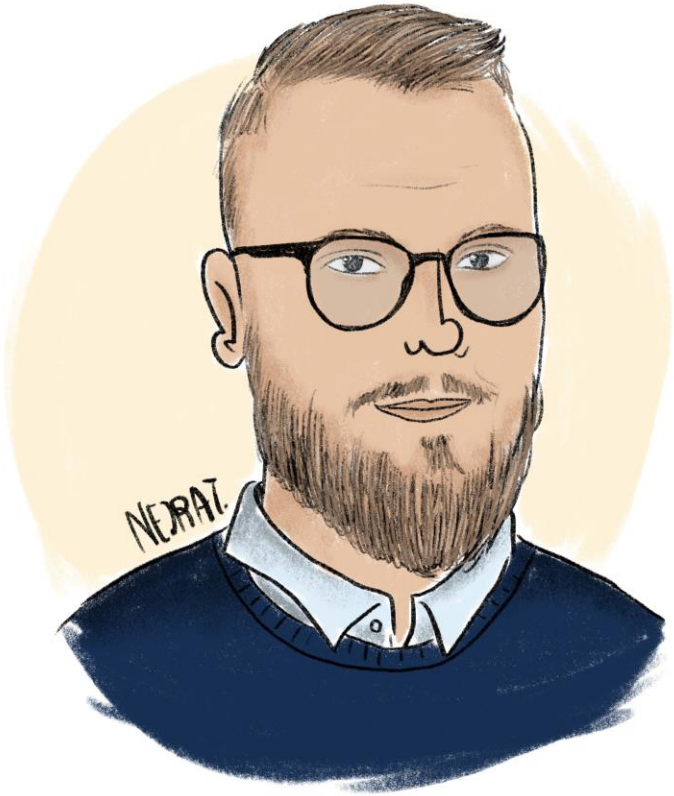
Azure SignalR Service

Almir Vuk

Software Engineer & Microsoft MVP
AgentLocator Inc.

Almir Vuk



I am **Software Development Engineer** and **Microsoft MVP** working with **.NET** and **C#,** crafting apps with **ASP.NET Core** for Web and **Xamarin** for mobile. Currently part of **AgentLocator Inc**.

Spending free time, running, playing chess, speaking on conferences, answering questions **StackOverflow** and committing to open-source projects on **GitHub.**

MVP Microsoft® Most Valuable Professional

AGENT LOCATOR

almirvuk.com
@almirvuk

ASP.NET Core SignalR and Azure

ASP.NET Core SignalR is an open-source library that simplifies adding real-time web functionality to apps.

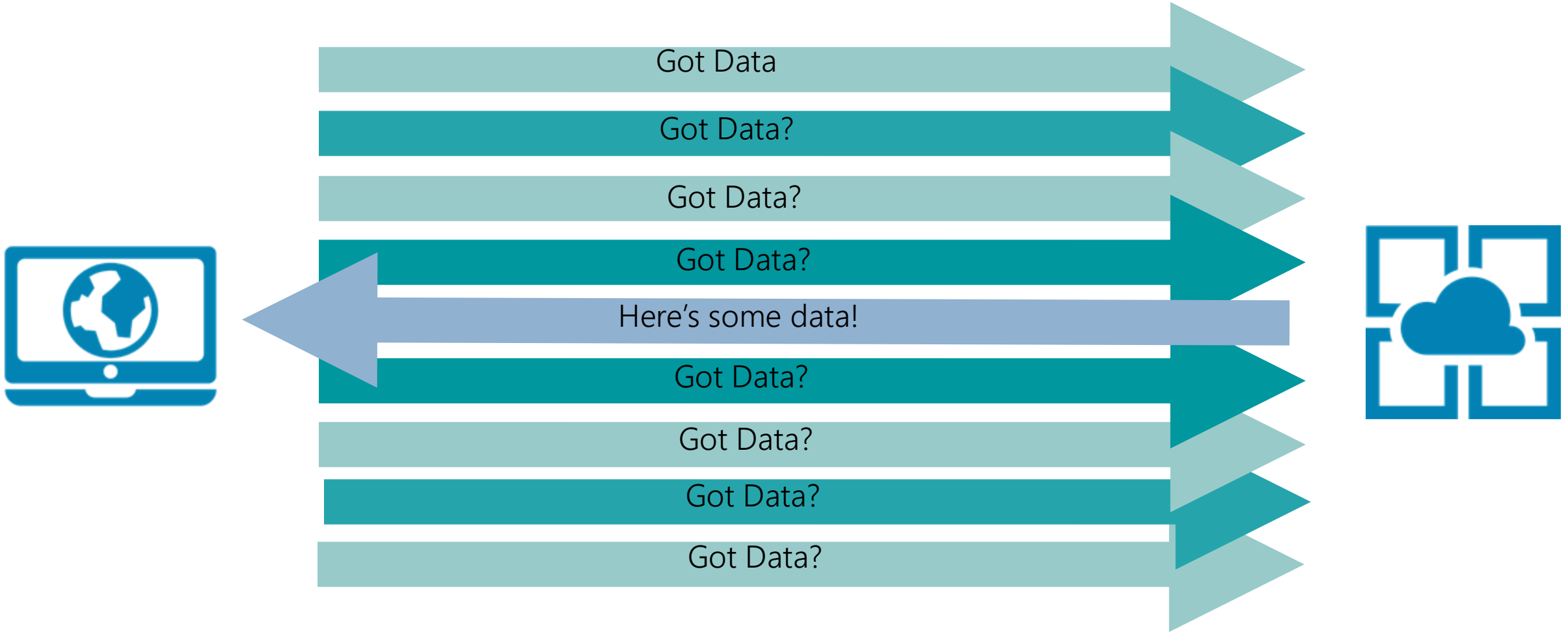
Real-time web functionality enables server-side code to push content to clients instantly.

ASP.NET Core SignalR

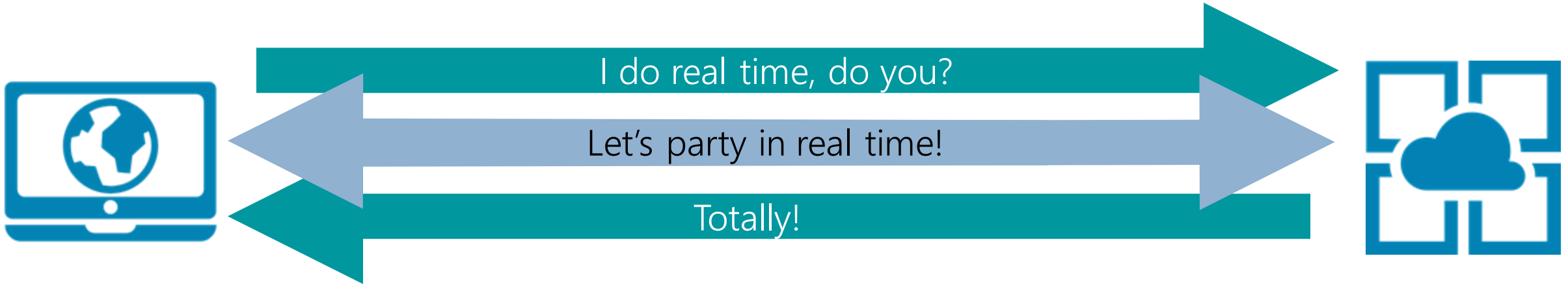
Good candidates for SignalR:

- Apps that require high frequency updates from the server. Examples are gaming, social networks, voting, auction, maps, and GPS apps.
- Dashboards and monitoring apps.
  Examples include company dashboards, instant sales updates, or travel alerts.
- Collaborative apps. Whiteboard apps and team meeting software are examples of collaborative apps.
- Apps that require notifications. Social networks, email, chat, games, travel alerts, and many other apps use notifications.

