Bringing the Supertanker to the next Island - Kotlin in the Enterprise

Edwin Günthner
Senior Software Developer, IBM Z

edwin.guenthner@de.ibm.com
IBM Deutschland Research & Development GmbH

(image generated by MS Copilot)

Bringing the Ausflugsdampfer to the next Island - Kotlin in the Enterprise

Edwin Günthner
Senior Software Developer, IBM Z

edwin.guenthner@de.ibm.com
IBM Deutschland Research & Development GmbH

IBM

Contents

Disclaimer:

Kotlin exists since 2014, it is at version 2.3 now … it is **mature**!

The focus of this presentation is:

Software Engineering aspects of introducing Kotlin in large enterprise contexts.

(Not: an introduction to Kotlin, a pro/con analysis, …)

Part 1: Context

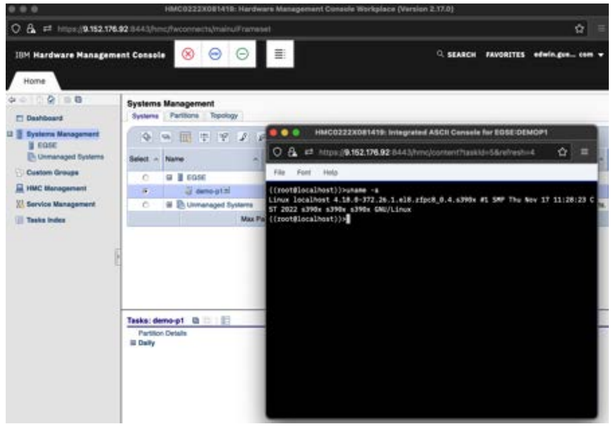
IBM z17

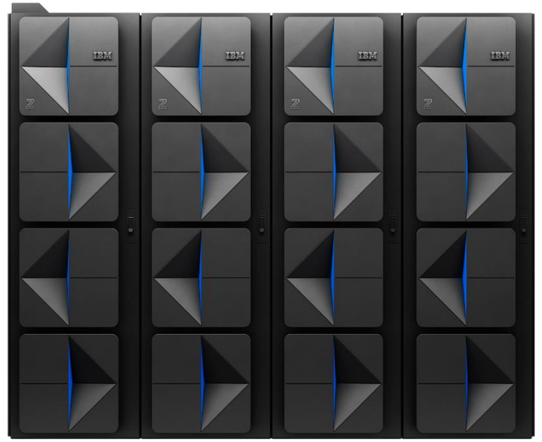
(available since June 2025)

- Up to 32 CPUs (8 cores each), running at 5.5 GHz
- RAM: up to 64 TB
- Up to 12 PCIe+ drawers (16 I/O adapters per drawer)
- Usable (theoretical) I/O bandwidth: 1536 GB/s

- Built-in virtualization (up to 85 logical partitions)
- 2nd level virtualization (via z/VM or Linux on IBM Z + kvm)
- Supported OSes:
  - z/OS
  - Linux on IBM Z
  - z/VM
  - ...

Managing an IBM z17

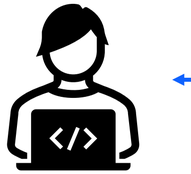




1 Mainframe

SE

controlled by 2
Support Elements



HMC

managed by 1 (or more)
Hardware Management Console(s)

The Code Stack

IBM Z **firmware** comprises all software components that get shipped with the mainframe, SEs and HMCs.

There are hundreds of developers around the globe, working on dozens of different projects, using all sorts of tools and languages:

- IBM Z assembler
- PL8
- C / C++
- Java / Kotlin
- Go
- Python

One of those projects: zEndCon

GitHub repository with:
- ~ 13 GB of content (current dev branch: ~ 4 GB)

- ~ 3M LOC C/C++
- ~ 6M LOC Java
- ~ 300K LOC Kotlin

For perspective:

The oldest journaling comment to be found in zEndCon:

```
/* 04/27/87  0001V00 Initial coding */
```

Part 2: Preparation

Why?

"Do something because you want to,

not because you can!”

— Anonymous

Why, refined!

"Do something because you want to,

not because you can!”

— Anonymous

Better strategy:

- Evaluate needs
- Look at benefits, but also: cost
- Ask yourself: “what’s my business case?”

Why, refined!

"Do something because you want to,

not because you can!”

— Anonymous

Our expectations for kotlin:

- Improved null pointer safety
- Code will be more concise (less boilerplate, too)
- Better abstractions, allowing for easy implementation of powerful DSLs
  …

In short:

We want a modern programming language!

