

Ballerina and Jolie: The Frontier of Microservices Programming

*Anjana Fernando; Saverio Giallorenzo; Claudio Guidi; Sameera Jayasoma;
Balint Maschio; Jacopo Mauro; Fabrizio Montesi; Marco Montesi; Marco
Peressotti; Matthias Dieter Wallnöfer; Lakmal Warusawithana*



What's on the frontier?

- Service Orientation
- Communication Primitives
- Manifest Workflow
- Access point
- Clearly defined dependencies
- Use of API's
- Messages types and value
- Interoperability
- Architectural design extraction
- Architectural programming
- Build-in observability



<https://www.jolie-lang.org/>

Jolie

The service-oriented
programming language

 Download

 Docs

 Vision

Balint Maschio

The Jolie Team

9-9-2020 Microservices Conference

```
type HelloRequest {
    name:string
}

interface HelloInterface {
    requestResponse:
        hello( HelloRequest )( string )
}

service HelloService {
    execution { concurrent }

    inputPort HelloService {
        location: "socket://localhost:9000"
        protocol: http { format = "json" }
        interfaces: HelloInterface
    }

    main {
        hello( request )( response ) {
            response = "Hello " + request.name
        }
    }
}
```

The content of this presentation is based on the syntax of jolie version 1.10.0 that has been released as a beta version



Interfaces and message types

```
type Op1Request: void{
  nodeA:string ( regex(".*@.*\\.\\.*)" )
  nodeB:string( enum(["hello","homer","simpsons"]))
}

type Op2Response:void{
  nodeA:string
}

type Op2Request:void{
  nodeA:int ( ranges( [1,4], [10,20]) )
}

interface MyInterface{
  RequestResponse:
    op1 (Op1Request) (Op1Response)
  OneWay:
    op2 (Op2Request)
}
```

This is an interface declaration in Jolie

- Definition of interfaces
- Definition of operation
- Definition of their types
- With some data validation mechanism build (refined types)

What impact this can have:

- A single point to define the contract of the service
- Contract first by design
- A strongly typed message definition reduce the need of control implementation on the formal validity of the data



Implementation

```
service FirstService () {  
  execution{ concurrent }  
  inputPort AccessPointA {  
    Location: "local"  
    Protocol: sodep  
    Interfaces: MyInterface  
  }  
  inputPort AccessPointB {  
    Location: "socket://localhost:80"  
    Protocol: http  
    Interfaces: MyInterface  
  }  
  main{  
    [op1(request)(response){  
      response.nodeA =  
        request.nodeA + " " + response.nodeB  
    }]  
    [op2(request)(response)]{ //do something }  
  }  
}
```

Implementing a microservice

- Clear boundary of service implementation
- Definition of two separate access points (inputPorts)
- Clear implementation area
- No extra code unmarshalling the incoming message

What impact this can have:

- Multiple service implementation in the same file
- Multiple inputPorts in the same service with different protocols defined
- Implementation is almost independent to the communication protocol



Invoking other services

```
from MyPackage import MyInterface

service ClientService (){

    outputPort CallingPort {
        Location: "socket://localhost:80"
        Protocol: http
        Interfaces: MyInterface
    }

    main{
        request.nodeA = "my@email.com"
        request.nodeB = "Hello"
        op1@CallingPort(request) (response)
        undef(request)
        request.nodeA = 1
        op2@CallingPort(request)
    }
}
```

A service can also play as a client

- Definition of outputPort (invoking endpoint)
- Specific primitives for service invocation (SolicitResponse and OneWay)
- No Marshalling code

What impact this can have:

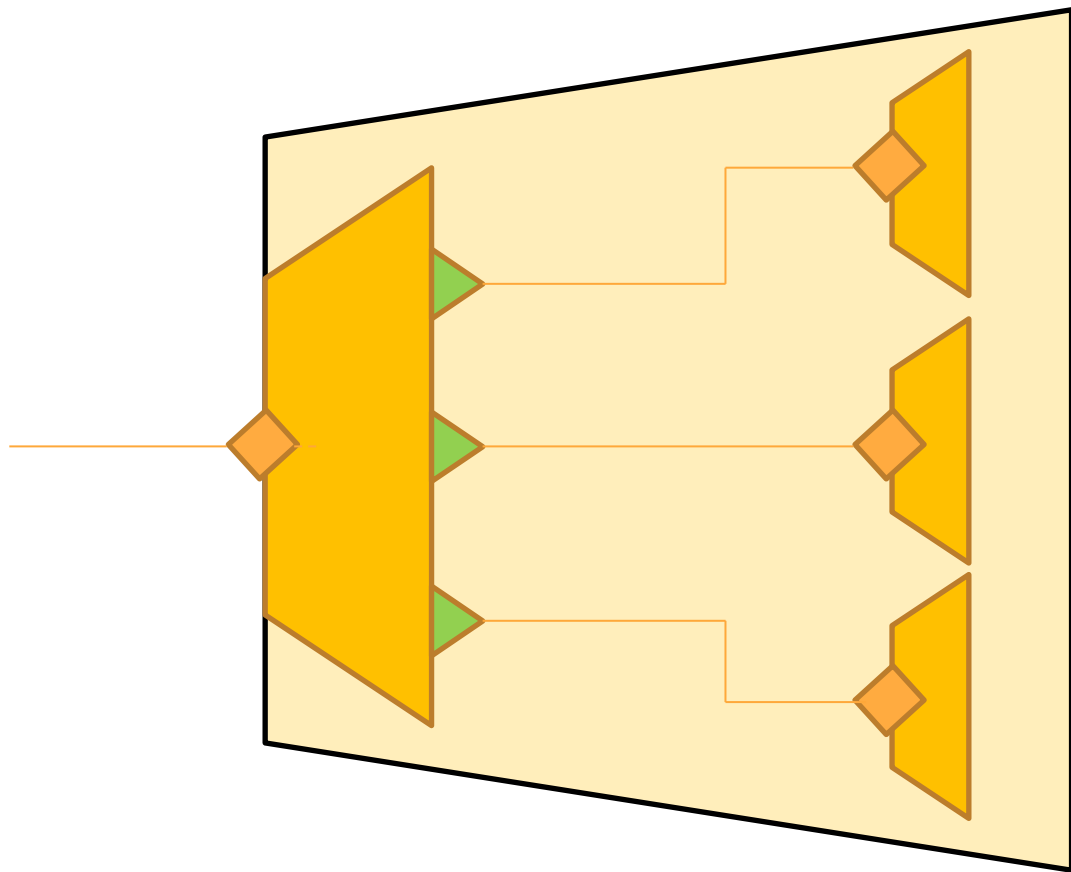
- It is possible to define multiple outputPorts in the same service, thus it is possible to invoke more services from within the same service
- We do not need anything about the communication in the behaviour, just using the communication primitive. All the communication details are managed by the ports



