Ballerina - A Cloud Native Programming Language

Anupama Pathirage
Director of Engineering
@
WSO2

anupama@wso2.com

@anupama_pathira

"APIs enable organizations to create new business models and revenue streams by exposing their data and services to third-party developers and partners."

- Forrester

INTEGRATION PRODUCTS & TECHNOLOGIES

ESB, BPM, EAI

NOT CLOUD NATIVE

Ballerina

The Integration Language

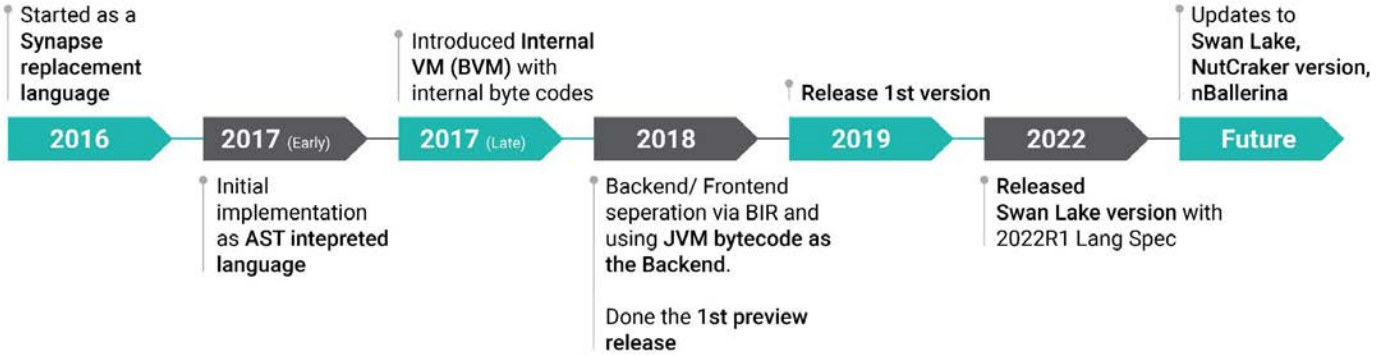
GENERAL PURPOSE LANGUAGES & FRAMEWORKS

Java	-	SpringBoot, Micronaut, VertX, Quarkus
NodeJS	-	Express, VueJS
Python	-	Flask, FastAPI

WRONG ABSTRACTIONS

What is Ballerina?

- Started in 2016 by WSO2.
- A general-purpose programming language.
- Specializes in solving integration & cloud-based problems by providing the right level of language abstractions and tools.
- Open-source project and driven by the community.



What is unique about Ballerina?

Data Oriented

Ballerina

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

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

// Prints the top 10 countries having the highest case-fatality ratio.
public function main() returns error? {
    http:Client diseaseEp = check new ("https://disease.sh/v3");
    Country[] countries = check diseaseEp->get("/covid-19/countries");

    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);
}
```

- Sharing code over the network is not considered safe! Instead move data.
- Emphasizes plain data - data that is independent of any code used to process data
- Make it easy to model the data that the program manipulates and move over the network.
- Plain data maps straightforwardly to and from JSON
- XML support - integrates functionality similar to XQuery
- Language-integrated query with SQL-like syntax

Network Oriented - Type system as a schema language

Ballerina

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

// Describes both the payload on the wire
// and data in memory.
type Country record {
    string country;
    int population;
    string continent;
    int cases;
    int deaths;
};

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

- Get some data over the wire and bind it to a data structure in the language in order to manipulate it.
- Ballerina's type system also works as a schema
  - Data binding is just a type cast
  - Most similar to TypeScript
- Eliminates the data binding concept for well-known data formats and network data structures

Network Oriented - Services and service typing

Ballerina

```
type Album readonly & record {|
    string id;
    string title;
    string artist;
    decimal price;
|};

service / on new http:Listener(port) {
    resource function get albums() returns Album[] {
        return albums.toArray();
    }

    resource function get albums/[string id]() returns Album|http:NotFound {
        Album? album = albums[id];
        if album is () {
            return http:NOT_FOUND;
        } else {
            return album;
        }
    }

    resource function post albums(@http:Payload Album album) returns Album {
        albums.add(album);
        return album;
    }
}
```

- 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, named by verbs, support RPC style (used for gRPC)
  - resources, named by method (e.g. GET) + noun, support RESTful style (used for HTTP and GraphQL)
- Inherently concurrent

Network Oriented - Service clients

Ballerina

