SignalR

DEVELOPMENTOR

DEVELOPING PEOPLE WHO DEVELOP SOFTWARE

Agenda

- Problem space
- What is ASP.NET SignalR
- Building a simple app with SignalR
  - Javascript client
  - C# client
- Scaling

Problem space

- Http protocol
  - Client sends request
  - Server responds
- Client needs to re-request to get update data
  - Manual user refresh
  - Timer based
- Not practical for
  - Web based collaboration
  - Real time updates (Stock trading, online gaming)

Solutions

- There are many solutions that could be used
  - HTML 5
    - Web Sockets
    - Server Send Events
  - Comet transports
    - Forever frame
    - Ajax long polling
- Not all solutions supported by all client/server combinations
- All achieve real time push updates from server

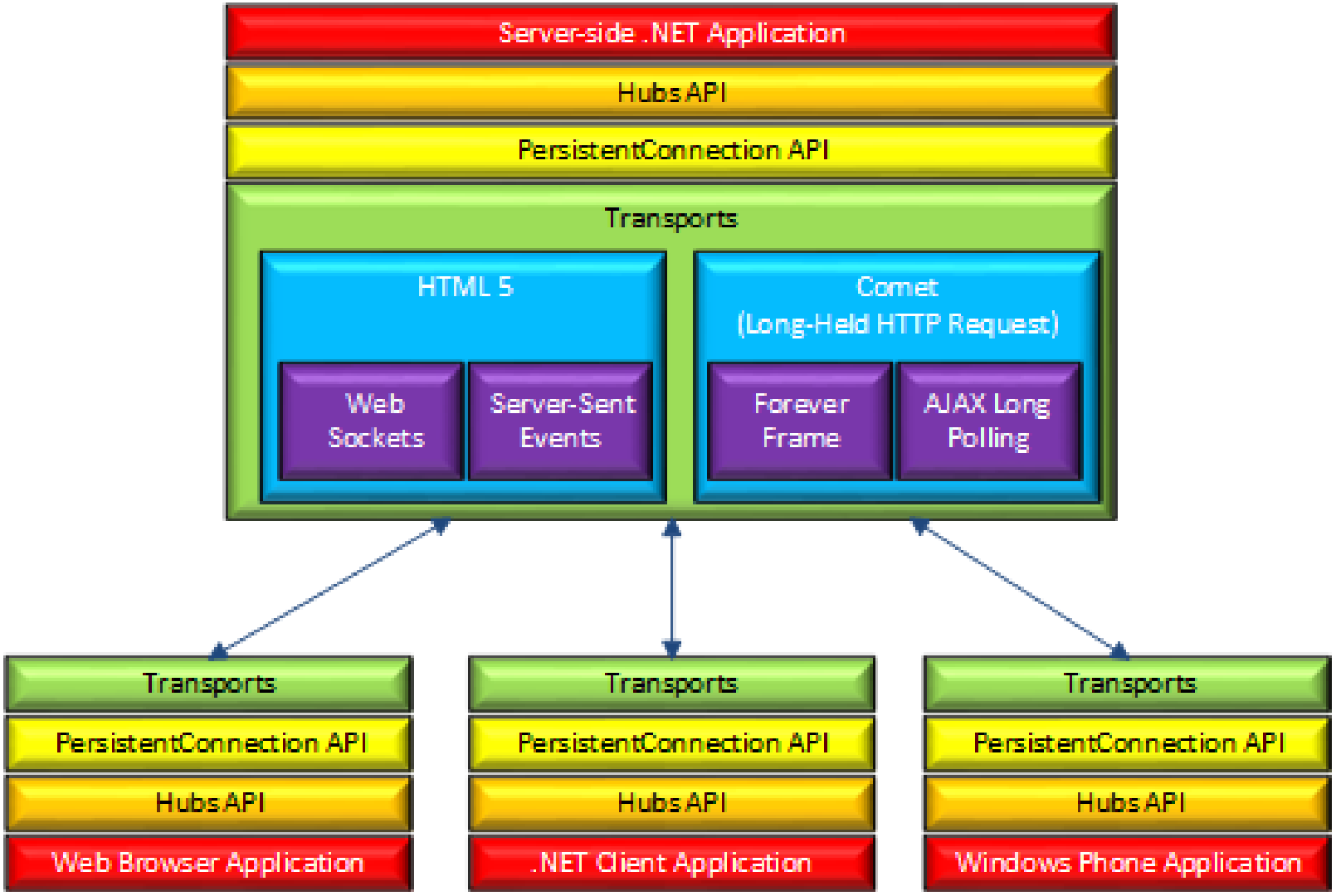
What is ASP.NET SignalR

- Provides abstraction for server push over http
  - Negotiates protocol
- Uses JSON for message format
- Support for the following clients
  - JavaScript
  - .NET
  - Windows Phone
- Open Source, accessible via GitHub