Architectural composition in Jolie

- Embedding
- Orchestration
- Aggregation
- Internal Service
- Couriers
- Rederiction
- Collections

Embedding





Embedding and orchestration

```
from MyPackage import MyInterface
service DummyService () {
  execution { concurrent }
  embed firstService at FirstServicePort
  outputPort CallingPort {
    Location: "socket://localhost:80"
    Protocol: http
    Interfaces: MyInterface }
  inputPort EntryPointB {
    Location: "socket://localhost:82"
    Protocol: sodep
    Interfaces: MyDummyInterface }
  main {
    [ dummyOp (request) (response) {
      requestOp1.nodeA = request.someNodeA
      requestOp1.nodeB = request.someNodeB
      op1@CallingPort (requestOp1) (responseOp1)
      requestOp2.nodeA = 1
      op2@FirstServicePort (requestOp2)
      response.someNodeA = responseOp1.nodeA
    }
  }
}
```

Implementing an orchestrator is like implementing a simple service

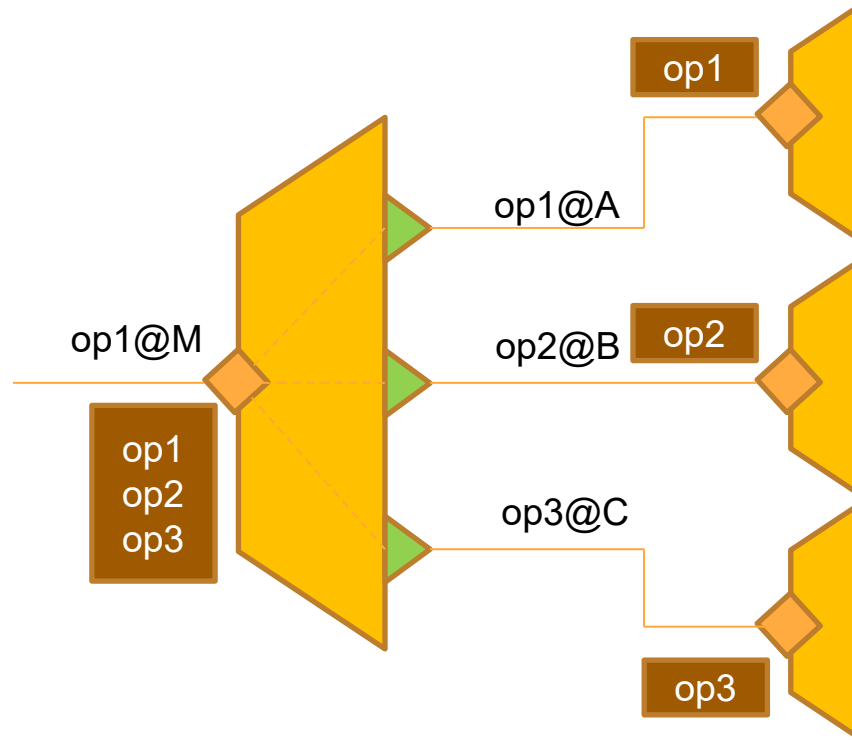
- Embedding of another service (firstService)
- Implementation of orchestration flow

What impact this can have:

- Evident orchestration workflow
- External services and embedded services are orchestrated similarly



Aggregation of services





Aggregation of services

```
from MyPackage import MyInterface

service ClientService () {
    execution { concurrent }
    outputPort CallingPort {
        Location: "socket://localhost:80"
        Protocol: http
        Interfaces: MyInterface
    }
    inputPort EntryPointB {
        Location: "socket://localhost:82"
        Protocol: http
        Interfaces: MyDummyInterface
        Aggregates: CalligPort
    }

    main {
        [dummyOp(request) (response) {
            //some implemetation
        }]
    }
}
```

It is just a line more in the input port

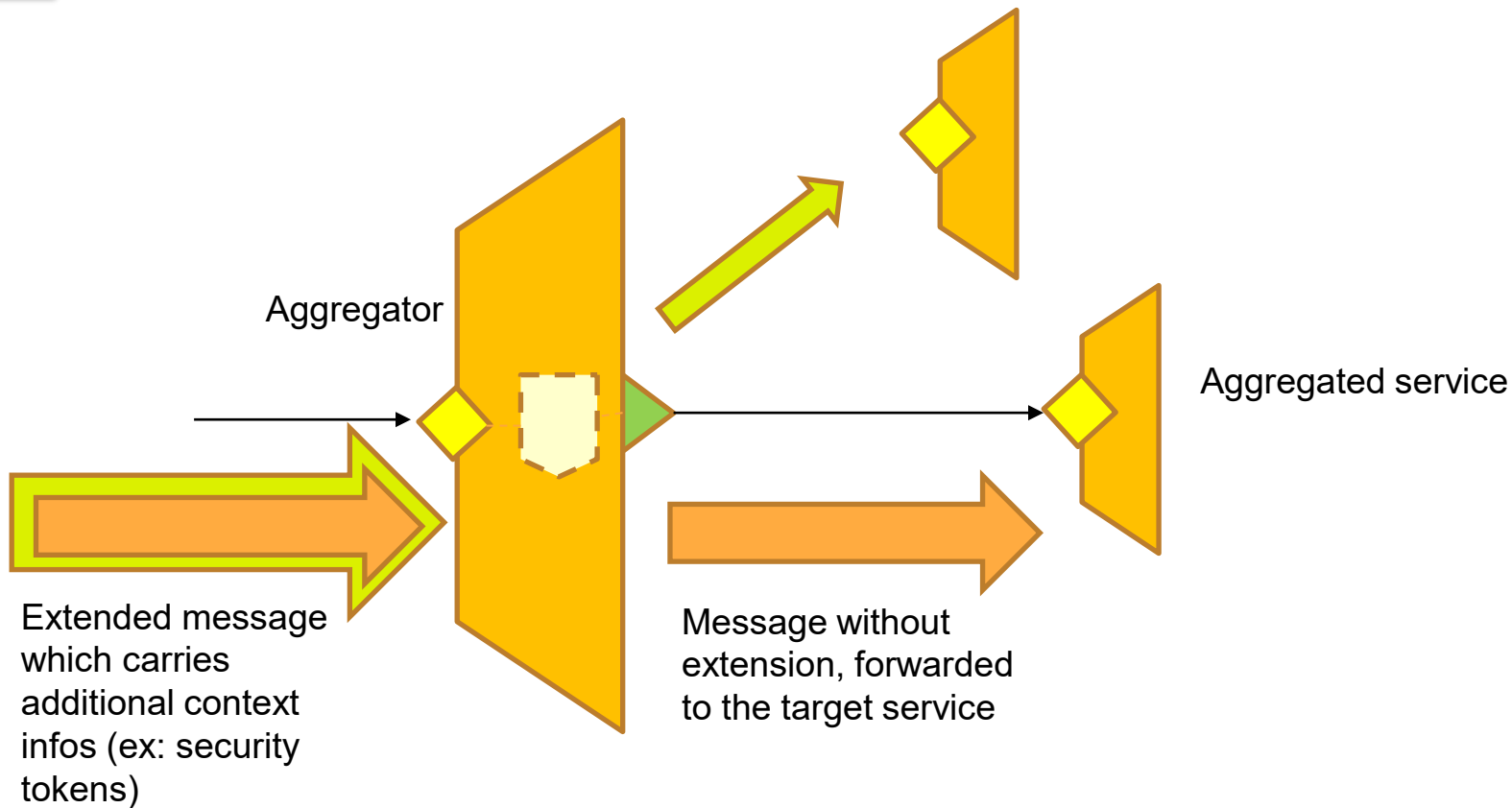
- Aggregations happens within an inputPort which aggregates outputPorts

