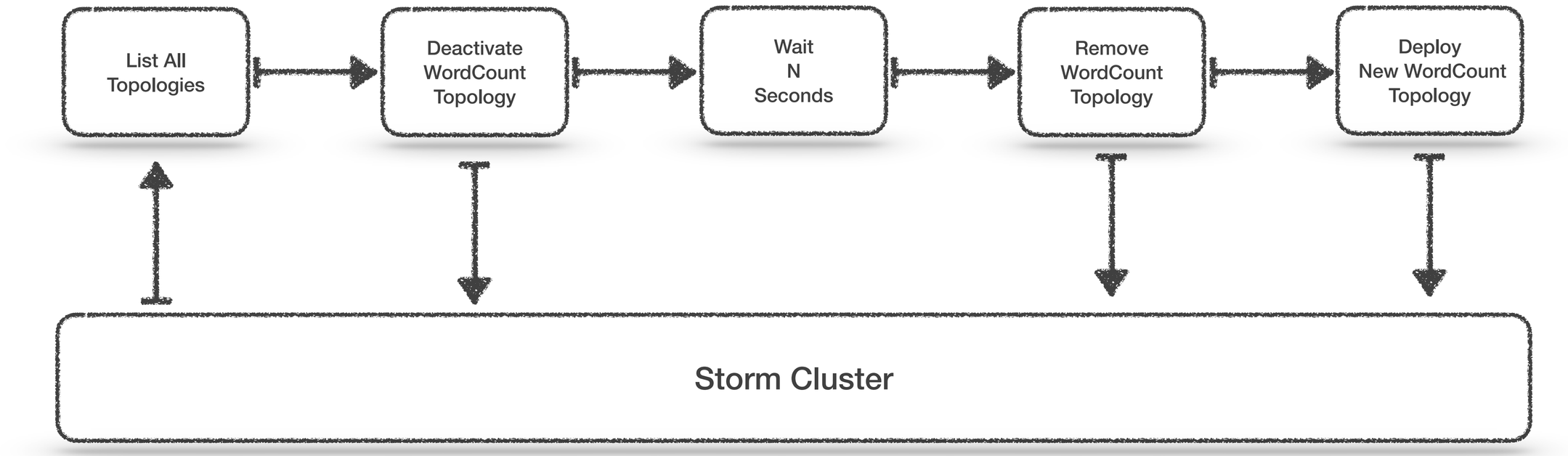
Topology Insights

Storm

1. Topology Deployment;
2. Dependency Management;
3. Why Scala;
4. Mock Tests;
5. Metrics.

Topology Deployment



Topology Deployment

Command Line Client

```
➜  apache-storm bin/storm list
11:43:00.512 [main] INFO  o.a.s.u.NimbusClient - Found leader nimbus : iopo0782.home:6627
Topology_name        Status     Num_tasks  Num_workers  Uptime_secs  Topology_Id          Owner
-------------------------------------------------------------------------------
WordCountTopology    ACTIVE     28         3            748          WordCountTopology-1-1637321432 veigam
```

```
➜  apache-storm bin/storm Kill WordCountTopology
11:49:29.138 [main] INFO  o.a.s.c.Deactivate - Killed topology: WordCountTopology -w 30
```

```
➜  apache-storm bin/storm jar wordCount-storm-1.0.0-SNAPSHOT.jar com.ppb.feeds.wordCount.storm.WordCountTopology
Start uploading file 'wordCount-storm-1.0.0-SNAPSHOT.jar' to ‘/apache-storm-2.3.0/storm-local/nimbus/inbox/
stormjar-2ab0d474-93a5-4eee-91ff-d920cd713503.jar' (18076737 bytes)
[==================================================] 18076737 / 18076737
```

Topology Deployment

Programmatically

```
object WordCountDeployer {

  def main(args: Array[String]): Unit = {

    val wordCount     = new WordCountTopology
    val nimbusClient  = NimbusClient.getConfiguredClient(wordCount.configs()).getClient
    val oldTopology   = getTopologiesName(nimbusClient, wordCount.name())

    killTopology(nimbusClient, oldTopology.head, 30)
    StormSubmitter.submitTopology(wordCount.name(), wordCount.configs(), wordCount.create())
 }

  def getTopologiesName(nimbusClient: Nimbus.Client, prefixName: String): Seq[String] = {…}
  def killTopology(nimbusClient: Nimbus.Client, name: String, waitBeforeKill: Int): Try[String] = Try {…}
```

/Flutter-Global/topology-management-lib

Dependency Management

Inversion of Control (IOC) is a software design principle where the responsibility of controlling the application components and its flow is transferred to external containers or frameworks.

