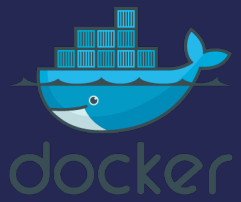
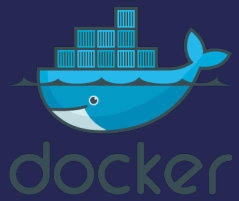



ACTOR CLUSTERING WITH DOCKER CONTAINERS AND AKKA.NET IN F#

"I have no special talents. I am only passionately curious."

- Albert Einstein

Riccardo Terrell

HOT ACTOR CLUSTERING WITH DOCKER CONTAINERS AND AKKA.NET IN F#

"I have no special talents. I am only passionately curious."

- Albert Einstein

Riccardo Terrell

Agenda

- Actor Model - quick intro
- Actor (Akka) Remoting & Clustering for distributed System
- F# Code-Quotations
- Remote Deploy Arbitrary Actor and dependencies… *HOT!*
- Docker integration

Goals – Presentation Roadmap

Building Distributed Systems cross-platform with ***hot*** loads *(behaviors and dependencies)*

One solution does not exist… combining different tools and programming models… *bend technonlogies to your need*

Introduction - Riccardo Terrell

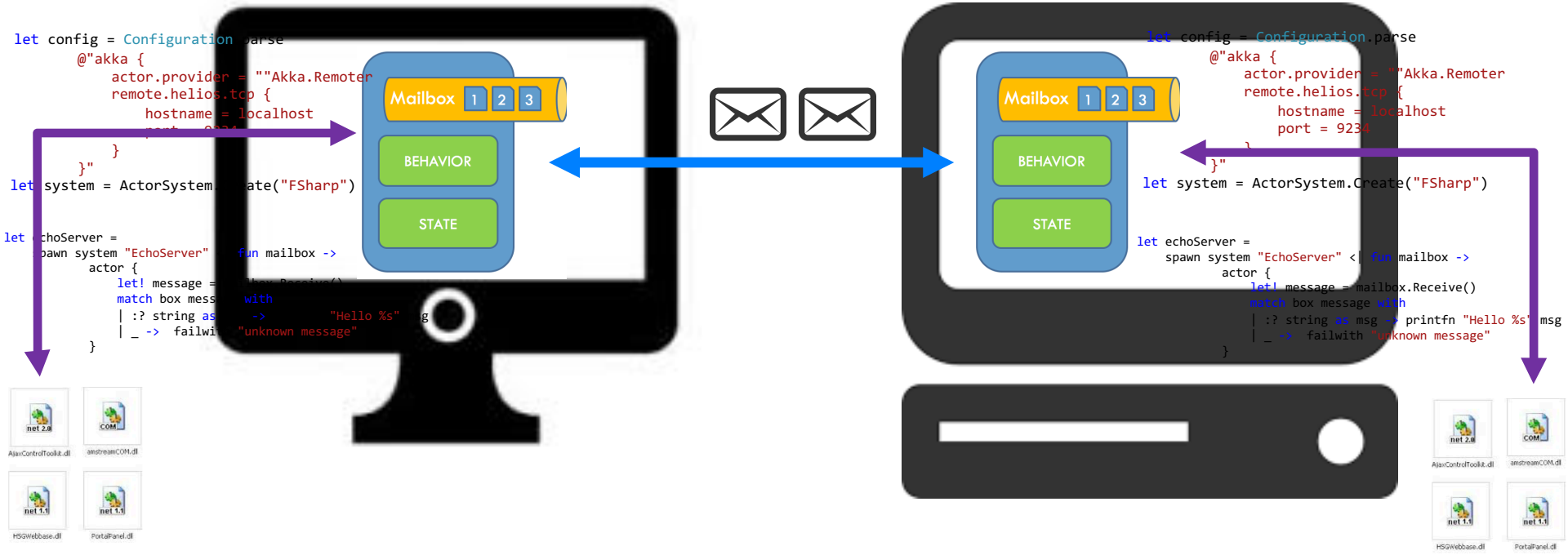
- Originally from Italy, currently - Living/working in Washington DC ~10 years
- +/- 19 years in professional programming
  - C++/VB → Java → .Net C# → Scala → Haskell → C# & F# → ??
- DC F# User Group - Organizer
- Working @ statmuse
- *Polyglot programmer - believes in the art of finding the right tool for the job*

@trikace *rickyterrell.com* *tericcardo@gmail.com*

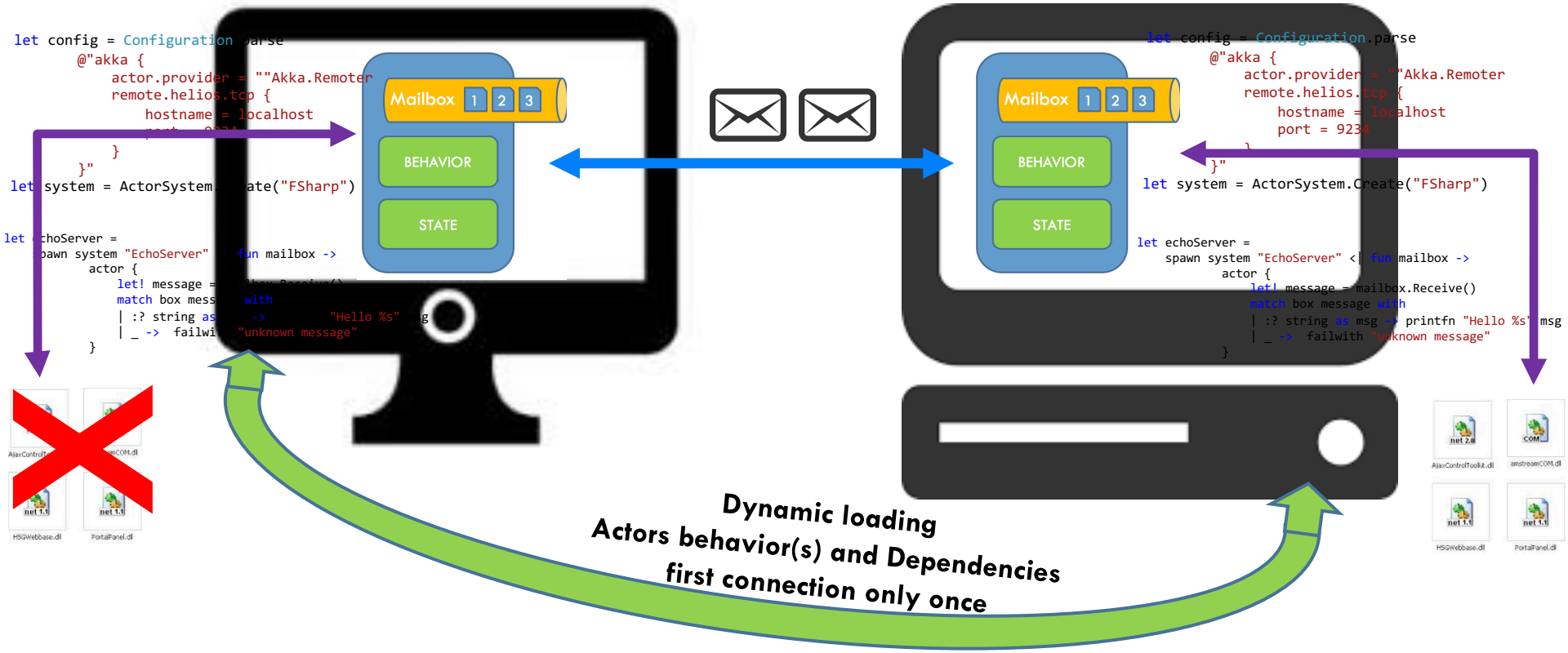
MVP Microsoft® Most Valuable Professional


What about you?

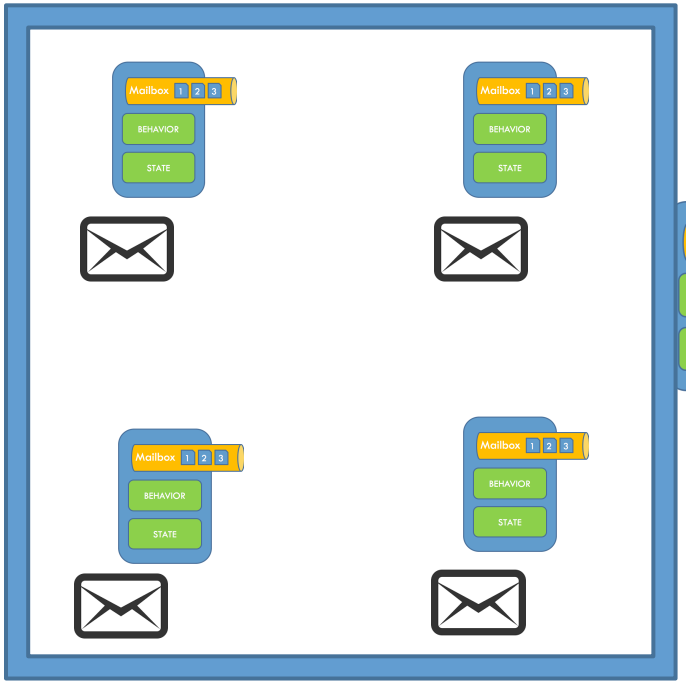
Scenario 1 – Remoting

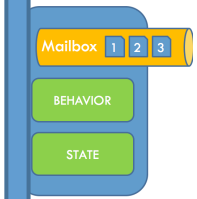


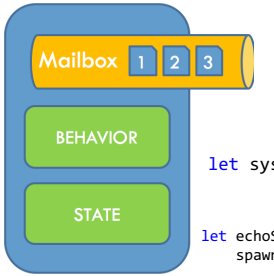
Scenario 1 – Remoting



Scenario 2 - Clustering








```
let config = Configuration.parse
        @"akka {
            actor.provider = ""Akka.Remoter
            remote.helios.tcp {
                hostname = localhost
                port = 9234
            }
        }"
let system = ActorSystem.Create("FSharp")
```

