

Ballerina

Swan Lake

An open-source programming language for the cloud that makes it
easier for integrations

July - 2023



Anupama Pathirage
Director of Engineering
@
WS02















anupama@wso2.com

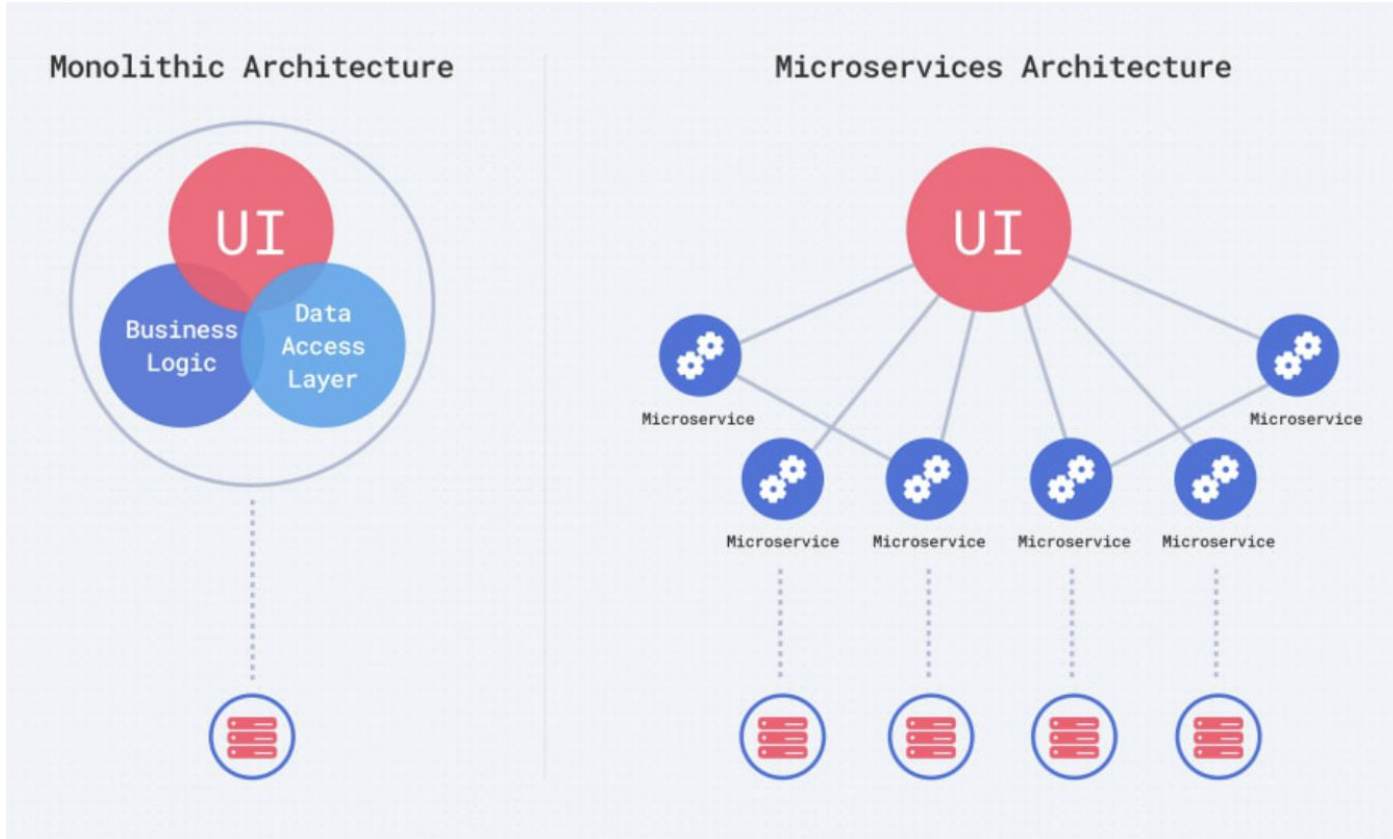


[@anupama_pathira](https://twitter.com/anupama_pathira)