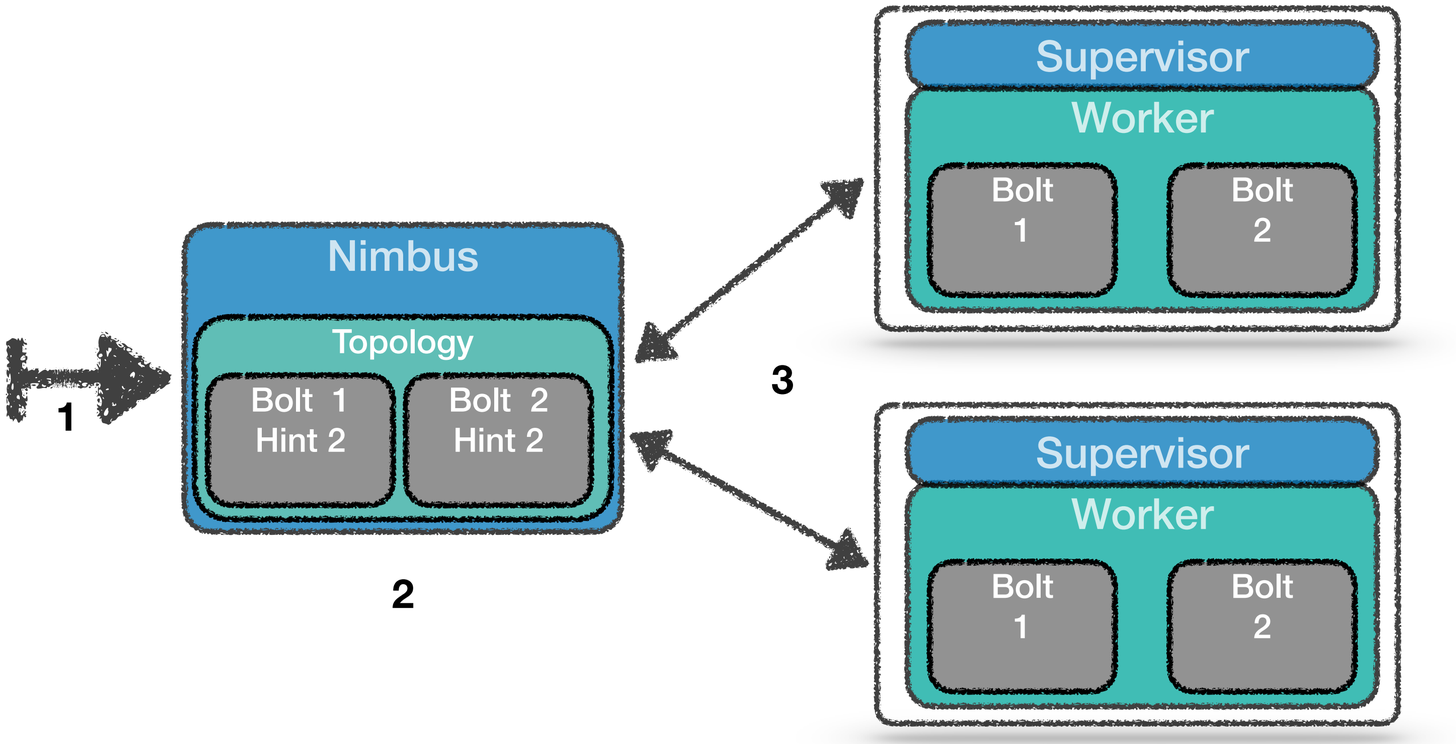
This allows:

- Decoupling the execution of a component from its implementation;
- Better switching between different implementations;
- Greater modularity by breaking an application into isolated components;
- Easier to test.

Dependency Management

Dependency injection (DI) is a design pattern that can be used to apply IOC. In DI, dependencies of an object are provided (injected) by means of an orchestrator instead of having the object construct them itself. Normally, this can be achieved with the help of IOC Container frameworks like Spring or libraries like Guice.

