

BEAM
SUMMIT

Beam ❤️ Kotlin : full pipeline with Midgard




Mazlum TOSUN



Head of Data and
Cloud GroupBees



- ❖ Google Cloud Evangelist, Data Architect, functional programming, Devops, Serverless...
- ❖  Fan

<https://www.youtube.com/@GCPLearning-ce9bq> 

<https://github.com/tosun-si> 

<https://twitter.com/MazlumTosun3> 

<https://www.linkedin.com/in/mazlum-tosun-900b1812/> 

<https://medium.com/@mazlum.tosun> 

<https://stackoverflow.com/users/9261558/mazlum-tosun> 



Some Idioms

Create object

```
data class Customer(val name: String, val email: String)
```

Default values for function parameters

```
fun foo(a: Int = 0, b: String = "") { ... }
```



Filter a list

```
val positives = list.filter { it > 0 }
```

Check presence element collection

```
if ("john@example.com" in emailsList) { ... }
```

```
if ("jane@example.com" !in emailsList) { ... }
```



String interpolation

```
println("Name $name")
```

Instance checks

```
when (x) {  
    is Foo -> ...  
    is Bar -> ...  
    else   -> ...  
}
```



Read-only map and list

```
val list = listOf("a", "b", "c")
```

```
val map = mapOf("a" to 1, "b" to 2, "c" to 3)
```

Extensions functions

```
fun String.spaceToCamelCase() { ... }
```

```
"Convert this to camelcase".spaceToCamelCase()
```



Singleton

```
object Resource {  
    val name = "Name"  
}
```

If-not-null shorthand

```
val files = File("Test").listFiles()  
  
println(files?.size) // size is printed if files is not null
```




If-not-null-else shorthand

```
val files = File("Test").listFiles()

println(files?.size ?: "empty") // if files is null, this prints "empty"

// To calculate the fallback value in a code block, use `run`
val fileSize = files?.size ?: run {
    return someSize
}
println(fileSize)
```



Execute a statement if null

```
val values = ...  
val email = values["email"] ?: throw IllegalStateException("Email is missin
```

Generic function that requires the generic type information

```
// public final class Gson {  
//     ...  
//     public <T> T fromJson(JsonElement json, Class<T> classOfT) throws JsonSyntaxException {  
//         ...  
  
inline fun <reified T: Any> Gson.fromJson(json: JsonElement): T = this.fromJson(json,  
T::class.java)
```



Check this link to have more examples

<https://kotlinlang.org/docs/idioms.html>



Why Kotlin instead of Scala ?



- ❑ Library for Beam Kotlin
- ❑ Proposes some extensions on PCollection
- ❑ Oriented to Functional Programming style
- ❑ Proposes map, flatMap and filter operator
- ❑ Executes operators and access to DoFn lifecycle and ProcessContext if needed



Example of usual Beam pipeline with map, flatMap and filter operations :

```
val resultPlayers: PCollection<Player> = pipeline
  .apply("Create", Create.of(listOf(psgTeam, realTeam)))
  .apply(
    "To Team with Slogan V2",
    MapElements
      .into(TypeDescriptor.of(Team::class.java))
      .via(SerializableFunction { it.copy(slogan = "${it.slogan} VERSION 2") })
  )
  .apply(
    "To Players",
    FlatMapElements
      .into(TypeDescriptor.of(Player::class.java))
      .via(SerializableFunction { it.players })
  )
  .apply("Filter age > 25", Filter.by(SerializableFunction { it.age > 25 })))
```



The same pipeline with Midgard library :

```
import fr.groupbees.midgard.*

val resultPlayersMidgard: PCollection<Player> = pipeline
  .apply("Create", Create.of(listOf(psgTeam, realTeam)))
  .map("To Team with Slogan V2") { it.copy(slogan = "${it.slogan} VERSION 2") }
  .flatMap("To Players") { it.players }
  .filter("Filter age > 25") { it.age > 25 }
```



Midgard allows to propose map and flatMap operators and extensions while interacting with this lifecycle

```
val resultTeamMidgardMapLifeCycle: PCollection<Team> = pipeline
  .apply("Create", Create.of(listOf(psgTeam, realTeam)))
  .mapFn(
    name = "To Team with Slogan V2",
    transform = { it.copy(slogan = "${it.slogan} VERSION 2") },
    setupAction = { println("Setup Action") },
    startBundleAction = { println("Start Bundle Action") },
    finishBundleAction = { println("Finish Bundle Action") },
    teardownAction = { println("Teardown Action") }
  )
```