Tooling (Initial)

Starting points:

- Existing Gradle build for Java (high quality)
- Excellent Gradle and Kotlin skills

Tooling (Initial)

Starting points:

- Existing Gradle build for Java (high quality)
- Excellent Gradle and Kotlin skills

“Time to MVP” (minimal viable product):

Half a week!

(MVP:
- Seamless integration of kotlin for production and test code
- Full interoperatibility between Java and Kotlin)

Empowerment & Education

More than 200 developers are contributing to our repository.

All of them *could* use Kotlin …

Empowerment & Education

More than 200 developers are contributing to our repository.

All of them *could* use Kotlin …

We focused on our component team (~ 50 developers).

Empowerment & Education

More than 200 developers are contributing to our repository.

All of them *could* use Kotlin …

We focused on our component team (~ 50 developers).

(“Our”) Team policy:

All **new** code should be written in Kotlin!

Empowerment & Education

Education principles:

- Everybody can participate
- Encourage community aspect

Education activities:

- Individual self learning (YouTube, Udemy, …)
- Joint team sessions
  - Team collects list of learning topics
  - 90 minute sessions, once per week, for ~3 months
  - Volunteers present about a specific topic, whole team then delves into details together

Part 3: Execution

Tooling (Advanced)

A lot of things …

simply worked out of the box!

Tooling (Advanced)

A lot of things ...

simply worked out of the box!

And they work ...

at compile time

and

at run time!

Tooling (Advanced)

A lot of things ...

simply worked out of the box!

And they work ...

at compile time

and

at run time!

Functionality added over time:

- gRPC code generation
- spotless linting

DSLs
(Why We Need Them)

Our component manages “resources”.

Resources need representation.

Each “interface” has its own representation.

DSLs
(Why We Need Them)

Our component manages “resources”.

Resources need representation.

Each “interface” has its own representation.

”Web Services API”

(RESTful API, offers services to users on the HMC)

Partition operations summary

The following table provides an overview of the operations provided for Partition objects.

Table 82. DPM - Partition: operations summary

Operation name	HTTP method and URI path
“List Partitions of a CPC” on page 248	`GET /api/cpcs/{cpc-id}/partitions`
“List Permitted Partitions” on page 251	`GET /api/console/operations/list-permitted-partitions`
“Create Partition” on page 254	`POST /api/cpcs/{cpc-id}/partitions`
“Delete Partition” on page 260	`DELETE /api/partitions/{partition-id}`
“Delete Partition Asynchronously” on page 262	`POST /api/partitions/{partition-id}/operations/async-delete`
“Get Partition Properties” on page 265	`GET /api/partitions/{partition-id}`
“Update Partition Properties” on page 269	`POST /api/partitions/{partition-id}`

Table 111. Partition object: class specific properties (continued)

Name	Qualifier	Type	Description
os-type	(pc)	String (0-8)	A human readable form of the operating system provided value for the type of the operating system active in this partition. If not provided by the operating system or if the partition **status** is **"stopped"**, an empty string is returned.
os-version	(pc)	String (0-32)	A human readable form of the operating system provided value for the version of the operating system active in this partition. If not provided by the operating system or if the partition **status** is **"stopped"**, an empty string is returned.
reserve-resources	(w)(pc)	Boolean	If true, resource reservation is enabled for this partition, and all physical resources backing the virtual resources configured for this partition are allocated and reserved, even when the partition is in **"stopped"** state. This guarantees that the partition start will not fail due to non-availability of resources. Default: **false**
degraded-adapters	(pc)	Array of String/ URI	Array of URIs referring to I/O adapters (NIC, crypto adapter, and virtual storage resource) attached to the partition that are degraded. Only used if the **status** property of the partition is **"degraded"**. If the partition has no degraded adapters, the array is empty.
processor-mode	(w)(pc)	String Enum	Defines how processors are allocated to the partition. One of the following values: • **"dedicated"** - All processors in the partition are to be exclusively available to this specific partition. • **"shared"** - All processors in the partition are to be shareable across partitions. Default: shared Constraint: This property can only be updated when the partition's **status** is **"stopped"**.
cp-processors	(w)(pc)	Integer	Defines the number of general purpose processors (CP) to be allocated for the partition. The value must not be larger than **processor-count-general-purpose** property of the hosting CPC. Default: 0 **Note:** Exactly one of the **cp-processors** and **ifl-processors** must have a value other than 0. Partitions can have only CPs or only IFLs but not a mix of both.