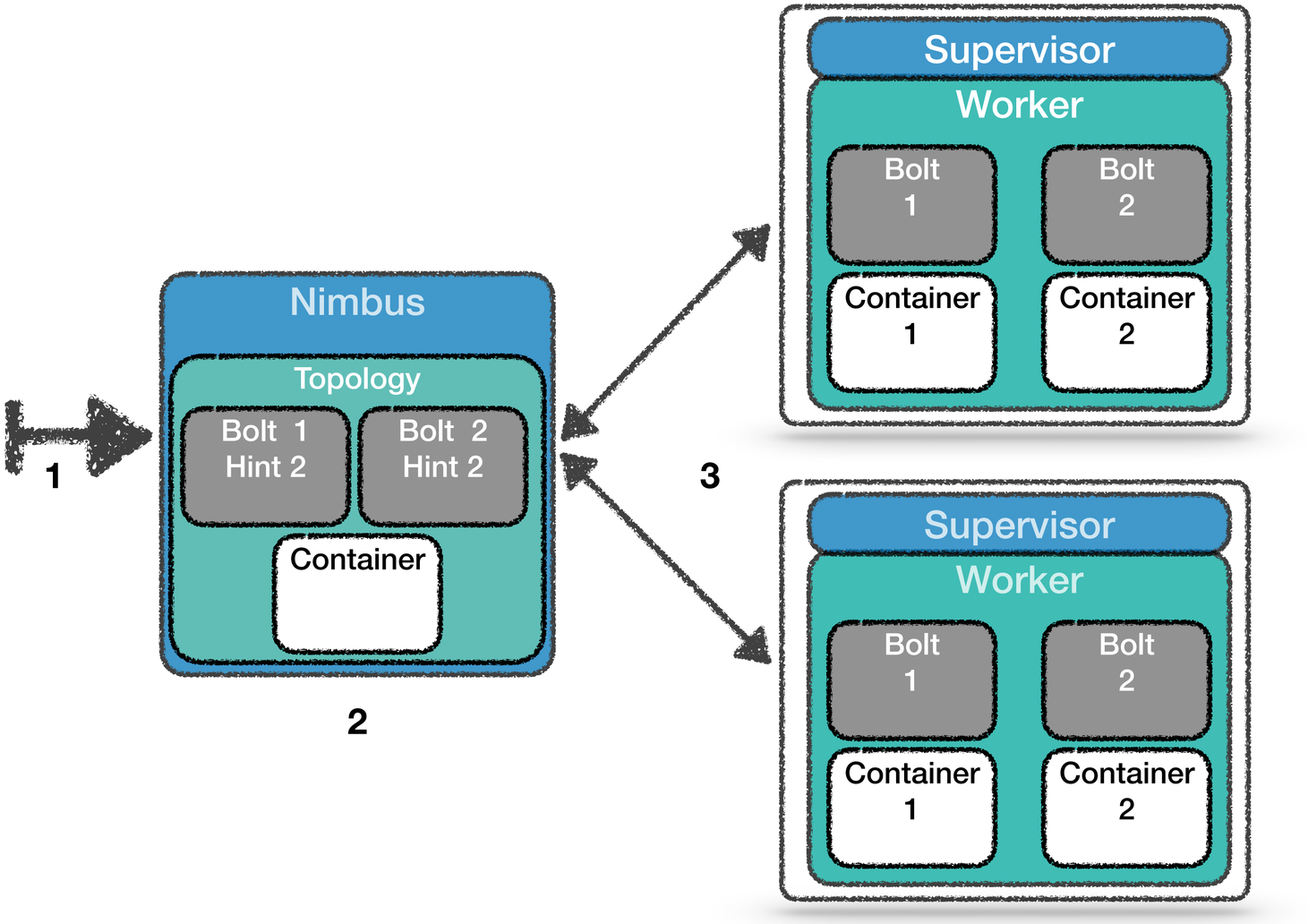
Dependency Management



Topology Submission

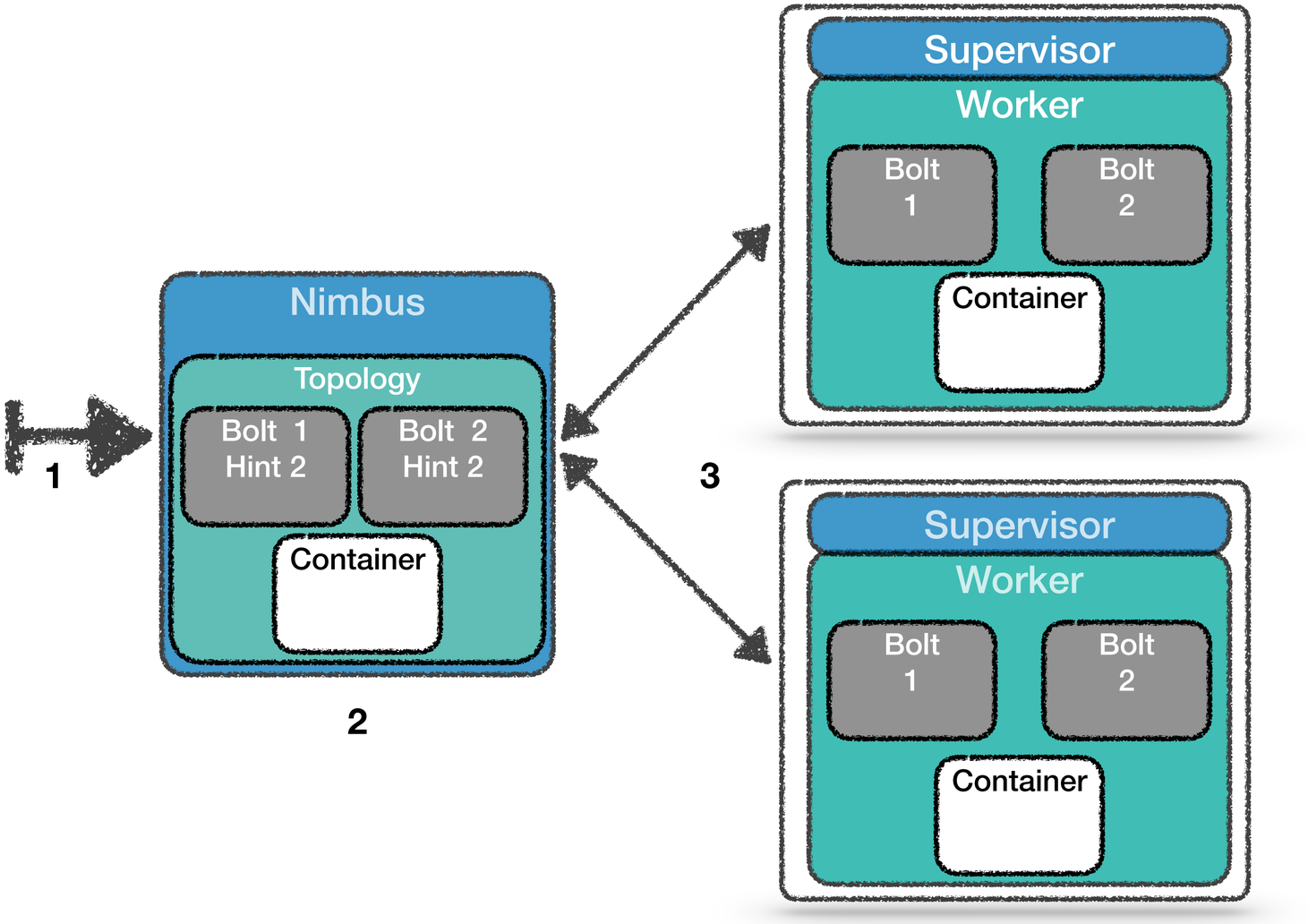
1. The topology's Jar and Config files are uploaded to Nimbus;
2. Nimbus builds the topology and serializes it;
3. The serialized topology is sent to the available workers;

Dependency Management



- Higher resource consumption because of each bolt having a dedicated IOC container;
- Various container configurations, more difficult manage.

Dependency Management



- One IOC container per worker shared by all bolt instances;
- It is necessary to synchronize the initialization/access to the container;
- This leads to a greater contention and increased topology startup times;
- Continue to depend on a external dependency (IOC container).

Why Scala?

Scala is a strongly statically typed language that:

- Allows for a more concise and expressive syntax;
- Supports object-oriented programming (OOP) paradigm;
- Supports functional programming (FP) paradigm;
- Has a sophisticated type inference system;
- Runs on the Java Virtual Machine (JVM)

Why Scala?

Thin Cake Pattern - A pure Scala approach to Modularity and DI, where the application is broken down into modules and orchestrated using Scala traits.

/using-scala-traits-as-modules-or-the-thin-cake-pattern

Why Scala?

```
trait SimpleTopology extends SpoutModule with ProcessorModule with OutputModule {

  val SPOUT     : String = "SimpleSpout"
  val PROCESSOR : String = "SimpleBolt"
  val OUTPUT    : String = "SecondSimpleBolt"


  def name() = "SimpleTopology"

  def create(): StormTopology = {

    val builder = new TopologyBuilder()

    builder.setSpout(SPOUT, getSpout(), 1)

    builder.setBolt(PROCESSOR, getProcessor(), 1).shuffleGrouping(SPOUT)

    builder.setBolt(OUTPUT, getOutput(), 1).shuffleGrouping(PROCESSOR)

    builder.createTopology()
  }

  def config(): Config = new Config
}
```

Why Scala?

```
trait ProcessorBolt extends BaseRichBolt {

  var output   : OutputCollector = _
  val processor: Processor[Instruction] = _

  override def prepare(map: util.Map[_, _], topologyContext: context, output: OutputCollector): Unit = {
    output = output
  }

  override def execute(tuple: Tuple): Unit = {

    val instruction = tuple.getValueByField(INSTRUCTION.name).asInstanceOf[Instruction]
    val results = processor.process(instruction)

    results.map(result => output.emit(tuple, new Values(result)))
    output.ack(tuple)
  }

  override def declareOutputFields(outputFieldsDeclarer: OutputFieldsDeclarer): Unit = {
    outputFieldsDeclarer.declare(new Fields(INSTRUCTION.name))
  }
}
```

