

# ColdFusion .NET Integration

Rupesh Kumar

Computer Scientist, ColdFusion Team

Adobe



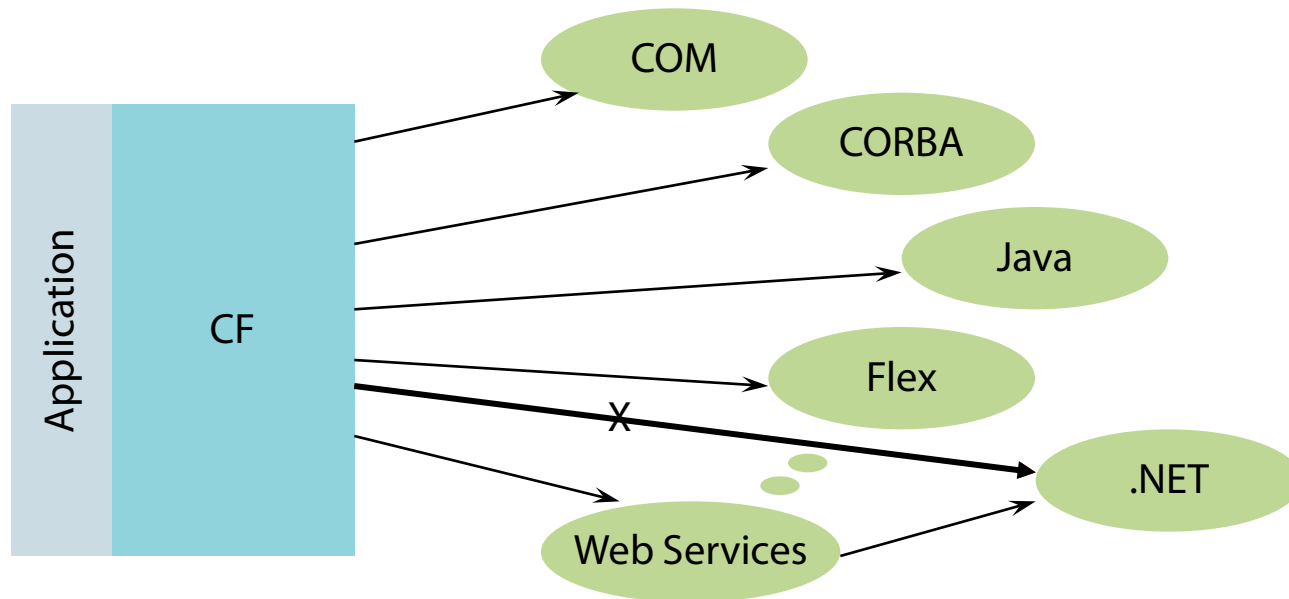
- Introduction
- Brief .NET Introduction
- Interoperability Strategies for .NET
- .NET Integration Features
- Syntax
- Demo
- Nuts and Bolts
- Deployment scenarios
- Limitation
- Q&A

- Why .NET integration ??
  - Leverage some functionality available in .NET
  - Leverage Microsoft products like Word, Excel, Outlook, Exchange server etc.
  - Existing investments.
  - Existing components and services
  - Acquired components and services

# What ColdFusion provides?



- Makes Hard Stuff Easy !!!
- Makes all the technologies available
- Keep it simple.
- Java, COM, corba, any webservice, flex



- Microsoft intermediate language (MSIL)
- CLR (like JVM for Java)
- Manages memory, threads, execution of MSIL etc
- .NET versions – 1.x and 2.0
- Assembly - .dll or .exe (C#, VB, VJ#, managed C++)
- Global assembly cache (GAC)

- Web Services
- Messaging
- Using COM
- Runtime Unification (NEW !!)

