Workshop on Play and Akka (Verizon VDSI)

Nirmalya Sengupta

Customary self-sell

- (very) old hat
- C -> C++ -> Java -> Scala (bash/perl)
- Only Unix, No Microsoft! :-)
- Working as a freelance backend stack developer/mentor/architect
- Integration framework (CORBA), telecom, multiplayer gaming, rule engine, streaming analytics
- India, Ireland, Germany
- Training / Consulting whenever it interests me

Sessions: a look ahead

- Get to know Akka and Actors: the story
- Get to know Play framework: separation of M,V and C
- Put to practise, our understanding so far
- Being Reactive: what it means

Rules of the game

- Overall concepts using slides
- Whiteboard explanations
- Download code snippets, then modify (dirtying our hands), then **write more**
- Get comfortable with tools
- Questions: any time | Answers: best effort (I don’t know, and you do)
- Learning: everyone (I, and you all)
- Enjoy, reflect, look for ***Aha*** moments

Your machines: ready?

- JDK
- Eclipse
- Activator
- Git (client)
- sbt
- cURL
- browser

Download the following using git:

https://github.com/nsengupta/Akka-Lab-Projects

We are in a new era

- The free lunch is over (Herb Sutter)
  - http://www.gotw.ca/publications/concurrency-ddj.htm
- Demands of users have grown manifold: stability, availability
- Cloud-based services
  - Underscore the necessity and importance of making software scalable
- Problems are complex
- .. but, solution should be simpler and manageable

Present day applications..

- Reactive manifesto
  - Require to be Responsive
  - Require to be Resilient
  - Require to be scalable
  - Require to be amenable to ***Let it crash*** approach

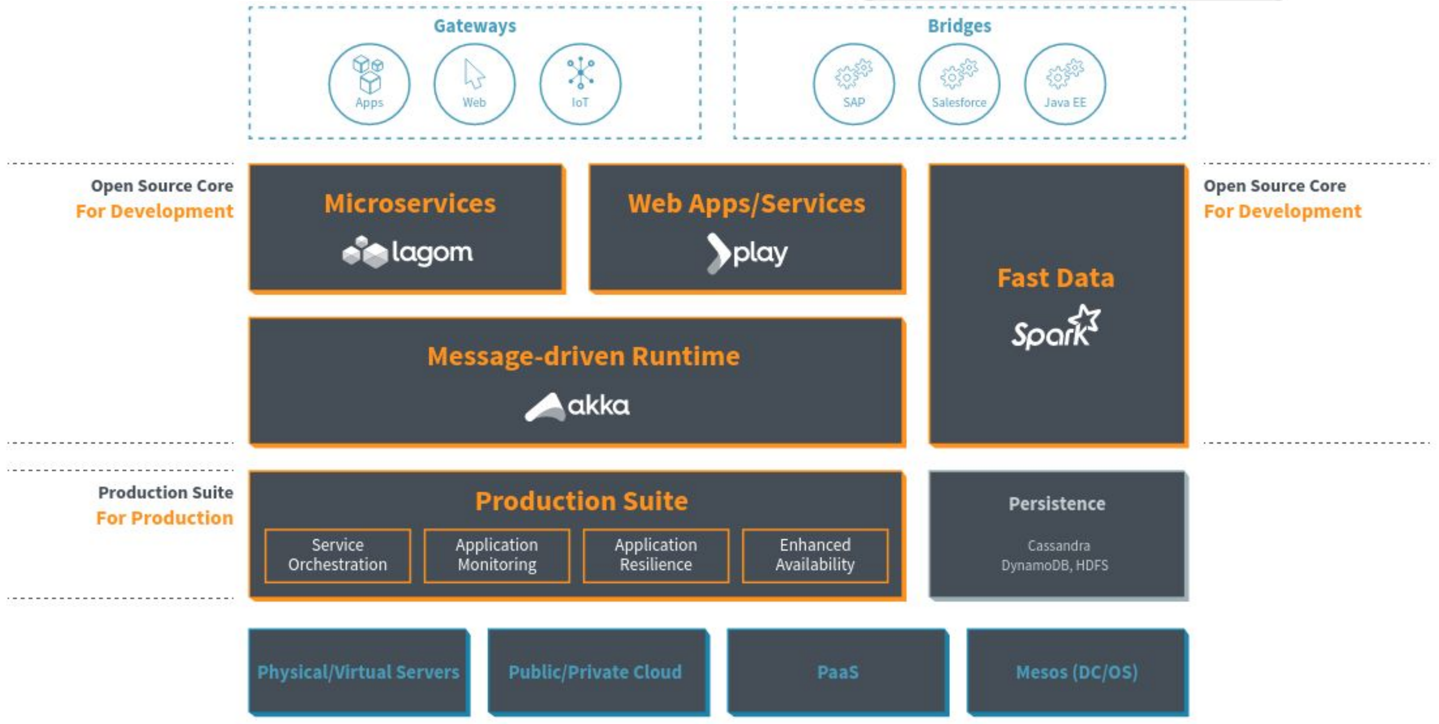
Actor model: brief overview

- A paper by Carl Hewitt, Peter Bishop, Richard Steiger (1973)
- Erlang in Ericsson
  - Search for suitable programming language for Next-G Switches (late 1980s)
  - Joe Armstrong, Robert Virding, Mike Williams (supervisor: Bjarne Dacker)
  - Akka's Actor model is heavily influenced by that of Erlang's
- An actor memory footprint is small (~400 bytes); millions can exist