What impact this can have:

- We can expose different services operations without replementing them or implementing orchestrations
- Automatic protocol conversions among inputOperatons and aggregated outputPorts

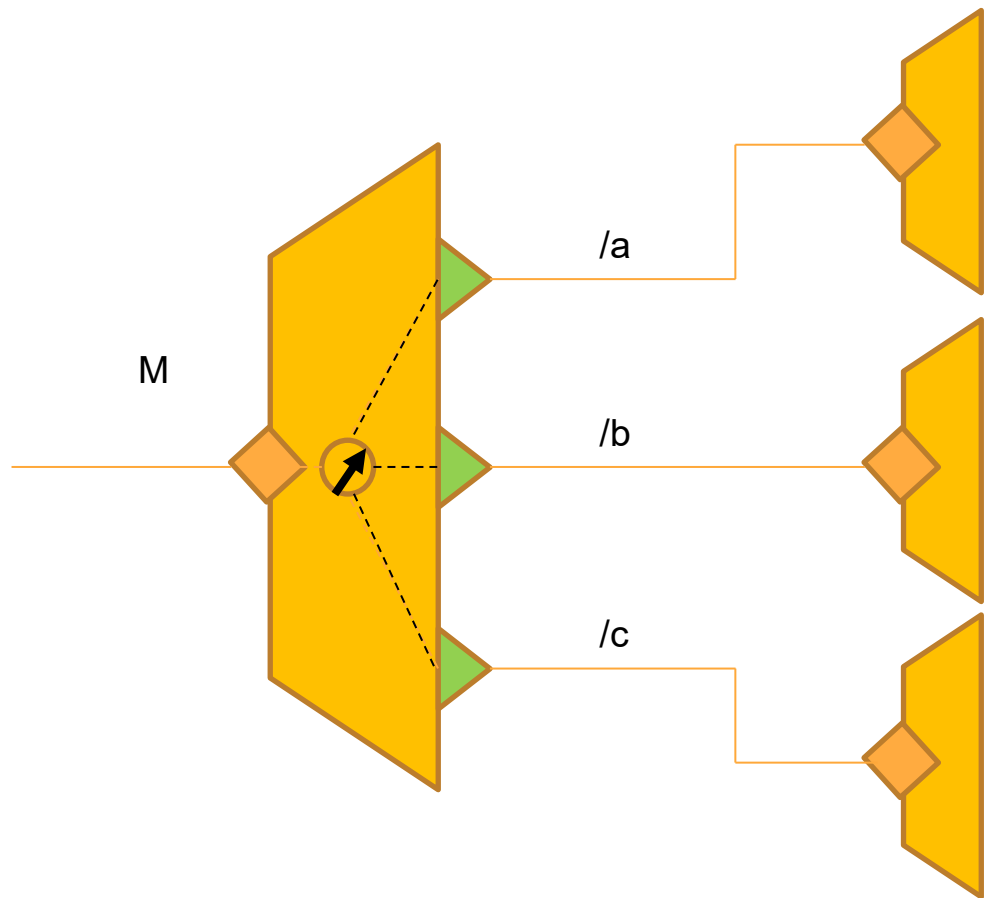


Couriers



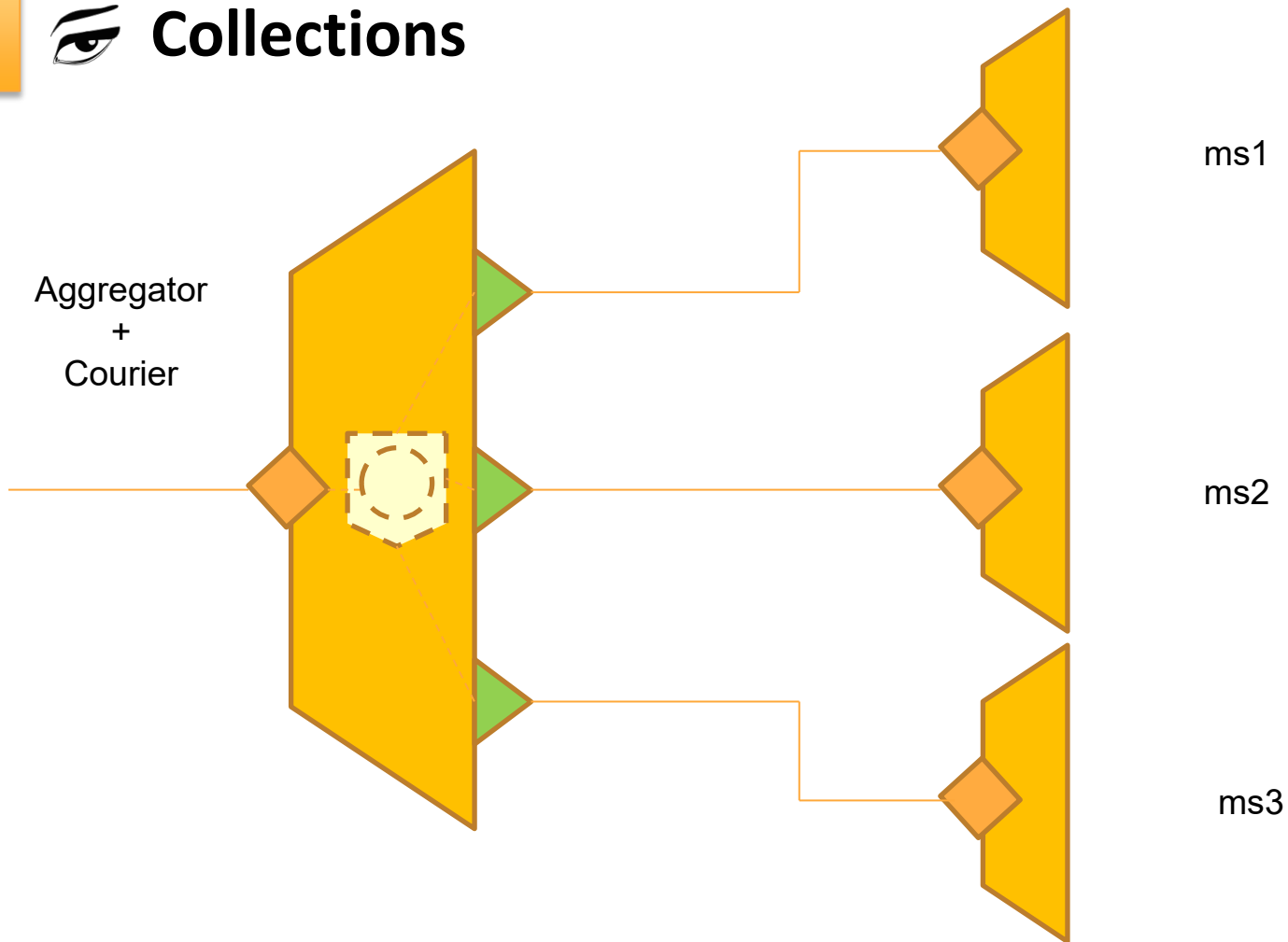


Redirection





Collections





...and all the rest

- **Java**

- There is a native possibility to integrate Java code into a Jolie service (JavaServices)
- It is possible to run a Jolie service within an hosting application server (Ex: JBoss)
- It is possible to send sodep messages to Jolie from a third party Java application

- **Javascript**

- As for Java, it is possible to embed javascript code into a Jolie service

- **Text Editors**

- Plugins for **Atom** and **Sublime Text**

- **HTTP and HTTPs**

- *HTTP and HTTPs are supported protocols*

- **SOAP Web Services**

- SOAP is a supported protocol
- jolie2wsdl and wsdl2jolie are tools which permit to convert Jolie interface into WSDL documents and viceversa

- **JSON**

- JSON format is supported as format for the http Protocol

- **Web**

- **Leonardo** is a web server written in Jolie.

- **REST**

- Thanks to the usage JSON messages can be exploited just parameterizing a port
- If necessary, it is possible to implement a standard REST services by exploiting **Jester**, a REST router for jolie services

- **Databases**

- SQL databases can be easily connected to a Jolie service using JDBC libraries.
- MongoDB connector

....



Supporters and stakeholders

Jolie in the real world...

- Used in production environments
- In Academy research about Jolie is still going on





Useful links

- ***Jolie website***

- <https://www.jolie-lang.org/>