Client Negotiation



Client Negotiation



Transports

- SignalR supports the following techniques for handling real-time communication (in order of graceful fallback):
  - WebSockets
  - Server-Sent Events
  - Long Polling

- SignalR automatically chooses the best transport method that is within the capabilities of the server and client.

Web Sockets

It is a protocol which provides a full-duplex communication channel over a single TCP connection.

For instance a two-way communication between the Server and Browser.

Since the protocol is more complicated, the server and the browser has to rely on library of websocket which is SignalR in .NET world.

Hello World example

```
public class Startup
{
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddMvc();
        services.AddSignalR()
                .AddAzureSignalR();
    }

    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        app.UseMvc();
        app.UseFileServer();

        app.UseAzureSignalR(routes =>
        {
            routes.MapHub<Chat>("/chat");
        });
    }
}
```

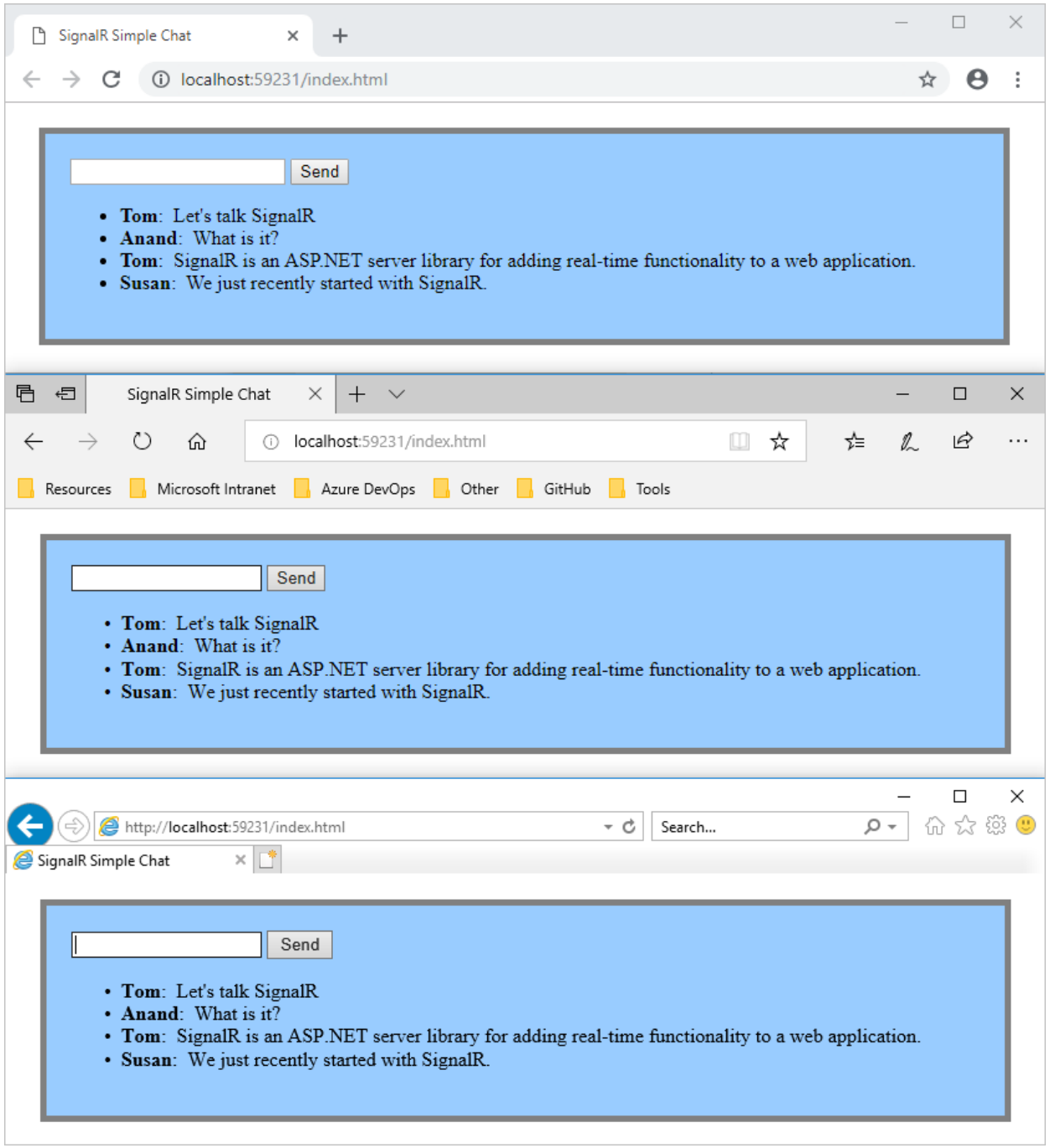
Hubs

- SignalR uses *hubs* to communicate between clients and servers.

