



# 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



INTEGRATION  
PRODUCTS

ESB, BPMN, EAI

NOT AGILE

GENERAL PURPOSE  
PROGRAMING LANGUAGES

Java / Spring  
JavaScript / Node

NOT INTEGRATION SIMPLE

INTEGRATION  
PRODUCTS

ESB, BPMN, EAI

NOT AGILE

The  
Integration  
Gap

GENERAL PURPOSE  
PROGRAMING LANGUAGES

Java / Spring  
JavaScript / Node

NOT INTEGRATION SIMPLE

# 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.

The image displays a side-by-side comparison of Ballerina code and its visual representation as a sequence diagram. On the left, the code editor shows the source code for `main.bal`, which includes imports, configurable strings, a struct definition for `PR record`, and a `main` function that interacts with GitHub and Google Sheets. On the right, the sequence diagram visualizes the execution flow, showing the creation of `github` and `gsheets` clients, the execution of `get` and `appendRowToSheet` operations, and a loop structure for processing pull request records.

```
github-pull-requests-to-gsheets > main.bal > url
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: {token: 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 sequence diagram on the right illustrates the following flow:

- START** node leads to a `new` operation for the `github` client.
- The `github` client performs a `get` operation, returning a `prs` array.
- A `new` operation creates the `gsheets` client.
- The `gsheets` client performs an `appendRowToSheet` operation.
- A loop structure (represented by a diamond and a box) processes each record in `prs`, with a `new` operation for each iteration.
- Each iteration performs an `appendRowToSheet` operation on the `gsheets` client.
- The flow concludes at an **END** node.

# Integration designer

The screenshot displays the Visual Studio Code interface for the REST API Designer. The left pane shows the Explorer and the service definition in `service.bal`. The center pane shows the Overview Diagram with sections for Services, Types, and ModuleVariables. The right pane shows the Configure Resource dialog for the `albums` resource.

```
1 import ballerina/http;
2
3 configurable int port = 8080;
4
5 type Album readonly & record {
6   string id;
7   string title;
8   string artist;
9   decimal price;
10 };
11
12
13 table<Album> key(id) albums = table {
14   (id: "1", title: "Blue Train", artist: "John Coltrane", price: 56.99),
15   (id: "2", title: "Jazz", artist: "Henry Mulligan", price: 27.99),
16   (id: "3", title: "Sarah Vaughan and Clifford Brown", artist: "Sarah Vaughan", price: 39.99)
17 };
18
19 run | debug | py | visualize
20
21 service / on new httpListener(port) {
22   visualize
23   resource function get albums() returns Album[] {
24     return albums.toArray();
25   }
26   visualize
27   resource function get albums(string id) returns Album|httpNotFound {
28     Album? album = albums[id];
29     if album is {} {
30       return http::NOT_FOUND;
31     } else {
32       return album;
33     }
34   }
35   visualize
36   resource function post albums(@http:Payload Album album) returns Album {
37     albums.add(new album);
38     return album;
39   }
40   visualize
41   resource function delete path/(string id)(string? param) returns error?|httpNotFound|httpAccepted {
42     Album? album = albums[id];
43     if album is {} {
44       return http::NOT_FOUND;
45     } else {
46       = albums.remove(k: id);
47       return http::ACCEPTED;
48     }
49   }
50 }
```

**Overview Diagram**

- Services: /
- Types: Album
- ModuleVariables: port, albums

**Configure Resource**

HTTP Method: GET, Resource Path: path

Parameters: Add Parameter

Advanced Parameters: Show

Responses: 500 error?

Buttons: Cancel, Save

# 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  }
5  pub
6  }
7  }
8  }
  service on email:ImapListener
  service on email:PopListener
  service on file:Listener
  service on ftp:Listener
  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
```

## 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 `ballerina` 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.

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

Search for `org.kafkas` or `ballerina:org.kafkas` or `org.ballerina`

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



ballerina/chrono  
0.4.12



ballerina/client.config  
1.0.1



ballerina/asyncapi.native.handler  
0.2.0



ballerina/googleapis.sheets  
3.4.0



ballerina/twitter



ballerina/mysql



ballerina/postgresql



ballerina/mssql.driver

### Language library Distribution (2021.7.2)

lang.array	lang.boolean	lang.decimal	lang.error
lang.float	lang.function	lang.future	lang.int
lang.map	lang.object	lang.overly	lang.regex
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.