Connections and Hubs

- Connections
  - Low level communication api
  - Ideal when complete control is required over message format
  - Client/Server receive messages and provides own dispatching logic
- Hubs
  - Provide RPC abstraction
  - Clients register with a hub
    - Can call methods on Server
    - Can call methods on all registered clients
  - The most common usage model for SignalR

Layered architecture



Hub Message Format

- Tight and compact
  - **Hubname**
  - **Method**
  - **Arguments**
- Serializes to JSON

HTTP BODY

{"H":"**stocktickerhub**","M**":"GetPriceAndSubscribe**","A":["**CSCO**"],"I":0}

Message type

- Single
  - Send a message to a specific client
- Group
  - Send to a group of clients
- Broadcast
  - Send to all clients

Hosting

- Typically inside a web server
  - IIS ASP.NET
    - Traditional ASP.NET
    - Using Katana/OWIN
- Self hosting via Katana/OWIN

Building server side Hub

- Server side code derives from Hub class
  - Similar to MVC Controllers
- Public methods can be called by client
- All return data serialized in JSON
- Can decorate with HubName and HubMethodName, otherwise member names are used

```
[HubName("stockTicker")]
public class StockTickerHub :Hub
{
  [HubMethodName("getPrice")]
  public object GetPrice(string symbol)
  {
    return new {Symbol = symbol, Price = 10.20m};
  }
}
```

Initialising SignalR

- Initialised via OWIN Startup class
- MapSignalR
  - Can supply URL for JavaScript proxy code
  - Can supply configuration (HubConfiguration class)
    - Enable Detailed error reporting to client
    - Disable JavaScript proxies

```
public class Startup
{
   public void Configuration(IAppBuilder appBuilder)
   {
     // Will generate Javascript client proxies
     // /signalR/hubs

     appBuilder.MapSignalR();
    }
}
```

JavaScript client

- Server generates client side JavaScript proxies
- **Hub name** and **method names** turned into javascript convention
- NOTE, if hub is decorated with HubName and HubMethodName attributes case is preserved.
- **Enable logging to console**

```
<script  src="Scripts/jquery-1.6.4.js"></script>
<script src="Scripts/jquery.signalR-2.1.0.js"></script>
<script src="/signalr/hubs"></script>

$(document).ready( function() {
  $.connection.hub.logging = true;
  $.connection.hub.start()
                    .done(function () {
                        var stockTicker = $.connection.stockTicker.server;

                      stockTicker.getPrice ("CSCO")
                        .done(addStockQuote);
                    });
} );
```

.NET client

- Nuget package Microsoft.ASPNET.SignalR.Client
- HubConnection connects to the SignalR base URL
- IHubProxy connects to a specific hub
  - Proxies **must** be created before the connection is started
- IHubProxy.Invoke to invoke a method on the server
  - returns Task<T>

```
var connection = new HubConnection("http://localhost:17724/signalr");
IHubProxy proxy= connection.CreateHubProxy("stockTicker");

connection.Start().Wait();

dynamic quote = proxy.Invoke<dynamic>("getPriceAndSubscribe","CSCO").Result;

Console.WriteLine("{0} {1}",quote.Symbol,quote.Price);
```

Server calling client

- From inside the hub
  - Client sends message to server
  - While processing the message the hub wishes update one or many registered clients
- From outside the hub
  - Server detects state change and needs to communicate change with clients
- Clients need to register interest in receiving messages from the hub

Calling clients from within the Hub

- From within the Hub
  - Use Clients property to direct call
  - Offers range of possible targets
  - Targets are dynamic, leaving binding to client

```
[HubName("speaker")]
public class Speaker:Hub
{
  [HubMethodName("shout")]
  public void Shout(string message)
  {
       Clients.All.acceptMessage(message);
  }
}
```