Source: Hardware Management Console Web Services API (Version 2.17.0)
https://www.ibm.com/docs/en/systems-hardware/zsystems/9175-ME1?topic=library-overview

DSLs
(Why We Need Them)

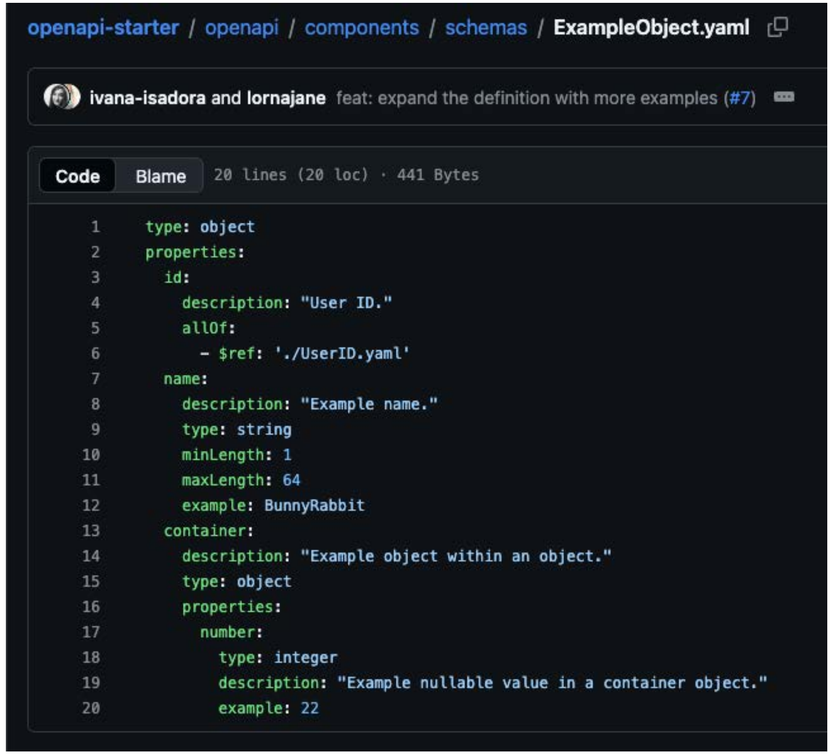
Our component manages “resources”.

Resources need representation.

Each “interface” has its own representation.

”Web Services API”

(RESTful API, offers services to users on the HMC)



Source: https://github.com/Redocly/openapi-starter/blob/main/openapi/components/schemas/ExampleObject.yaml

DSLs
(Why We Need Them)

Our component manages “resources”.

Resources need representation.

Each “interface” has its own representation.

gRPC: protobuf definitions of data structures and services

(Services are offered by SE, to be used by HMC)

```
message Person {
  string name = 1;
  int32 id = 2;
  bool has_ponycopter = 3;
}
```

```
// The greeter service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply) {}
}

// The request message containing the user's name.
message HelloRequest {
  string name = 1;
}

// The response message containing the greetings
message HelloReply {
  string message = 1;
}
```

Source: https://grpc.io/docs/what-is-grpc/introduction/

DSLs
(Why We Need Them)

Our component manages “resources”.

Resources need representation.

Each “interface” has its own representation.

Additional resource models:

- Legacy WS API endpoint implementations
- Legacy HMC<->SE communication
  (based on proprietary object model and RMI)

Each “interface” representation deviates from the others:

- (Slightly) different data types
- (Slightly) different field names

Kotlin to the Rescue!

We built a “mapping DSL” that implements a **declarative**, uni-directional mapping between two objects.

Our DSL supports:

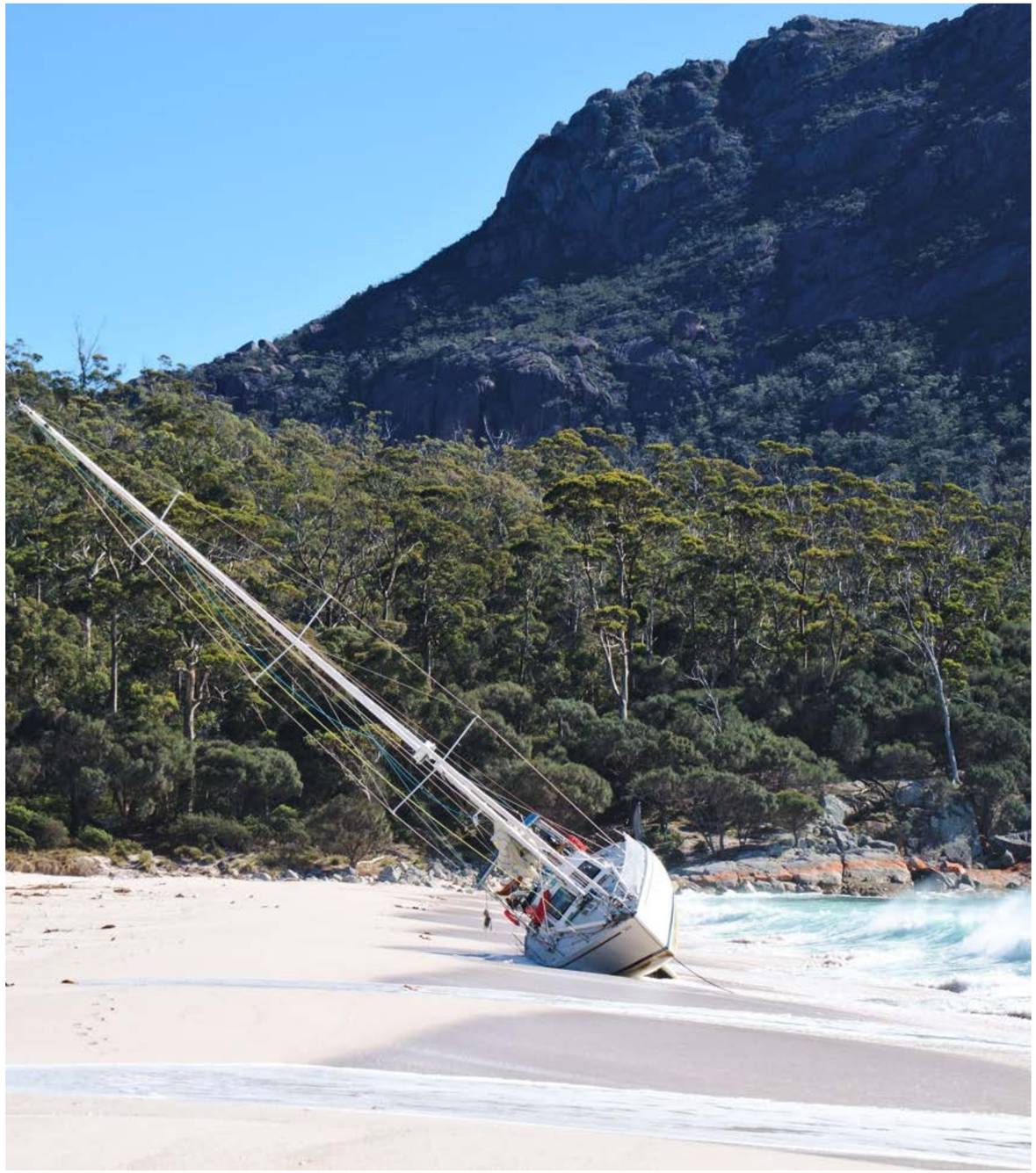
- Map a full source object to a target object
- Map individual fields
- Compute delta between two source objects, and apply to target

```
val petStoreMapping = newMapping<PetStoreGrpcModel, Map<String, Any?>, MappingContext> {
    assign(PetStoreGrpc.NAME to PetStoreLegacy.NAME_PROPERTY)
    assign(PetStoreGrpc.TYPE to PetStoreLegacy.TYPE_PROPERTY) {
        WeirdPetEnumMappings.petTypeMapping.mapFrom(it)?.toObscureRepresentation()
    }

    assign(PetStoreGrpc.NICK_NAME to PetStoreLegacy.NICK_NAME_PROPERTY) {
        if (it.isEmpty()) null else NickName(it)
    }

    compute(PetStoreLegacy.DIET_PLAN_PROPERTY) { determineDiet(theGrpcPetObject) }
    extractItems(PetStoreGrpc.CUDDLING_NEEDS) {
        matchFirstOrSetDefaults({ it.type == PetTypes.SNAKE }) {
            ...
        }
    ...
    }
}
```

Part 4: Surprises



Developer Environment

Before Kotlin:

- Gradle build works …
  - fine with IntelliJ
  - somehow with Eclipse and VS Code

Developer Environment

Before Kotlin:

- Gradle build works …
  - fine with IntelliJ
  - somehow with Eclipse and VS Code

Initial support for Kotlin:

- Gradle build works …
  - fine with IntelliJ
  - somehow, but less good with Eclipse and VS Code

Developer Environment

Before Kotlin:

- Gradle build works …
  - fine with IntelliJ
  - somehow with Eclipse and VS Code

Initial support for Kotlin:

- Gradle build works …
  - fine with IntelliJ
  - somehow, but less good with Eclipse and VS Code