Evolution of Application Architecture

	Development Process	Application Architecture	Deployment & Packaging	Application Infrastructure
~ 1980	Waterfall 	Monolithic 	Physical Server 	Datacenter 
~ 1990				
~ 2000	Agile 	N-Tier 	Virtual Servers 	Hosted 
~ 2010	DevOps 	Microservices 	Containers 	Cloud 

Disaggregation leads to more endpoints



INTEGRATION
PRODUCTS &
TECHNOLOGIES

ESB, BPM, EAI

NOT CLOUD-NATIVE

GENERAL-PURPOSE
LANGUAGES &
FRAMEWORKS

Java - SpringBoot,
Micronaut,
VertX, Quarkus

NodeJS - Express, VueJS

Python - Flask, FastAPI

WRONG ABSTRACTIONS

INTEGRATION
PRODUCTS &
TECHNOLOGIES

ESB, BPM, EAI

NOT CLOUD-NATIVE

Ballerina

Swan Lake

The
Integration
Language

GENERAL-PURPOSE
LANGUAGES &
FRAMEWORKS

Java - SpringBoot,
Micronaut,
VertX, Quarkus

NodeJS - Express, VueJS

Python - Flask, FastAPI

WRONG ABSTRACTIONS

Object Oriented Programming (OOP)

Everything is an object.

Encourages developers to create complex entities and processes with encapsulation and polymorphism using objects.

- Ideal for monolithic applications with complex logic and several boundaries.

But...

OOP increases the complexity of creating a system.

Data Oriented Programming (DOP)

Great for smaller services

- Communicate over the wire by sending and receiving data
- Typically, manipulate data from multiple services

Goal

decrease system complexity by separating code from data.

It is all about abstractions..

Data

- Representing the data - shape of data, separation from code (behaviour)
- 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

Data abstractions in Ballerina



Data abstractions in Ballerina

Ballerina records - Data is a first class citizen

```
type Instructor record {
    string id;
    string firstName;
    string lastName;
};

type Course record {
    string title;
    Instructor instructor;
};

type Student record {
    string id;
    string firstName;
    string lastName;
    int age;
    Course[] courses;
};
```

```
Student john = {
    id: "S12345678",
    firstName: "John",
    lastName: "Joe",
    age: 17,
    courses: [
        {
            title: "Data oriented programming",
            instructor: {
                id: "I12345678",
                firstName: "Anne",
                lastName: "Miller"
            }
        },
        {
            title: "Data structures and algorithms",
            instructor: {
                id: "I87654321",
                firstName: "Robert",
                lastName: "Williams"
            }
        }
    ]
};
```

- Make it easy to model the data that the program manipulates and move over the network.
- Can create custom record types to represent the data model.

Data abstractions in Ballerina

Flexible type system - Control openness

```
// Open record type allows additional fields
// with `anydata` values.
type Student record {
    string name;
    int age;
};

Student s1 = {
    name: "John Doe",
    age: 25,
    "country": "UK"
};

// Closed record type doesn't allow additional fields
type Instructor record {|
    string name;
    string language;
|};

Instructor i1 = {
    name: "Anne Williams",
    language: "Java"
};
```

- Robustness principle : Be conservative in what you send, be liberal in what you accept.
- Open records allow fields other than those specified. The type of unspecified fields is anydata.

Data abstractions in Ballerina

Flexible type system - Represent optionality and unions

```
type Student record {|
    string firstName;
    string lastName;
    string country?; //Optional field
    string language?; //Optional field
    int? score;      //Nilable field
    int|string systemId; //Union field
|};

Student s1 = {
    firstName: "John",
    lastName: "Doe",
    country: "USA",
    score: (),
    systemId: 123
};
```

- Optional fields can be omitted when creating a value of the record type.
- T1|T2 is the union of the sets described by T1 and T2.