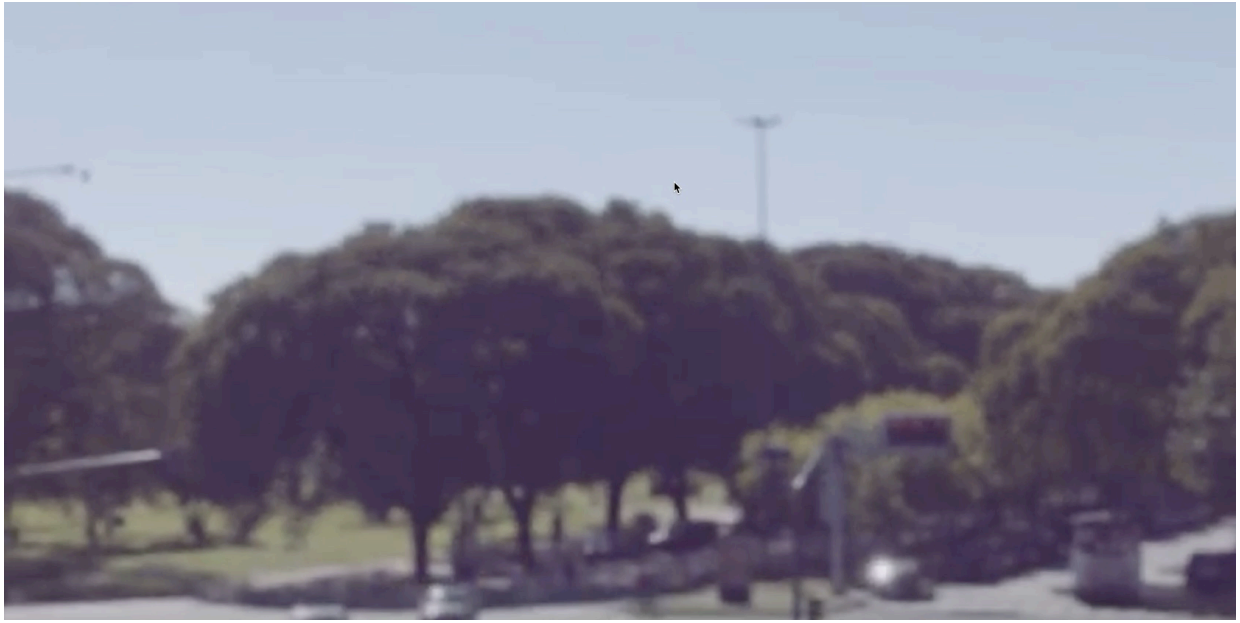
```
let echoServer =
    spawn system "EchoServer" <| fun mailbox ->
            actor {
                let! message = mailbox.Receive()
                match box message with
                | :? string as msg -> printfn "Hello %s" msg
                | _ ->  failwith "unknown message"
            }
```

Actor Model

What & Why

The need parallelism

Multithreading in practice



The issue is Shared of Memory

- Shared Memory Concurrency
- Data Race / Race Condition
- Works in sequential single threaded environment
- Not fun in a multi-threaded environment
- Not fun trying to parallelize
- Locking, blocking, call-back hell

What is an Actor?

- **Share Nothing**
- **Message are passed by value**
- Light weight processes/threads communicating through messaging
- Communication only by messages
- Lightweight object
- Processing
- Storage – State
- Running on it’s own thread.
- Messages are buffered in a “mailbox”

The Solution is Immutability and Isolation

Reactive Manifesto & Actor Model

Responsive	Event Driven
Message-Driven	Communication by messages
Resilient	Fault tolerant by Supervision
Elastic	Clustering and Remoting across multiple machines

Akka.NET

- Akka.Remote
- Akka.Cluster
- Akka.Persistence
- Akka.Streams
- Supervision
- Routing

** Concurrent*

** Resilient*

** Distributed*

** Scalable*

Akka.NET

- Akka.Remote
- Akka.Cluster
- Akka.Persistence
- Akka.Streams
- Supervision
- Routing

** Concurrent*

** Resilient*

** Distributed*

** Scalable*

Actor – Akka.NET in C#


```
var system = ActorSystem.Create("fizz-buzz");

public class FizzBuzzActor : ReceiveActor {
    public FizzBuzzActor()     {
        Receive<FizzBuzzMessage>(msg => {
            // code to handle the message
        });
    }
}

var actor = system.ActorOf(Props.Create<FizzBuzzActor>(), "fb-actor");
actor.Tell(new FizzBuzzMessage(5));
```

Actor – Akka.NET in F#


```
let system = ActorSystem.Create("FSharp")

type EchoServer =
    inherit Actor

    override x.OnReceive (message:obj) =
        match message with
        | :? string as msg -> printfn "Hello %s" msg
        | _ ->  printfn "What should I do with this thing?”

let echoServer = system.ActorOf(Props(typedefof<EchoServer>))
echoServer.Tell 42
```

Actor – Akka.NET in better F#


```
let system = System.create "System" <| Configuration.load()

let greeter =
   // ActorFactory -> Name -> f(Actor<Message> -> Cont<'Message, 'return>) -> ActorRef
     spawn system "Greeter-Functional"
     <| fun mailbox ->
         let rec loop() = actor {
             let! msg = mailbox.Receive()
             match msg with
             | Greet(w) ->printfn "Hello %s" w
             return! loop() }
         loop()

greeter <! GreetMsg.Greet("AKKA.Net!!")
```

Remoting

Remotely Deploying Actors

akka.net

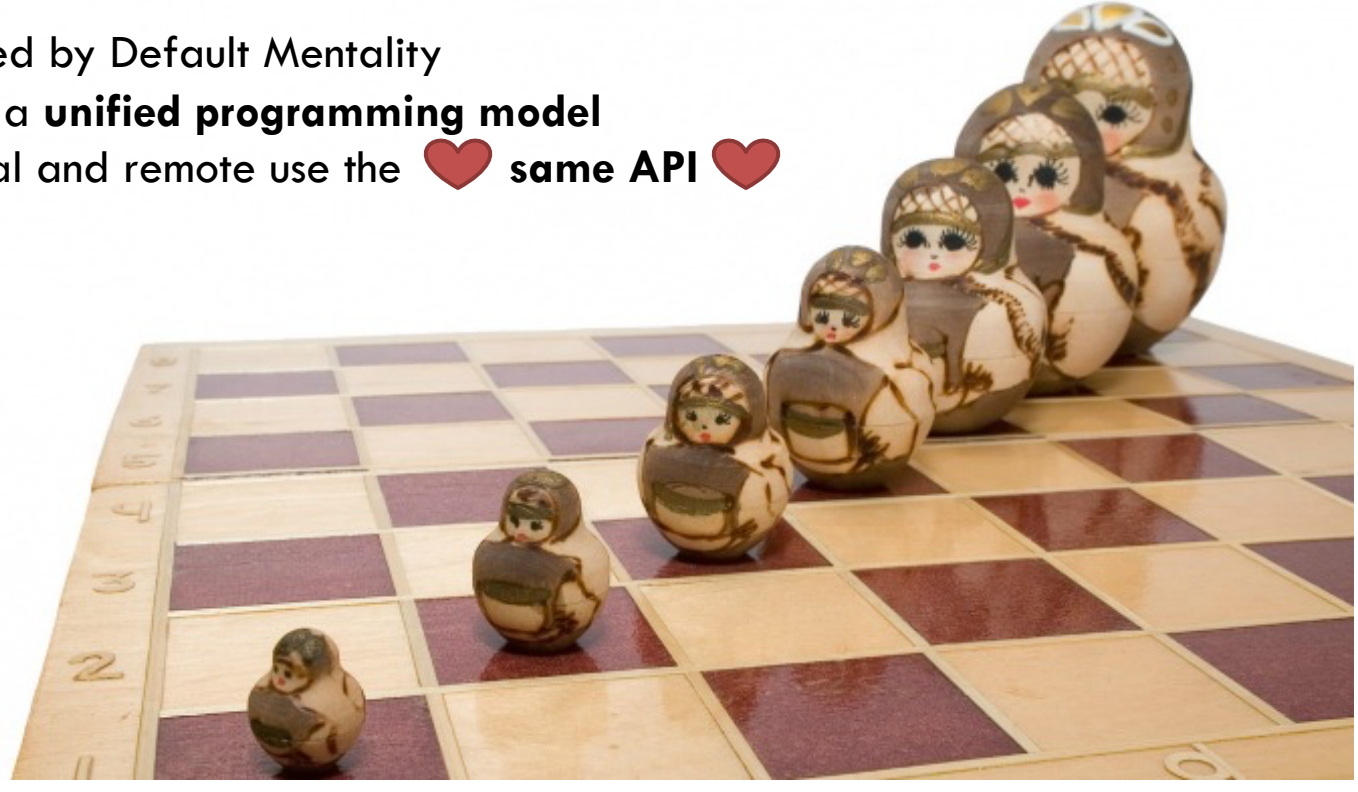
Deploying an actor means two things simultaneously:

- Creating an actor instance with specific, explicitly configured properties
- Getting an ActorRef to that actor

```
using (var system = ActorSystem.Create("Deployer", ConfigurationFactory.ParseString(@" // CLIENT
          akka {  ... CONFIG AS BEFORE ... }
              remote {
                  helios.tcp {
                      port = 0
                      hostname = localhost
                  }
              }
          }")))
      {
        var remoteEcho1 = system.ActorOf(Props.Create(() => new EchoActor()), "remoteecho");

        var echoActor = system.ActorOf(Props.Create(() => new HelloActor(remoteEcho1)));
        echoActor.Tell("Hello")
```

Going Scalable – Akka Remoting

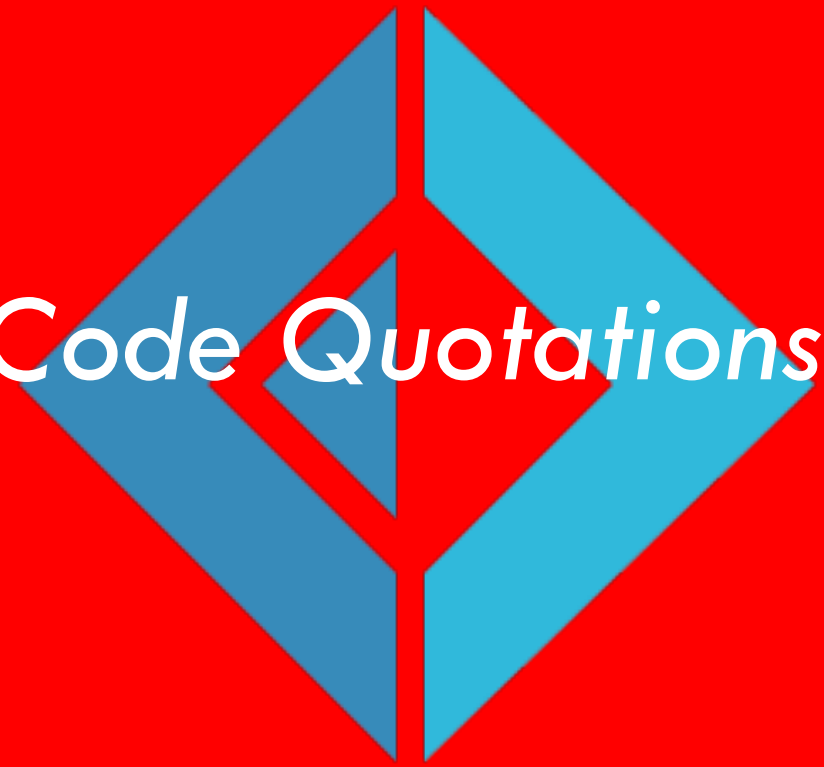
- Distributed by Default Mentality
- Provides a **unified programming model**
  - Local and remote use the ❤ **same API** ❤



Location Transparency

Protocol | Actor System | Address | Path

Akka.tcp://System@10.55.1.25:9234/actorName



<@ Code Quotations @>

<@ F# Code Quotations @>

```
// A typed code quotation.
let sum5 : Expr<int> = <@ 2 + 3 @>
```

```
val sum5 : Quotations.Expr<int> =
  Call (None, op_Addition, [Value (2), Value (3)])
```

<@ F# Code Quotations @>

```
let rec print expr =
    match expr with
    | Call(exprOpt, methodInfo, exprList) ->
        match exprOpt with
        | Some expr -> print expr
        | None -> printf "%s" methodInfo.DeclaringType.Name
        for expr in exprList.Tail do  print expr
    | Int32(n) ->printf "%d" n
    | Value(value, typ) -> printf ”%d” value
```

<@ F# Code Quotations @>

```
let fsum42 = fun x -> x + 42
```

<@ F# Code Quotations @>

```
let fsum42 = fun x -> x + 42
let serializer:BinarySerializer = FsPickler.CreateBinarySerializer()

let sum42:byte[] = serializer.Pickle<@  fsum42  @>
```

<@ F# Code Quotations @>

```
let fsum42 = fun x -> x + 42
let serializer:BinarySerializer = FsPickler.CreateBinarySerializer()

let sum42:byte[] = serializer.Pickle<@  fsum42  @>
let sum42 = serializer.UnPickle<Quotations.Expr<int -> int>> sum42
```

<@ F# Code Quotations @>

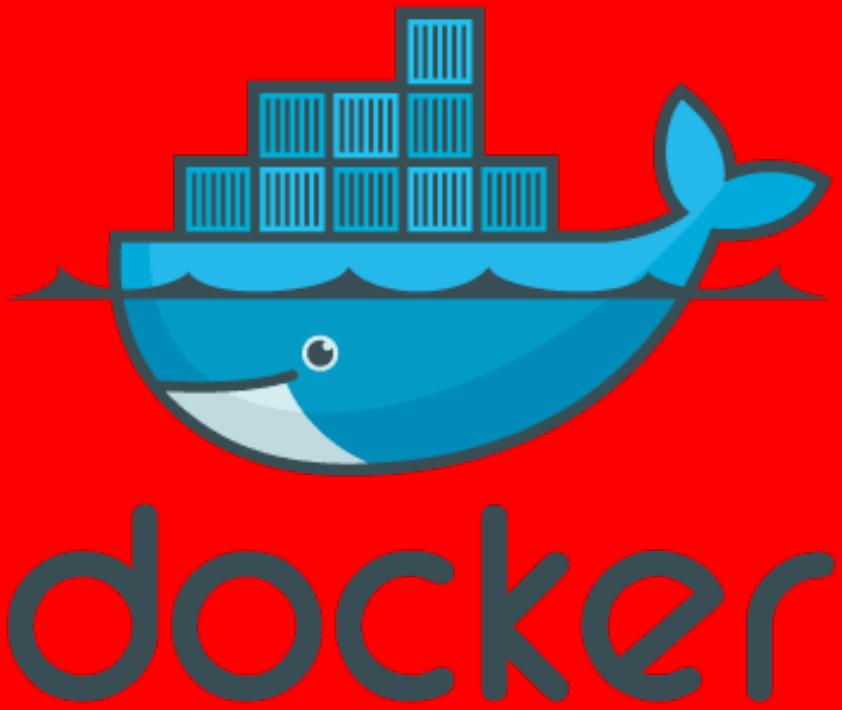
```
let fsum42 = fun x -> x + 42
let serializer:BinarySerializer = FsPickler.CreateBinarySerializer()

let sum42:byte[] = serializer.Pickle<@  fsum42  @>
let sum42 = serializer.UnPickle<Quotations.Expr<int -> int>> sum42

let sumfun = QuotationEvaluator.Evaluate sum42

printfn "%d" (sumfun  4) // print 46
```

<@ F# Code Quotations @>

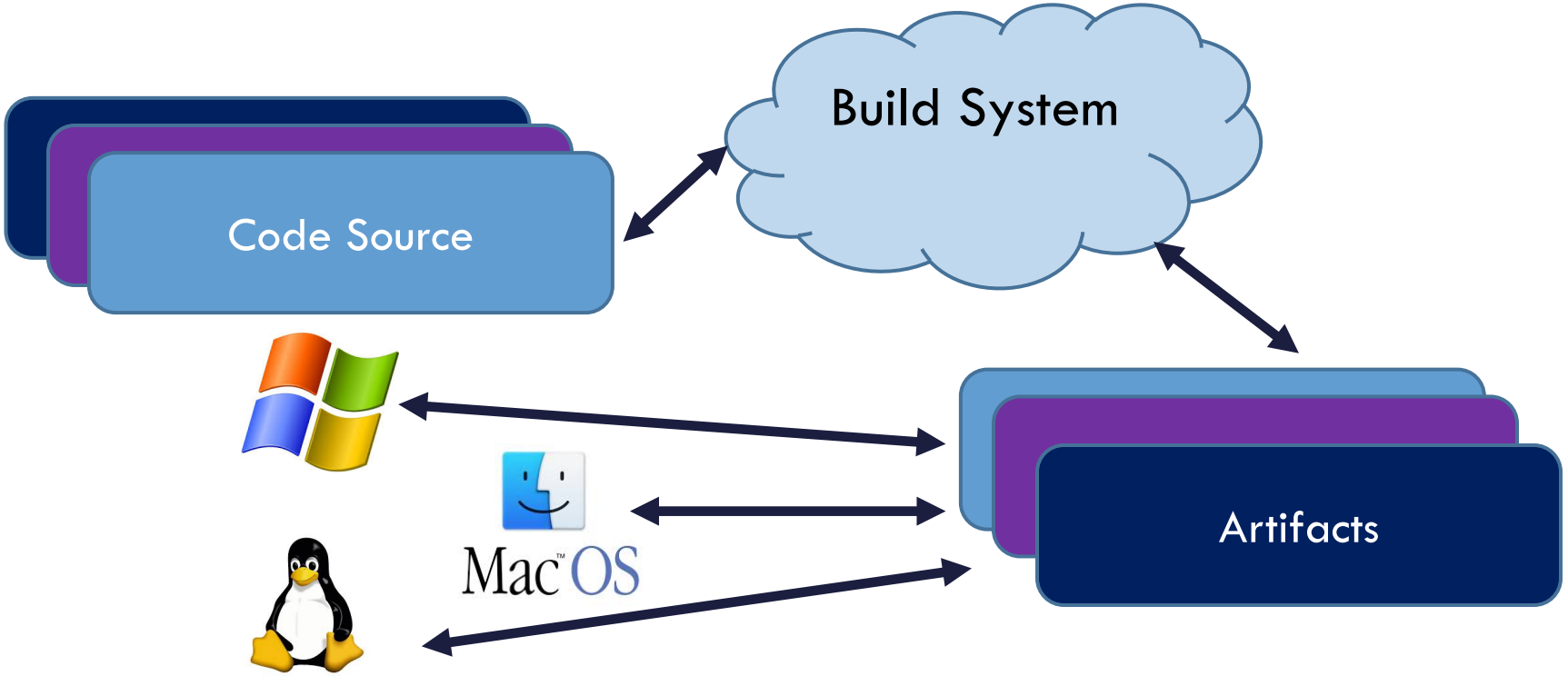
```
let remoter =
        spawne system "remote"
                fun mailbox ->
                let rec loop(): Cont<int * string, unit> =
                    actor {
                        let! msg = mailbox.Receive()
                        match msg with
                        | (REQ, m) ->
                            printfn "Remote actor received: %A" m
                            mailbox.Sender() <! (RES, "ECHO " + m)
                        | _ -> logErrorf "Received unexpected message: %A" msg
                        return! loop()
                    }
                loop()
                [ SpawnOption.Deploy(remoteDeploy remoteSystemAddress) ]
```