- ***Jolie documentation***

- <https://jolielang.gitbook.io/docs/>

- ***Jolie Discord Community***

- <https://discord.gg/yQRTMNX>

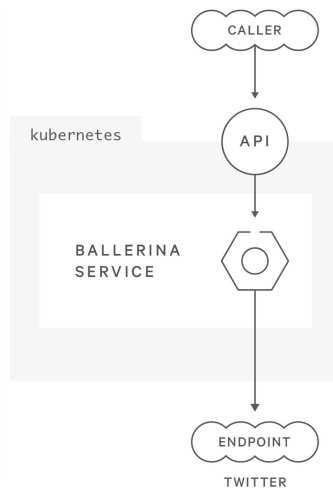
- ***Social***

- *Fb: <https://www.facebook.com/jolieprogramminglanguage>*
- *Twitter: <https://twitter.com/jolielang>*

Ballerina

Cloud Native Programming Language

```
@kubernetes:Deployment {  
  image: "corp/microsocial"  
}  
listener http:Listener ep = new(9090);  
service myService on ep {  
  @http:ResourceConfig {  
    methods:["POST"]  
  }  
  resource function foo(http:Caller  
    caller, http:Request req) {  
    twitter:client t = new(Config);  
    transaction {  
      var response = t->tweet(...);  
      _ = caller->respond(...);  
    }  
  }  
}
```



Batteries Included

Networking NATS ActiveMq WebSockets



HTTP/HTTPS

Client Endpoint
Redirects
Base Path and Path
Query Path Matrix Param
Restrict By Media Type
HTTP Caching
Disable Chunking
Trace Logs
HTTPS Listener
Basic HTTPS Listener Client
Mutual SSL
Request With Multiparts
Response With Multiparts
CORS
Access Logs
Data Binding
100 Continue
Handling Different Payload Types
HTTP Streaming
HTTP Interceptors/Filters

HTTP2

HTTP 1.1 to 2.0 Protocol Switch
HTTP 2.0 Server Push

WebSockets

Client Endpoint
Basic Server Functionalities
HTTP To WebSocket Upgrade
Proxy Server
Chat Application

Routing

Header-Based Routing
Passthrough
Content-Based Routing

Resiliency

Circuit Breaker
Load Balancing
Failover
Retry
Timeout

Database

JDBC Client CRUD Operations
JDBC Client Batch Update
JDBC Client Call Procedures
Streaming a Big Dataset

gRPC

Unary Blocking
Unary Non-Blocking
Server Streaming
Client Streaming
Bidirectional Streaming
Secured Unary
Proto To Ballerina