```
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]);
    }
}
```

- Client applications consume network services.
- Therefore, Ballerina supports defining client objects to allow a program to interact with remote network services using remote methods.

Concurrent and Reliable

Ballerina

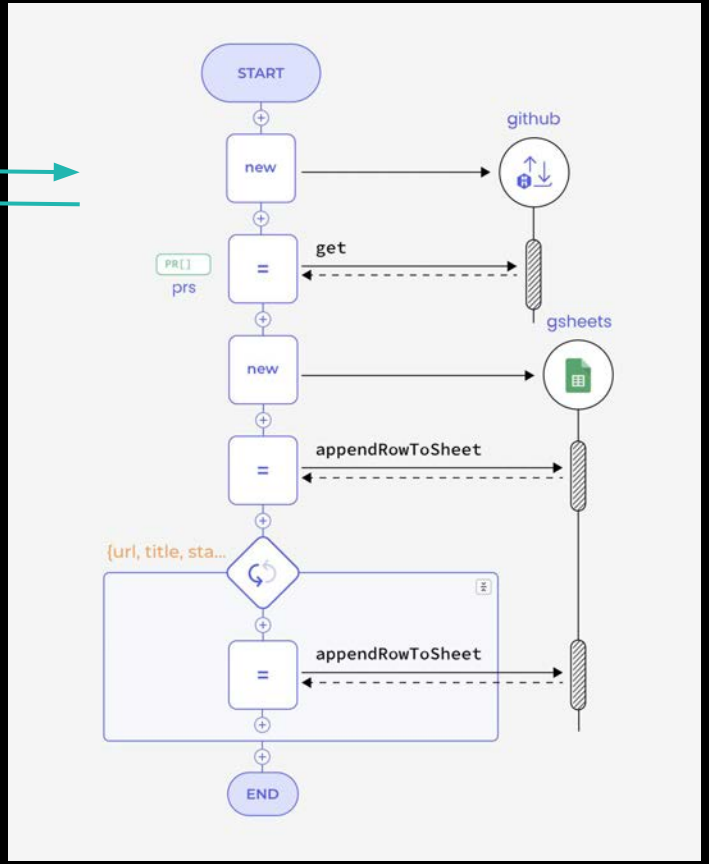
```
// Asynchronous function calls
future<int> fut = start foo(); // `start` calls a function asynchronously.
int|error x = wait fut; // `wait` for `future<T>` gives `T|error`.

// 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;
}
```

- With more and more applications needing to support network interaction, concurrency becomes important for handling scale.
    - **Asynchronous function calls** - calls a function asynchronously and the function runs on a separate logical thread
    - **Workers** - Represents a single strand of a function invocation.
    - **A strand** - is a logical thread of control assigned to every worker, which is multitasked cooperatively instead of preemptively.
- Ballerina runtime has built-in support for interacting with a transaction manager.
    - language provides syntax for delimiting transactions.

Text and graphical syntax parity

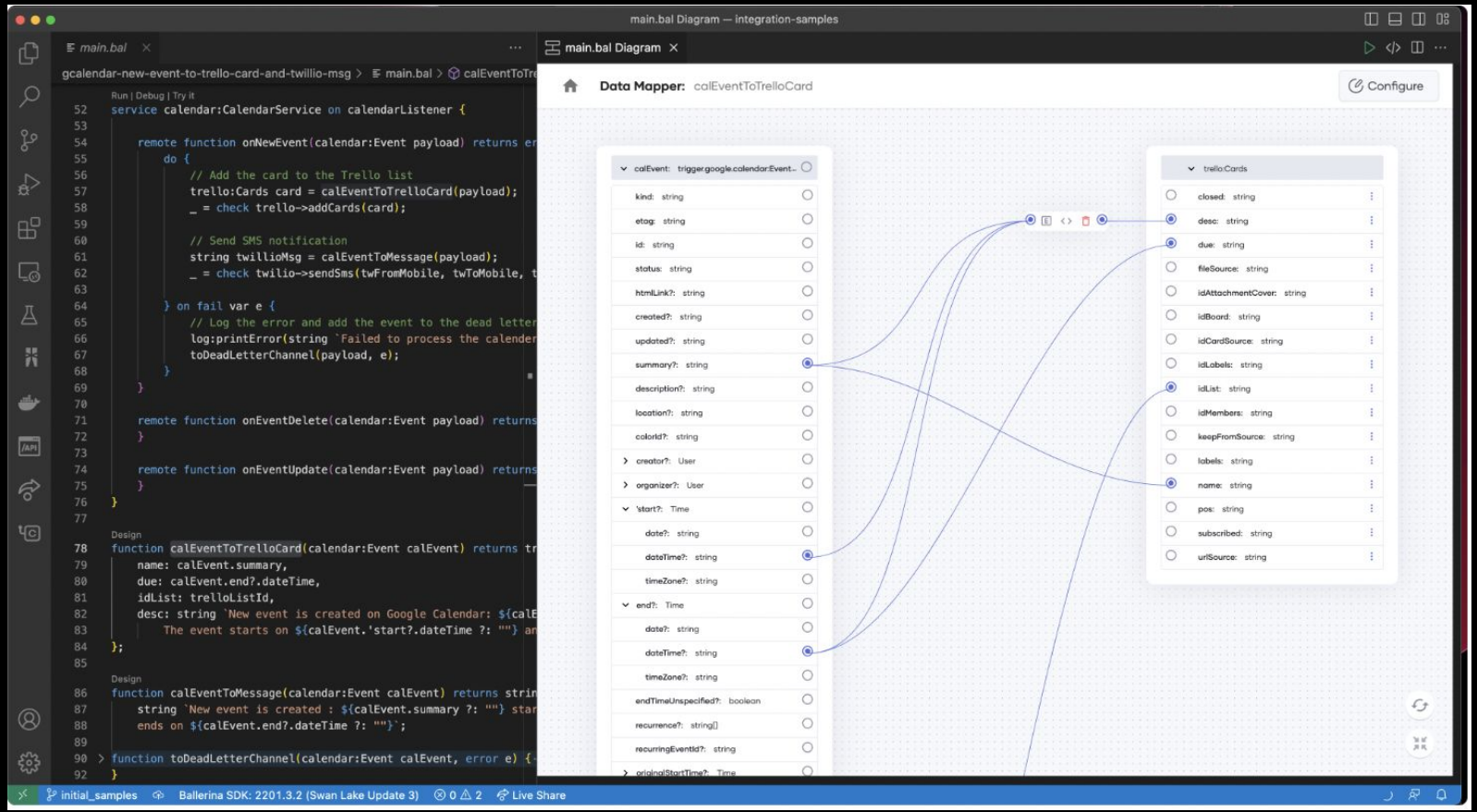
Ballerina