<@ F# Code Quotations @>

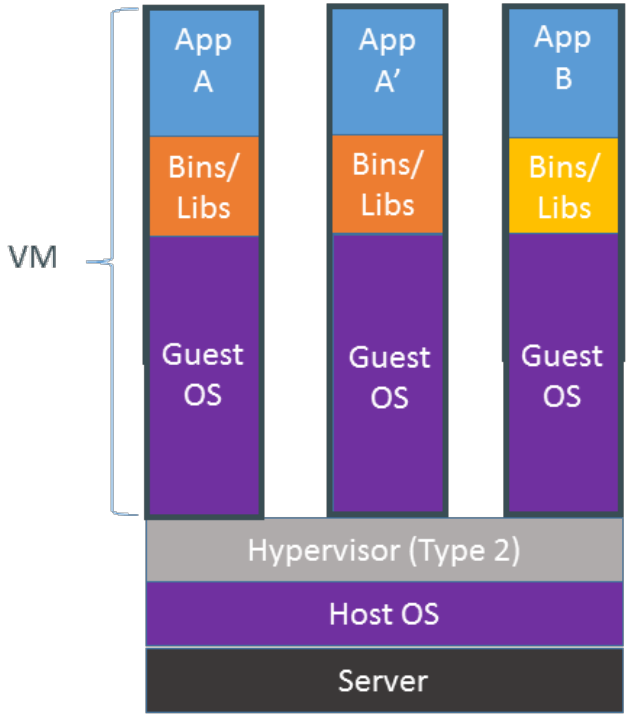
```
let remoter =
        spawne system "remote"
            <@  fun mailbox ->
                let rec loop(): Cont<int * string, unit> =
                    actor {
                        let! msg = mailbox.Receive()
                        match msg with
                        | (REQ, m) ->
                            printfn "Remote actor received: %A" m
                            mailbox.Sender() <! (RES, "ECHO " + m)
                        | _ -> logErrorf "Received unexpected message: %A" msg
                        return! loop()
                    }
                loop()
             @> [ SpawnOption.Deploy(remoteDeploy remoteSystemAddress) ]
```


docker

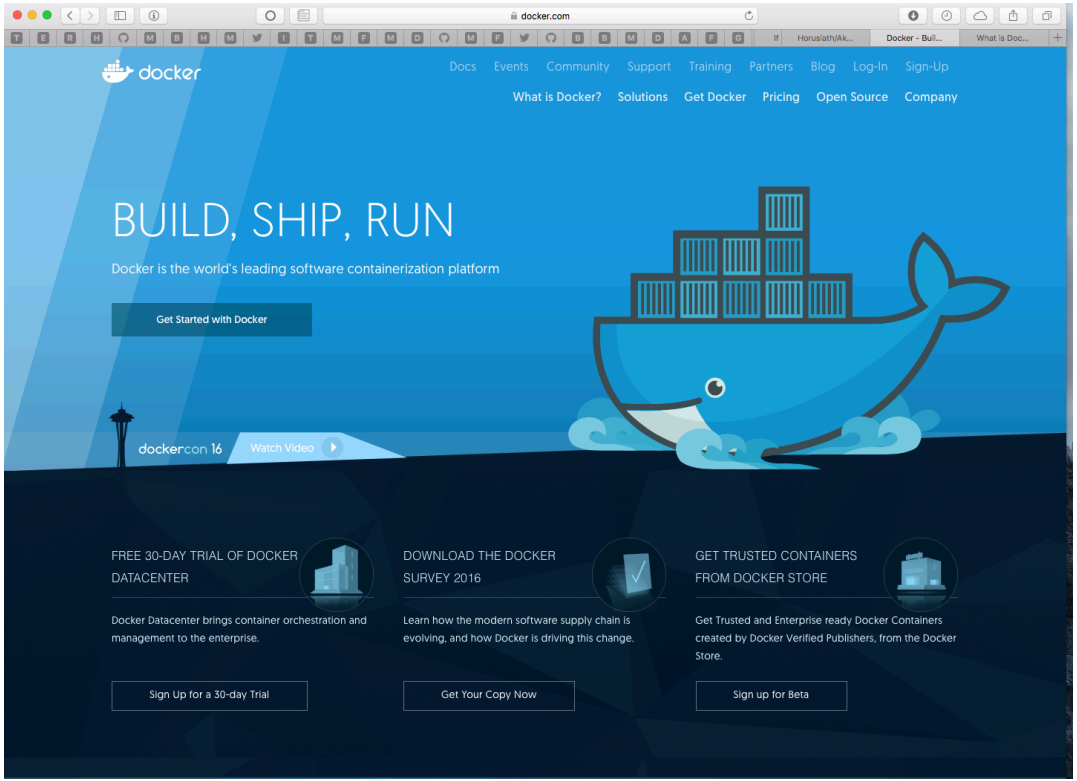
Build Processes without Docker



Why Docker ?


App A
Bins/ Libs
Guest OS
App A'
Bins/ Libs
Guest OS
App B
Bins/ Libs
Guest OS
VM
Hypervisor (Type 2)
Host OS
Server