Data abstractions in Ballerina

JSON/XML are first class types

```
// Create a `json` value.
json n = null;
json i = 21;
json s = "str";
json a = [1, 2];
json m = {"x": n, "y": s, "z": a};
json[] arr = [m, {"x": i}];

// Get the `json` value from the string.
string rawData = "{\"id\": 2, \"name\": \"Georgy\"}";
json j = check rawData.fromJsonString();

// Access the fields of `j` using field access.
string name = check j.name;

// Convert the `json` into a user-defined type.
Student student = check j.cloneWithType();
```

```
// An XML element. There can be only one root element.
xml x1 = xml `<book>The Lost World</book>`;

// An XML processing instruction.
xml x2 = xml `<?target data?>`;

// An XML comment.
xml x3 = xml `<!--I am a comment-->`;

// An XML text.
xml x4 = xml `Hello, world!`;

// `xml:createText` can be used to convert a string to `xmlText`.
string hello = "Hello";
string world = "World";
xml:Text xmlString = xml:createText(hello + " " + world);

// Creates an XML value.
xml xmlValue = xml `<name>Sherlock Holmes</name>
<details>
  <author>Sir Arthur Conan Doyle</author>
  <language>English</language>
</details>`;

//Access XML value
xml xmlHello = xml `<para id="greeting">Hello</para>`;
string id = check xmlHello.id;
```

Data abstractions in Ballerina

Powerful query support

```
public function main() returns error? {
    Country[] countries = getCountries();

    json summary =
        from var {country, continent, population, cases, deaths} in countries
        where population >= 100000 && deaths >= 100
        let decimal caseFatalityRatio = <decimal>deaths / <decimal>cases * 100
        order by caseFatalityRatio descending
        limit 10
        select {country, continent, population, caseFatalityRatio};
    io:println(summary);
}
```

Network abstractions in Ballerina

Clients

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

type Country record {
    string country;
    int population;
    string continent;
    int cases;
    int deaths;
};

Run | Debug | Visualize
public function main() returns error? {
    http:Client diseaseEp = check new ("https://disease.sh/v3");
    Country[] countries = check diseaseEp->/covid\~19/countries;
    io:println(countries);
}
```

- Client applications consume network services.
- Ballerina supports defining client objects to allow a program to interact with remote network services using remote methods.
- Payload data-binding allows directly binding the response payload to a given subtype of anydata. It does this by mapping a given HTTP content-type to one or more Ballerina types.

Network abstractions in Ballerina

Services

```
import ballerina/http;

Run | Debug | Try it | Visualize
service / on new http:Listener(9090) {

    Visualize
    resource function get albums() returns Album[] {
        return albums.toArray();
    }

    Visualize
    resource function post albums(Album album) returns Album {
        albums.add(album);
        return album;
    }

    Visualize
    resource function put albums(Album album) returns Album|http:NotFound {...}
    }

    Visualize
    resource function delete albums/[string id]() returns Album|http:NotFound {...}
    }
}
```

- First-class language concepts for providing and consuming services.
- Libraries provide protocol-specific Listeners, which receive network input and dispatch to services
- Service support two interface styles
 - remote methods - support RPC style (used for gRPC)
 - Resources methods - support RESTful style (used for HTTP and GraphQL)

Network abstractions in Ballerina

Services

```
import ballerina/graphql;
```

Run | Debug | Try it | Visualize

```
service /graphql on new graphql:Listener(9090) {  
    |  
    | Visualize  
    | resource function get greeting() returns string {  
    |     | return "Hello, World";  
    | }  
}
```

```
import ballerina/grpc;
```

```
@grpc:Descriptor {  
    | value: GRPC_SERVICE_SIMPLE_DESC  
    | }  
}
```

Run | Debug

```
service "HelloWorld" on new grpc:Listener(9090) {  
    |  
    | Visualize  
    | remote function hello(string request) returns string {  
    |     | return "Hello " + request;  
    | }  
}
```

Concurrency in Ballerina

```
// Asynchronous function calls
future<int> fut = start foo(); // `start` calls a function asynchronously
int|error x = wait fut; // `wait` for `future<T>` gives `T|error`.