The screenshot displays the Ballerina IDE interface. On the left, a code editor shows the implementation of a data transformation function. On the right, a 'Data Mapper' diagram visualizes the mapping between the source and target data structures.

```
52 service calendar:CalendarService on calendar:Listener {
53
54   remote function onNewEvent(calendar:Event payload) returns error? {
55     do {
56       // Add the card to the Trello list
57       trello:Cards card = calEventToTrelloCard(payload);
58       _ = check trello->addCards(card);
59
60       // Send SMS notification
61       string twilioMsg = calEventToMessage(payload);
62       _ = check twilio->sendSms(twFromMobile, twToMobile, twMsg);
63     }
64     on fail var e {
65       // Log the error and add the event to the dead letter channel
66       log:printlnError(string "Failed to process the calendar event: ", e);
67       toDeadLetterChannel(payload, e);
68     }
69   }
70
71   remote function onEventDelete(calendar:Event payload) returns error? {
72   }
73
74   remote function onEventUpdate(calendar:Event payload) returns error? {
75   }
76 }
77
78 Design
79 function calEventToTrelloCard(calendar:Event calEvent) returns trello:Cards {
80   name: calEvent.summary,
81   due: calEvent.end7.dateTime,
82   idList: trelloListId,
83   desc: string "New event is created on Google Calendar: ${calEvent.summary}
84         The event starts on ${calEvent.start7.dateTime ? ""} and ends on
85         ${calEvent.end7.dateTime ? ""}";
86 }
87
88 Design
89 function calEventToMessage(calendar:Event calEvent) returns string {
90   string "New event is created: ${calEvent.summary ? ""} starts on
91         ${calEvent.start7.dateTime ? ""} and ends on
92         ${calEvent.end7.dateTime ? ""}";
93 }
```

The Data Mapper diagram, titled 'Data Mapper: calEventToTrelloCard', shows the mapping between the source 'calEvent: trigger.google.calendar.Event' and the target 'trello:Cards'. The source fields include kind, slug, id, status, htmlLink, createdAt, updatedAt, summary, description, location, color, creator, organizer, start Time, date, dateTime, timeZone, end Time, date, dateTime, timeZone, and endTimeIsRepeat. The target fields include closed, desc, due, Resource, idBoard, idCover, idSource, idLabel, idList, idMembers, keepFromSource, label, name, pos, subscribed, and urlSource. Lines connect the source fields to the target fields, illustrating the data transformation logic.

# Data persistence

Entity Relationship Diagram — ballroom-backend

```
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

Collapsible

The diagram illustrates the following entities and their relationships:

- Contest**: Attributes include `id` (string), `title` (string), `readmeFile` (byte[]), `startTime` (time:Civil), `endTime` (time:Civil), `imageUrl` (string), `challenges` (ChallengesOnContests[]), `submissions` (Submission[]), and `moderator` (User).
- Challenge**: Attributes include `id` (string), `title` (string), `createTime` (time:Civil), `templateFile` (byte[]), `readmeFile` (byte[]), `difficulty` (string), `testCasesFile` (byte[]), `contests` (ChallengesOnContests[]), `author` (User), and `submissions` (Submission[]).
- User**: Attributes include `id` (string), `username` (string), `fullname` (string), `role` (string), `challenge` (Challenge[]), `moderatedContests` (Contest[]), and `submissions` (Submission[]).
- Submission**: Attributes include `id` (string), `submittedTime` (time:Civil), `score` (float), `challenge` (Challenge), `contest` (Contest), and `user` (User).
- ChallengesOnContests**: A join entity with attributes `id` (string), `challenge` (Challenge), `contest` (Contest), and `assignedTime` (time:Civil).

Relationships:

- Contest** (1..1) to **Challenge** (0..\*): One-to-many relationship.
- Contest** (1..1) to **Submission** (0..\*): One-to-many relationship.
- Challenge** (1..1) to **Submission** (0..\*): One-to-many relationship.
- User** (1..1) to **Submission** (0..\*): One-to-many relationship.
- Contest** (1..1) to **ChallengesOnContests** (0..\*): One-to-many relationship.
- Challenge** (0..\* to **ChallengesOnContests** (0..\*): Many-to-many relationship.

Footer: m2\* Live Share Ballerina 2201.7.1 (Swan Lake Update 7)



# Debug your integration

The screenshot displays the Ballerina IDE interface during a debug session. The main editor shows the source code for `main.bal` with a breakpoint set at line 38. The `getGeoData` function is currently paused at this line. The `Variables` panel on the left shows the state of local variables, with `resp` being a `Response` object containing status code 200 and a JSON payload. The `Call Stack` shows the current function call `getGeoData` at line 38.

On the right, the `Service /geodata` viewer shows two resources: `weather/{string ip}` and `location/{string ip}`. The `Responses` table for the `weather` resource is as follows:

Code	Description
200	json
500	error

The `Terminal` at the bottom shows the command used to run the application in debug mode:

```
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
```

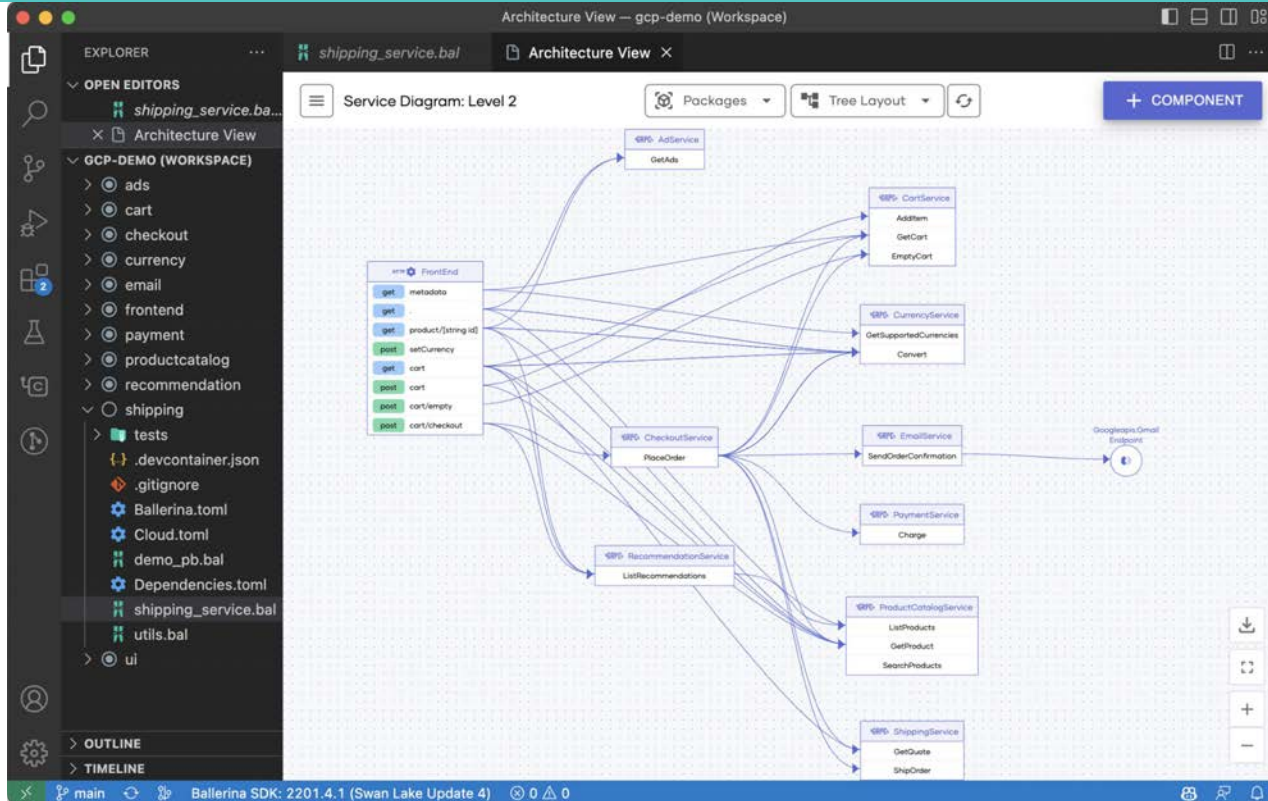
# Application Architecture View

The screenshot displays an IDE interface with the following components:

- EXPLORER:** Shows a workspace named "GCP-DEMO (WORKSPACE)" with a tree view of folders and files. The "shipping" folder is expanded, showing sub-folders like "tests" and "ui", and files like ".devcontainer.json", ".gitignore", "Ballerina.toml", "Cloud.toml", "demo\_pb.bal", "Dependencies.toml", "shipping\_service.bal", and "utils.bal".
- EDITOR:** The main window shows the "shipping\_service.bal" file. A context menu is open over the code, listing actions such as "Developer: Reload Window", "Preferences: Color Theme", "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)", and "WSO2 Choreo: Focus on Account View".
- Code Snippet:** The visible code in the editor includes:

```
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()}
26 service "ShippingService" on ep {
27     final float SHIPPING_COST = 8.99;
28
29     isolated remote function GetQuote(GetQuoteRequest value) returns GetQuoteResponse|error {
30         CartItem[] items = value.items;
31         int count = 0;
```
- STATUS BAR:** Shows the current file is "main", the SDK is "Ballerina SDK: 2201.4.1 (Swan Lake Update 4)", and the cursor is at line 17, column 23.

# 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

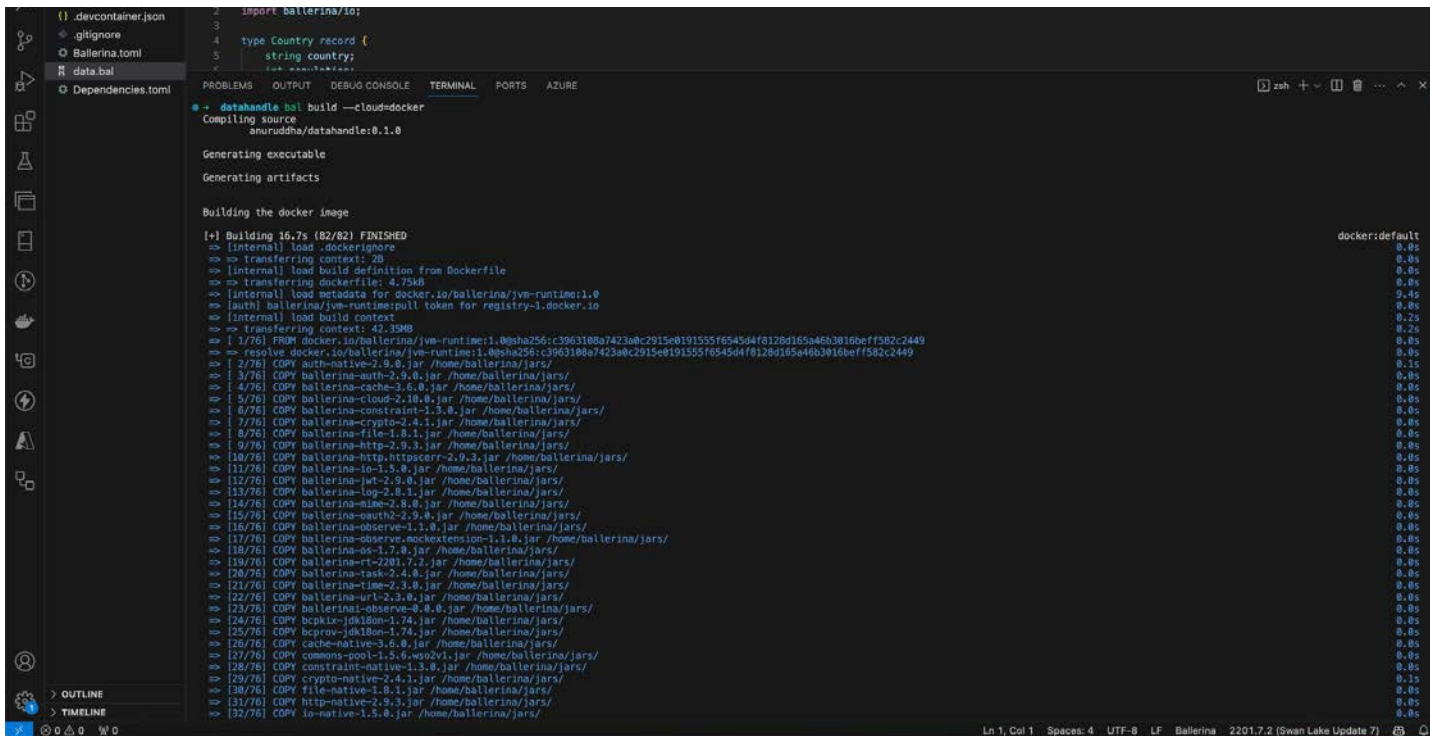
Compiling source
  user/hello_world:0.1.0

=====
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

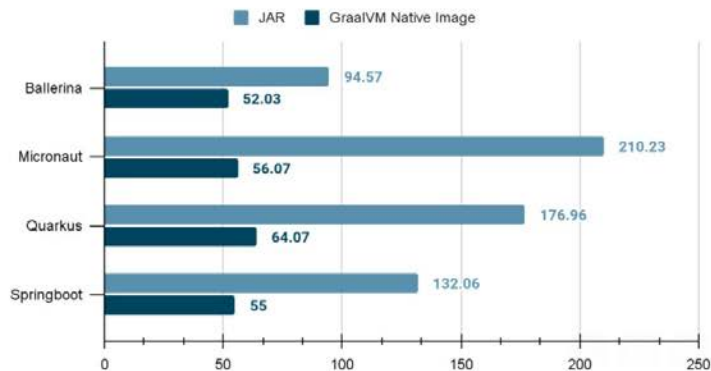
```
bal build -cloud=docker
```



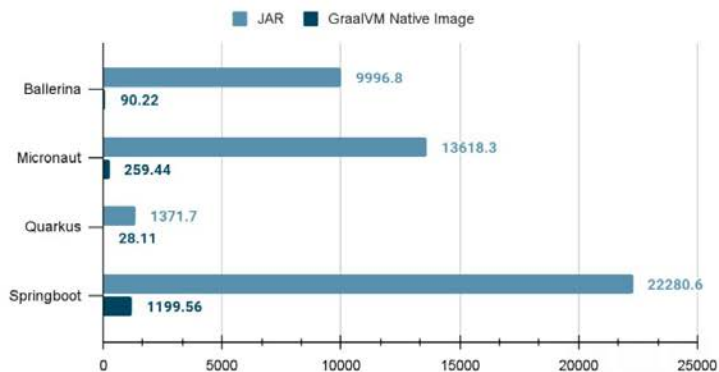
```
1 import ballerina/io;
2
3
4 type Country record {
5     string country;
6     int latitude;
7 }
8
9 }
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