Actor model: key points

- Actors are Data Structures, reside in memory
- Actors act on messages, internals are encapsulated
- Messages may or may not reach: be prepared
- For a pair of Actors: order of arrival of messages is guaranteed
- If an Actor receives messages from two Actors, the messages may be interspersed

A quick look at Lightbend stack



https://www.lightbend.com/platform

Finally, we meet Akka

- Is centered on philosophy of Actors (Scala and Java)
- Provides Actors with built-in (and mostly unseen) MessageBoxes
- Allows Actors to send and receive messages (and act upon them)
- Expects the messages to be **immutable** data structures (& serializable)
- Helps identifying an Actor but irrespective of its location (in a separate JVM)
- Facilitates hierarchy (Parent-Child relationship)
- Provides a regular life-cycle of an Actor

Akka: Our own little lab

- Download from:

  https://github.com/nsengupta/Akka-Training-Elem

Akka: Execution order

- We require an ActorSystem before we can employ Actors
- org.training.nirmalya.**sampleCodeOne**
- Difference between the behaviour of
  - org.training.nirmalya.sampleCodeOne.{ **HelloWorldWithDriver** | **HelloWorldWithSleepingDriver** }
- Order of output is not predictable

Akka: Responder needs to know the Requestor

- Conversation requires sending and receiving
- org.training.nirmalya.**sampleCodeTwo**
- Explain the output
- The question is: who does an Actor reply to?

Akka: Importance of conversation

- A protocol is necessary
  - Messages are immutable
- An Actor <u>needs</u> another Actor to talk to
- InBox provides us with a quickly raised, *temporary* Actor
- org.training.nirmalya.**sampleCodeThree**
- TODO: SmartPongActor should respond to Pong with a Ping!

Akka: Creation of Actors

- The mode of construction is important
  - **Props** is recommended (and mandatory for this course)
- org.training.nirmalya.**sampleCodeFour**
  - PongActor contains a prototype (run: Driver)
- Create SmartPingActor (responds to only Pong messages)
- Create SmartPongActor (responds to only Ping messages)
- Implement PingPongDriver
  - It should converse with Smart {Ping | Pong} Actors

Akka: Actors can carry other Actors

- An Actor can carry other actors
- We can construct Actors using other Actors
- org.training.nirmalya.**sampleCodeFive**
- Implement Actors *Kalia* and *Sambha*
  - They only understand Pong message
  - They identify themselves, while responding with a Ping message
- Update protocol if and as necessary
- Use Props while implementing *Kalia* and *Sambhas*

Akka: Actors can have children

- Ownership: Any Actor (Parent) can create another Actor (Child)
  - getContext().actorOf(..);
  - Remember: only ActorSystem is responsible for creating Actors
  - Every Actor knows its Parent (mandatorily exists) and Children (may not exist)
- org.training.nirmalya.**sampleCodeSix**
- Remember: *tell()* carries the 2nd parameter, as the sender's ActorRef
- Tip: *getSelf().path().name()* provides the name of an Actor

Akka: Testing Actors

- We will always need an ActorSystem
- We will need a special Actor to **tell** (and *hear from*) Actors that we implement
- Akka provides JavaTestKit
- Usual JUnit asserts available
- **ExpectMsg** series
- Time is an inseparable part of testing
  - Remember, we don’t know **if** and **when** an Actor responds

Akka: Testing Actors - Constraints

- tell() is **fire-and-forget**
  - There may be late or no reply
- An Actor may interact with other Actors underneath
- An Actor’s internal state is never publicly available
- Protocol must include all possible messages
- Messages are immutable

Akka: Let’s taste the water!

- org.training.nirmalya.testBed.sampleCodeThree
- org.training.nirmalya.testBed.sampleCodeFive
- Useful
  - Object[] expectMsgAllOf(Duration max, Object... msg)
  - Object expectMsgAnyOf(Duration max, Object... msg)
  - T expectMsgEquals(Duration max, T msg)
  - T expectMsgClass(Duration max, Class<T> c)

Akka: Aspects of testing Actors

- Remember: Timing
- Remember: Asynchronous
- Remember: Statelessness
- Remember: Collaboration
- JUnit (Scalatest)

Akka: Exercise

- A multi-lingual thesaurus actor provides synonyms of English words, in English and Hindi
- It employs two linguist actors to help it
  - One understands English and the other, Hindi
- There may be zero, one or more synonyms available for each English word supplied
- An user specifies which language is she interested in (may be both), and supplies an English word
- The linguist actors learn too
  - collects synonyms they haven’t known so far