Observability

Distributed Tracing
Counter-Based Metrics
Gauge-Based Metrics

OpenAPI

Client Generation
Ballerina To OpenAPI
OpenAPI To Ballerina

NATS

Basic Publisher and Subscriber
Basic Streaming Publisher and Subscriber
Streaming Publisher and Subscriber With Data Binding
Durable Subscriptions
Queue Groups
Historical Message Replay

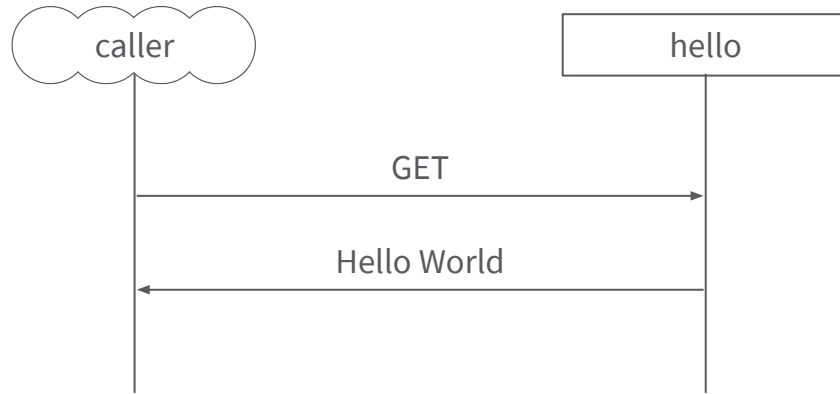
Socket

Basic TCP Socket
Basic UDP Client Socket

Access Control

Secured Service with JWT
Secured Service with Basic Auth
Secured Client with Basic Auth
Secured Client with JWT Auth
Secured Client with OAuth2

Hello World



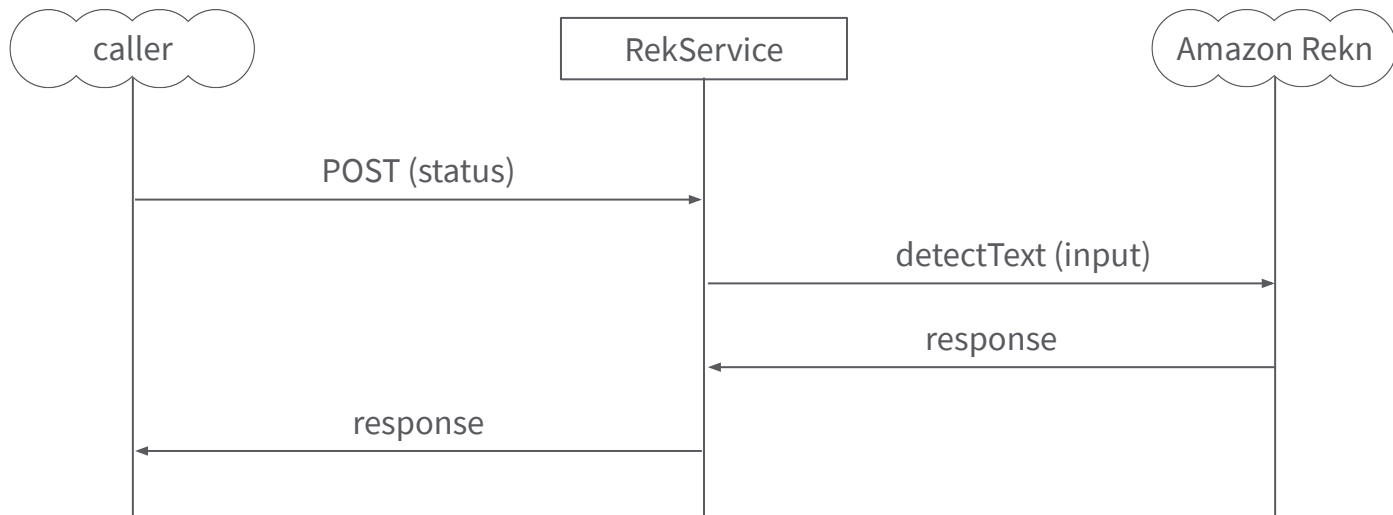
```
1  import ballerina/http;
2
3  service hello on new http:Listener(9090) {
4
5      resource function sayHello(http:Caller caller, http:Request req) returns error? {
6          check caller->respond("Hello, World!");
7      }
8
9  }
```

Annotations




```
1  import ballerina/http;
2
3  @http:ServiceConfig {
4      basePath: "/"
5  }
6  service hello on new http:Listener(9090) {
7
8      @http:ResourceConfig {
9          path: "hello",
10         methods: ["POST"]
11     }
12     resource function sayHello(http:Caller caller, http:Request req) returns @tainted error? {
13         string msg = <@untainted> check req.getTextPayload();
14         check caller->respond(string `Hello, ${msg}!`);
15     }
16
17 }
```