Lifecycle and access DoFn ProcessContext while applying the current transformation

```
// Simulate a side input for the slogan suffix.
val slogansSideInput: PCollectionView<String> = pipeline
    .apply("Read slogans", Create.of("VERSION 2"))
    .apply("Create as collection view", View.asSingleton())

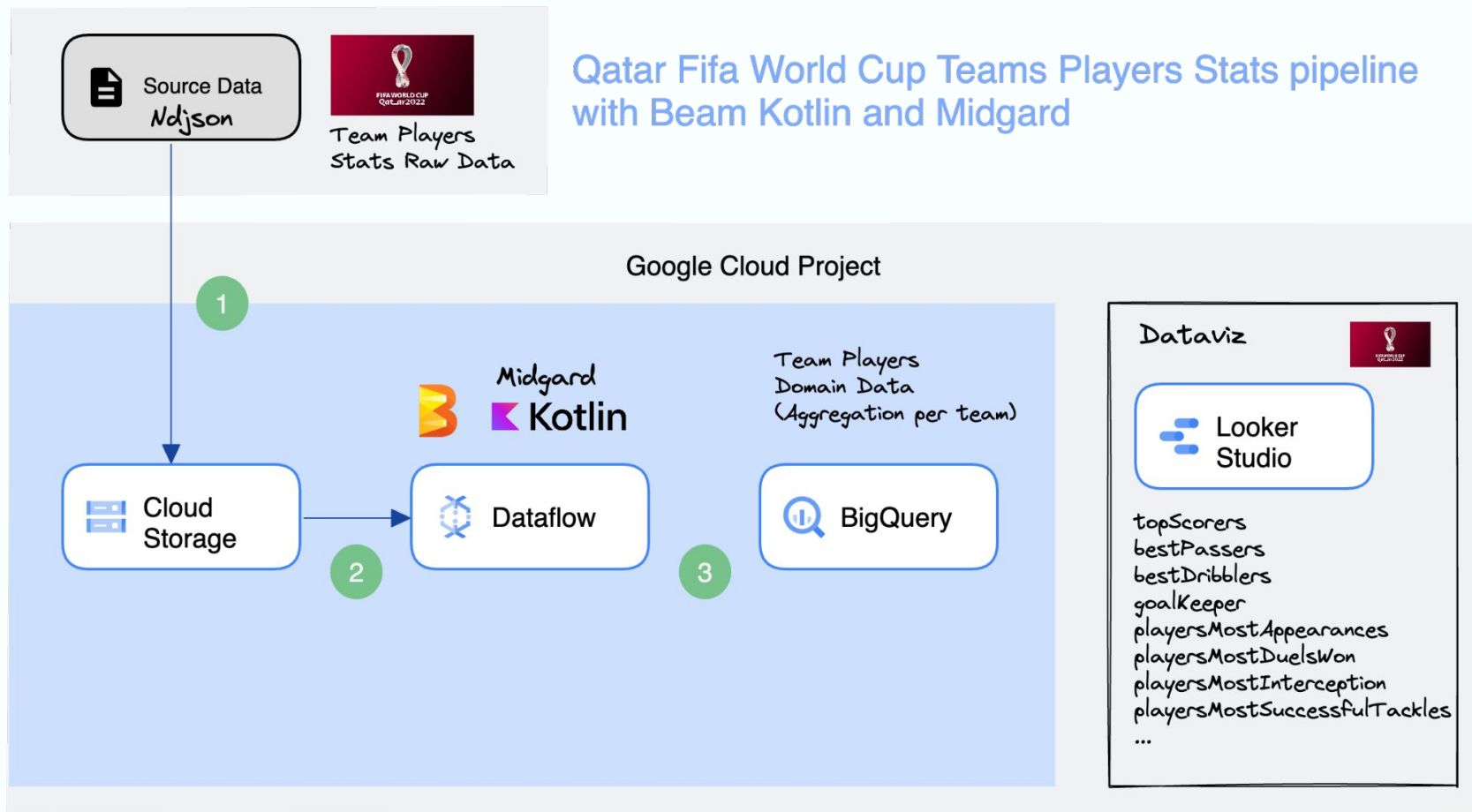
val resultTeamMidgardMapContextLifeCycle: PCollection<Team> = pipeline
    .apply("Create", Create.of(listOf(psgTeam, realTeam)))
    .mapFnWithContext(
        name = "To Team with Slogan V2",
        transform = { context -> toTeamWithSloganSuffixFromSideInput(slogansSideInput, context) },
        setupAction = { println("Setup Action") },
        sideInputs = listOf(slogansSideInput),
        startBundleAction = { println("Start Bundle Action") },
        finishBundleAction = { println("Finish Bundle Action") },
        teardownAction = { println("Teardown Action") }
    )
```



Lifecycle and access DoFn ProcessContext : side input

```
private fun toTeamWithSloganSuffixFromSideInput(
    sideInput: PCollectionView<String>,
    context: DoFn<Team, Team>.ProcessContext
): Team {
    val currentTeam: Team = context.element()
    val sloganSuffixSideInput: String = context.sideInput(sideInput)

    return currentTeam.copy(slogan = "${currentTeam.slogan} $sloganSuffixSideInput")
}
```





- ❑ Add extensions for existing IOs :
 - TextIO
 - BigQueryIO
 -

- ❑ Add extensions for built in transform :
 - WithKey
 - GroupByKey
 -



<https://github.com/tosun-si/world-cup-qatar-team-stats-java>

<https://github.com/tosun-si/world-cup-qatar-team-stats-kotlin-midgard>



<https://github.com/tosun-si/midgard>

Feel free to contribute to the project, give me feedback, try it and support us with a Github star 



Thank you :)

QUESTIONS?