```
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]);
    }
}
```

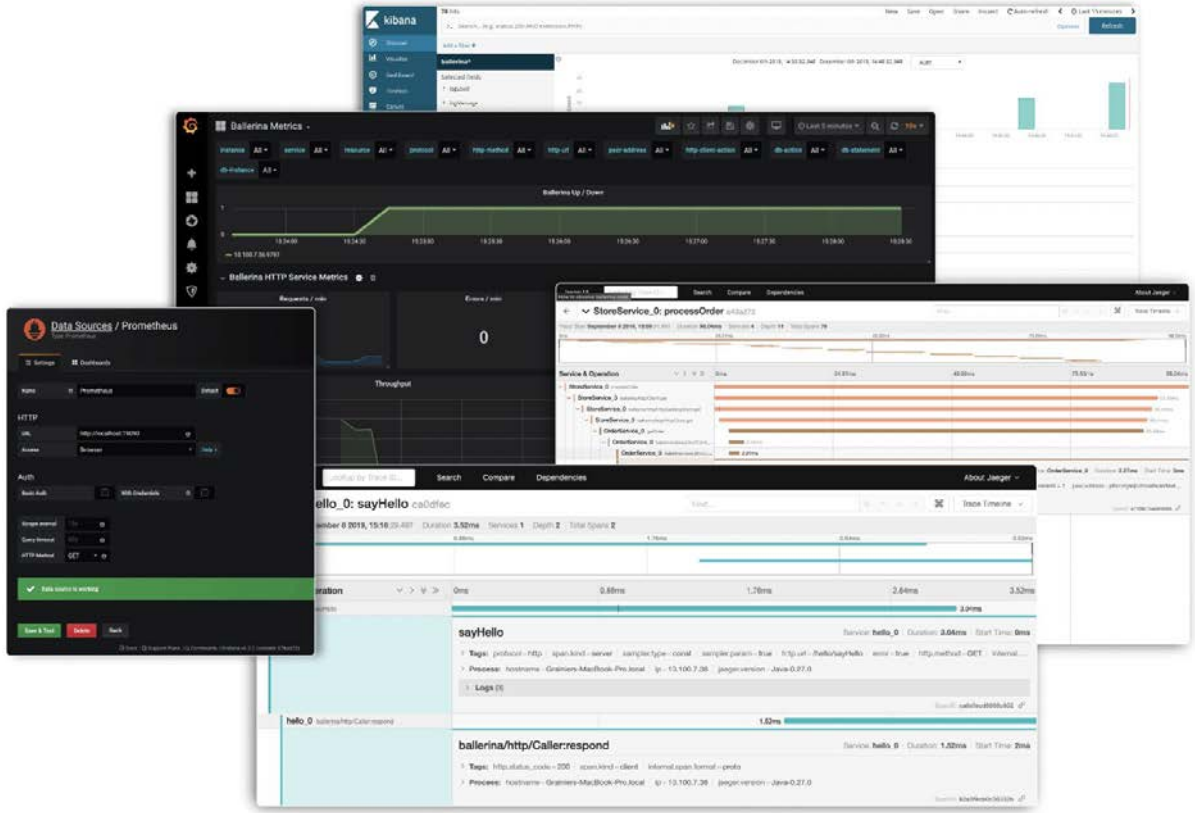


Data Mapping

Ballerina



Built-in Observability



- 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

Code to cloud

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: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
```

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

```
[container.image]
repository= "ballerina"
name="hello-world"
tag="v1"

[cloud.deployment]
min_memory="100Mi"
max_memory="256Mi"
min_cpu="200m"
max_cpu="500m"
```

Ballerina is NOT a JVM language!

- Compiler is written in Java and generates JVM bytecode.
  - Provides Java interoperability
- Semantics of the language is carefully designed to be independent from Java and JVM.
- Working on another implementation of Ballerina that generates native bytecode.
- Recently introduced the --native flag, which generates a GraalVM native executable when building a Ballerina project.

Ballerina offers not just the language, but the 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
- Integration to Choreo by WSO2 for observability, CI/CD and DevOps

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

Thanks!

wso2.com