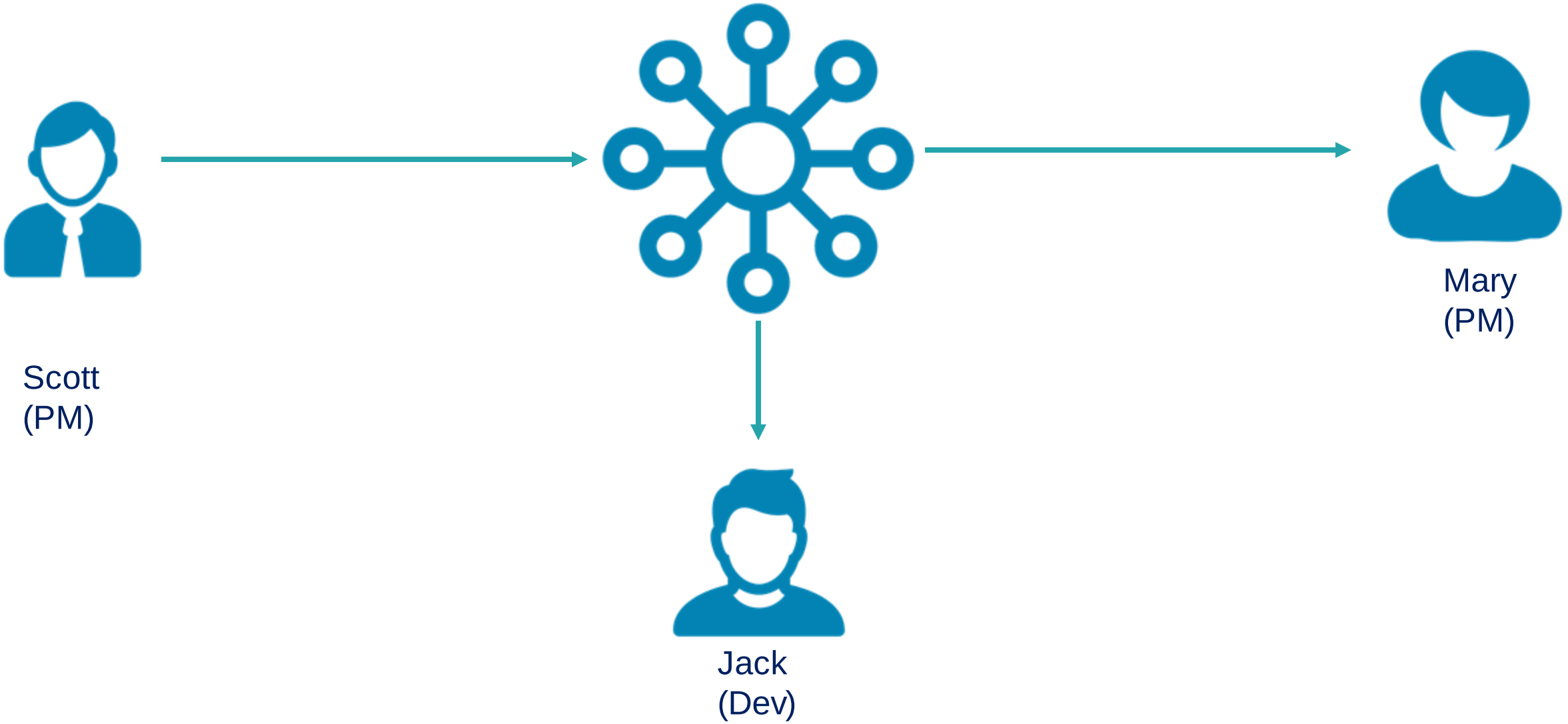
- The SignalR Hubs API enables connected clients to call methods on the server. The server defines methods that are called from the client and the client defines methods that are called from the server. SignalR takes care of everything required to make real-time client-to-server and server-to-client communication possible.

```
public class Chat : Hub
{
    public void BroadcastMessage(string name, string message)
    {
        Clients.All.SendAsync("broadcastMessage", name, message);
    }

    public void Echo(string name, string message)
    {
        Clients.Client(Context.ConnectionId).SendAsync("echo", name,
            message + " (echo from server)");
    }
}
```

```
function bindConnectionMessage(connection) {

  var messageCallback = function (name, message) {

  if (!message) return;
    alert("message received:" + message);
  };

  // Create a function that the hub can call to broadcast messages.
  connection.on('broadcastMessage', messageCallback);
  connection.on('echo', messageCallback);
}

var connection = new signalR.HubConnectionBuilder()
.withUrl('/chat')
.build();

bindConnectionMessage(connection);

connection.start()
  .then(function () {
      onConnected(connection);
})

  .catch(function (error) {
      console.error(error.message);
});
```


SignalR Simple Chat
localhost:59231/index.html
Send
Tom: Let's talk SignalR
Anand: What is it?
Tom: SignalR is an ASP.NET server library for adding real-time functionality to a web application.
Susan: We just recently started with SignalR.
SignalR Simple Chat
localhost:59231/index.html
Resources
Microsoft Intranet
Azure DevOps
Other
GitHub
Tools
Send
Tom: Let's talk SignalR
Anand: What is it?
Tom: SignalR is an ASP.NET server library for adding real-time functionality to a web application.
Susan: We just recently started with SignalR.
http://localhost:59231/index.html
Search...
SignalR Simple Chat
Send
Tom: Let's talk SignalR
Anand: What is it?
Tom: SignalR is an ASP.NET server library for adding real-time functionality to a web application.
Susan: We just recently started with SignalR.

Client Targeting with SignalR Hubs



Send messages to clients

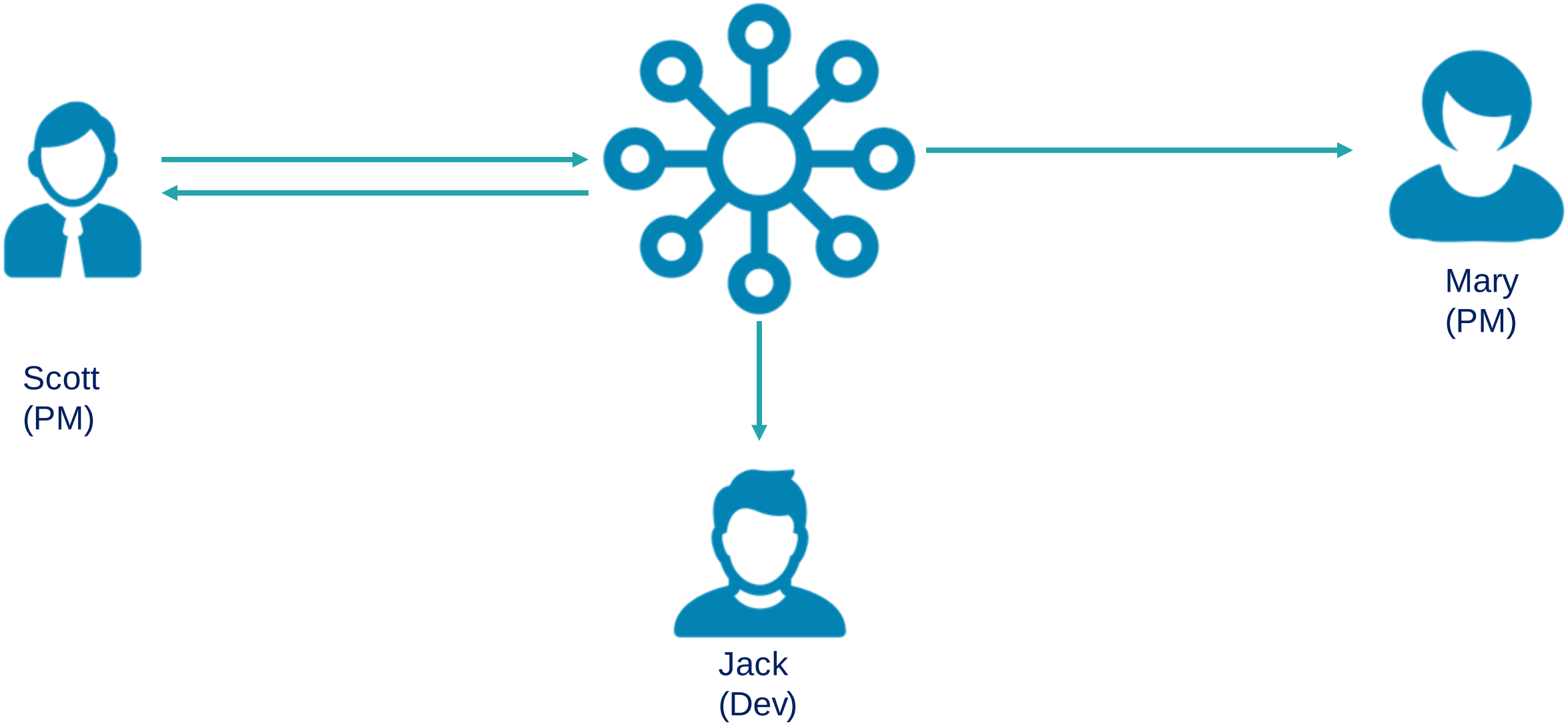
- To make calls to specific clients, use the properties of the Clients object.