- .NET → CF (CF components exposed as web service)

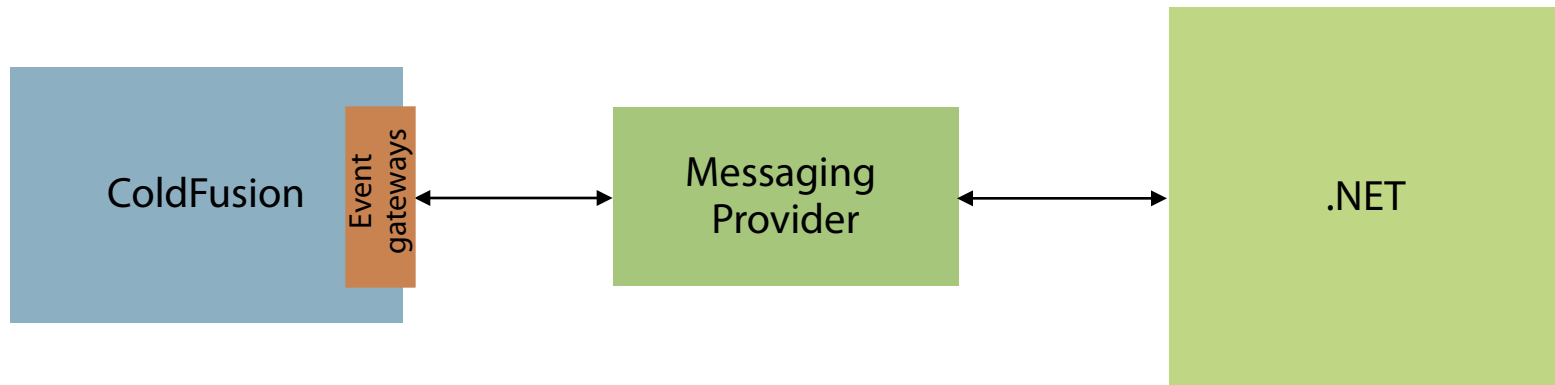
```
<cfcomponent>
  <cffunction name='echo' returnType='string' access='remote'>
    <cfargument name='input' type='string'>
      <cfreturn #arguments.input#>
    </cffunction>
</cfcomponent>
```

- Access the wsdl using component's URL  
<http://localhost/echo.cfc?wsdl>

- CF → .NET (.net components exposed as webservice)

```
<cfobject webservice="http://xyz/TempService.wsdl"
  name="obj">
<cfset temp=obj.getTemp("55987")>
```

- Messaging (JMS, MQSeries...)
- Event Gateways

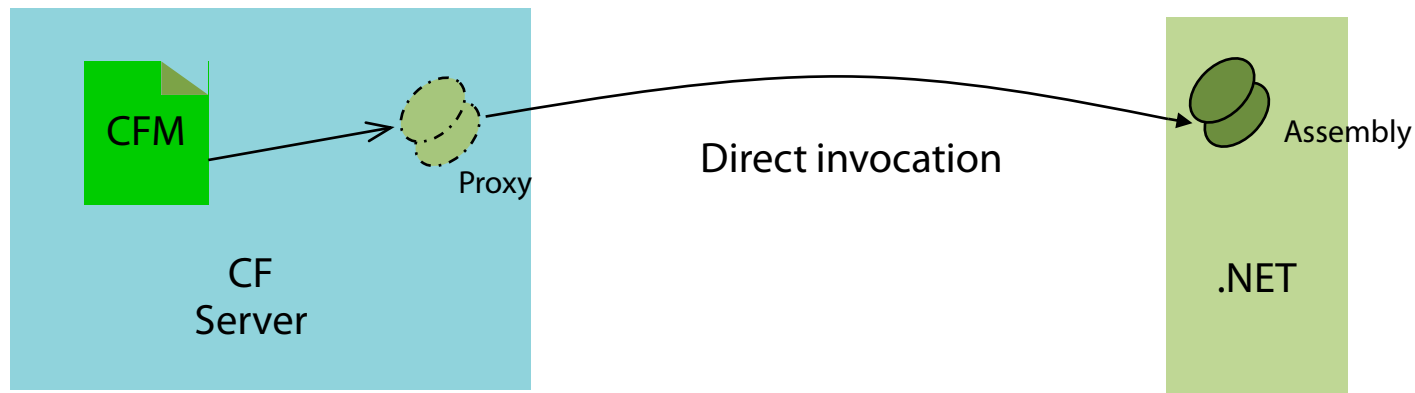


- COM
  - Create COM wrapper for .NET assembly
  - Create Java proxy using some tool
  - Invoke proxy from CF

# Runtime Unification – New in Scorpio !



- Make .NET components locally available
- For more fine grained control and invocations
- Invoke .NET components directly from cfm.