How to start with



TOOL BOX

Docker Client

Docker Machine

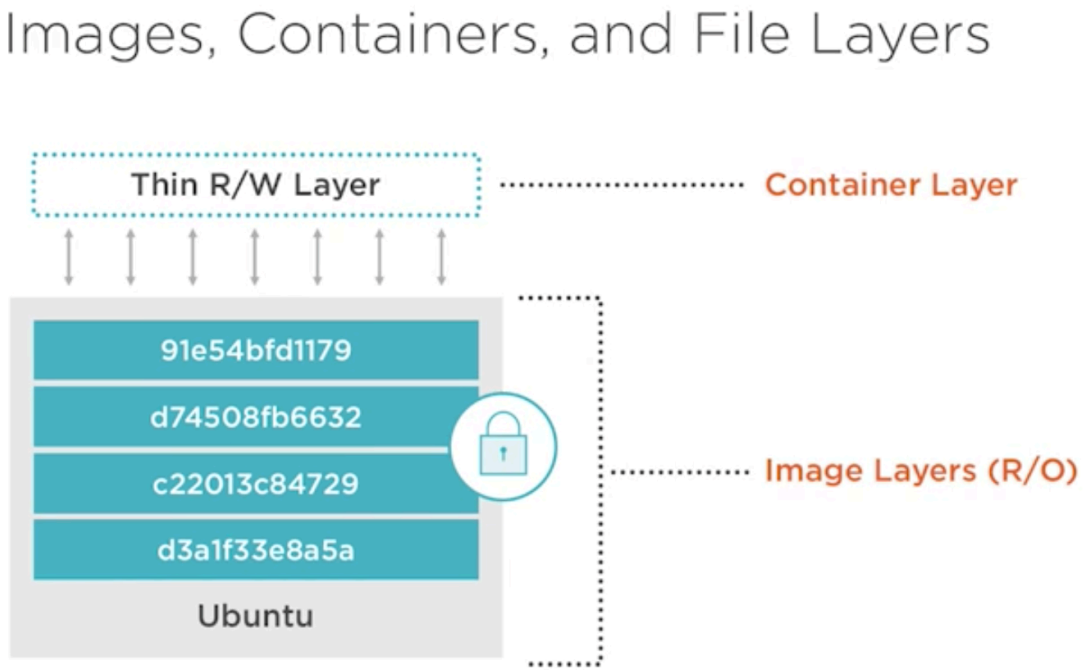
Docker Kitmaatic

Docker Compose

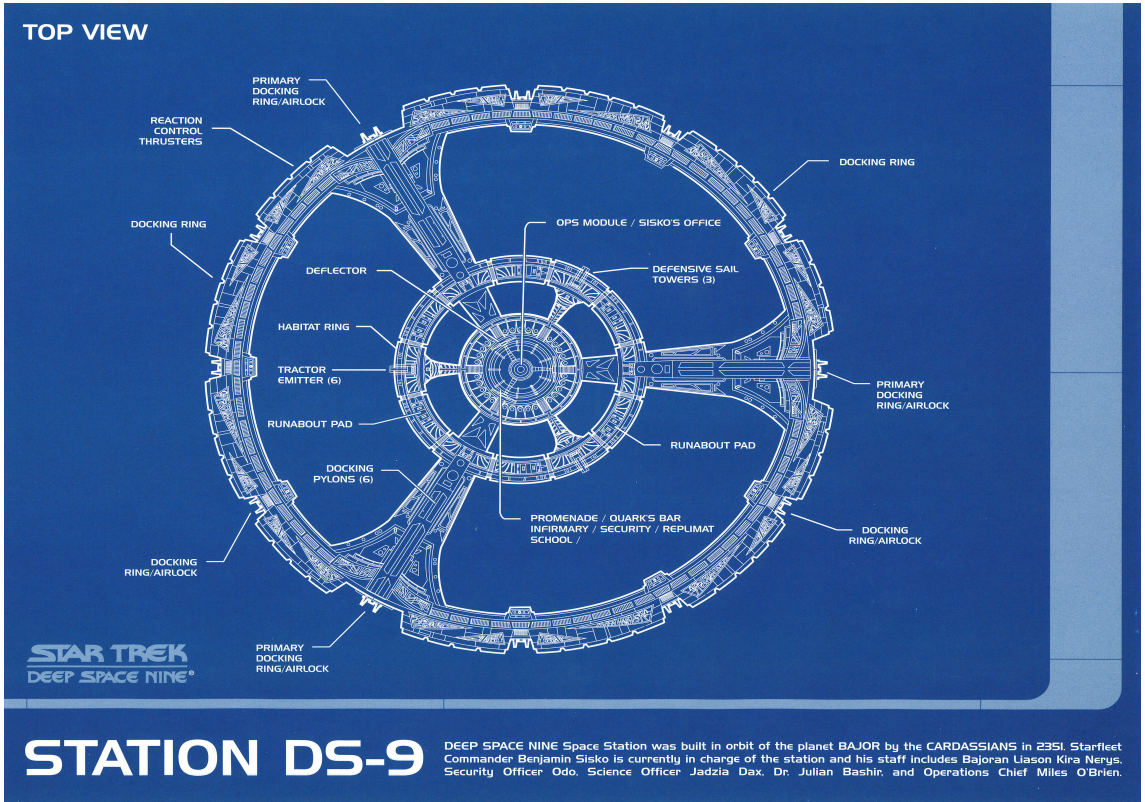
Docker Tools

VirtualBox

Docker Images

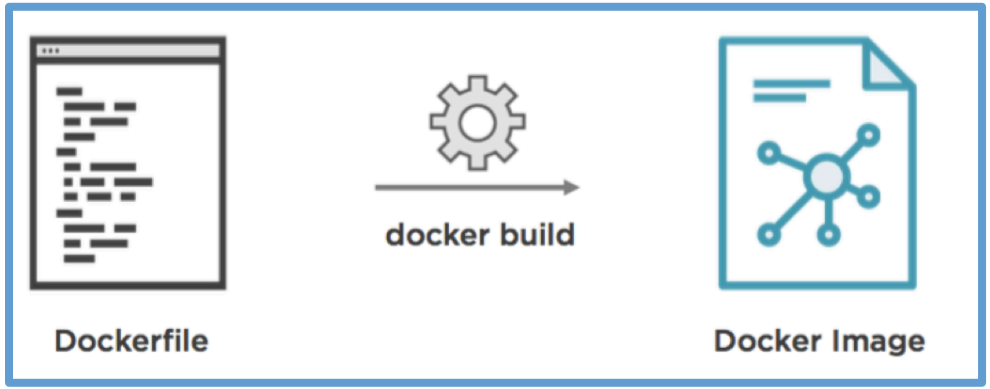


Docker Images



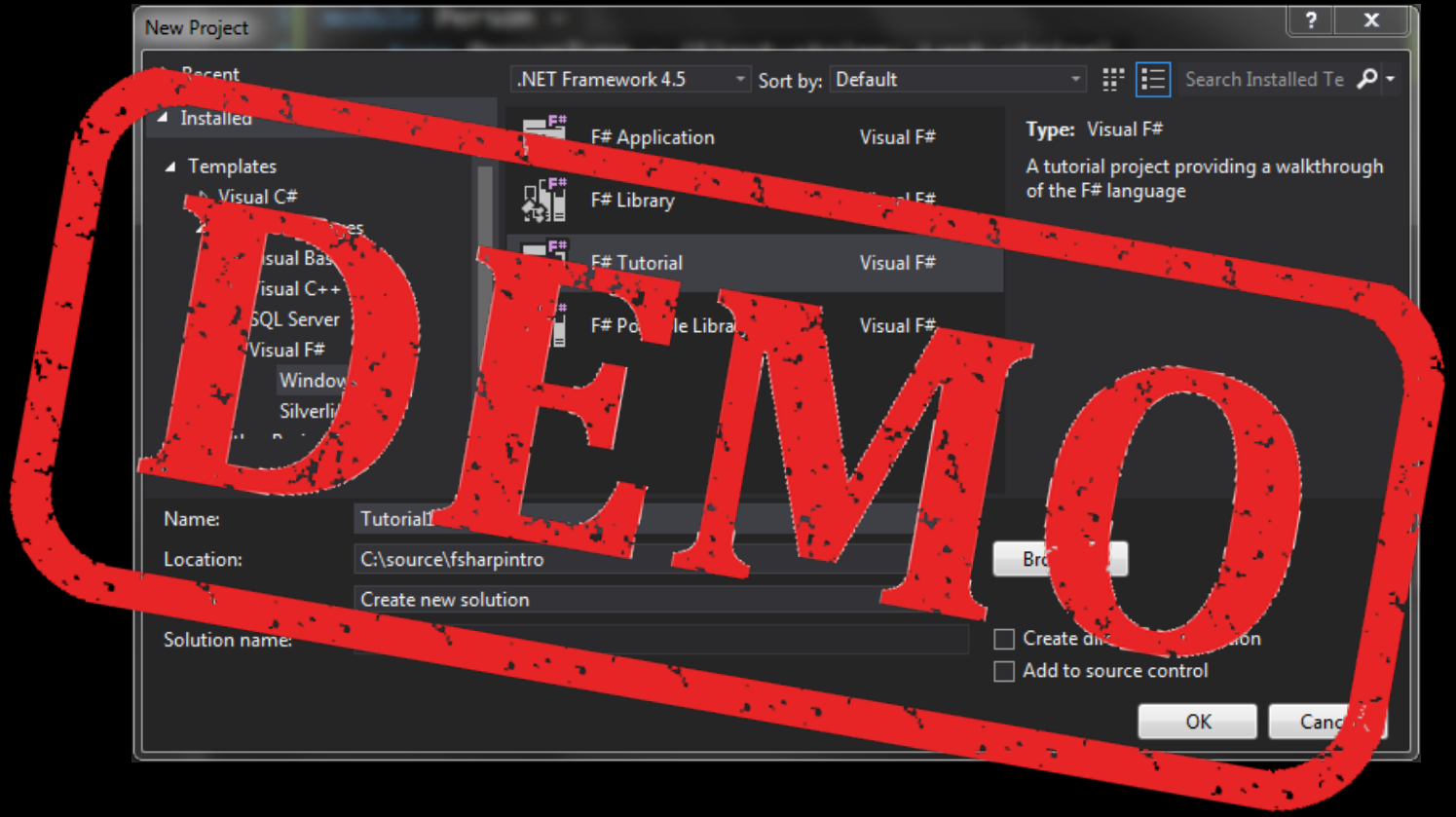
Dockerfile

```
FROM        mono:latest

MAINTAINER  Riccardo Terrell

ENV         CODEDIR /var/code
RUN         mkdir -p $CODEDIR

WORKDIR     $CODEDIR

RUN         chmod +x $CODEDIR
            && xbuild ./ActorRemote.sln /property:Configuration=Release

EXPOSE   9234

ENTRYPOINT          ["mono", "./AkkaExamples/bin/Release/AkkaExamples.exe"]
```



