AKKA AND MAINFRAMES: I WENT THERE

Ahmad Rana
Accenture - Associate Manager - Emerging Technology

Syed Moid
Accenture - Senior Manager – Technology Architecture
WHAT IS THIS PRESENTATION IS REALLY ABOUT

• Not about deploying Actor Systems on Mainframes... where-ever you can run JVM, you can run an Akka Actor system (z\Akka anyone? 😊)

• How can actor topology, Akka toolkit and reactive concepts be used to integrate / expose business functionalities
PATTERNS & USE CASES
SOCKET CONNECTION (STATE-LESS)

Dependencies:
• Socket Server on mainframe supports persistent socket connection command execution
• I.e. Actor does not have to keep alive socket connection
• User executes authorization command using any available actor

Actor onStart: Establish connection to socket server
Actor receives message. Execute command.

Actor onError: Depending on error, utilize Supervisor Strategy to either:
• Restart: Re-establish connection
• Stop: Disconnect

Red flag: Some socket server integrations allow for continuous connect / disconnect (i.e. non-persistent connection). However, care should be taken when establishing these types of connection, as actor system with mis-aligned supervisor strategy could overwhelm socket server
Application (Root Actor System)
- Functionality A System
- Functionality B System
- Functionality C System

Every Actor → One TCP/IP (TLS/SSL) Connection

Actor System
- Functionality A
- Functionality B
- Functionality C

Sub-set of Actors → TCP/IP (TLS/SSL)

Mainframe

Socket Server
SOCKET CONNECTION (STATEFUL)

Dependency / Rationale:
• Mainframe socket server requires socket to be specific to one user
• Socket Server on mainframe must support persistent socket connection command execution
• I.e. Actor does have to keep alive socket connection

Actor onStart: Establish connection to socket server. Establish user session with authentication.

One application interface must accept user system credentials to login into mainframe and allow for authorized results.

Actor receives message. Execute command.

Actor onError: Depending on error, utilize Supervisor Strategy to either:
• Restart: Re-establish connection with authentication details.
• Stop: Disconnect. Poison Pill.
• Escalate: non-fatal errors to caller.
WHAT ABOUT SHARDING?
Post successful login, some token or reference marker must be used by client to ensure future requests are sent to correct actor.
Approach #2: Cluster Sharding

Post successful login, some token or reference marker must be used by client to ensure future requests are sent to correct actor.

Application(s)

User 1 Login
Input: Credentials
Create User 1 Actor
Create User N Actor

Socket Conn Distributed Actor System

Mainframe

User 1 Action
Input: Command
* Actor Ref
Find User 1 Actor
Execute command
Find User N Actor
Execute command

Socket Server

New Actor → One TCP/IP (TLS/SSL) Conn

Socket Server

Re-use Actor → Re-use user’s TCP/IP (TLS/SSL) Conn
SCREEN-SCRAPING (EX. 3270 EMULATION)

Dependencies:
• TN3270 server component on mainframe
• Any client has to keep alive socket connection
• User executes authorization command first

Actor onStart: Establish connection.
Actor receives message. Execute command(s).
Actor onError: Depending on error, utilize Supervisor Strategy to either:
• Restart: Disconnect (if possible). Destroy
• Stop: Disconnect. Destroy

Screen-scraping derided, scoffed but holds major business value in many contexts
• Faster time to market for new solutions or exposure points
• Less costly than famous early 2000s’ adage: ‘rip and replace’
Actor System

- Functionality A
- Functionality B
- Functionality C

Master Socket Actor System

Socket Server

Mainframe

Sub-set of Actors → Each with one TN3270 Conn

Commands
YOU GOT “RPCED”
Functionality A
Functionality B
Functionality C

Actor System

Master Socket Actor System (gRPC or Akka Remoting)

CICS ONC RPC

Mainframe

Sub-set of Actors → Each with one RPC or direct TCP connection
Note: akka-gRPC not suitable in some setups if mainframe does not support IBM’s Linux on z/OS
STRANGLER APPROACH

Build on top of existing mainframe services and ‘strangle’ capability piece by piece until it is ported to replacement system(s)

Dependencies:

• Mainframe exposes commands, operations and data through accessible network interfaces
• Web Services, SSH execution of SPs (oh my), external invocation of COBOL / IMS / CICS TXs (lol whut)

Actor system(s) sit one layer above: orchestrate and execute logic, while at same time populating replacement system’s data-store(s)

• Sounds like event sourcing. Does it work? Sometimes.

Example:

• Akka Persistence + Akka Clustering ← IMS TX over Web Services ← Web Client
Functionality A Logic Actor

Functionality B

Functionality C

Actor System(s)

Akka Persistence Data Actors

Data

Mainframe

- SSH
- IMS TX
- WEB SERVICES
- ESB

ESB / Service GW
Functionality A Logic Actor

Functionality B

Functionality C

Actor System(s)

Akka Persistence Data Actors

Data

Mainframe

- SSH
- IMS TX
- WEB SERVICES
- ESB

ESB / Service GW
EVENT SOURCING
Persistence Actor (Executes decoupled command)

Operator Actor (Receives Command)

- Populate data set using existing business logic from mainframe as divided into actors
- As each actor does its duty (CRUD ops, enrichment, persistence), create event journal for replay and/or auditing purposes
- If necessary, can send orchestrate in async manner with mainframe SoR using available SOA abstractions or previously defined approaches