Connectors



```

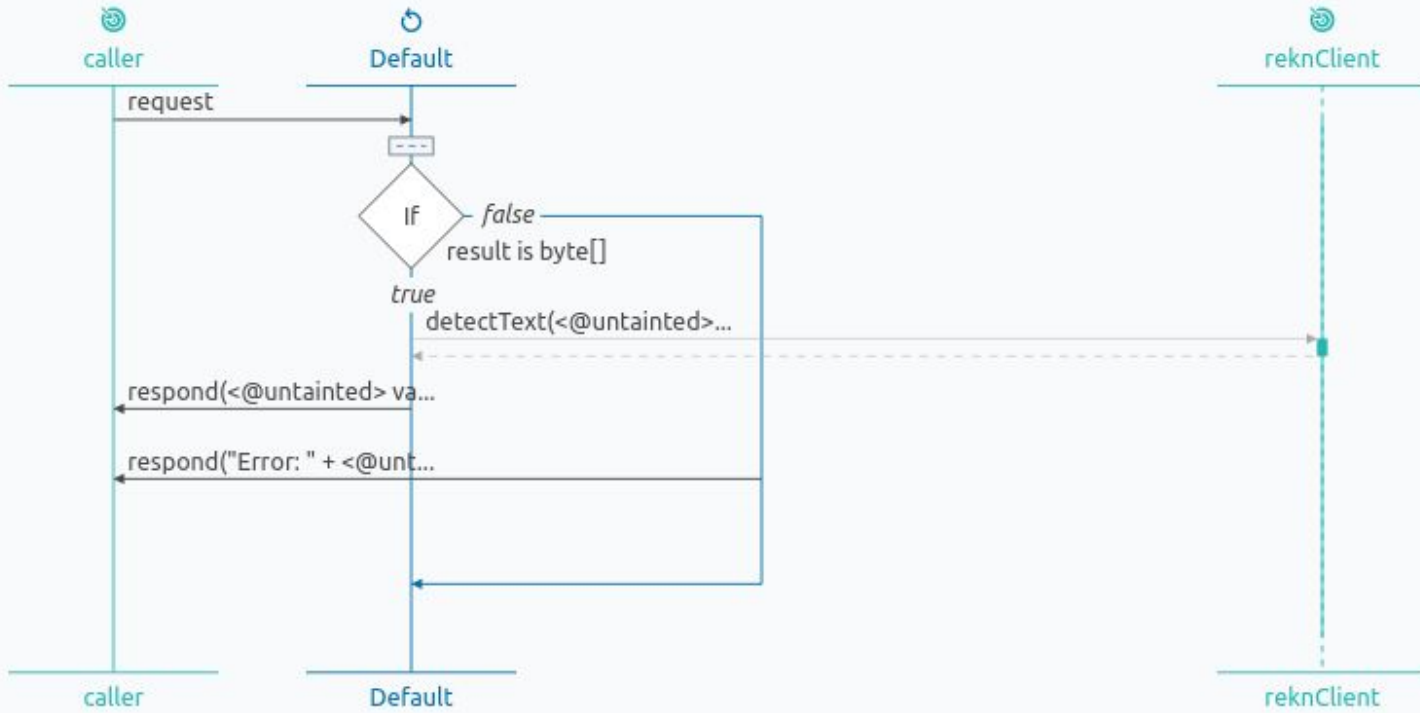
1  import wso2/amazonrekn;
2  import ballerina/config;
3  import ballerina/http;
4
5  amazonrekn:Configuration conf = {
6      accessKey: config.getAsString("AK"),
7      secretKey: config.getAsString("SK")
8  };
9
10 amazonrekn:Client reknClient = new (conf);
11
12 @http:ServiceConfig {
13     basePath: "/"
14 }
15 service ocrservice on new http:Listener(8080) {
16
17     resource function process(http:Caller caller,
18                             http:Request request) returns @tainted error? {
19         var result = request.getBinaryPayload();
20         if result is byte[] {
21             string value = check reknClient->detectText(<@untainted> result);
22             check caller->respond(<@untainted> value);
23         } else {
24             check caller->respond("Error: " + <@untainted> result.reason());
25         }
26     }
27 }
28

```

ocrservice

process

Zoom : 90% Depth : 0 Design : Interaction

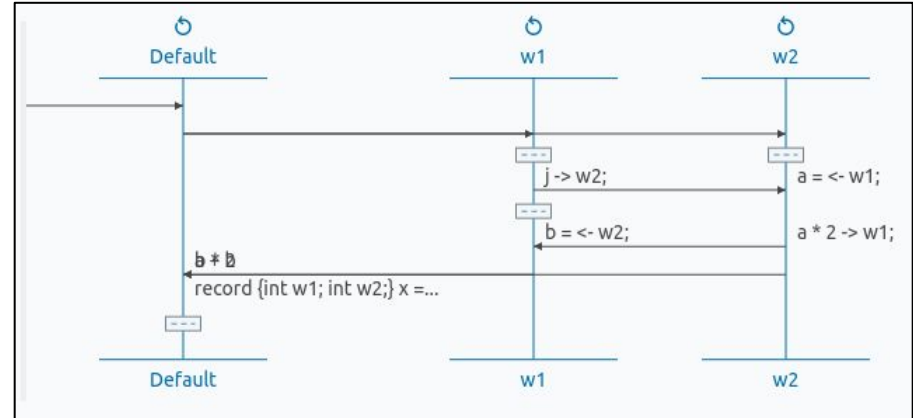


Concurrency

Workers

```
import ballerina/io;

public function main() {
    worker w1 returns int {
        int j = 10;
        j -> w2;
        int b;
        b = <- w2;
        return b * b;
    }
    worker w2 returns int {
        int a;
        a = <- w1;
        a * 2 -> w1;
        return a + 2;
    }
    record {int w1; int w2;} x = wait {w1, w2};
    io:println(x);
}
```



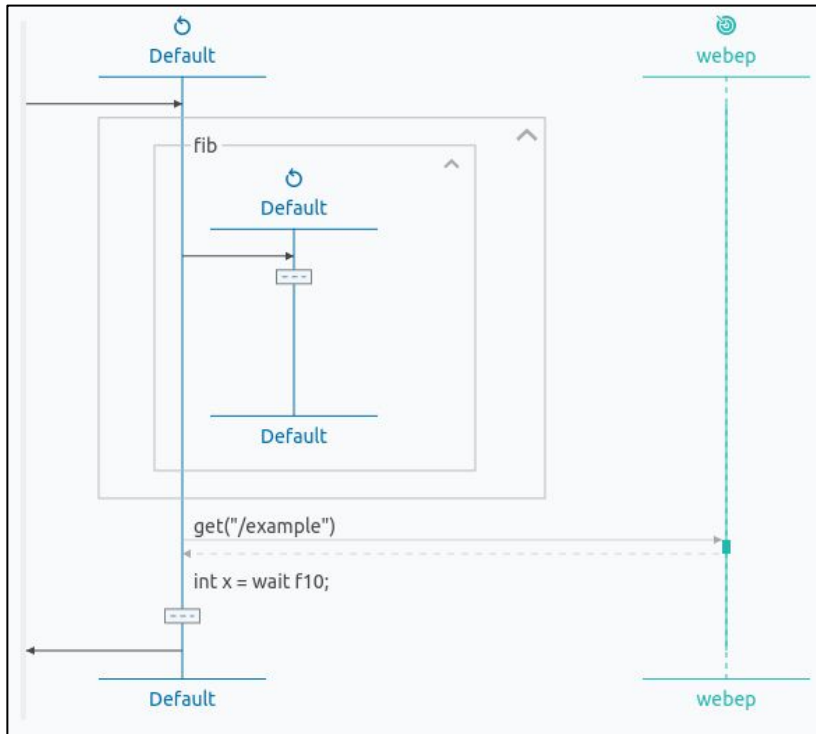
Futures

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

http:Client webep = new("http://example.com");

public function main() {
    future<int> f10 = start fib(10);
    var result = webep->get("/");
    int x = wait f10;
    if (result is http:Response) {
        io:println(result.getTextPayload());
        io:println(x);
    }
}

function fib(int n) returns int {
    if (n <= 2) {
        return 1;
    } else {
        return fib(n - 1) + fib(n - 2);
    }
}
```




Async I/O

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

http:Client webep = new("http://example.com");

public function main() returns error? {
    int x = 10;
    io:println(x * 2);
    var result = webep->get("/foo");
    if (result is http:Response) {
        io:println(check result.getJsonPayload());
    } else {
        io:println("Error: ", result);
    }
}
```

This looks like a typical blocking call,
but it's a bit more than that...



