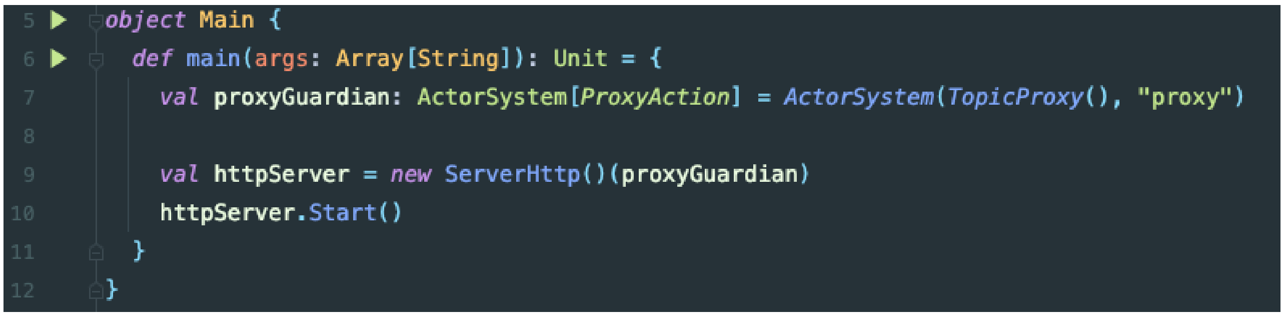
Checkpoint 1

Actor List

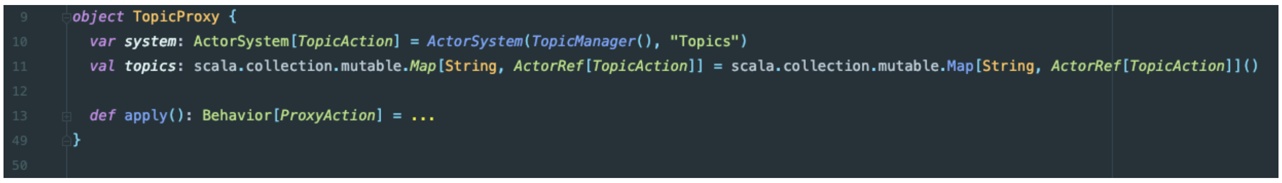
proxy (TopicProxy)

Proxy guardian (a.k.a **TopicProxy**) is the entry point of whole message broker. It accepts messages from servers and route them to appropriate topic managers. Defining multiple proxy guardians will effectively create multiple completely isolated brokers:



```scala
object Main {
  def main(args: Array[String]): Unit = {
    val proxyGuardian: ActorSystem[ProxyAction] = ActorSystem(TopicProxy(), "proxy")

    val httpServer = new ServerHttp()(proxyGuardian)
    httpServer.Start()
  }
}
```

TopicProxy holds an actor system which is used to create **TopicManagers** and a map of **string -> TopicManager**

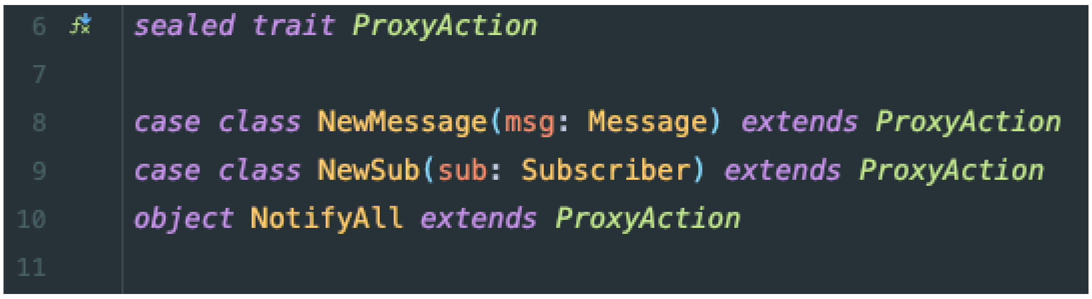


```scala
object TopicProxy {
  var system: ActorSystem[TopicAction] = ActorSystem(TopicManager(), "Topics")
  val topics: scala.collection.mutable.Map[String, ActorRef[TopicAction]] = scala.collection.mutable.Map[String, ActorRef[TopicAction]]()

  def apply(): Behavior[ProxyAction] = ...
}
```

Incoming actions:

Proxy can accept 3 different actions:

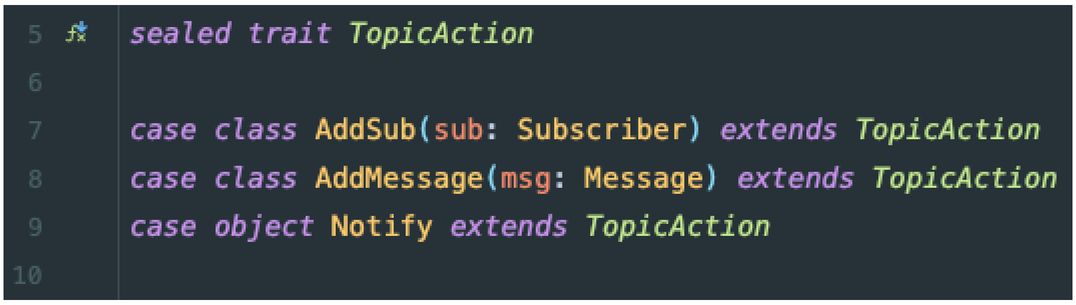
- **NewMessage** which corresponds to producer sending a new message
- **NewSub** which corresponds to subscriber subscribing to message broker
- **NotifyAll** which triggers every topic manager to dequeue a message and send it to appropriate subscribers. **NotifyAll** is a temporary development workaround because every topic should have each own custom notification policy



```scala
sealed trait ProxyAction

case class NewMessage(msg: Message) extends ProxyAction
case class NewSub(sub: Subscriber) extends ProxyAction
object NotifyAll extends ProxyAction
```

Outcoming actions:

Proxy can send 3 type of different actions to each topic manager:



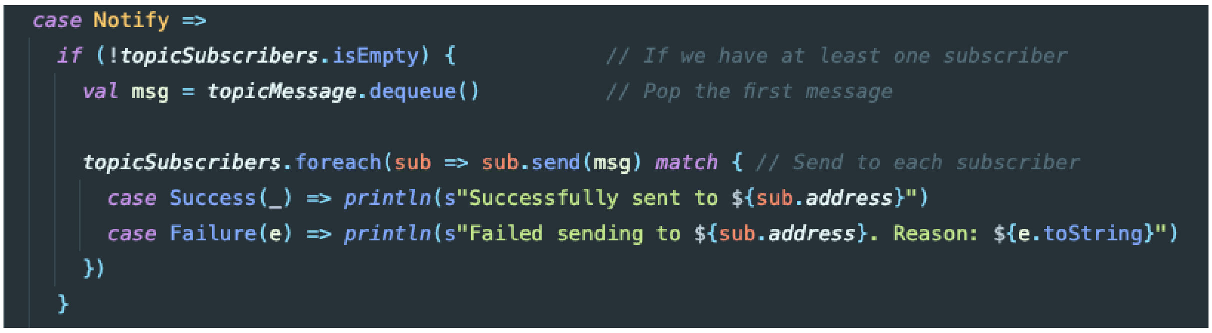
```scala
sealed trait TopicAction

case class AddSub(sub: Subscriber) extends TopicAction
case class AddMessage(msg: Message) extends TopicAction
case object Notify extends TopicAction

```

Behaviour of **TopicActions** and **ProxyActions** are same, just in different context. **TopicAction** is limited to single topic, while **ProxyAction** is being routed to target topic from Topics array (this is why it's a proxy)

<topic> (TopicManager)

TopicManager is responsible for all operations on his topic. **TopicManagers** names topic names themselves, that's why you can only have one TopicManager per topic (because actors names have to be unique).
TopicManager is being created if a consumer subscribers to a non-existent topic or producer is sending a message to a non-existent topic. Since subscribers are normal classes (not actors), **TopicManager** only receives actions and is not sending any actions

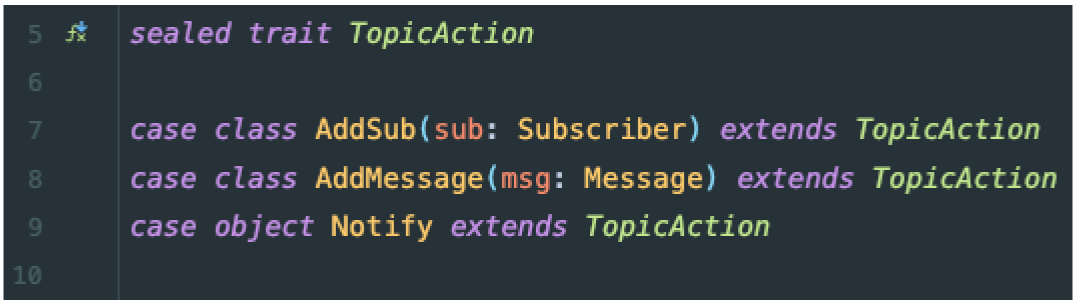
This is how **TopicManager** handles **Notify** action:



```scala
case Notify =>
  if (!topicSubscribers.isEmpty) {            // If we have at least one subscriber
    val msg = topicMessage.dequeue()          // Pop the first message

    topicSubscribers.foreach(sub => sub.send(msg) match { // Send to each subscriber
      case Success(_) => println(s"Successfully sent to ${sub.address}")
      case Failure(e) => println(s"Failed sending to ${sub.address}. Reason: ${e.toString}")
    })
  }
```