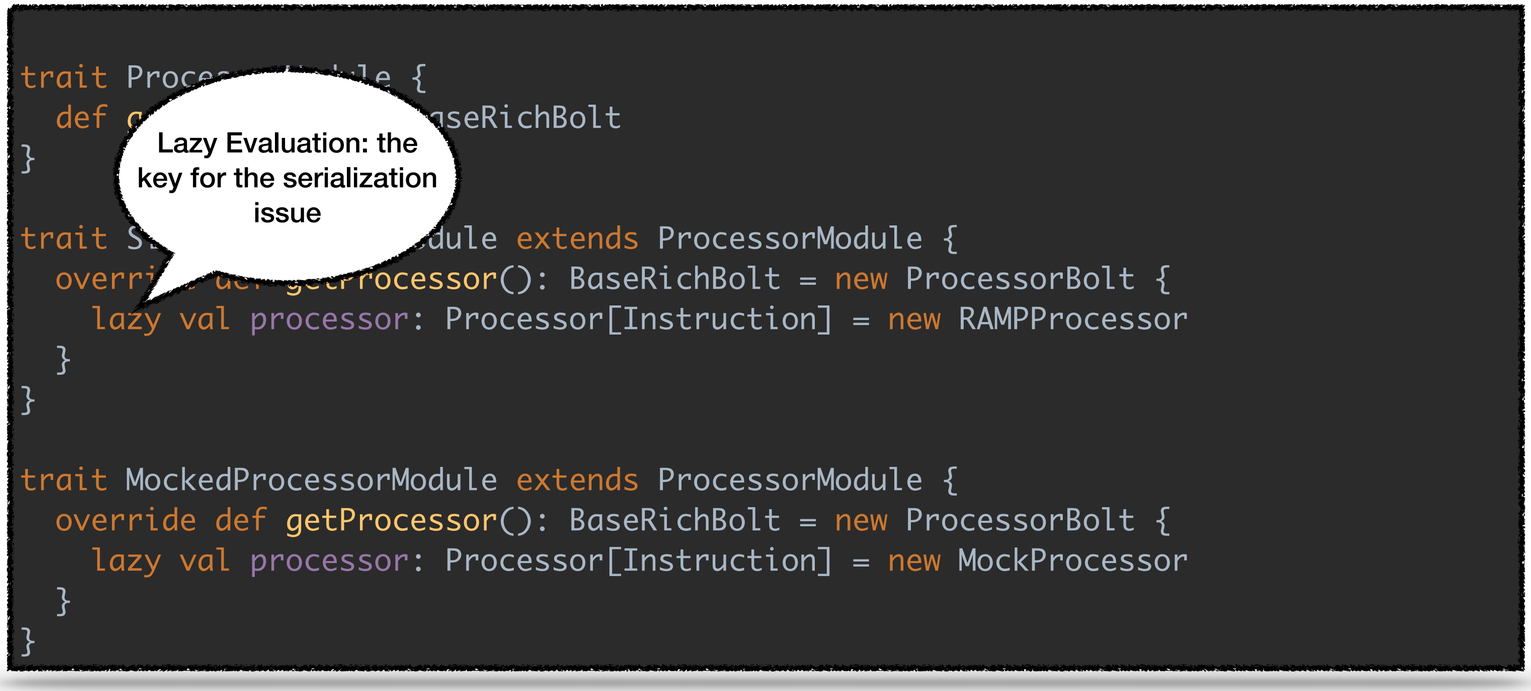
Why Scala?

```
trait ProcessorModule {
  def getProcessor(): BaseRichBolt
}

trait SimpleProcessorModule extends ProcessorModule {
  override def getProcessor(): BaseRichBolt = new ProcessorBolt {
    lazy val processor: Processor[Instruction] = new RAMPProcessor
  }
}

trait MockedProcessorModule extends ProcessorModule {
  override def getProcessor(): BaseRichBolt = new ProcessorBolt {
    lazy val processor: Processor[Instruction] = new MockProcessor
  }
}
```

Why Scala?


Lazy Evaluation: the key for the serialization issue
trait Proces le {
def g seRichBolt
}
trait S dule extends ProcessorModule {
overri def getProcessor(): BaseRichBolt = new ProcessorBolt {
lazy val processor: Processor[Instruction] = new RAMPProcessor
}
}
trait MockedProcessorModule extends ProcessorModule {
override def getProcessor(): BaseRichBolt = new ProcessorBolt {
lazy val processor: Processor[Instruction] = new MockProcessor
}
}

Why Scala?

```
object SimpleTopologyDeployer {

  def main(args: Array[String]): Unit = {

    val simpleTopology = new SimpleTopology with KafkaSpoutModule with SimpleProcessorModule with KafkaOutputModule

    StormSubmitter.submitTopology(simpleTopology.name(), simpleTopology.config(), simpleTopology.create())
  }
}
```

```
object MockedTopologyDeployer {

  def main(args: Array[String]): Unit = {

    val simpleTopology = new SimpleTopology with MockedSpoutModule with MockedProcessorModule with MockedOutputModule

    StormSubmitter.submitTopology(simpleTopology.name(), simpleTopology.config(), simpleTopology.create())
  }
}
```

Mock Tests

```
"SimpleTopology" should “consume an instruction, process it and output the result" in {

  Given("a simple create instruction")

  val instructions = List(Instruction()
    .withIType(CREATE)
    .withHeaders(Map(MeterKey.ID.name -> "5769609"))
    .withEvent("mocked event"))

  val simpleTopology = new SimpleTopology
    with MockSpoutModule
    with MockProcessorModule
    with MockOutputModule

  When(“the instruction is processed by the topology")

  val results = runTopology(Map(simpleTopology.SPOUT -> instructions), simpleTopology)

  Then("the result contains two valid tuples emitted by the Kafka Spout and Processor Bolt”)

  val instructionsTuple = results.get(simpleTopology.SPOUT)
  val processorTuple    = results.get(simpleTopology.PROCESSOR)

  instructionsTuple.isDefined shouldBe true
  processorTuple.isDefined shouldBe true
```