Calls acceptMessage on all clients connected to this hub

JavaScript client registration

- Client needs to provide binding from server message method to code
- MUST register before calling start

```
function showMessage(message)
{
   alert(message);
}

 $(document).ready(function () {
  var client = $.connection.speaker.client;
  client.shout = showMessage;

  $.connection.hub.start()
                  .done(function () {

   }
 });
```

.NET Client Registration

- Use IHubProxy.On method to bind server method message to code
- Type arguments used to define expected parameters up to 7

```
 static void Main(string[] args)
 {
    HubConnection connection = new HubConnection("http://localhost:6944/");

    IHubProxy proxy = connection.CreateHubProxy("speaker");

    connection.Start().Wait();

    proxy.On<dynamic>("acceptMessage", msg =>
    {
       Console.WriteLine("{0} {1}", msg.from, msg.text);
    });

    Console.ReadLine();
}
```

Directing calls to clients

- Variety of ways to direct calls

Target	Description
Caller	The client that issued the executing request
Others	All other clients connected to this hub, but not the one that issued the executing request
All	All clients connected to this hub
Group("groupname")	All clients part of the named group
OthersInGroup("groupname")	All clients part of this group but not the client that issued the executing request
Groups(List<string>)	All clients in any of the listed groups
OthersInGroups(List<string>)	All clients part of any of the supplied group accept excluding the client that issued the executing request

Calling clients from outside the Hub

- Server state has changed, needs to notify clients connected to a hub
  - E.g. Database updated
- Use IConnectionManager to obtain context for Hub

```
var connectionManager =
    GlobalHost
    .DependencyResolver
    .GetService(typeof(IConnectionManager)) as IConnectionManager;

IHubContext hub = connectionManager.GetHubContext<SpeakerHub>();

hub.Clients.All.update();
```

Hub Groups

- Clients can be organized into groups
- Hub can target messages to specific groups
- Group membership maintained by client
  - Allows group membership to be maintained across re-start and failover

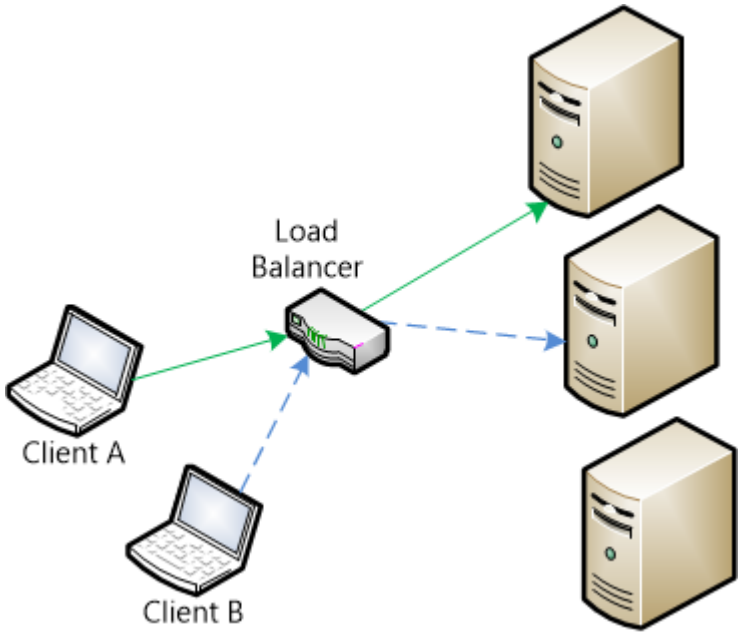
```
public class Speaker:Hub
{
  public void KeepTellingMeTheTime()
  {
     HubCallerContext callerContext = Context;
     Groups.Add(callerContext.ConnectionId, "time");
  }
}
. . .
var connectionManager = GlobalHost.DependencyResolver
    .GetService(typeof(IConnectionManager)) as
     IConnectionManager;

IHubContext hub = connectionManager.GetHubContext<SpeakerHub>();

hub.Clients.Group("time").speakTime(DateTime.Now.ToString());
```