So Far…

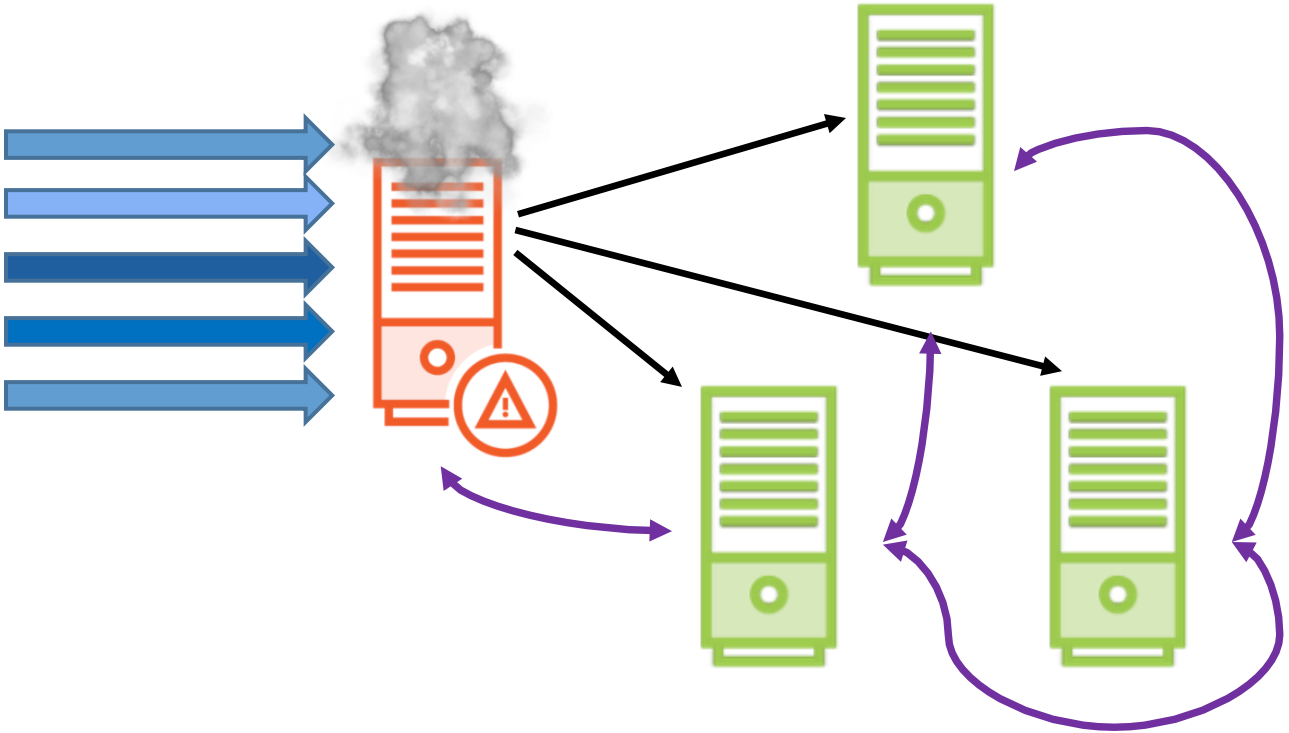
- What & Why Actor Model
- Akka.Net
- Remoting
- Code Quotations
- Docker


New Project
.NET Framework 4.5
Sort by:
Default
Search Installed Te
Installed
Templates
Visual C#
Visual C++
SQL Server
Visual F#
F# Application
Visual F#
F# Library
F# Tutorial
Visual F#
Type: Visual F#
A tutorial project providing a walkthrough of the F# language
Name:
Location:
C:\source\fsharpintro
Create new solution
Solution name:
Add to source control
OK
DEMO

Clustering

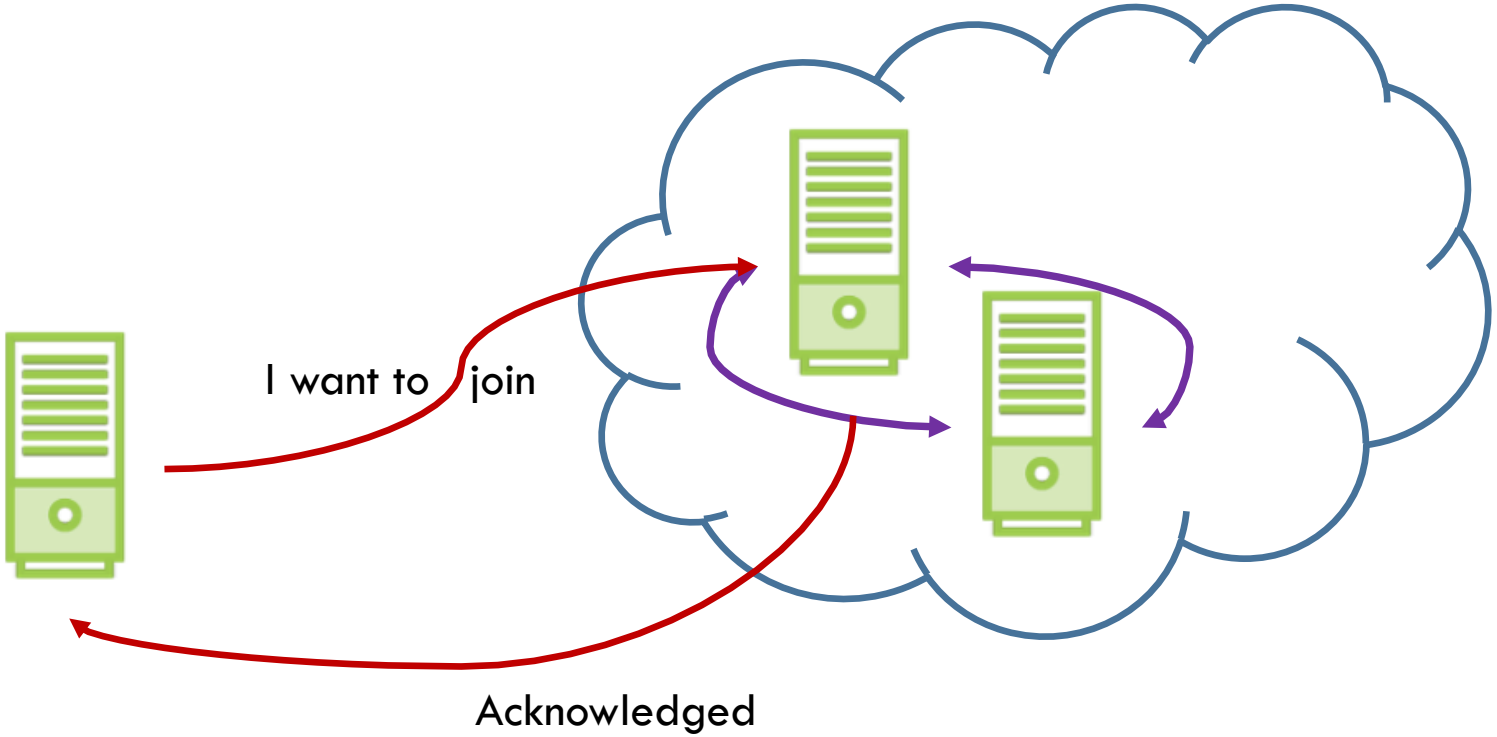
- Load balancing
- Fault-tolerant
- Scalability

Actor Clustering

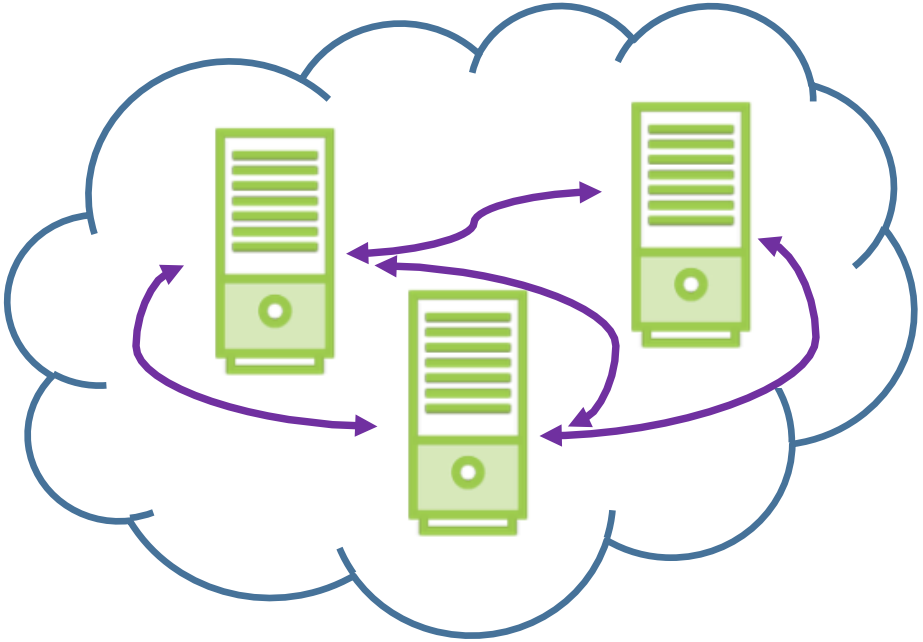


Gossip

Joining the Actor Cluster

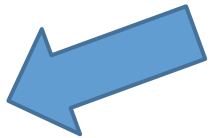


Joining the Actor Cluster



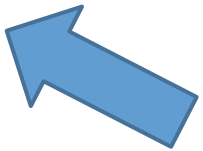
Clustering – Just a configuration matter

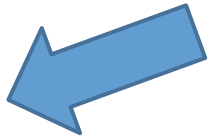

```
let workerSystem = System.create "cluster" <| Configuration.parse """
akka {
    actor {
        provider = "Akka.Cluster.ClusterActorRefProvider, Akka.Cluster"
        }
    }
    remote {
        helios.tcp {
            hostname = "localhost"
            port = 0
        }
    }
    cluster {
        seed-nodes = [ "akka.tcp://cluster@localhost:8091/" ]
    }
}
"""
```



Clustering – Just a configuration matter


```
let masterSystem = System.create "cluster" <| Configuration.parse """
akka {
    actor {
        provider = "Akka.Cluster.ClusterActorRefProvider, Akka.Cluster"
        }
    }
    remote {
        helios.tcp {
            hostname = "localhost"
            port = 8091
        }
    }
    cluster {
        seed-nodes = [ "akka.tcp://cluster@localhost:8091/" ]
    }
}
"""
```