Mock Tests

ScalaTest: A simple Testing Library for Scala

```
"SimpleTopology" should “consume an instruction, process it and output the result" in {

  Given("a simple create instruction")

  val instructions = List(Instruction()
    .withIType(CREATE)
    .withHeaders(Map(MeterKey.ID.name -> "5769609"))
    .withEvent("mocked event"))

  val simpleTopology = new SimpleTopology
    with MockSpoutModule
    with MockProcessorModule
    with MockOutputModule

  When(“the instruction is processed by the topology")

  val results = runTopology(Map(simpleTopology.SPOUT -> instructions), simpleTopology)

  Then("the result contains two valid tuples emitted by the Kafka Spout and Processor Bolt”)

  val instructionsTuple = results.get(simpleTopology.SPOUT)
  val processorTuple    = results.get(simpleTopology.PROCESSOR)

  instructionsTuple.isDefined shouldBe true
  processorTuple.isDefined shouldBe true
```

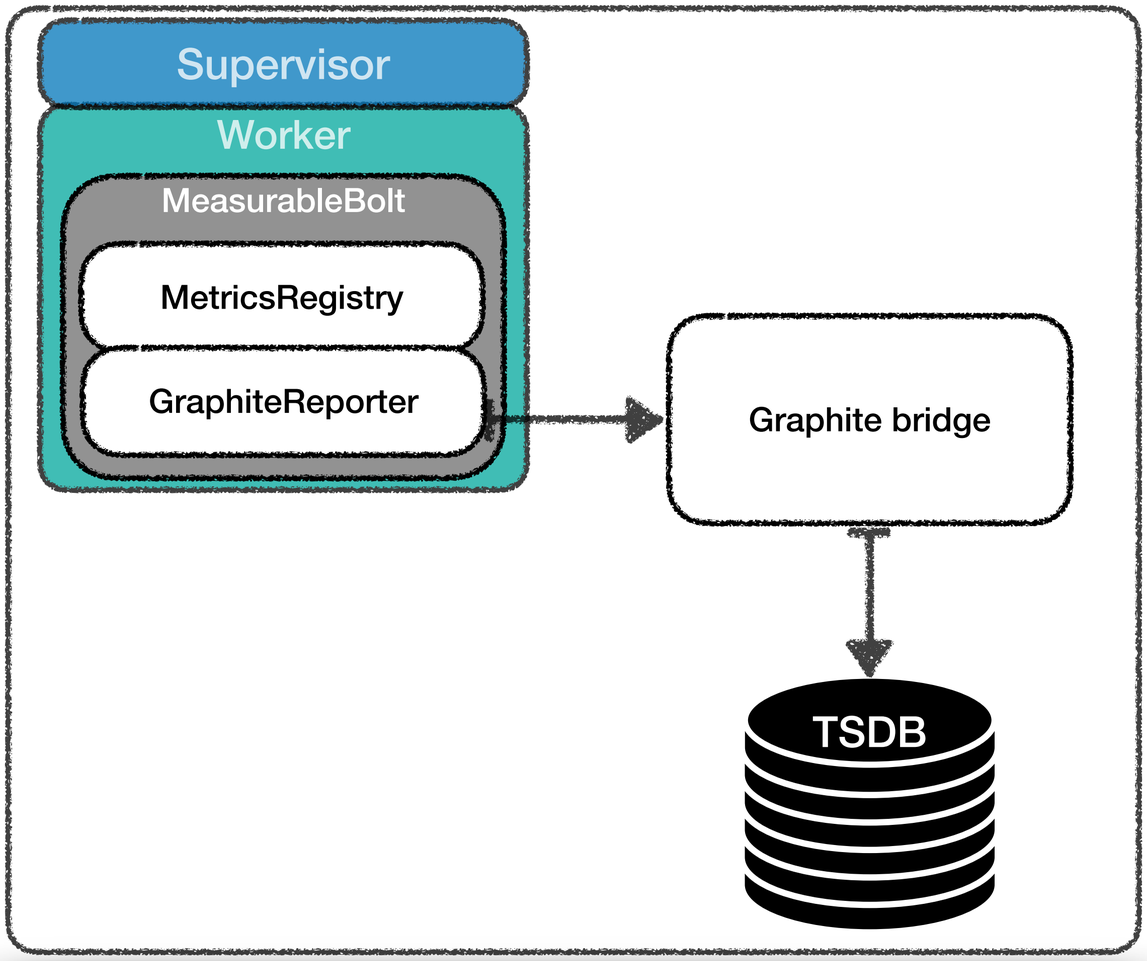
Mock Tests


ScalaTest: A simple Testing Library for Scala
"SimpleTopology" should “consume an instruction, process it and output the result" in {
Given("a simple create instruction")
val instructions = List(Instruction()
.withIType(CREATE)
.withHeaders(Map(MeterKey.ID.name -> "5769609"))
.withEvent("mocked event"))
val simpleTopology = new SimpleTopology
with MockSpoutModule
with MockProcessor
with MockOutputMo
It’s using storm.Testing LocalCluster
When(“the instruction
the topology")
val results = runTopology(Map(simpleTopology.SPOUT -> instructions), simpleTopology)
Then("the result contains two valid tuples emitted by the Kafka Spout and Processor Bolt”)
val instructionsTuple = results.get(simpleTopology.SPOUT)
val processorTuple = results.get(simpleTopology.PROCESSOR)
instructionsTuple.isDefined shouldBe true
processorTuple.isDefined shouldBe true