  There are three hub methods:

- **`SendMessage`** sends a message to all connected clients, using **`Clients.All`**.
- **`SendMessageToCaller`** sends a message back to the caller, using **`Clients.Caller`**.
- **`SendMessageToGroup`** sends a message to all clients in the SignalR Users group.

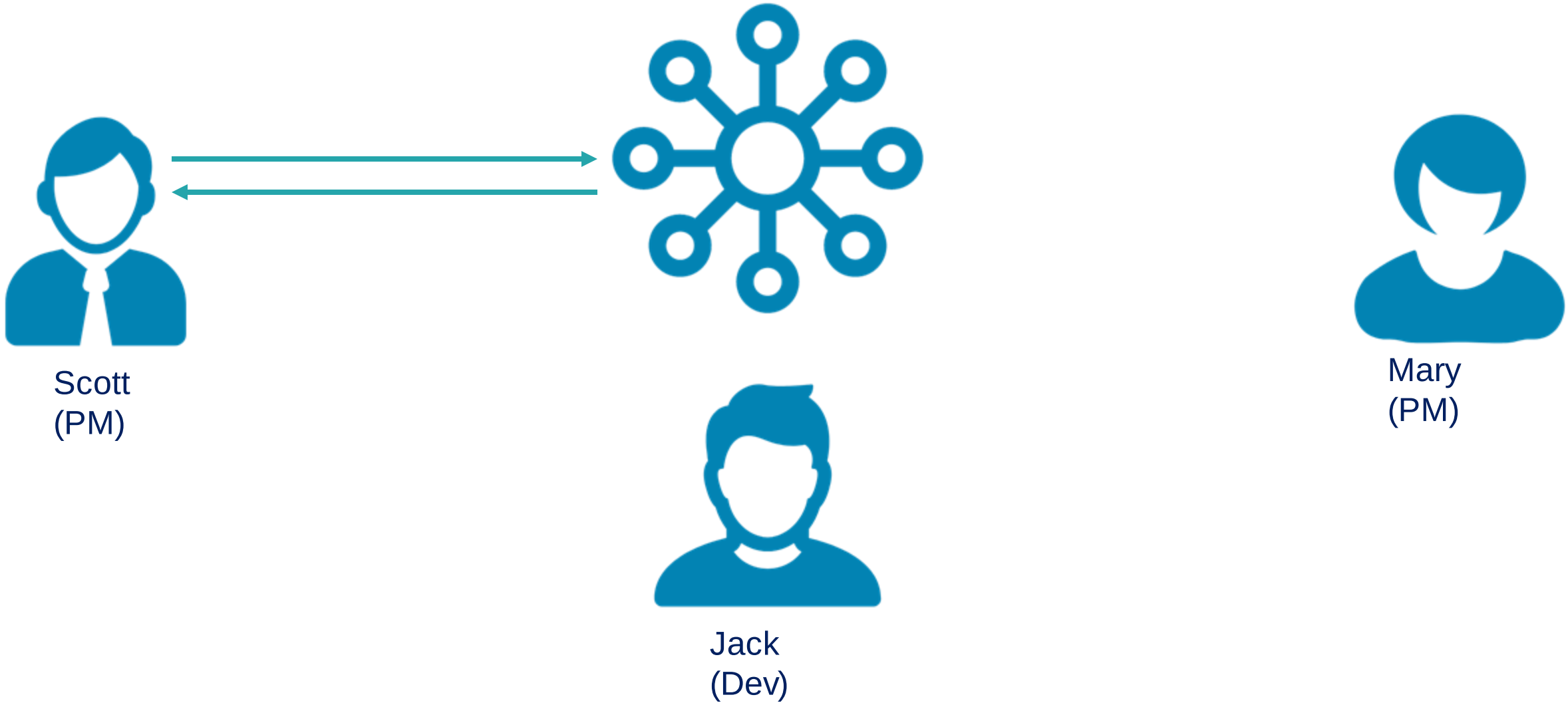
Client Targeting with SignalR Hubs

```
Clients.All.SendAsync(...)
```



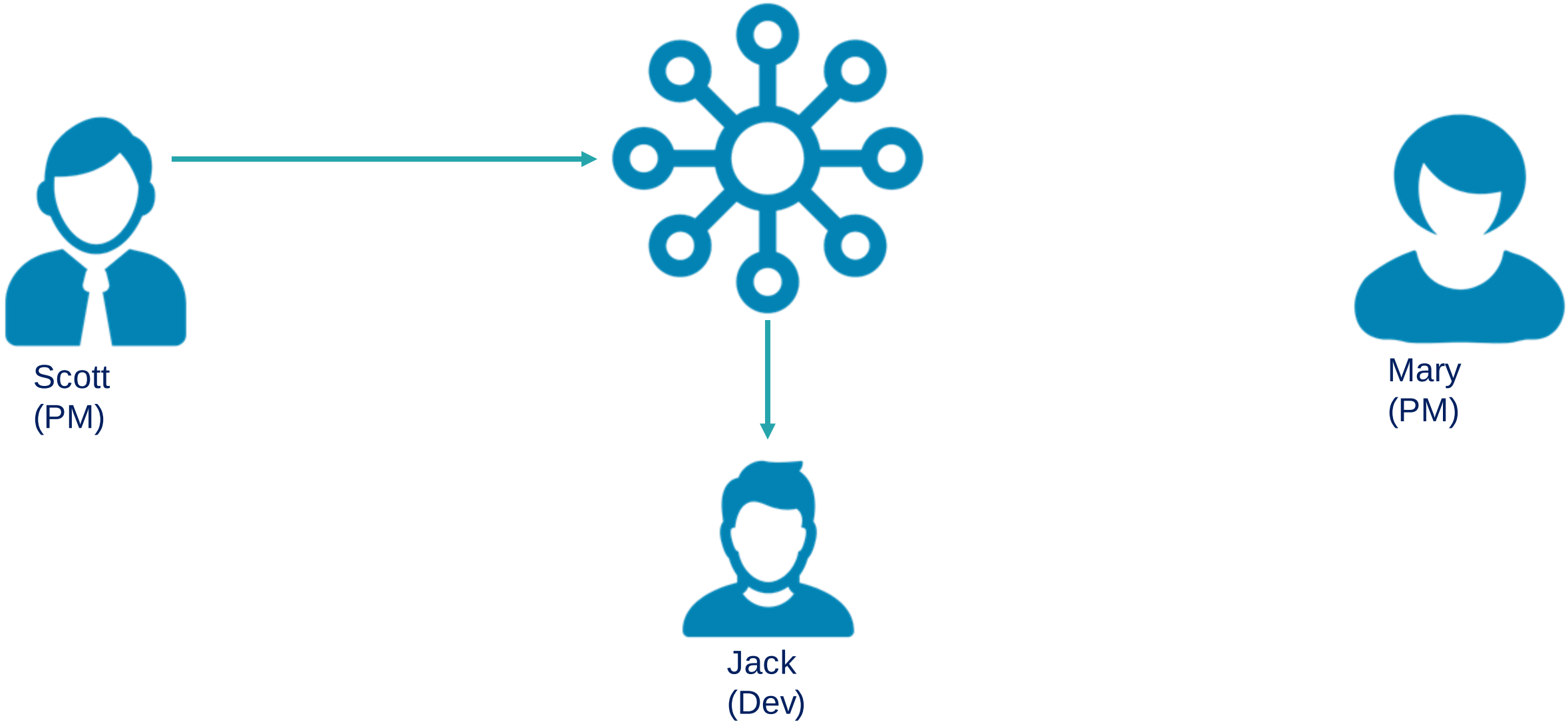
Client Targeting with SignalR Hubs

```
Clients.Caller.SendAsync(...)
```