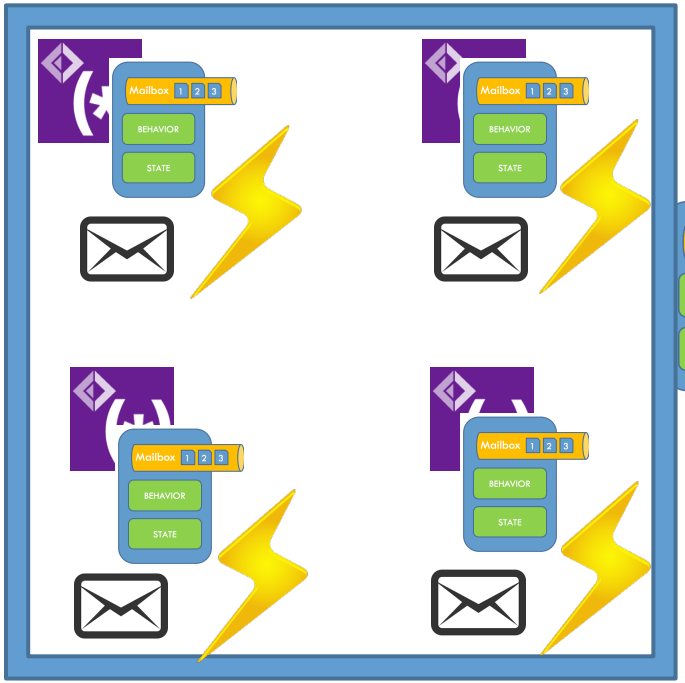


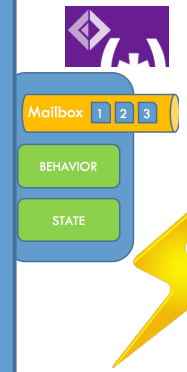


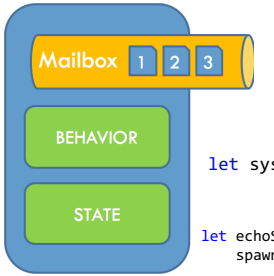
Tabasco

Dynamically load modules
types and dependencies
between different actor systems

Scenario 2 - Clustering



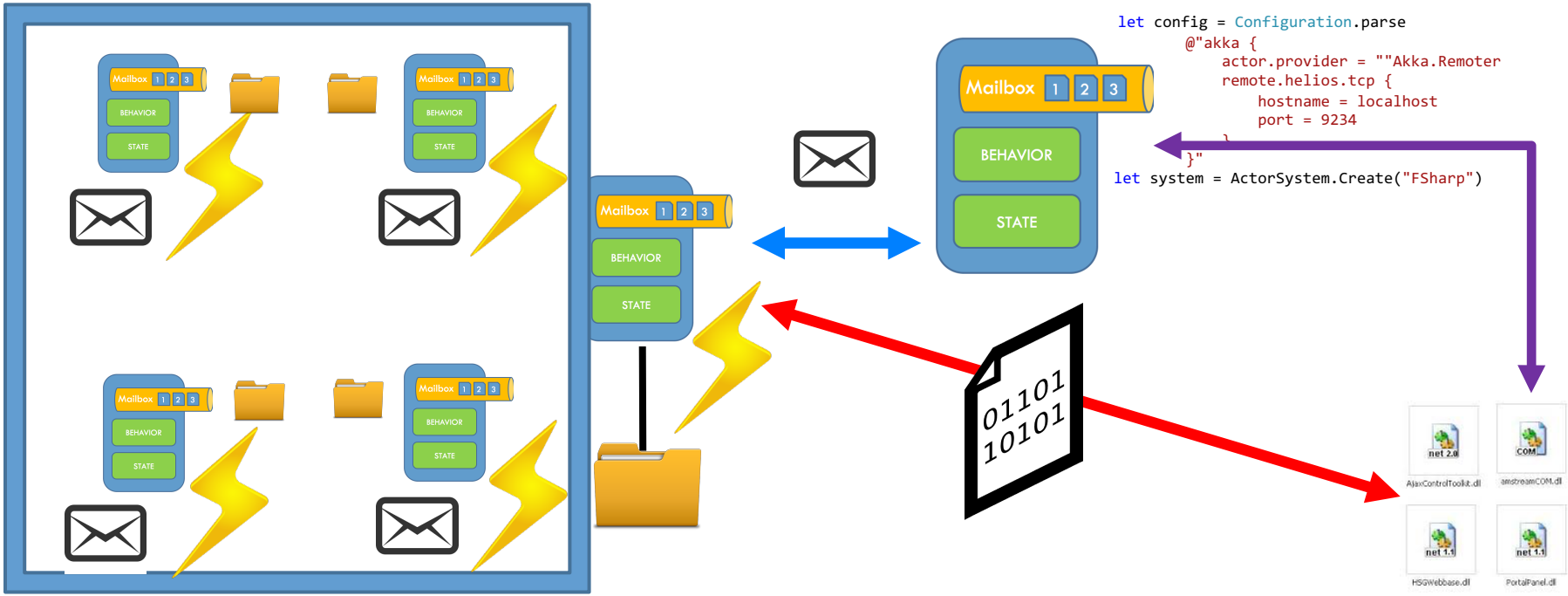




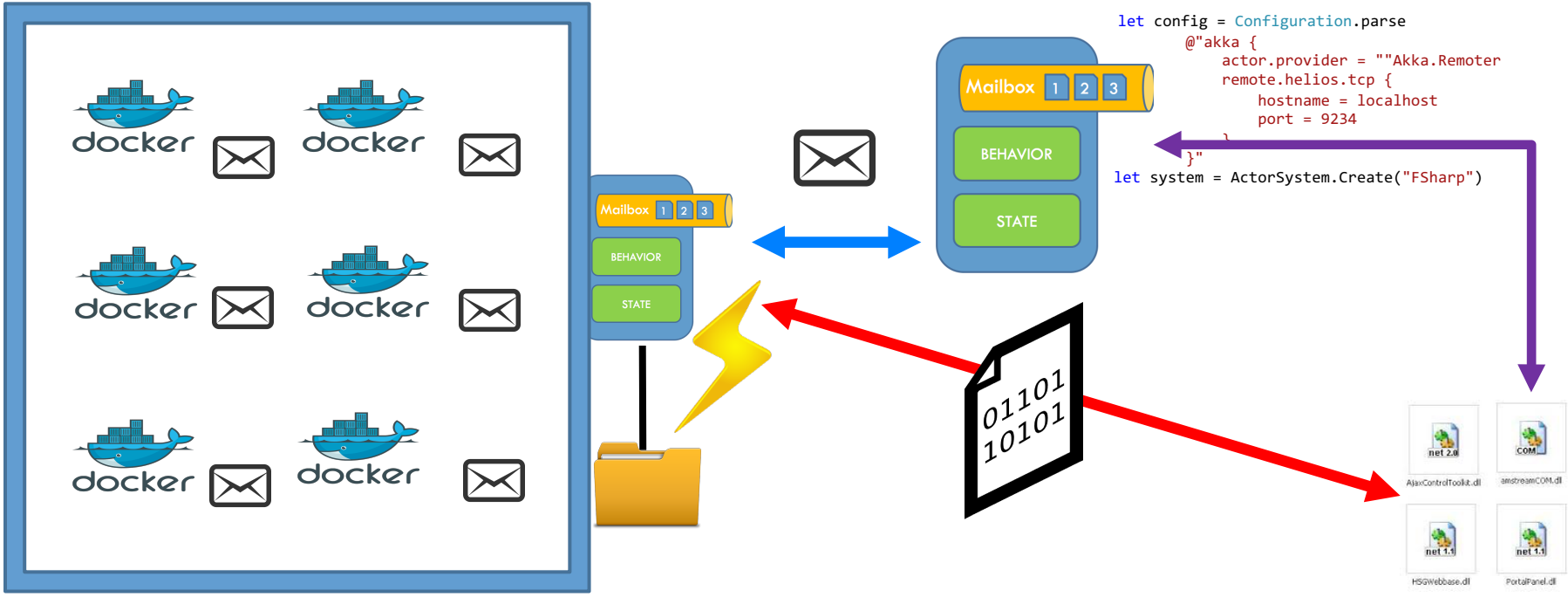

```
let config = Configuration.parse
        @"akka {
            actor.provider = ""Akka.Remoter
            remote.helios.tcp {
                hostname = localhost
                port = 9234
            }
        }"
let system = ActorSystem.Create("FSharp")
```

```
let echoServer =
    spawn system "EchoServer" <| fun mailbox ->
            actor {
                let! message = mailbox.Receive()
                match box message with
                | :? string as msg -> printfn "Hello %s" msg
                | _ ->  failwith "unknown message"
            }
```