Mock Tests

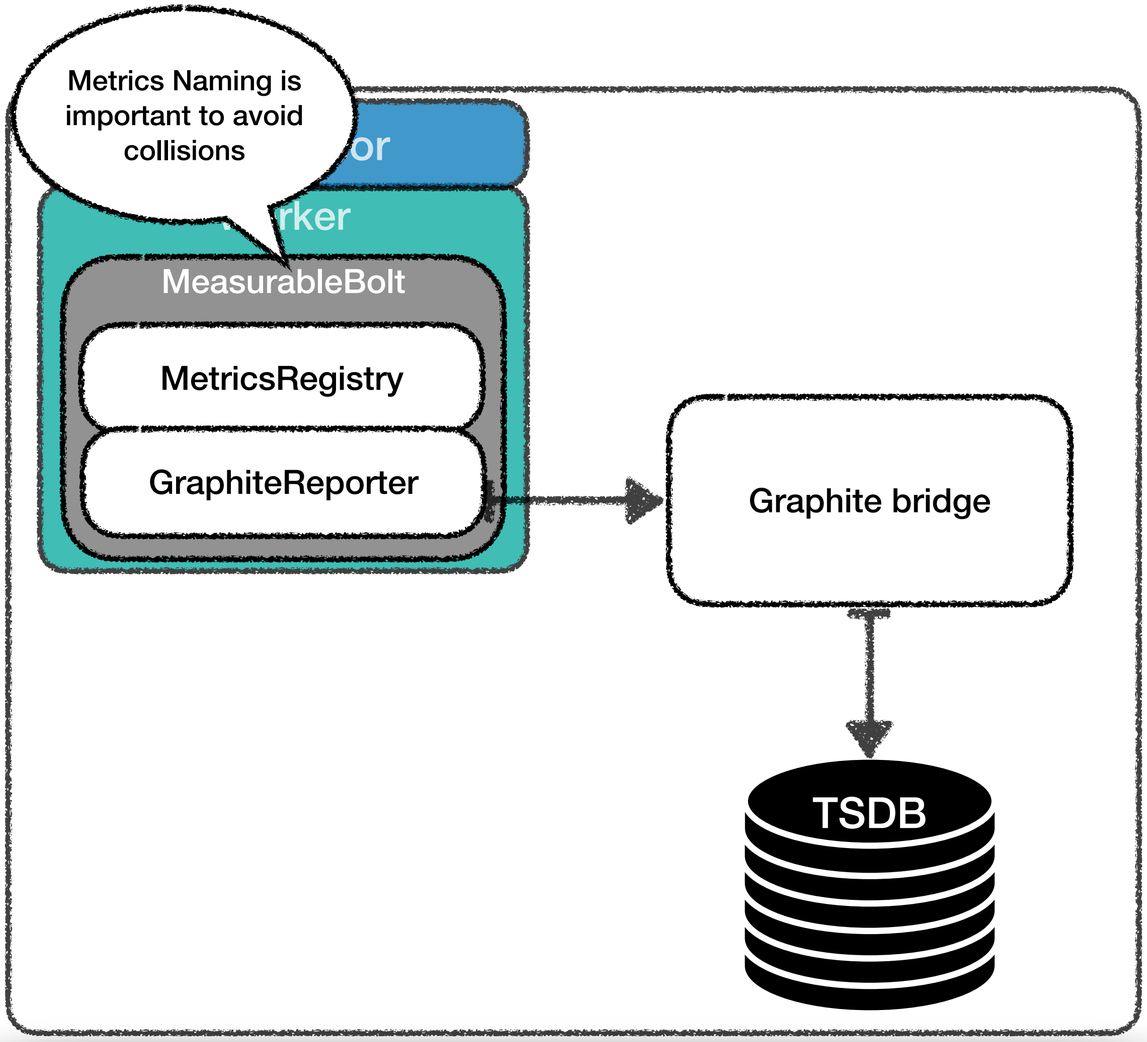

ScalaTest: A simple Testing Library for Scala
"SimpleTopology" should “consume an instruction, process it and output the result" in {
Given("a simple create instruction")
val instructions = List(Instruction()
.withIType(CREATE)
.withHeaders(Map(MeterKey.ID.name -> "5769609"))
.withEvent("mocked event"))
val simpleTopology = new SimpleTopology
with MockSpoutModule
with MockProcessor
with MockOutputMo
It’s using storm.Testing LocalCluster
When(“the instruction
the topology")
val results = runTopology(Map(simpleTopology.SPOUT -> in
opology)
It only captures emitted tuples from Spouts/Bolts
Then("the result contains two valid tuples emitted by the
rocessor Bolt”)
val instructionsTuple = results.get(simpleTopology.SPOUT)
val processorTuple = results.get(simpleTopology.PROCESSOR)
instructionsTuple.isDefined shouldBe true
processorTuple.isDefined shouldBe true