# Docker + GraalVM

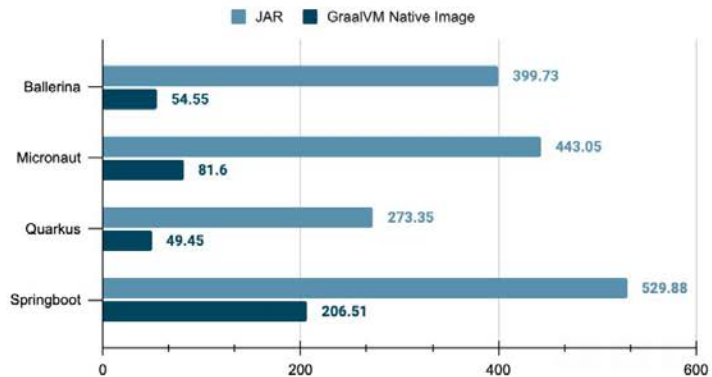
## Docker Image Compressed Size(MB)



## Docker Startup Time(ms)



## Docker Memory Footprint(MB)



# Build Kubernetes artifacts

```
→ geo bal build --cloud=k8s
```

```
Compiling source  
  anuruddha/geo:0.1.0
```

```
Generating executable
```

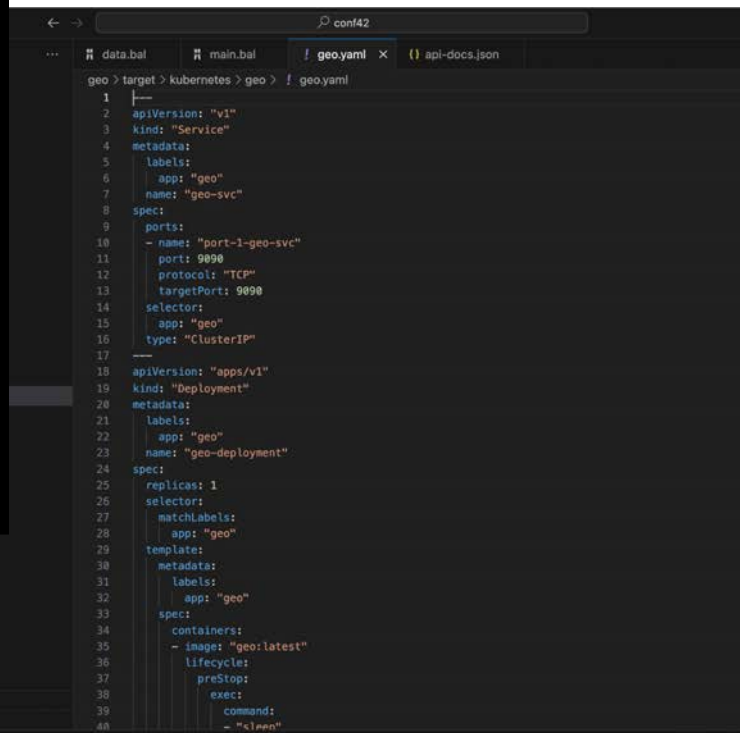
```
Generating artifacts
```

```
@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  
=> transferring build context: 17.20kB
```

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



```
geo > target > kubernetes > geo > ! geo.yaml  
1 |---  
2 |  apiVersion: "v1"  
3 |  kind: "Service"  
4 |  metadata:  
5 |    labels:  
6 |      app: "geo"  
7 |      name: "geo-svc"  
8 |  spec:  
9 |    ports:  
10 |      - name: "port-1-geo-svc"  
11 |        port: 9090  
12 |        protocol: "TCP"  
13 |        targetPort: 9090  
14 |    selector:  
15 |      app: "geo"  
16 |    type: "ClusterIP"  
17 |---  
18 |  apiVersion: "apps/v1"  
19 |  kind: "Deployment"  
20 |  metadata:  
21 |    labels:  
22 |      app: "geo"  
23 |      name: "geo-deployment"  
24 |  spec:  
25 |    replicas: 1  
26 |    selector:  
27 |      matchLabels:  
28 |        app: "geo"  
29 |    template:  
30 |      metadata:  
31 |        labels:  
32 |          app: "geo"  
33 |      spec:  
34 |        containers:  
35 |          - image: "geolatest"  
36 |            lifecycle:  
37 |              preStop:  
38 |                exec:  
39 |                  command:  
40 |                    - "clean"
```



```
api-docs.json  
...  
api-docs.json.zip
```

# 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 services.

### Supported triggers and bindings

An Azure Function consists of a trigger and optional bindings. A trigger defines the 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 flows into the function, and an output binding is a destination to which the function is outputting data from the function to an external resource. For more information, see [concepts](#).

The following Azure Functions triggers and bindings are currently supported in Ballerina.