Enhanced support for Kotlin:

- Gradle build works …
  - fine with IntelliJ
  - (almost) no more with Eclipse and VS Code

Developer Environment

Before Kotlin:

- Gradle build works …
  - fine with IntelliJ
  - somehow with Eclipse and VS Code

Initial support for Kotlin:

- Gradle build works …
  - fine with IntelliJ
  - somehow, but less good with Eclipse and VS Code

Enhanced support for Kotlin:

- Gradle build works …
  - fine with IntelliJ
  - (almost) no more with Eclipse and VS Code

Enterprise penalty:

- IntelliJ UE: costs money
- Some people are still using Eclipse / VS Code, so gradle setup can not be simplified

Hidden Cracks

“What changed?”

When you work on a problem,
and make changes, you have to
identify all affected output files
(for testing).

(Easy when you ship JARs,
but we ship .class files 😳)

Hidden Cracks

“What changed?”

When you work on a problem, and make changes, you have to identify all affected output files (for testing).

(Easy when you ship JARs, but we ship .class files 😳)

”Correct” solution:

- Run two full (remote) “verification” builds (with and without changes)
- Identify delta in output folders

(each build: 30,40 minutes 😳)

Hidden Cracks

“What changed?”

When you work on a problem, and make changes, you have to identify all affected output files (for testing).

(Easy when you ship JARs, but we ship .class files 😳)

”Correct” solution:

- Run two full (remote) “verification” builds (with and without changes)
- Identify delta in output folders

(each build: 30,40 minutes 😳)

“Affordable” solution:

- Run a local Gradle build (2, 3 minutes)
- Use heuristics to identify .class files that changed

Hidden Cracks

“What changed?”

When you work on a problem, and make changes, you have to identify all affected output files (for testing).

(Easy when you ship JARs, but we ship .class files 😳)

”Correct” solution:

- Run two full (remote) “verification” builds (with and without changes)
- Identify delta in output folders

(each build: 30,40 minutes 😳)

“Affordable” solution:

- Run a local Gradle build (2, 3 minutes)
- Use heuristics to identify .class files that changed

Some Kotlin features (e. g. default parameters) break binary compatibility.

The heuristics don’t work any more!

Larger Zoo

The “old” world:

stuck with Java 8/11 ”forever”.

(we are using Java 17 for a
while though)

Larger Zoo

The “old” world:

stuck with Java 8/11 ”forever”.

(we are using Java 17 for a while though)

Backlevel? No more!

”Prepare for Java 25. Now!”

Larger Zoo

The “old” world:

stuck with Java 8/11 ”forever”.

(we are using Java 17 for a while though)

Backlevel? No more!

”Prepare for Java 25. Now!”

Java 25 needs **Gradle 9**

Gradle 9 needs **Kotlin 2**

… affecting Kotlin deliveries!

Larger Zoo

The “old” world:

stuck with Java 8/11 ”forever”.

(we are using Java 17 for a while though)

Backlevel? No more!

”Prepare for Java 25. Now!”

Java 25 needs **Gradle 9**

Gradle 9 needs **Kotlin 2**

… affecting Kotlin deliveries!

Enterprise penalty:

Multiple teams are using Kotlin by now, so more and more synchronisation is needed.

Skill Gaps

Kotlin allows for a very high level of abstraction...

That is what we want, right?!

Skill Gaps

Kotlin allows for a very high level of abstraction…

That is what we want, right?!

Of course!

But remember: don’t do things because you can – but because they make sense!

Instead: make sure you understand the consequences of your decisions!

Skill Gaps

Kotlin allows for a very high level of abstraction…

That is what we want, right?!

Of course!

But remember: don’t do things because you can – but because they make sense!

Instead: make sure you understand the consequences of your decisions!

“Clean code” still applies!

See

Kotlin clean code and best practices

for example.

Skill Gaps

Kotlin allows for a very high level of abstraction…

That is what we want, right?!

Of course!

But remember: don’t do things because you can – but because they make sense!

Instead: make sure you understand the consequences of your decisions!

“Clean code” still applies!

See

Kotlin clean code and best practices

for example.

This is not a new problem!

See

Scala levels: beginner to expert, application programmer to library designer

for example.

TL;DR (Summary)

Kotlin is a mature technology:

- Adding it to an existing project can be straight forward
- The promised gains are real

But: it is not a silver bullet, and it introduces new challenges!

Education is essential:

- “Reading” Kotlin code is rather easy
- Taking full advantage of all benefits requires plenty of learning

Questions?

Comments?

Thank you!