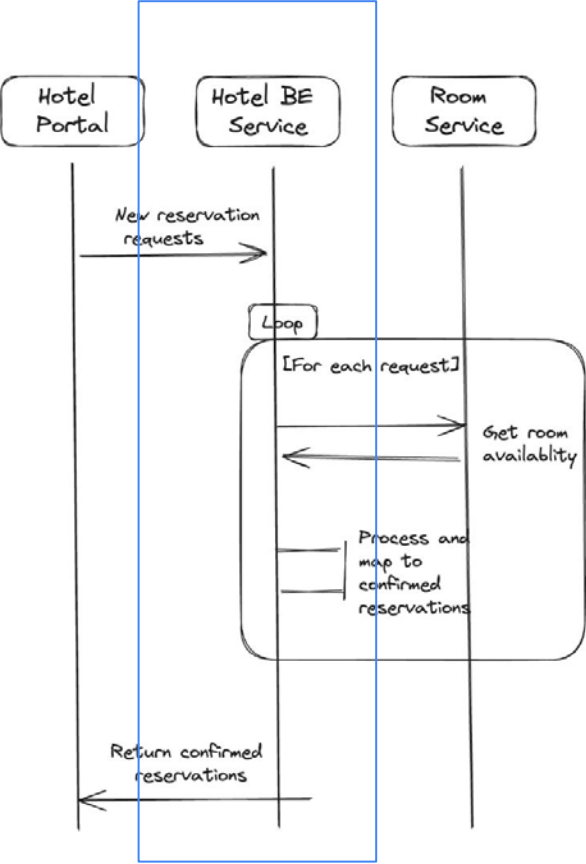
Set up Elastic Stack

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.

Demo

Hotel Management Service - REST Service



Ballerina Community



Tech Talks



Training



University Program



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](#)

Thank you!

If you have any further questions, please email contact@ballerina.io or raise them in the **Ballerina Discord server**.