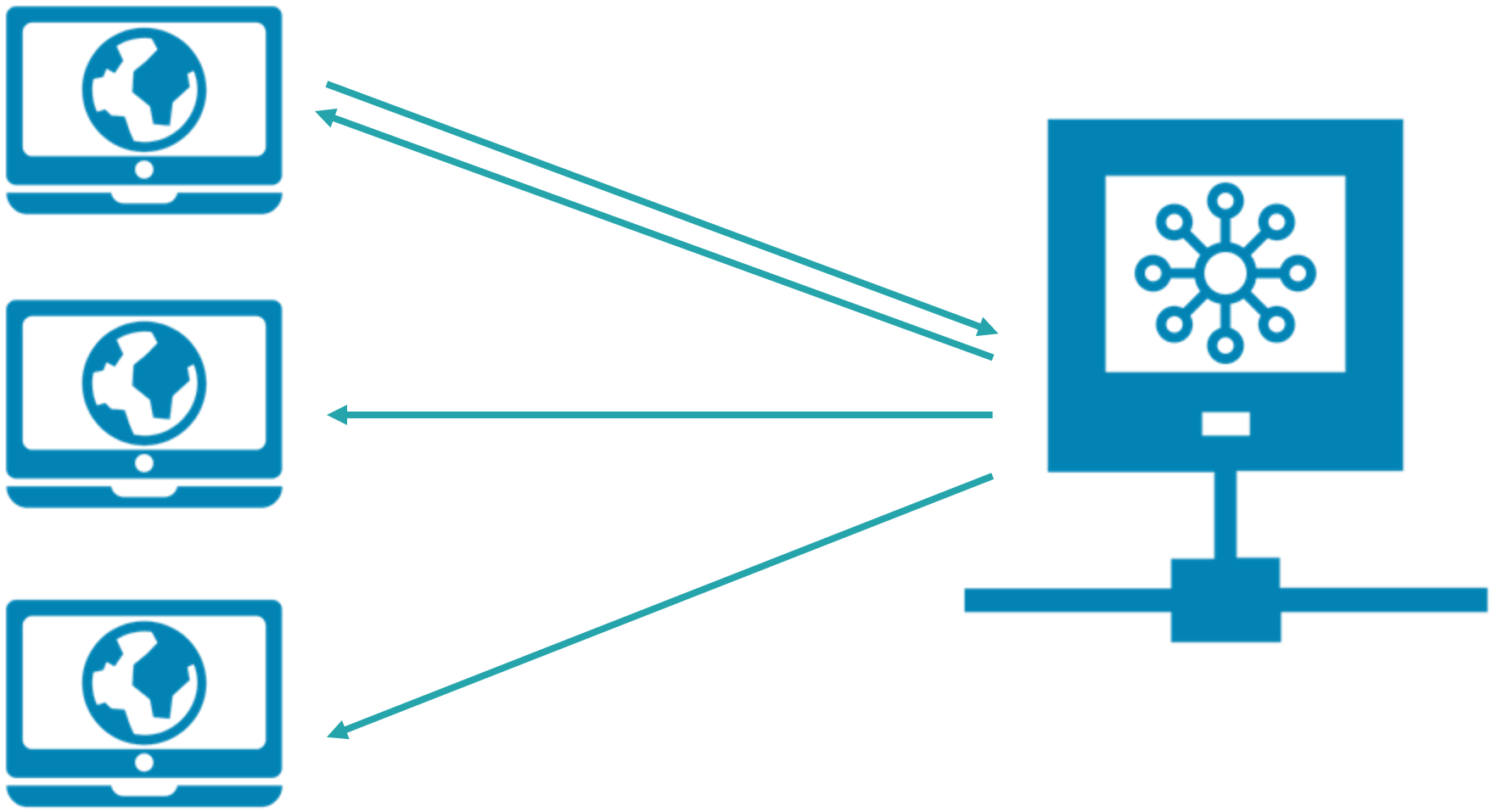
Client Targeting with SignalR Hubs

```
Clients.Groups(''Dev'').SendAsync()
```