// Named Workers - run concurrently with the function's default worker
// and other named workers.
final string greeting = "Hello";
worker A {
    io:println(greeting + " from worker A");
}

worker B {
    io:println(greeting + " from worker B");
}

// Transactions
retry transaction {
    doStage1();
    doStage2();
    check commit;
}
```

Concurrency-related features are built into the language as first-class citizens, and they map directly onto sequence diagrams.

- **Strand**
a logical thread of control assigned to every worker
- **Named workers**
run concurrently with the function's default worker and other named workers.
- **Asynchronous function calls**
a function asynchronously and the function runs on a separate logical thread (strand).

Concurrency safety in Ballerina

Ballerina's main goal for service concurrency is to achieve decent performance and a decent level of safety.

- Listener can have multiple threads serving incoming requests concurrently
- There are no undetected data races that lead to wrong results

Locks

- allows the access of mutable state from multiple strands running on separate threads

Isolated functions

- a function that is concurrency safe if its arguments are safe.
- Allowed to access a mutable state only through its parameters

The readonly type

- Represents immutable values

Code to cloud support in Ballerina

```
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:Docke              - complete 2/2

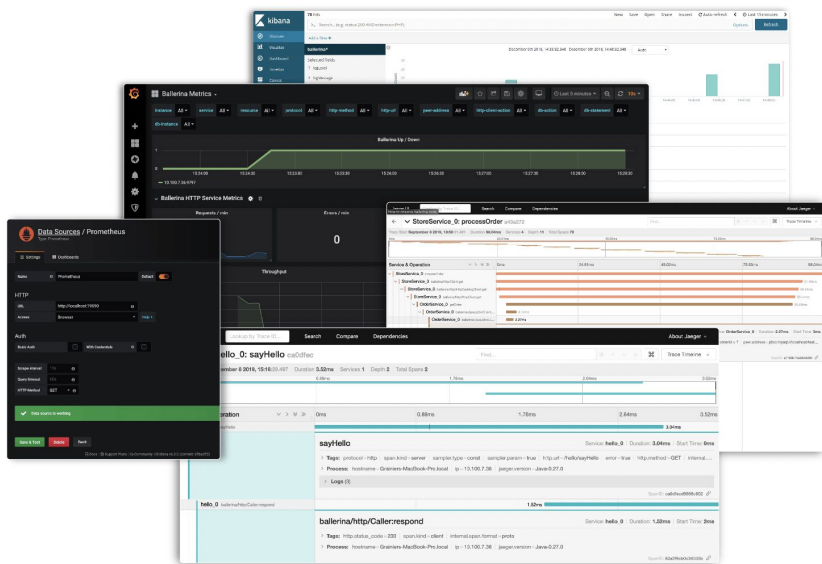
Execute the below command to deploy the Kubernetes artifacts:
kubect apply -f /Volumes/data/ballerina/code/testBalProject/target/kubernetes/helloworld