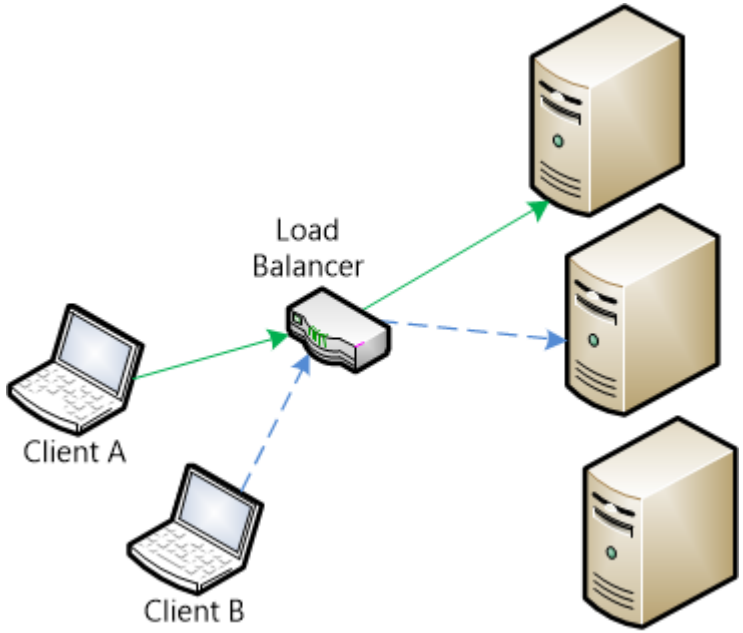
Scaling

- Server side scaling two basic models
  - Scale Up
    - Bigger faster server
    - Simple, but has obvious limits
  - Scale Out
    - Load balancer + more servers
    - Can be more complex, cache coherency issues



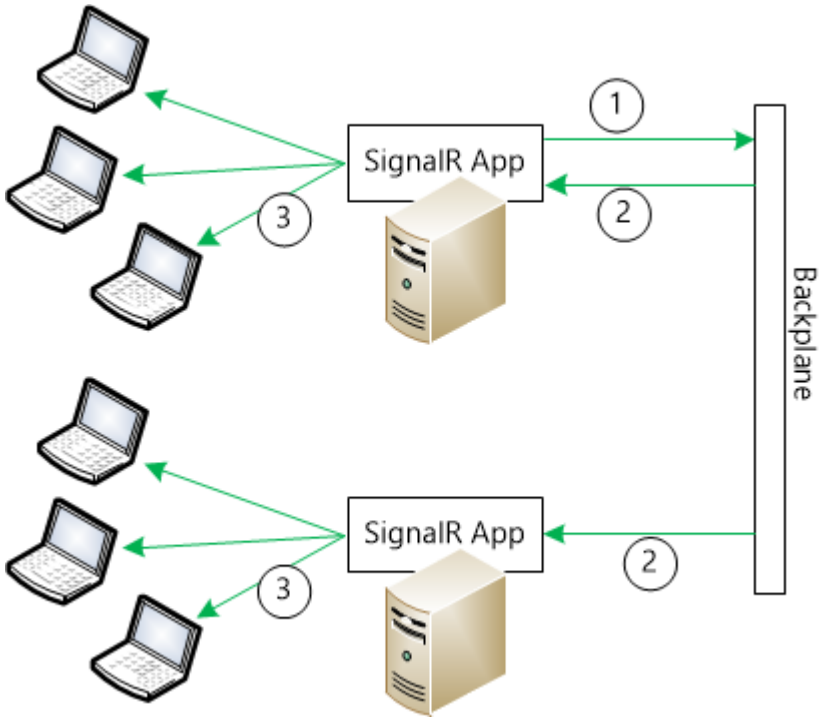
Scaling out

- Scaling out creates a problem
  - A hub wants to send a message to all clients connected to the same hub on any server in the farm
  - Each server hubonly knows about its own clients
- Solution
  - SignalR backplane



SignalR backplane

- [1] Server directs message to backplane
- [2] Backplane sends message to all servers registered on the backplane
- [3] Each server sends message to its connected clients



Backplane implementations

- Windows Azure Service Bus
  - Good for Azure based deployments
- Redis
  - Good for own server farm
- SQL Server
  - Good for own server farm

Summary

- SignalR,
  - Provides duplex communication over HTTP
  - Negotiates best technology
- HTTP can now be used for duplex communication
  - Delivers real time updates to browsers
  - Ideal for non browser apps in heavily firewalled environments