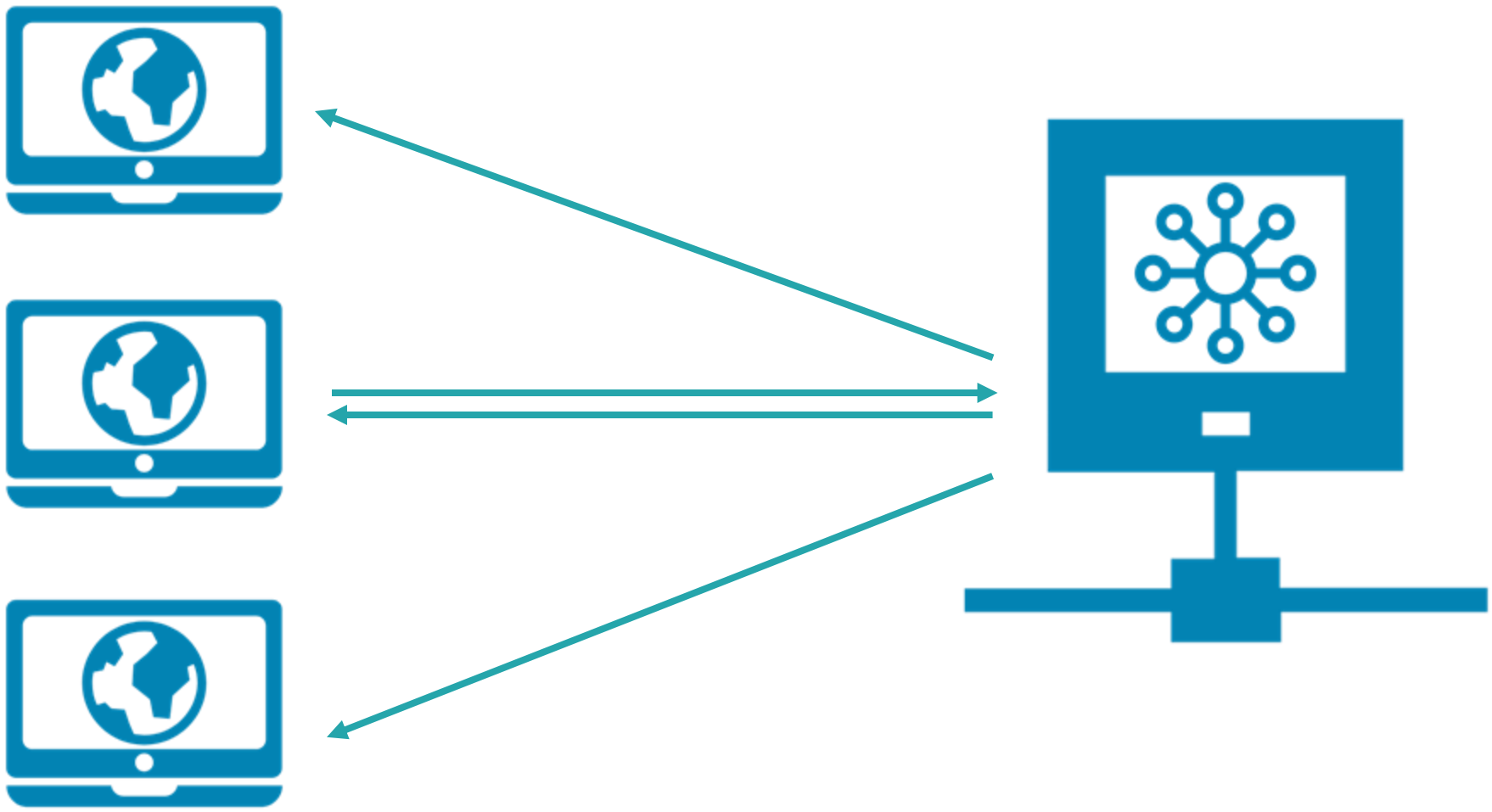


…now the interesting part begins 😂

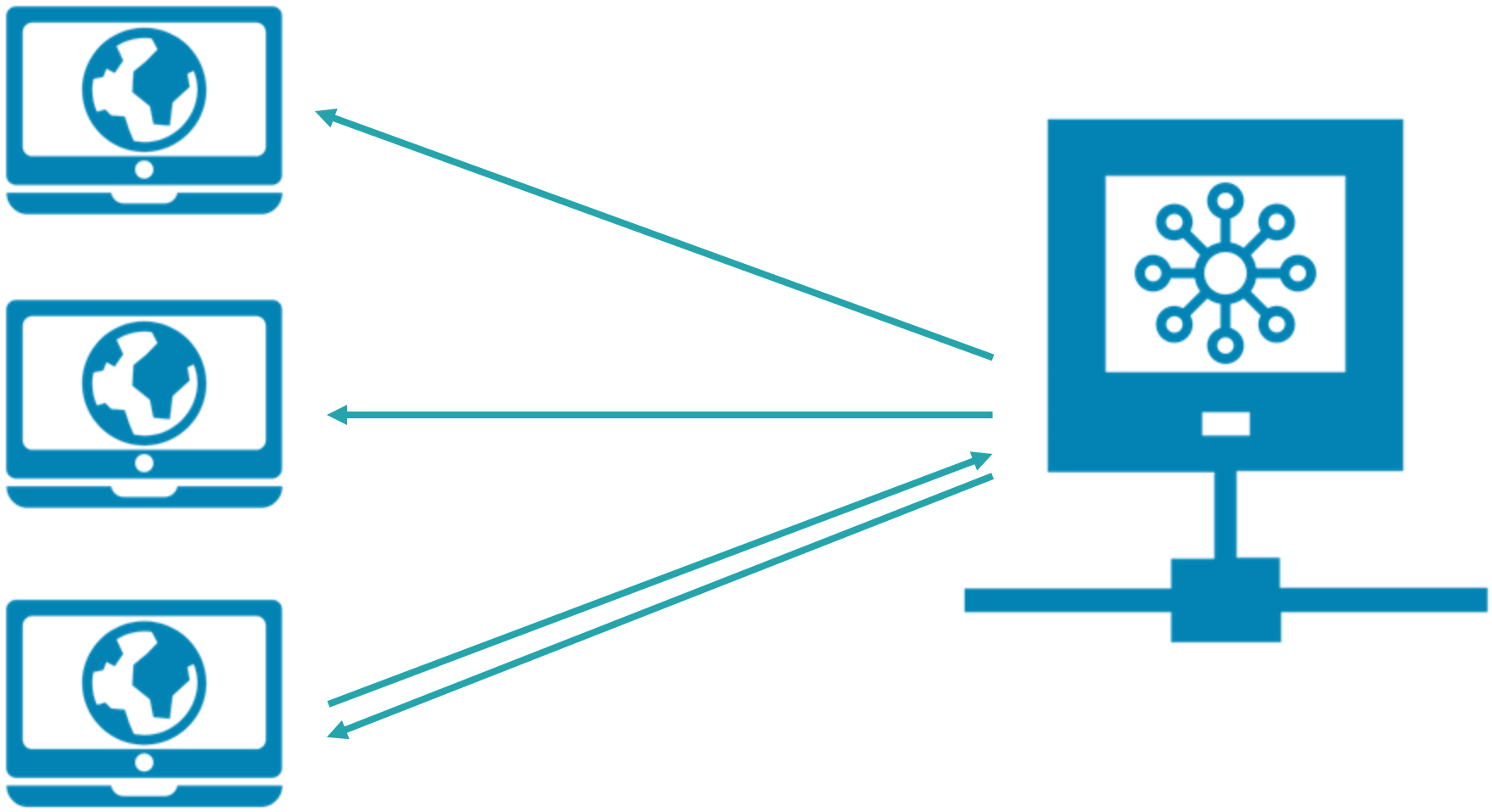
SignalR Hubs are Server-bound



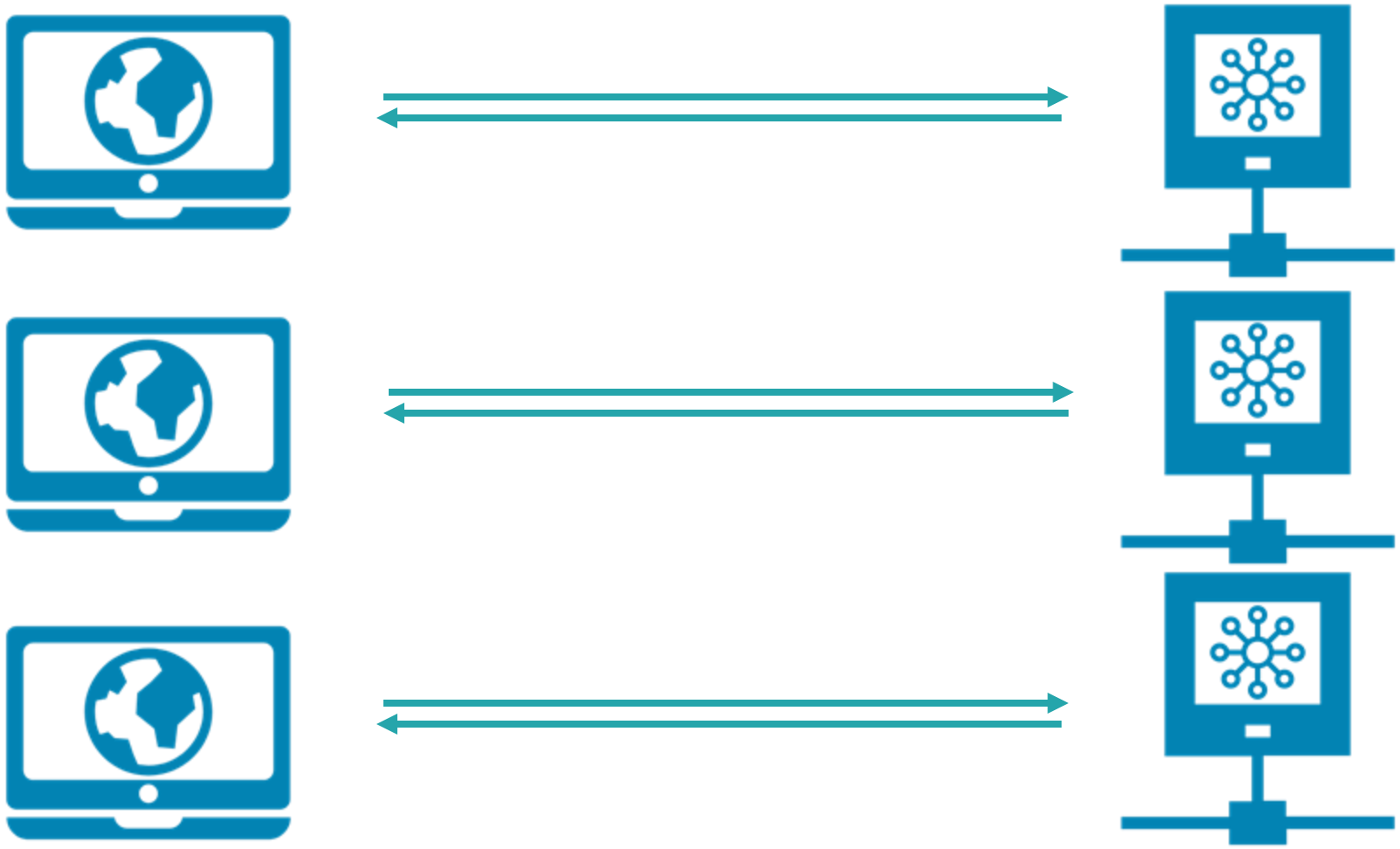