Metrics



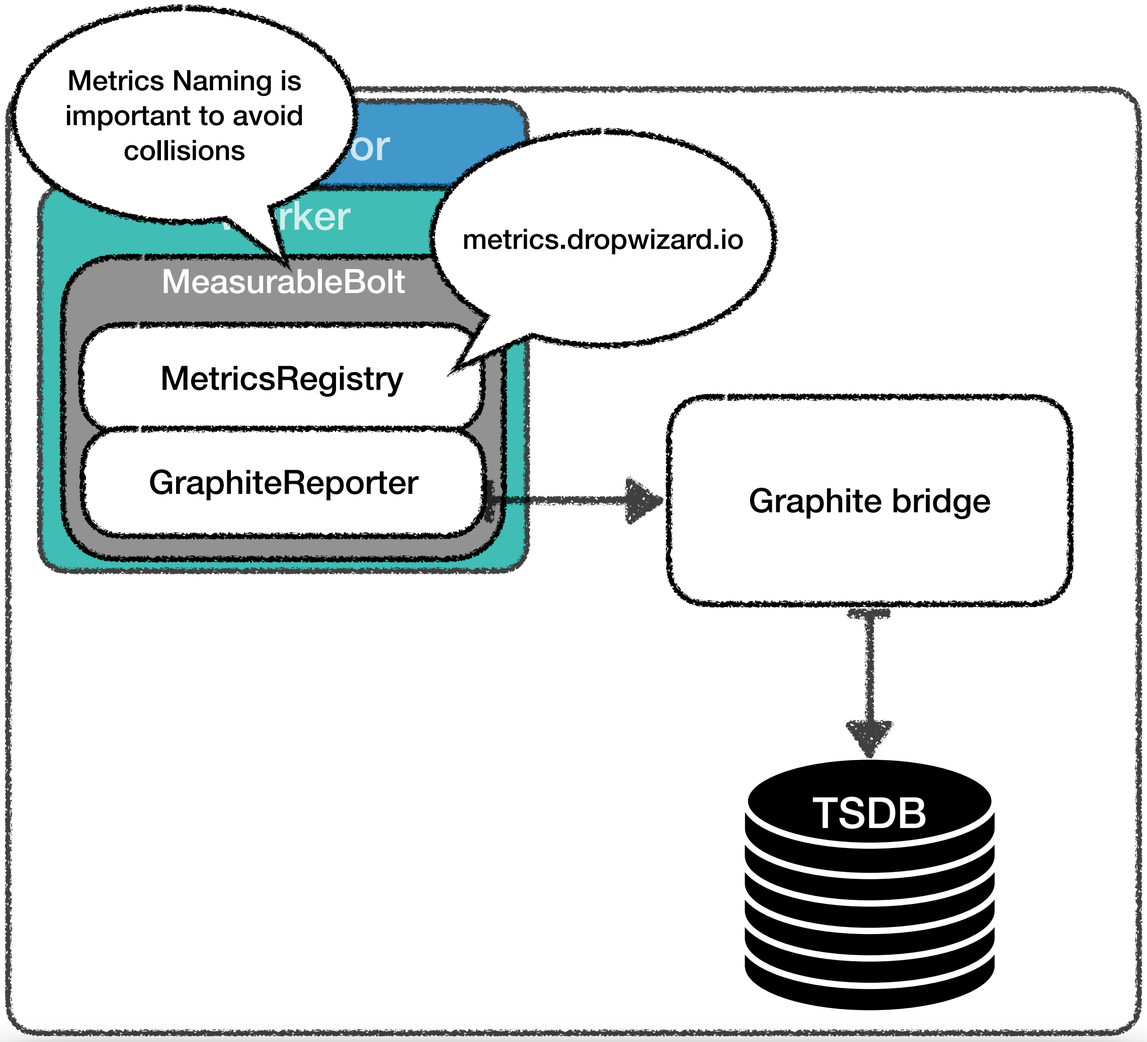
/Flutter-Global/topology-management-lib

Metrics



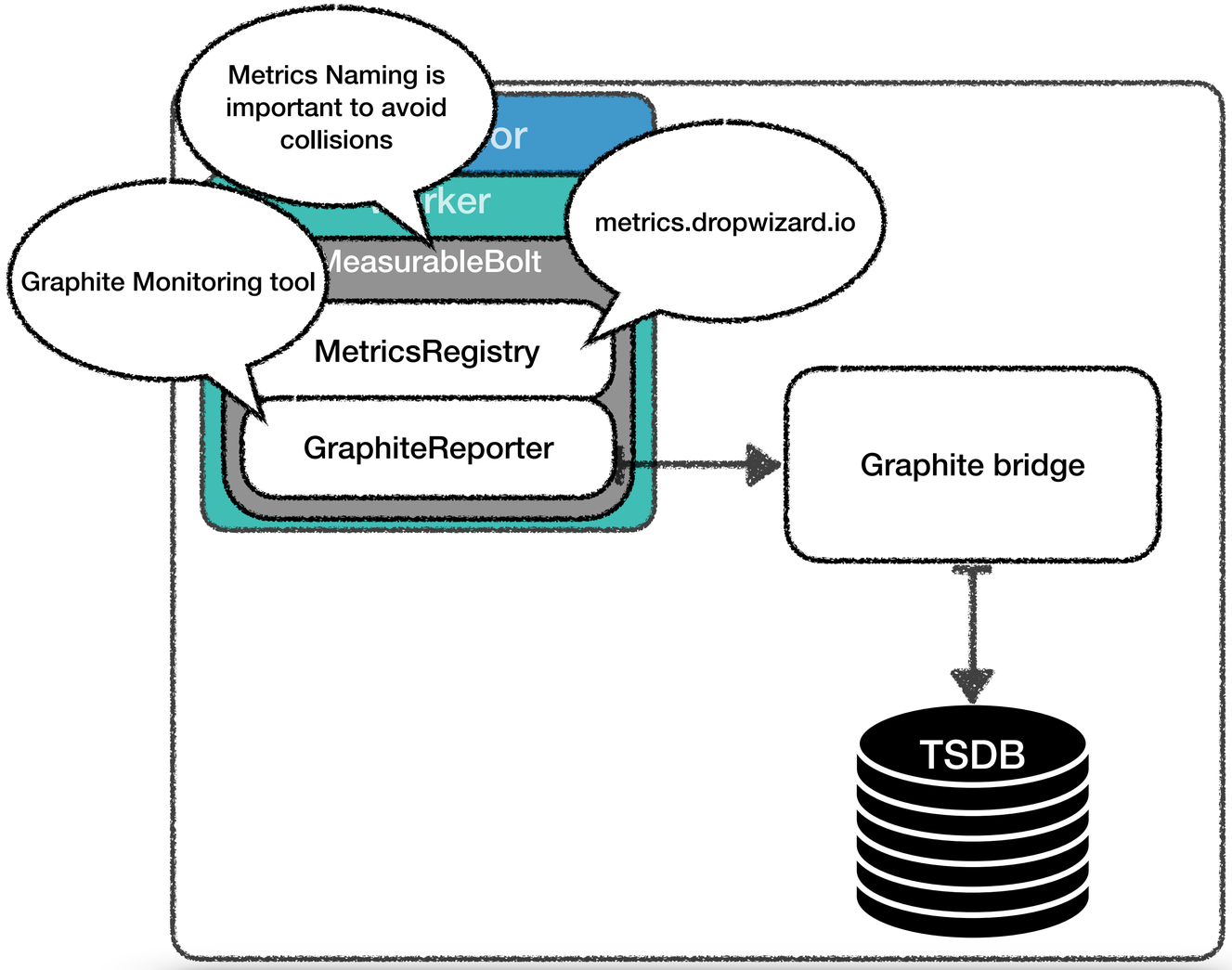
/Flutter-Global/topology-management-lib

Metrics



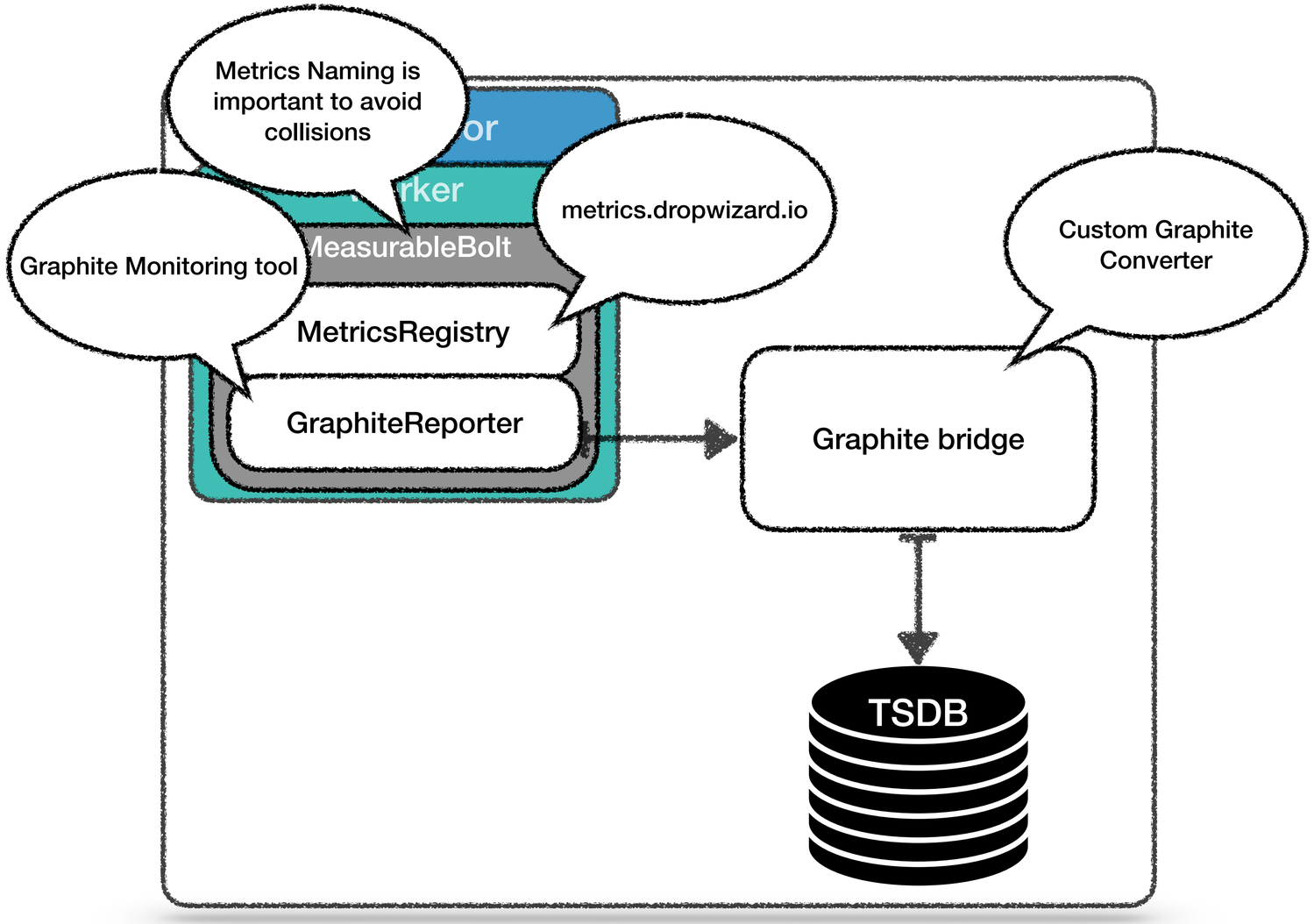
/Flutter-Global/topology-management-lib

Metrics



/Flutter-Global/topology-management-lib

Metrics



/Flutter-Global/topology-management-lib

Metrics

```
trait ProcessorBolt extends MeasurableRichBolt {

  val processor: Processor[Instruction]

  override def apply(tuple: Tuple, output: OutputCollector): Unit = {

   val instruction = tuple.getValueByField(INSTRUCTION.name).asInstanceOf[Instruction]

    val results = processor.process(instruction)

    results.map(result => output.emit(tuple, new Values(result)))
    output.ack(tuple)
    getCounter("BoltCounter").inc()
  }

  override def declareOutputFields(outputFieldsDeclarer: OutputFieldsDeclarer): Unit = {
    outputFieldsDeclarer.declare(new Fields(INSTRUCTION.name))
  }
}
```

/Flutter-Global/topology-management-lib

Metrics

Custom Bolt in management library

```
trait ProcessorBolt extends MeasurableRichBolt {

  val processor: Processor[Instruction]

  override def apply(tuple: Tuple, output: OutputCollector): Unit = {

   val instruction = tuple.getValueByField(INSTRUCTION.name).asInstanceOf[Instruction]

    val results = processor.process(instruction)

    results.map(result => output.emit(tuple, new Values(result)))
    output.ack(tuple)
    getCounter("BoltCounter").inc()
  }

  override def declareOutputFields(outputFieldsDeclarer: OutputFieldsDeclarer): Unit = {