Transparent non-blocking I/O usage promotes optimal resource usage and handling of large number of active connections

Docker & Kubernetes

```

13 @kubernetes:ConfigMap {
14     conf: "ballerina.conf"
15 }
16 @kubernetes:Service {
17     serviceType: "NodePort"
18 }
19 @kubernetes:Deployment { }
20 @http:ServiceConfig {
21     basePath: "/"
22 }
23 service ocrservice on new http:Listener(8080) {
24
25     resource function process(http:Caller caller, http:Request request) returns @tainted error? {
26         byte[] result = check request.getBinaryPayload();
27         string value = check reknClient->detectText(<@untainted> result);
28         check caller->respond(<@untainted> value);
29     }
30
31 }

```

```
laf@laf-ThinkPad-X1-Carbon:~/dev/samples/ballerina/aws-demo$ ballerina build demo.bal
```

```
Compiling source  
demo.bal
```

```
Generating executables  
demo.jar
```

```
Generating artifacts...
```

@kubernetes:Service	- complete 1/1
@kubernetes:ConfigMap	- complete 1/1
@kubernetes:Deployment	- complete 1/1
@kubernetes:Docker	- complete 2/2
@kubernetes:Helm	- complete 1/1

Run the following command to deploy the Kubernetes artifacts:

```
kubectl apply -f /home/laf/dev/samples/ballerina/aws-demo/kubernetes
```

Run the following command to install the application using Helm:

```
helm install --name demo-deployment /home/laf/dev/samples/ballerina/aws-demo/kubernetes/demo-deployment
```

AWS Lambda

```
1  import ballerina/awslambda;
2  import ballerina/system;
3
4  @awslambda:Function
5  public function uuid(awslambda:Context ctx, json input) returns json|error {
6      return system:uuid();
7  }
```

Azure Functions


```
22 @af:Function
23 function processImage(@af:QueueTrigger { queueName: "requests" } json jobInfo,
24                       @af:BlobInput { path: "images/{jobId}" } string? encodedData,
25                       @af:QueueOutput { queueName: "results" } af:StringOutputBinding queueOut)
26     returns @tainted error? {
27       byte[] data = check array:fromBase64(encodedData.toString());
28       azurecv:Client cvClient = new({ key: system:getEnv("AZURE_CV_KEY") });
29       string text = check cvClient->ocr(data);
30       json result = { jobInfo, text };
31       queueOut.value = <@untainted> result.toJsonString();
32     }
```