Supported triggers	Supported output bindings
<a href="#">HTTP</a>	<a href="#">HTTP</a>
<a href="#">Queue</a>	<a href="#">Queue</a>
<a href="#">Blob</a>	<a href="#">Blob</a>
-	<a href="#">Twilio SMS</a>
<a href="#">Cosmos DB</a>	<a href="#">Cosmos DB</a>
<a href="#">Timer</a>	-

## 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 `@aws:lambda:Function` adhering to the following function signature:

```
function (aws:lambda: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.

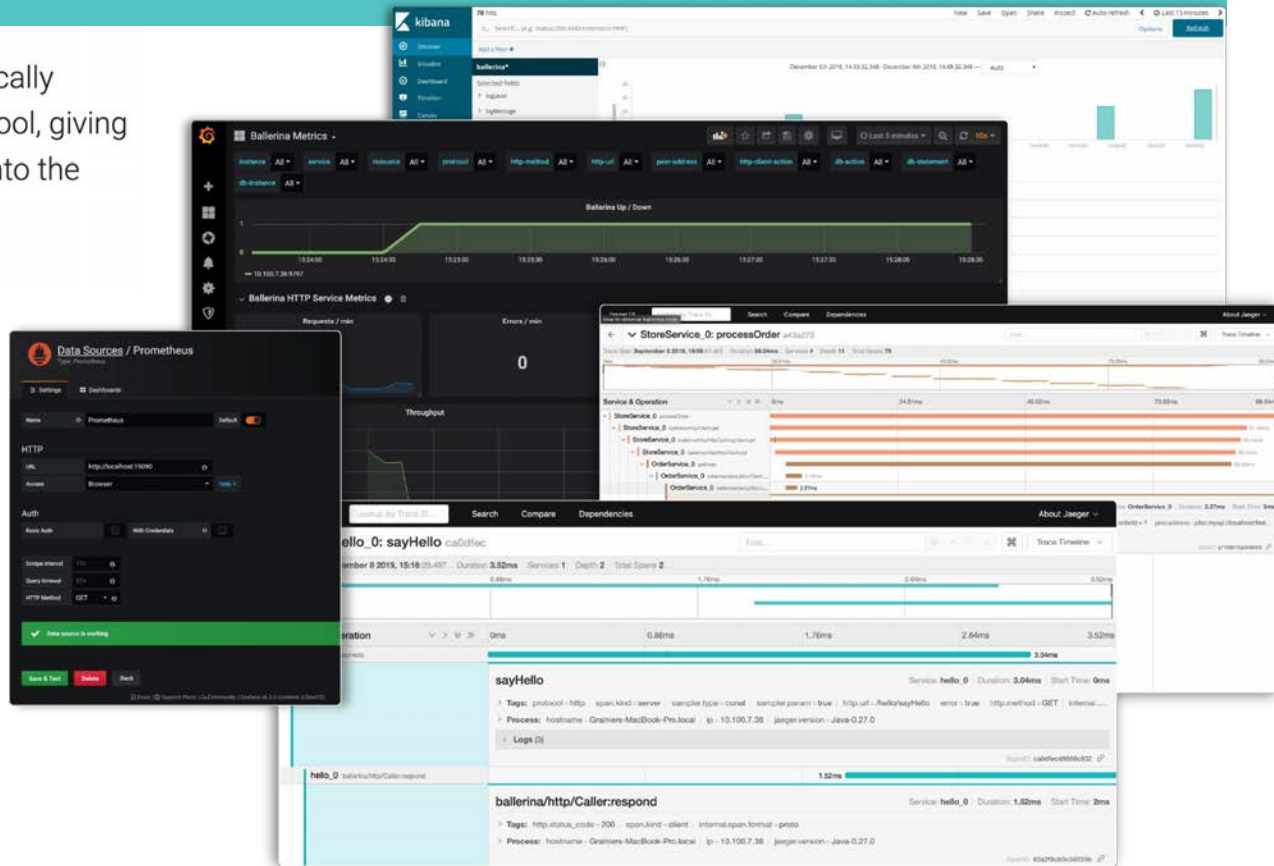
- [Direct invocation](#)
- [Simple Queue Service \(SQS\)](#)
- [Simple Storage Service \(S3\)](#)
- [DynamoDB](#)
- [Simple Email Service \(SES\)](#)
- [API Gateway](#)



# 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**