SignalR Hubs are Server-bound



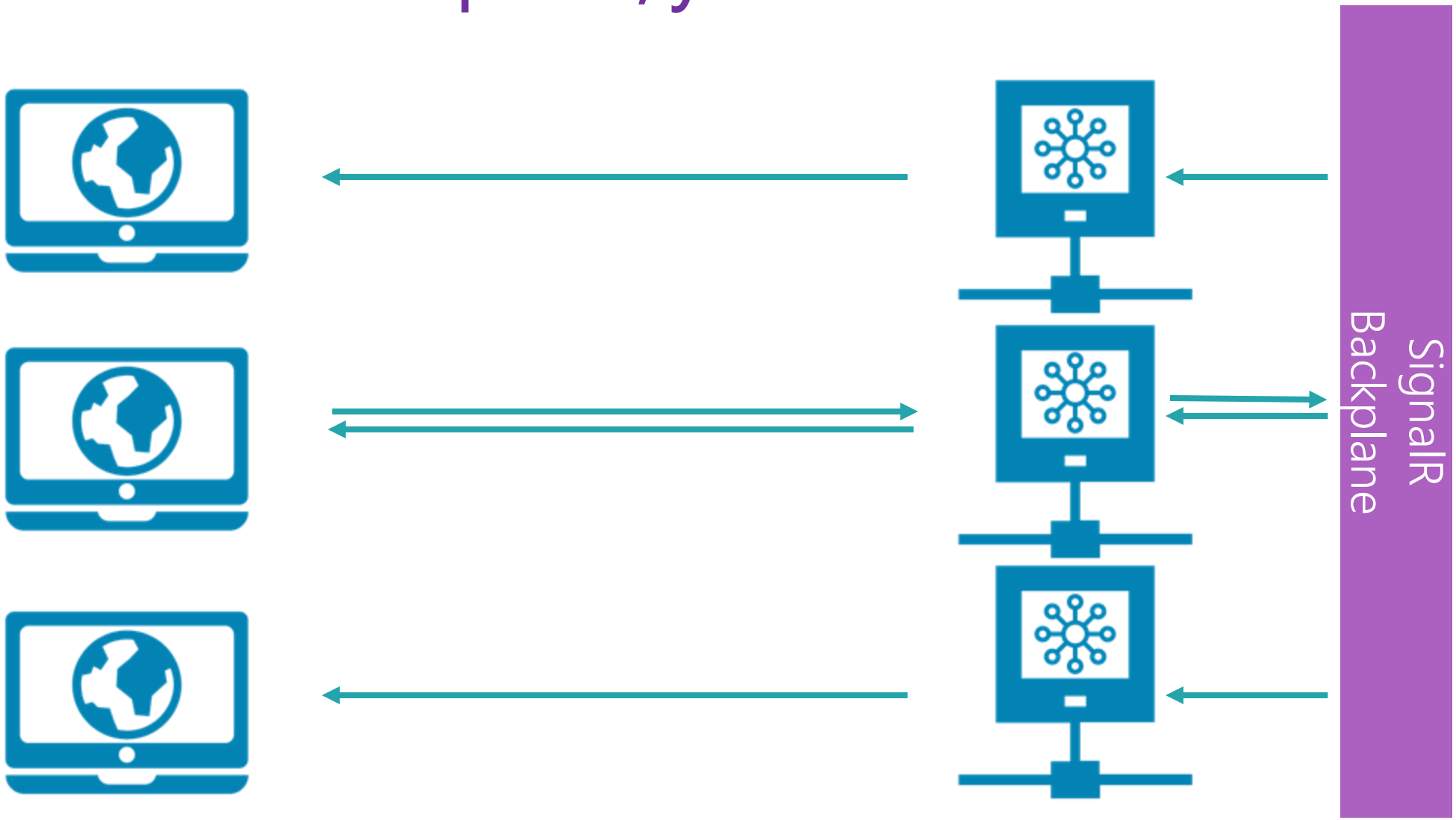
SignalR Hubs are Server-bound



So, does SignalR not work in a server farm?



Once you add a backplane, yes!