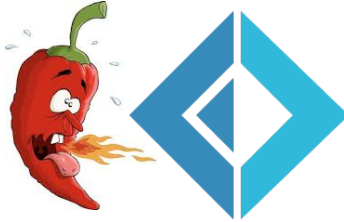
Scenario 2 - Clustering



Scenario 2 - Clustering

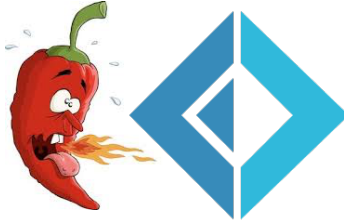


Tabasco - Dynamic Actor

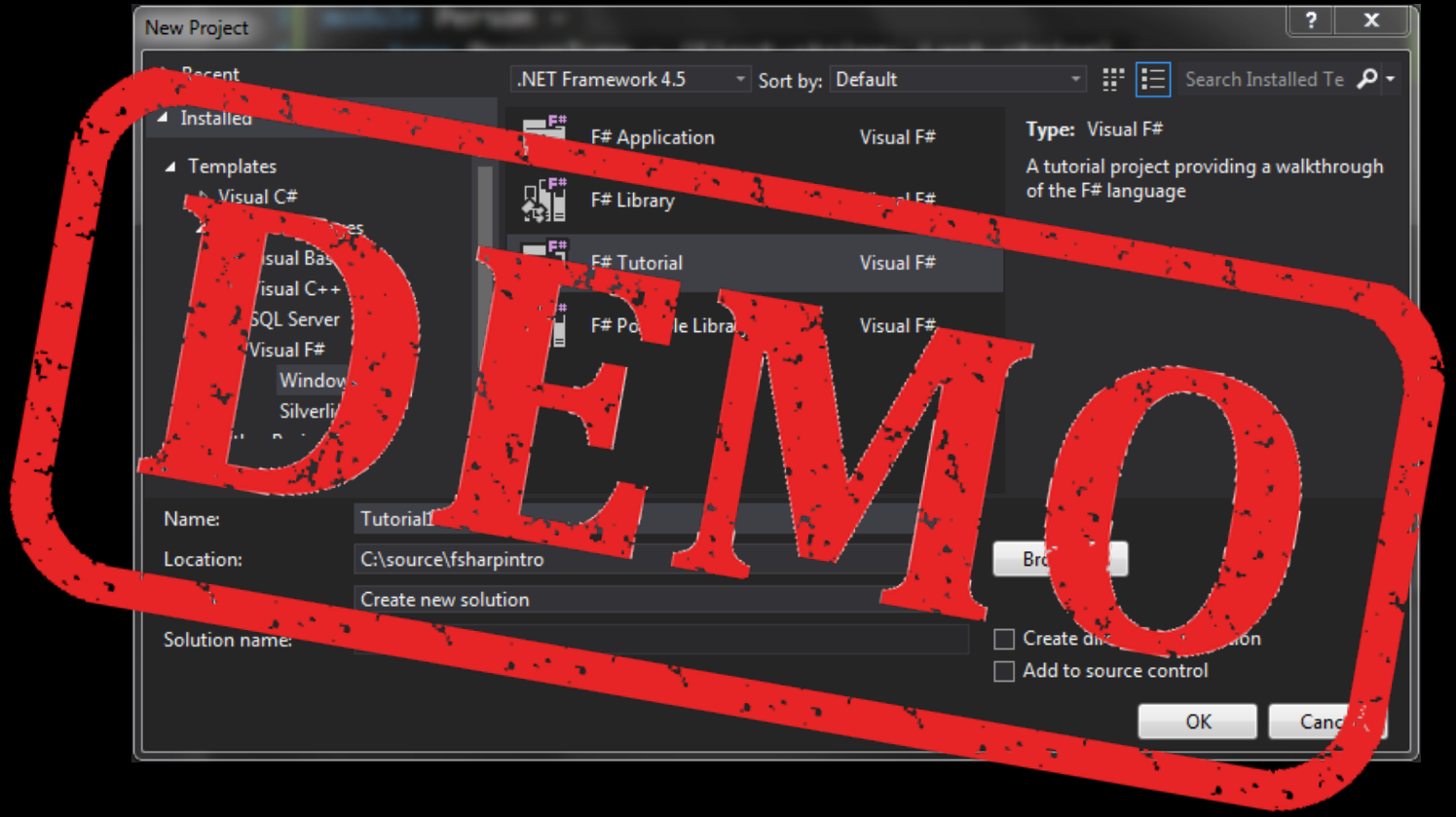


```
type DynamicActor(receive: Receive, zero: obj) as this =
    inherit UntypedActor()

    let untypedContext = UntypedActor.Context :> IActorContext
    let ctx = { Context = untypedContext}
    let mutable currentState: obj = zero
    let mutable currentReceive: Receive = receive

    override this.OnReceive(msg:obj) =
        match msg with
        | :? ActorMessage as message ->
            match message with
            | GetState -> untypedContext.Sender.Tell currentState
            | Stop -> untypedContext.Stop(untypedContext.Self)
            | Load(newReceive) -> currentReceive <- newReceive
        | message -> currentState <- currentReceive ctx
                                     currentState message
```

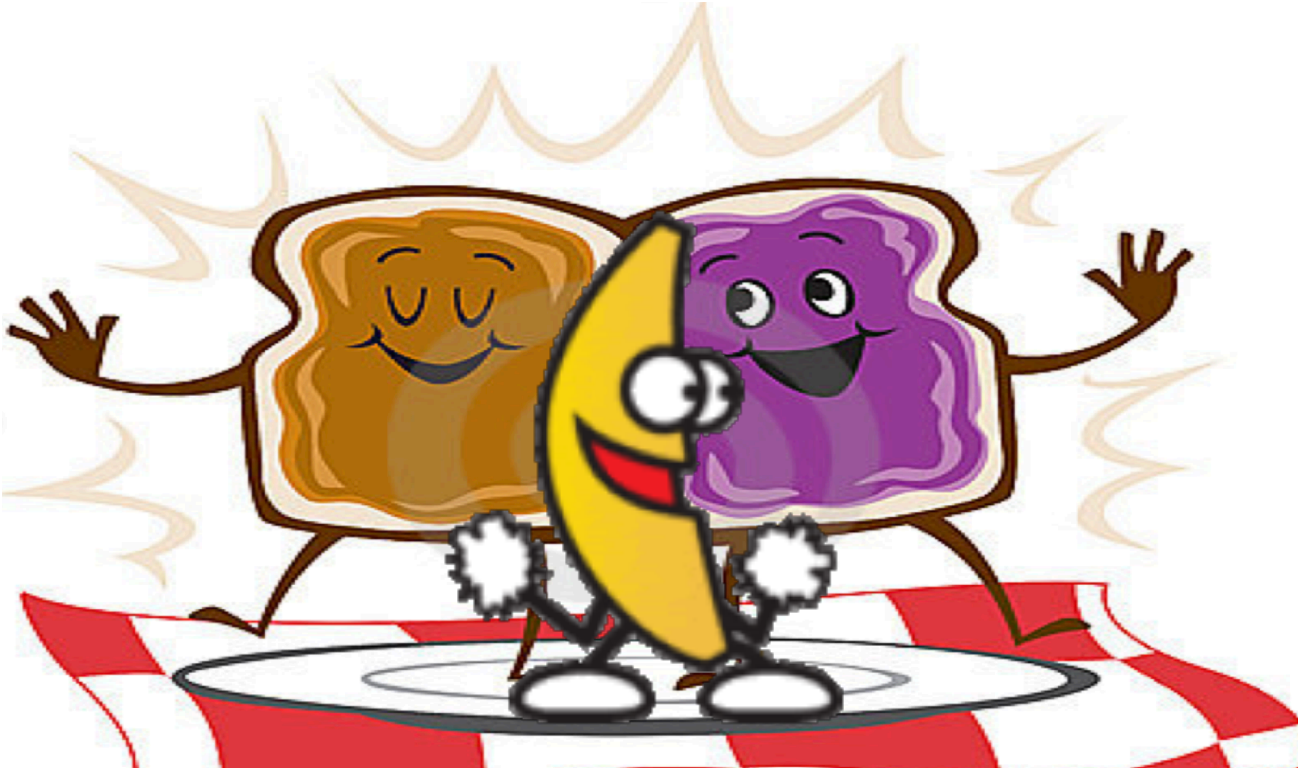
Tabasco – Assemblies Resolver



```
type internal AssemblyReceiver (vm: VagabondManager, serverRef: IActorRef) =
    member __.ServerRef = serverRef
    interface IRemoteAssemblyReceiver with

        member __.GetLoadedAssemblyInfo (ids:AssemblyId[]) = async {
            let! reply = serverRef <? GetAssemblyInfo ids
            return downcast reply
        }
        member __.PushAssemblies vas = async {
            let! reply = serverRef <? LoadAssemblies (vm.CreateRawAssemblies vas)
            return downcast reply
        }
```


New Project
.NET Framework 4.5
Sort by:
Default
Search Installed Te
Installed
Templates
Visual C#
Visual C++
SQL Server
Visual F#
F# Application
Visual F#
F# Library
F# Tutorial
Visual F#
Type: Visual F#
A tutorial project providing a walkthrough of the F# language
Name:
Location:
C:\source\fsharpintro
Create new solution
Solution name:
Add to source control
OK
DEMO

Docker & Akka.Net & F#



Summary

Building Distributed Systems cross-platform with ***hot*** loads *(behaviors and dependencies)*

One solution does not exist… combining different tools and programming models… *bend technonlogies to your need*


That's all Folks!

The tools we use have a profound (and devious!) influence on our thinking habits, and, therefore, on our thinking abilities.

-- Edsger Dijkstra

How to reach me

github.com/rikace/Presentations/AkkaActorModel

meetup.com/DC-fsharp

@DCFsharp @TRikace

tericcardo@gmail.com

https://www.surveymonkey.com/r/S6PV89W