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

target/bin/helloworld.jar
```

- Greatly simplifies the experience of developing and deploying Ballerina code in the cloud.
- Supports generating the deployment artifacts for the Docker, K8s, Azure functions.
- Use Cloud.toml to change the generated artifact values.

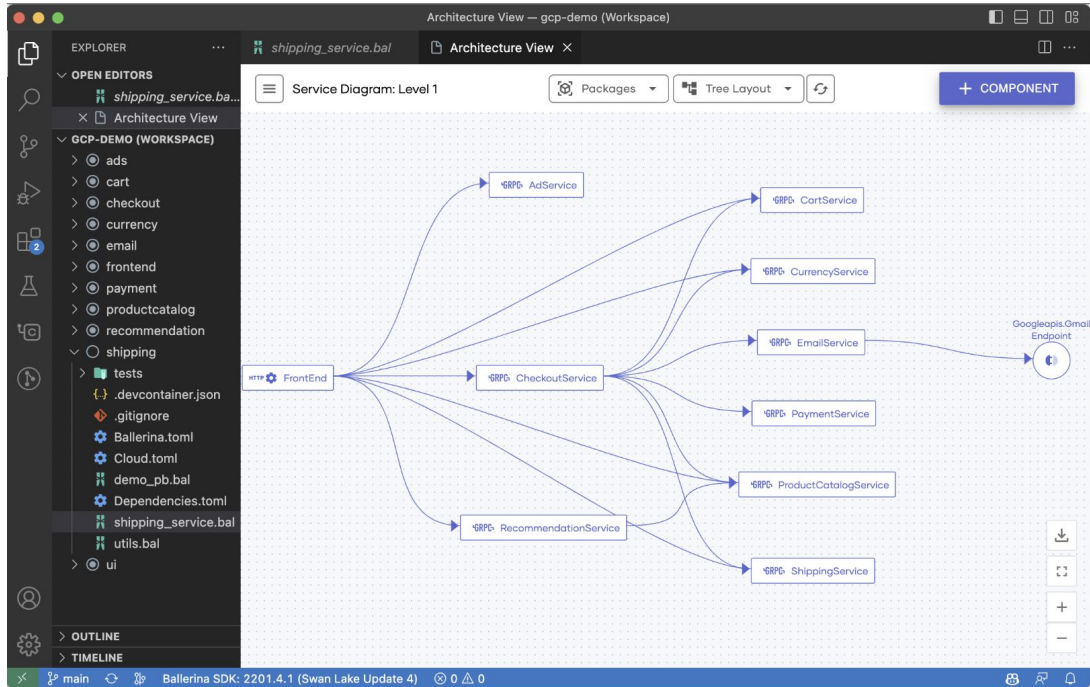
Observability in Ballerina



- 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
 - Logging - Elastic Stack

Powerful visualizing and design capabilities

Architectural design view



Powerful visualizing and design capabilities

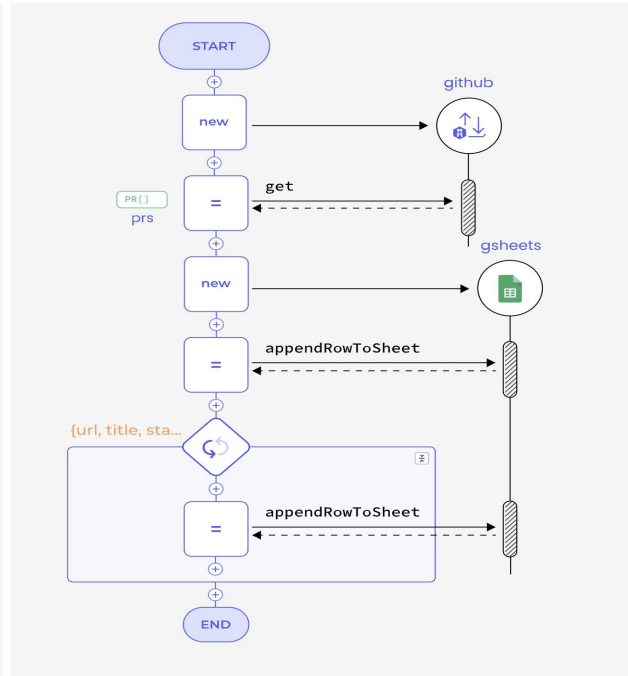
Text and graphical syntax parity

```
type PR record {
  string url;
  string title;
  string state;
  string created_at;
  string updated_at;
};

public function main() returns error? {
  http:Client github = check new ("https://api.github.com/repos");
  map<string> headers = {
    "Accept": "application/vnd.github.v3+json",
    "Authorization": "token " + githubPAT
  };
  PR[] prs = check github->get(string `${repository}/pulls`, headers);

  sheets:Client gsheets = check new ({auth: {token: sheetsAccessToken}});
  check gsheets->appendRowToSheet(spreadSheetId, sheetName,
    ["Issue", "Title", "State", "Created At", "Updated At"]);

  foreach var {url, title, state, created_at, updated_at} in prs {
    check gsheets->appendRowToSheet(spreadSheetId, sheetName,
      [url, title, state, created_at, updated_at]);
  }
}
```



Powerful visualizing and design capabilities

Data mapping

The image displays a development environment with two main panels. The left panel shows a code editor with Go code for a service named `calendar`. The code includes functions for handling new events, deleting events, and updating events. A design section shows the function signature for `calEventToTrelloCard` and its implementation, which maps fields from a `calendar.Event` to a `calendar.Event` object.