- Using cfunction and CreateObject
- Works very much same as cfunction/CreateObject for java

```
<cfunction type=".net" class="com.comp.AccountMgr"  
assembly="act.dll" name="actMgr">
```

```
<cfset act = actMgr.getAccount(101)>
```

```
<cfset bal = act.getBalance(>
```

## Web Services

- Loose Coupling
- Coarse granular and less programmatic control (as it is XML based)
- Low on performance, reliability and security
- Stateless
- Most useful to use it with external systems (third party services like weather/stock price webservices)

## Runtime Unification

- Tight Coupling
- Fine granular and more programmatic control as it is like invoking local objects
- High on performance, reliability and security
- Stateful
- Most useful when used with internal systems of enterprise.

- Seamless. You don't need to know much about .NET
- Location independent.
  - Can be local or remote
  - Can be even outside the firewall (over the web).
- Platform independent.
  - CF on any platform
  - .NET will of course be on a windows
- Hot deployment of assemblies
- Communication with multiple .NET side
- Secure
- Auto conversion from/to basic CF data types to/from .NET

```
<cfobject type=".Net"  
  class=".Net class"  
  name="Object name"  
  assembly="list of assemblies" [optional]  
  protocol="tcp|http" [optional]  
  secure="true|false" [optional]  
  server="IPAddress" [optional]  
  port="port"> [optional]
```

- `CreateObject(".Net", ".Net class")`
- `CreateObject(".Net", ".Net class", "assembly list")`
- `CreateObject(".Net", ".Net class", "assembly list", "IP Address", "port")`
- `CreateObject(".Net", ".Net class", "assembly list", "IP Address", "port", "protocol", "secure")`

- Assembly
  - List of dll's and/or exe's and/or proxy jars.
  - mscorlib.dll assembly will always be included.
  - Any referenced assembly if present in GAC will automatically be included.
- Protocol
  - Binary – default. Better performance
  - HTTP – can be used across firewall
- Secure CF-.NET communication using SSL
  - ColdFusion acts as SSL client
  - .NET side certificate should be trusted by Coldfusion.

- Constructors – use init

```
< cfoject type=".net" class="com.comp.Account"  
assembly="act.dll" name="act">
```

```
<cfset act.init("Rupesh", 1000)>
```

- Static method

```
<cfset types = act.getAccountTypes(>
```

```
<!--- no need to init() to call static method --->
```

- Calling methods

```
<cfset balance = act.getBalance(>
```

- Accessing public fields (using Get\_fieldname() and Set\_fieldname())

```
<cfset types = act.Get_name(> <!--- to access 'name' --->
```

- No setter if the field is final

DEMO

- Automatic conversion of primitive .NET datatypes to CF and CF (java) datatypes to .NET
- decimal type not supported
- use javacast() if required
- javacast enhanced to support byte, short and char.

## .NET

- sbyte
- byte
- short
- ushort
- int
- uint
- char
- long
- ulong
- float
- double
- bool
- String
- decimal

## Java

- byte
- short
- short
- int
- int
- long
- char
- long
- float
- float
- double
- boolean
- String
- NOT Supported