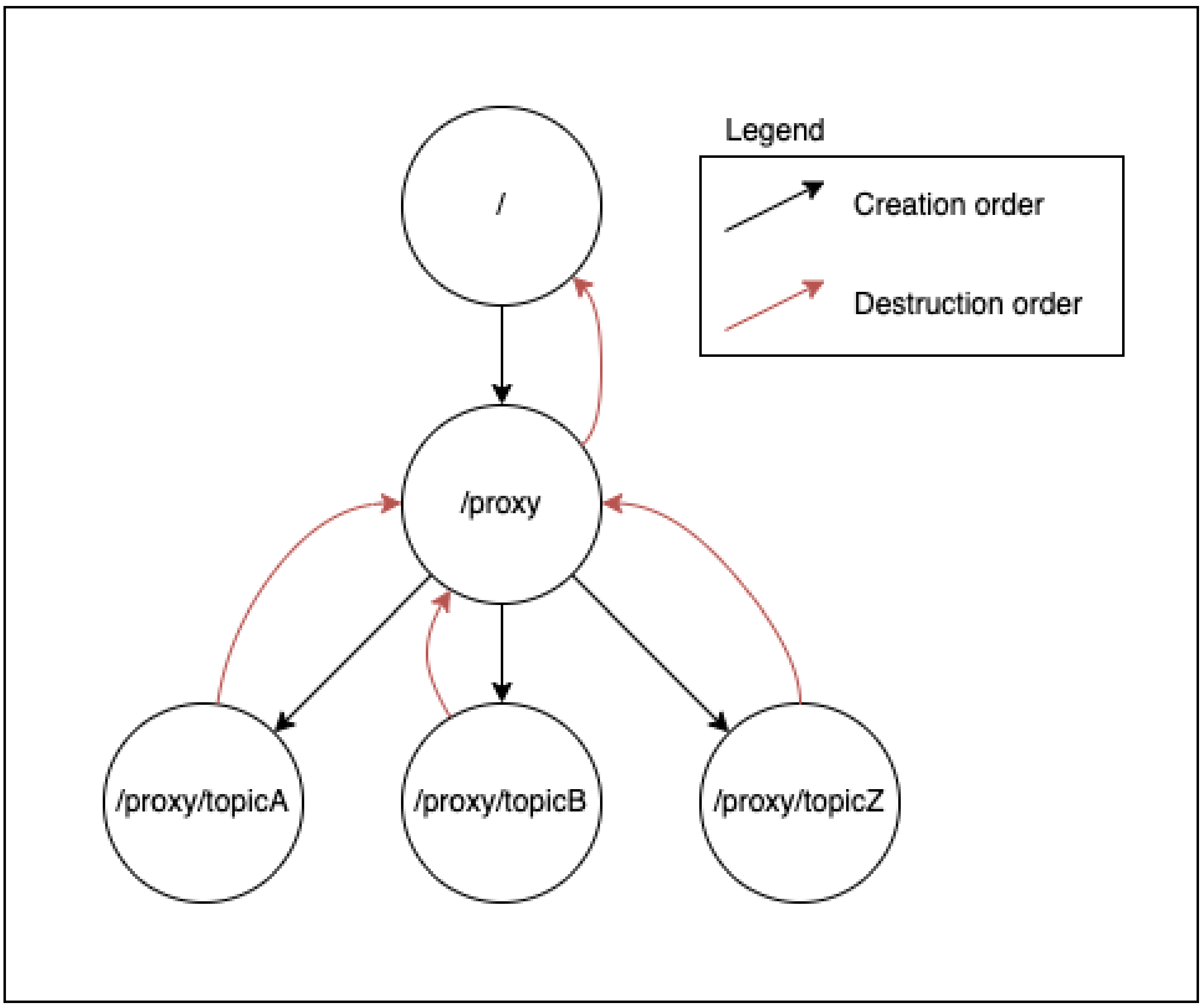
Incoming actions:

As you can guess, actions which income to **TopicManagers** are actions with outcome from **TopicProxy**



```scala
sealed trait TopicAction

case class AddSub(sub: Subscriber) extends TopicAction
case class AddMessage(msg: Message) extends TopicAction
case object Notify extends TopicAction

```

Supervision diagram

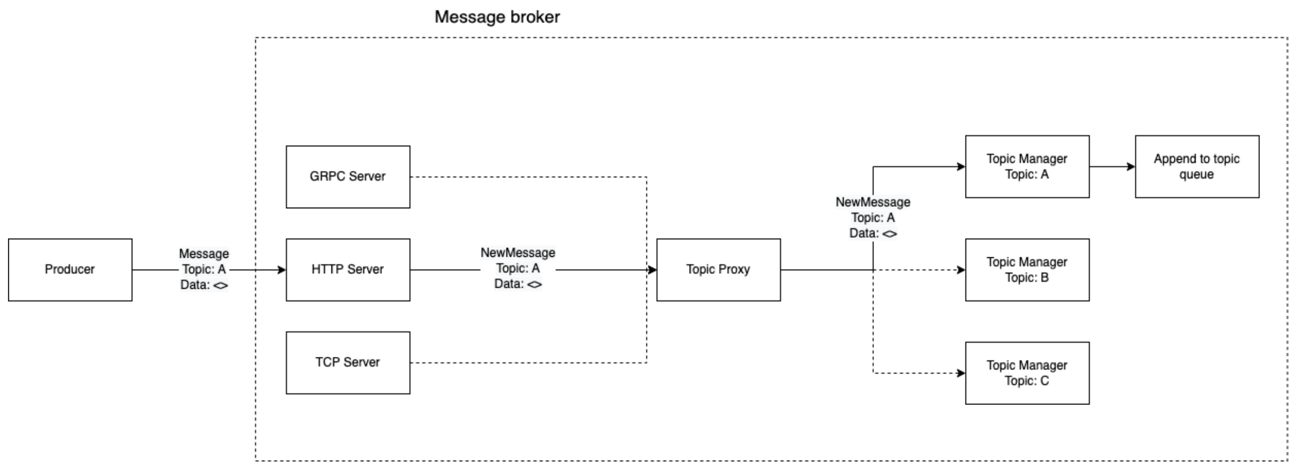


Technology list

- Scala 2.13.8
- Akka Typed Actors 2.6.19
- Akka HTTP 10.2.9
- OpenJDK 18

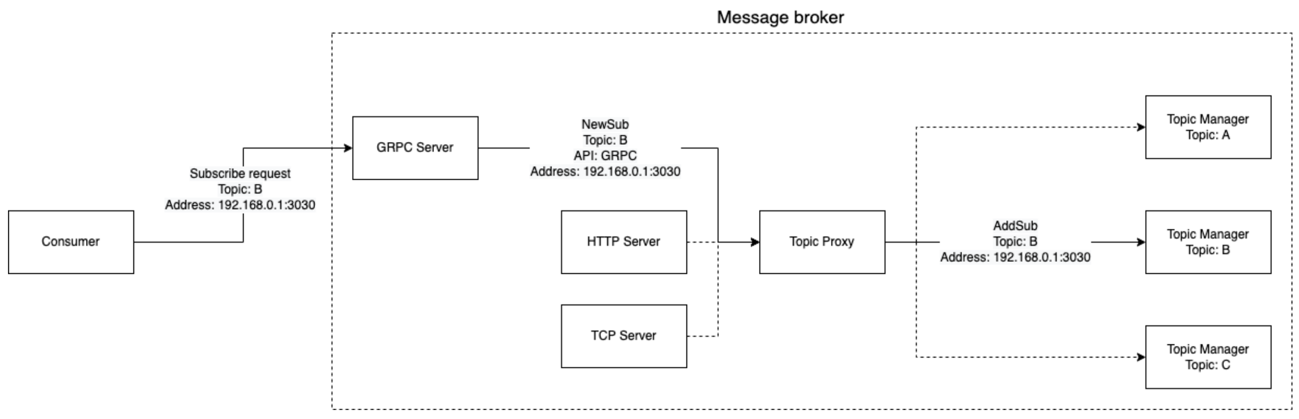
Diagram

Producer sending a message

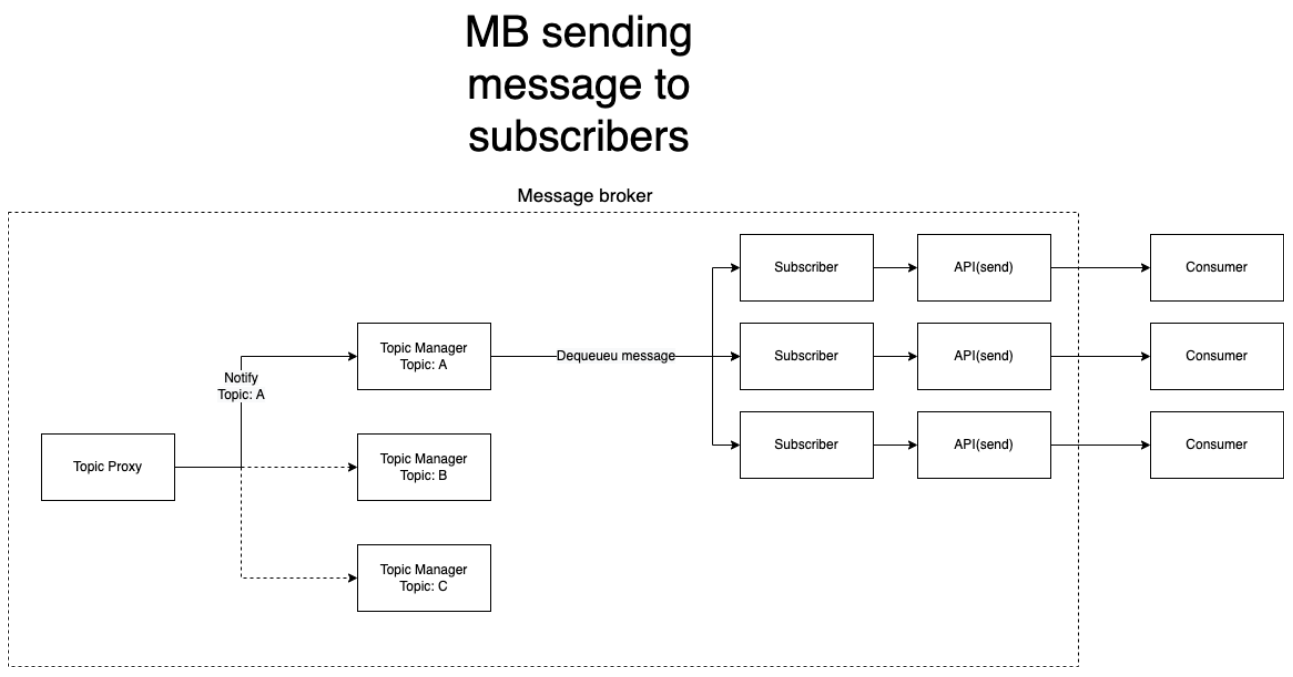


A producer is sending a request with a message on one of the servers. Server, which handles this request, creates a NewMessage action and sends it to TopicProxy actor. TopicProxy after this finds the TopicManager which is working with topic from request and sends NewMessage action. After this, TopicManager is adding this message to his message queue

Consumer subscribing to MB


Message broker
Subscribe request Topic: B Address: 192.168.0.1:3030
Consumer
GRPC Server
NewSub Topic: B API: GRPC Address: 192.168.0.1:3030
HTTP Server
TCP Server
Topic Proxy
AddSub Topic: B Address: 192.168.0.1:3030
Topic Manager Topic: A
Topic Manager Topic: B
Topic Manager Topic: C

A consumer is sending a subscribe request to one of the servers. Server composese NewSub action and sends it to TopicProxy. TopicProxy finds TopicManager which is working with topic from request and sends AddSub action. TopicManager is adding new subscriber to his subscribers array



Notification mechanism can be triggered by notification policies: timed policy, success-repeat policy, buffer-size policy and etc. Each TopicManager should have it’s own notification policy. Upon trigger, TopicProxy sends Notify action to triggered TopicManager, and this TopicManager dequeues a message and broadcasts it to every consumer

Bibliography

- https://doc.akka.io/docs/akka/2.5.32/typed/actors.html
- https://doc.akka.io/docs/akka/2.5.32/typed/actor-lifecycle.html
- https://doc.akka.io/docs/akka/current/actors.html
- https://doc.akka.io/docs/akka/2.5/guide/tutorial_1.html
- https://doc.akka.io/docs/akka-http/current/introduction.html
- https://livebook.manning.com/book/akka-in-action-second-edition/chapter-2/v-2/28