```
52 Run | Debug | Try It
53 service calendar:CalendarService on calendarListener {
54
55     remote function onNewEvent(calendar:Event payload) returns error {
56         do {
57             // Add the card to the Trello list
58             trello:Cards card = calEventToTrelloCard(payload);
59             _ = check trello->addCards(card);
60
61             // Send SMS notification
62             string twilioMsg = calEventToMessage(payload);
63             _ = check twilio->sendSms(twFromMobile, twToMobile, twMsg);
64
65         } on fail var e {
66             // Log the error and add the event to the dead letter channel
67             log:printError(string "Failed to process the calendar event: ", e);
68         }
69     }
70
71     remote function onEventDelete(calendar:Event payload) returns error {
72
73     }
74
75     remote function onEventUpdate(calendar:Event payload) returns error {
76
77     }
78
79     Design
80     function calEventToTrelloCard(calendar:Event calEvent) returns trello:Cards {
81         name: calEvent.summary,
82         due: calEvent.end?.dateTime,
83         idList: trelloListId,
84         desc: string "New event is created on Google Calendar: ${calEvent.summary}
85             The event starts on ${calEvent.start?.dateTime ? ""} and ends on
86             ${calEvent.end?.dateTime ? ""}";
87     };
88
89     Design
90     function calEventToMessage(calendar:Event calEvent) returns string {
91         string "New event is created : ${calEvent.summary} ?:" start
92         ends on ${calEvent.end?.dateTime ? ""}";
93     };
94
95     function toDeadLetterChannel(calendar:Event calEvent, error e) {
96
97     }
98 }
```

The right panel shows a "Data Mapper" tool titled "Data Mapper: calEventToTrelloCard". It visualizes the mapping between the source object `calEvent: trigger.google.calendar.Event` and the target object `trello:Cards`. The source object has fields like `kind`, `etag`, `id`, `status`, `htmlLink`, `created`, `updated`, `summary`, `description`, `location`, `calendar`, `creator`, `organizer`, `start`, `date`, `dateTime`, `timeZone`, `end`, `endTimeUnspecified`, `recurrence`, `recurrenceStart`, and `recurrenceEnd`. The target object has fields like `closed`, `desc`, `due`, `fileSource`, `idAttachmentCover`, `idBoard`, `idCardSource`, `idLabels`, `idList`, `idMembers`, `keepFromSource`, `labels`, `name`, `pos`, `subscribed`, and `urlSource`. Blue lines connect the source fields to the target fields, illustrating the data mapping process.

Powerful visualizing and design capabilities

Service designing

The image shows a side-by-side comparison of a code editor and a service design tool. The left pane displays a Scala file named `consuming_services.bal` with the following code:

```
19 resource function get albums() returns Album[] {
20     return albums.toArray();
21 }
22
23 Visualize
24 resource function get albums/[string id]() returns Album|http:NotFound
25     Album? album = albums[id];
26     if album is () {
27         return http:NOT_FOUND;
28     } else {
29         return album;
30     }
31
32 Visualize
33 resource function post albums(@http:Payload Album album) returns Album
34     albums.add(album);
35     return album;
36
37 Visualize
38 resource function put albums/[string id](@http:Payload Album album) returns Album|http:NotFound
39     Album? existingAlbum = albums[id];
40     if existingAlbum is () {
41         return http:NOT_FOUND;
42     } else {
43         albums.add(album);
44         return album;
45     }
46
47 Visualize
48 resource function delete albums/[string id]() returns Album|http:NotFound
49     Album? album = albums[id];
50     if album is () {
51         return http:NOT_FOUND;
52     } else {
53         _ = albums.remove(id);
54         return album;
55 }
```

The right pane shows the service design tool interface for a service named `consuming_services`. It displays a list of resources with their methods and responses:

- GET** `albums`: Returns `Album[]` (200).
- GET** `albums/[string id]`
- POST** `albums`
- PUT** `albums/[string id]`
- DELETE** `albums/[string id]`

The interface includes a '+ Resource' button, an 'Expand All' toggle, and icons for editing and deleting each resource.

Ballerina is a full platform

- VSCode plugin
 - Source and graphical editing
 - Debugging
- Tools for working with OpenAPI, GraphQL schemas, gRPC schemas
- Generate API Documentation & test framework
- Ballerina standard library and extended library
- Ballerina Central (<https://central.ballerina.io/>)
 - Module sharing platform

Join with
Ballerina
Community



Discord : <https://discord.gg/ballerinalang>



SO <https://stackoverflow.com/questions/tagged/ballerina>



Twitter <https://twitter.com/ballerinalang>



GitHub : <https://github.com/ballerina-platform>

THANK YOU