- Ambiguous Method Arguments

- Example 1

```
public void foo(int param)
public void foo(short param)
```

foo(12) - X

foo(javacast("short", 12)) - ✓

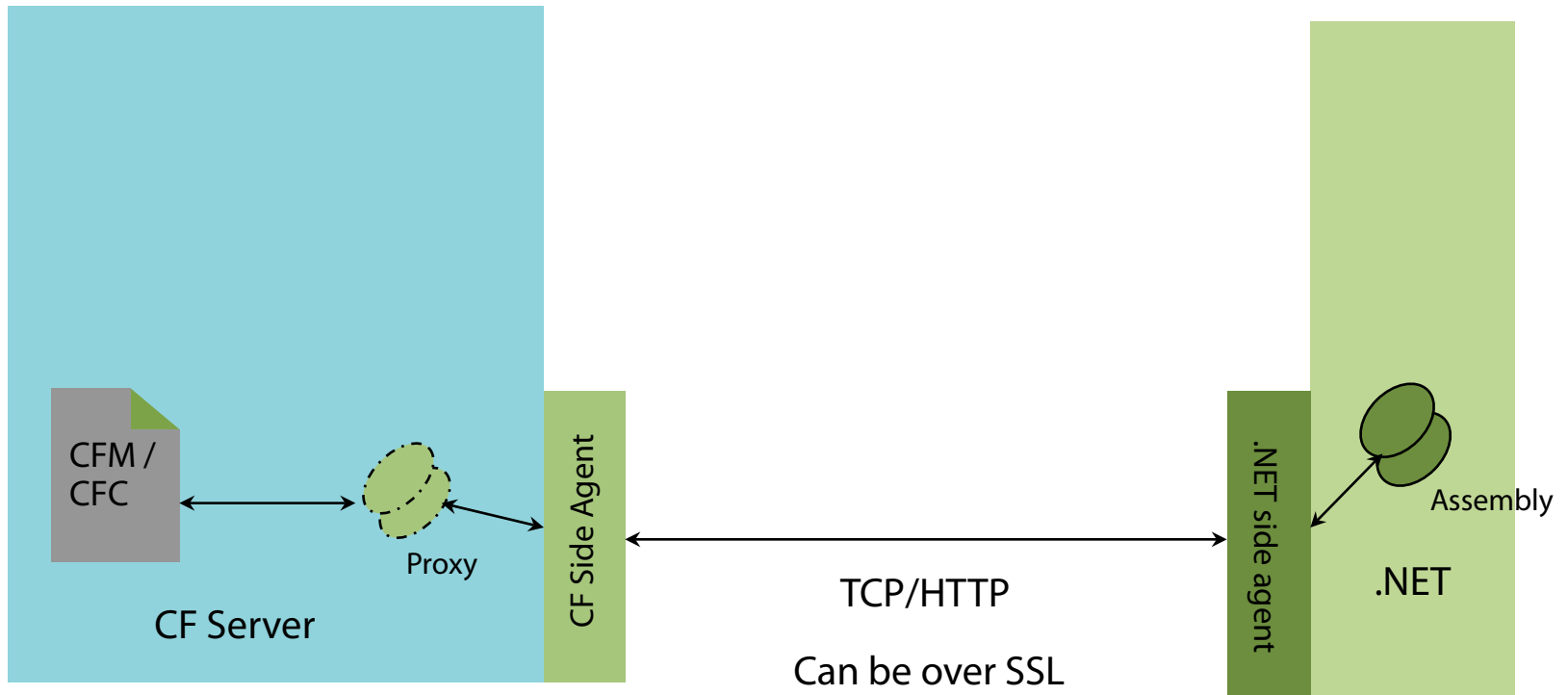
- Example 2

```
public void foo (String arg1)
public void foo (int arg1)
```

foo("123") - X

foo(javacast("String", "123")) - ✓

# What's going on inside?



- A proxy is what is used to interface between CF and the .NET runtime
- Automatically generated by CF if assembly is used
- To be generated by the user using a CF tool if CF is on non-windows machine.
- Generate once and use anywhere
- Proxies generated for 1.x can be used for 1.x as well as 2.0
- Proxies generated for 2.0 will only be used for 2.0
- Pass by reference (default)
- Pass by value
  - Reduced network calls
  - Bigger objects go over network. so can degrade performance
  - User should judge if this is required for a class
  - Ideal for simple data objects

- Agent that runs with .NET
- Registration of assemblies.
- Accepts calls from CF side proxy for invocation
- Delegates the call to the actual assembly
- Can configure port and protocol to be used
- Separate installer to install only the agent.

- CF and .NET on same machine
  - Of course it will be a windows machine 😊
  - No user configuration required
  - .NET side agent will be started by CF
  - Automatic registration of the assembly with the .NET side agent
  - Uses tcp protocol and default port by default
  - Auto generation of proxy if assembly changes

- CF and .NET on different machines - Both Windows
  - .NET side agent (shipped with CF) needs to be installed
  - Register the assemblies with .NET side agent
  - Ensure that .NET side agent is running on the remote machine
  - In cfoject/CreateObject, use host and port

- CF on non-Windows
  - .NET side agent (shipped with CF) needs to be installed
  - Generate the proxy using CF tool – jnbproxy.exe on .NET machine
  - Register the assembly with the .NET side agent
  - Ensure that .NET side agent is running on the remote machine
  - Copy the generated jar on the CF-machine
  - Use proxy jar in the assembly list.
  - In cfoject/CreateObject, use host and port

- Enum and Decimal data type
- methods with out parameters as arguments
- methods with pointers as arguments or return type
- .NET UI components
- Callbacks (events and Delegates) from .NET side

Q & A

**Better by Adobe.™**