    outputFieldsDeclarer.declare(new Fields(INSTRUCTION.name))
  }
}
```

/Flutter-Global/topology-management-lib

Metrics

Custom Bolt in management library

```
trait ProcessorBolt extends MeasurableRichBolt {

  val processor: Processor[Instruction]

  override def apply(tuple: Tuple, output: OutputCollector): Unit = {

   val instructi                etValueByField(INSTRUCTION.name).asInstanceOf[Instruction]

    val resu                    ess(instruction)

    results.map                 tput.emit(tuple, new Values(result)))
    output.ack(tuple)
    getCounter("BoltCounter").inc()
  }

  override def declareOutputFields(outputFieldsDeclarer: OutputFieldsDeclarer): Unit = {
    outputFieldsDeclarer.declare(new Fields(INSTRUCTION.name))
  }
}
```

Also has Timers, Meters, Gauges & Histograms

/Flutter-Global/topology-management-lib

Metrics

DEMO Grafana Dashboards

Exercise

Checkout STT Project

Implement a Topology that consumes strings of sentences from a Kafka topic:

- Consume from the “test-input-topic” **stream** using a KafkaSpout (“localhost:31092”);
- Process each sentence:
  - Split them into **words**;
  - Have a **counter** for each **word** that is incremented at every occurrence of that word in any sentence.
  - For each word emit a **resulting** string containing the following format "word = <word>, count = <count>"
- Publish the **results** to the “test-output-topic” **stream**;

Deploy topology:

```
bin/storm jar /stt/stt-storm/target/stt-storm-1.0.5-SNAPSHOT.jar
      com.ppb.feeds.stt.storm.topology.deploy.STTDeployer
```