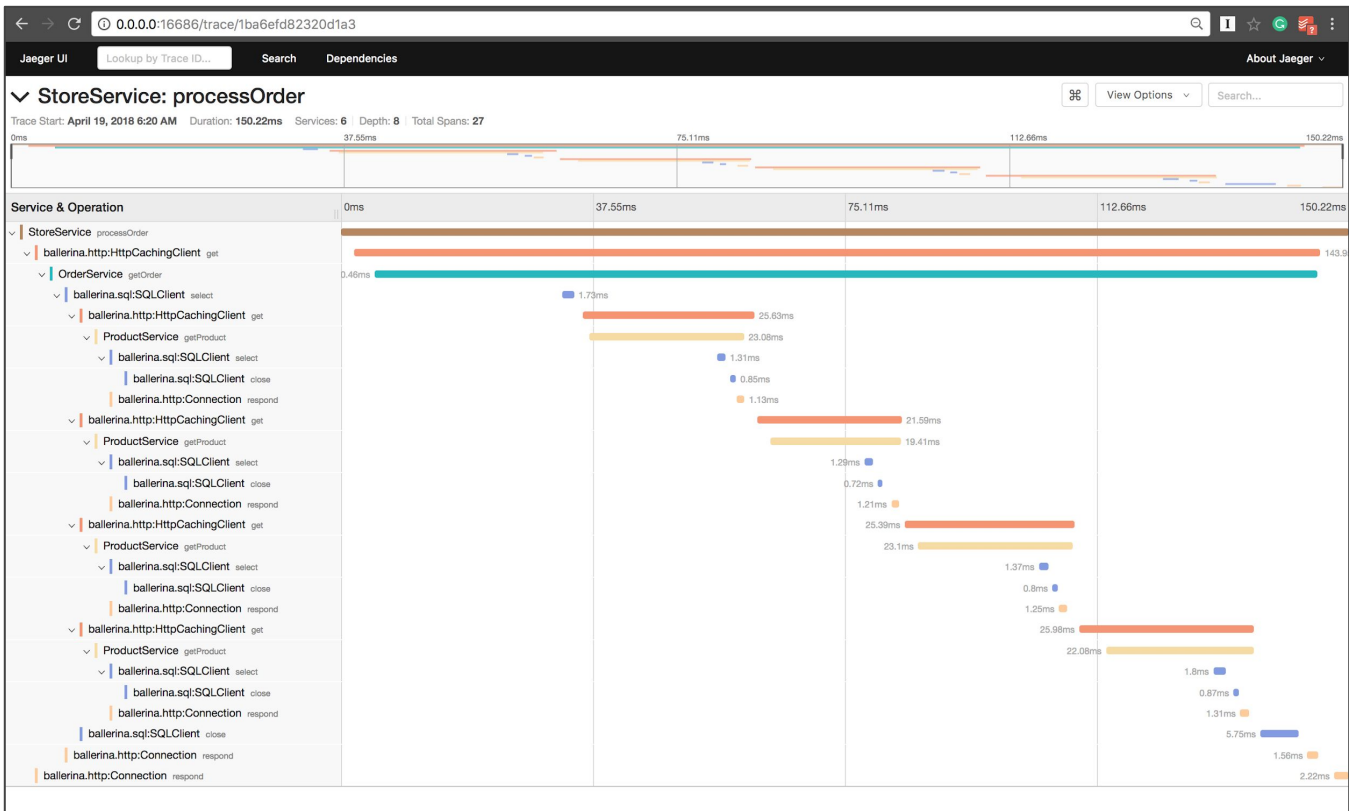
```
laf@laf-ThinkPad-X1-Carbon:~/dev/samples/ballerina/azure/ocr-serverless$ ballerina build ocr-functions.bal
Compiling source
    ocr-functions.bal

Generating executables
    ocr-functions.jar
    @azure.functions:Function: submitJob, processImage, publishResults

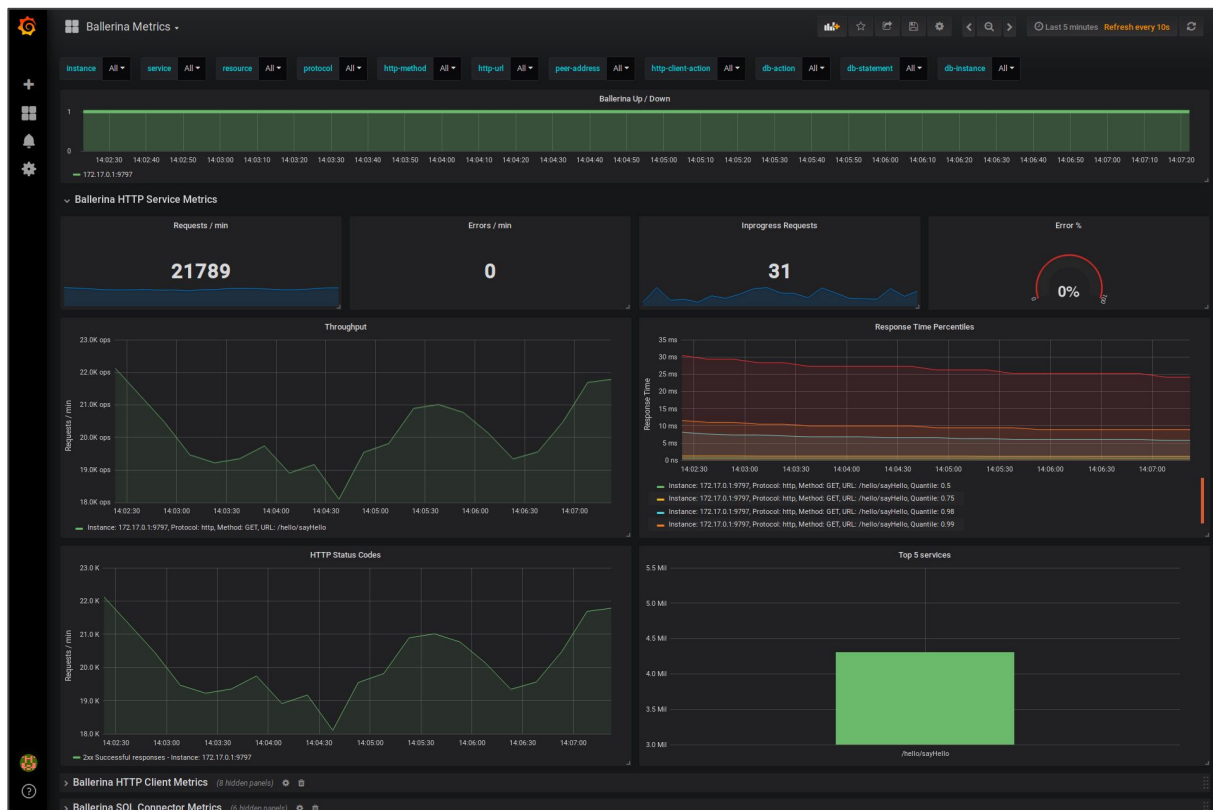
Run the following command to deploy Ballerina Azure Functions:
az functionapp deployment source config-zip -g <resource_group> -n <function_app_name> --src azure-functions.zip
```

Observability

Distributed Tracing: Jaeger/Zipkin



Metrics: Prometheus + Grafana



Learn - Ballerina by Example

<https://ballerina.io/learn/by-example/>

Language concepts

Hello World

- Hello World Service
- Hello World Main
- Hello World Parallel
- Hello World Client

Ballerina Basics

- Modules
- Variables
- Var
- Functions
- Defaultable Parameters
- Rest Parameters
- Documentation

Functions

- Function Pointers
- Anonymous Functions
- Closures
- Functional Iteration
- The Main Function

Flow Control

Values and Types

- Values
- Arrays
- Tuples
- Maps
- Table
- Union Types
- Byte Type
- Optional Type
- Anydata Type
- Any Type
- Constants
- Integer Ranges
- String Template Literal

Record

- Record
- Anonymous Records
- Optional Fields
- Record Type Reference

Object

- Object

JSON / XML

- JSON
- JSON Arrays
- JSON/Record/Map Conversion
- XML
- XML Literal
- XML Namespaces
- XML Attributes
- XML Functions
- XML Access
- XML/JSON Conversion

Binding Patterns

- Tuple Type Binding Pattern
- Record Type Binding Pattern
- Error Type Binding Pattern
- Tuple Destructure Binding Pattern
- Record Destructure Binding Pattern
- Error Destructure Binding Pattern

Errors

- Error Handling

I/O

- Byte I/O
- Character I/O
- Record I/O
- CSV I/O
- JSON I/O
- XML I/O
- JSON to CSV Transform

Common Libraries

- Time
- Caching
- Config
- Log
- Math
- String
- File Path
- Task Scheduler Appointment
- Task Scheduler Timer
- Task Service Appointment
- Task Service Timer

Security

How to get involved

Learn more

<http://ballerina.io>

Open source

<http://github.com/ballerina-platform/>

Get support

Stack Overflow - #ballerina tag

Slack - <https://ballerina.io/community/slack/>

Demos

<https://github.com/lafernando/samples/tree/master/ballerina/aws-demo>

<https://github.com/lafernando/samples/tree/master/ballerina/azure/ocr-serverless>

Questions?

Thanks!