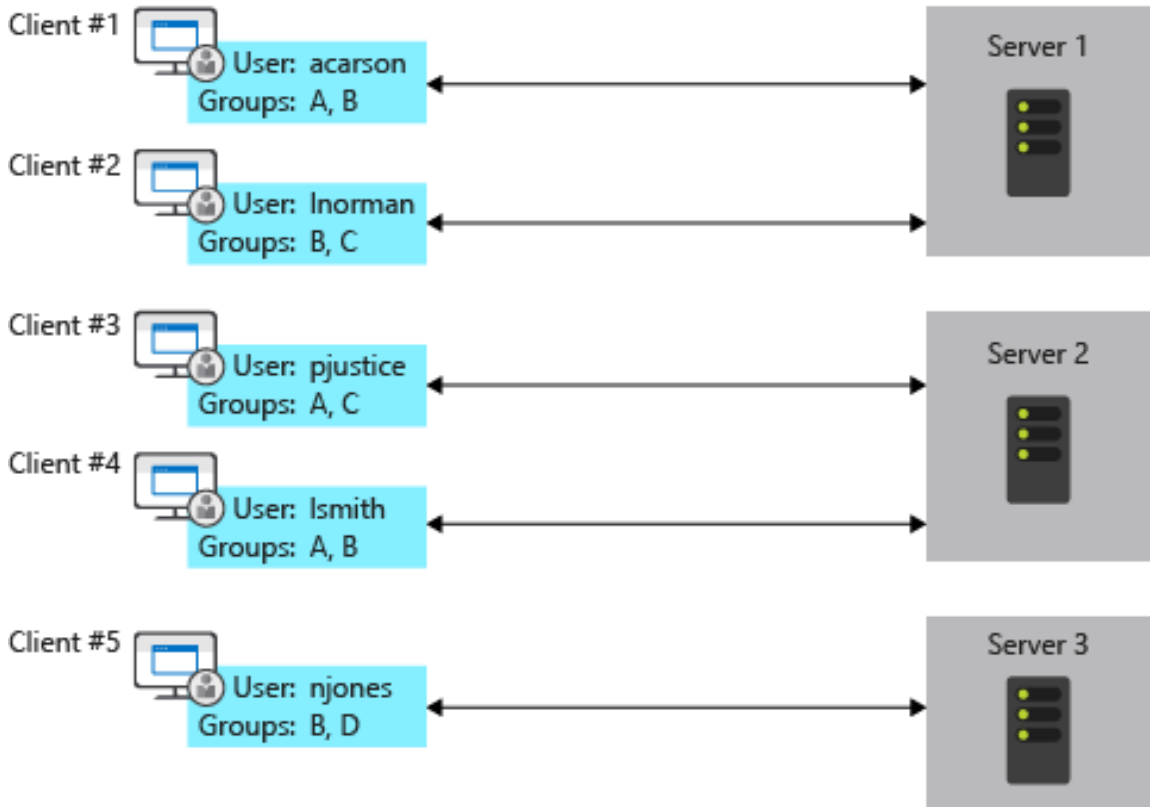
First issue? … Scale out

An app that uses SignalR needs to keep track of all its connections, which creates problems for a server farm.

Add a server, and it gets new connections that the other servers don't know about.

For example, SignalR on each server in the following diagram is unaware of the connections on the other servers.

When SignalR on one of the servers wants to send a message to all clients, the message only goes to the clients connected to that server.



Solution? … Load balancers and sticky session

When the app runs in the cloud scaling out is a matter of setting the number of servers you want to run.

A mechanism called a load balancer will then pick a server on each incoming request.

The load balancer can pick a different server in sequence or have some other logic going on to pick one.

Load balancers and sticky session

We can solve this problem by using sticky sessions.

There are several implementations of this but most of the time it works as follows.

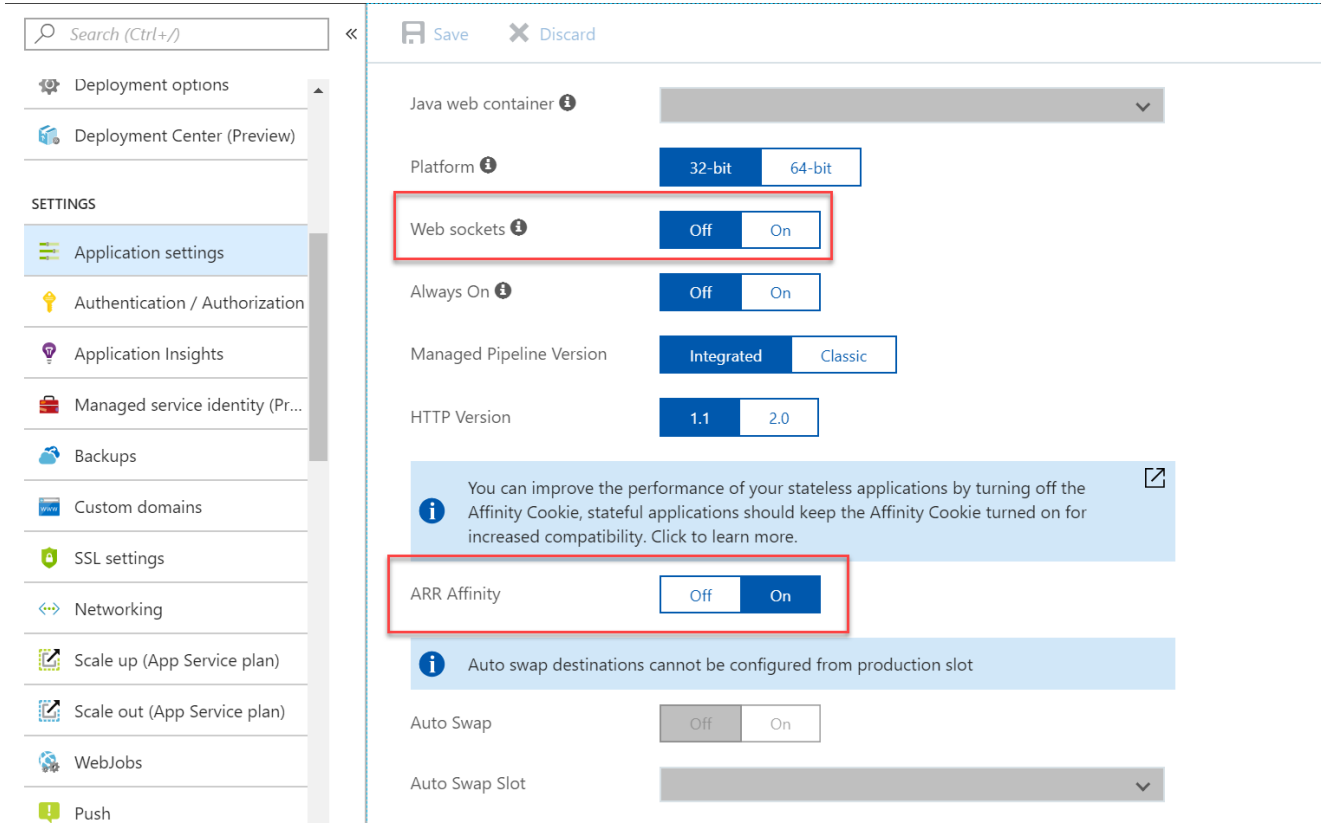
As part of the response of the first request the load balancer sets a cookie in the browser indicating the server that was used.

On subsequent requests the load balancer then reads the cookie and assigns the request to the same server.

Load balancers and sticky session

The IIS and Azure web apps version of sticky sessions is called **Application Request Routing Affinity** or **ARR Affinity**.

Since SignalR could use non-websocket transports you should turn this on on all servers where your application is on. When using an on-premise server with **IIS** install the **ARR Affinity module**.



… But there’s another problem.

Let’s say a user is working on a web document using Office 365 and she invites others to join her.

The other might end up at server. Now when the user on server 1 changes the document a message has to be sent to the others.

But server 1 doesn’t know about users that are connected to hubs in other servers.

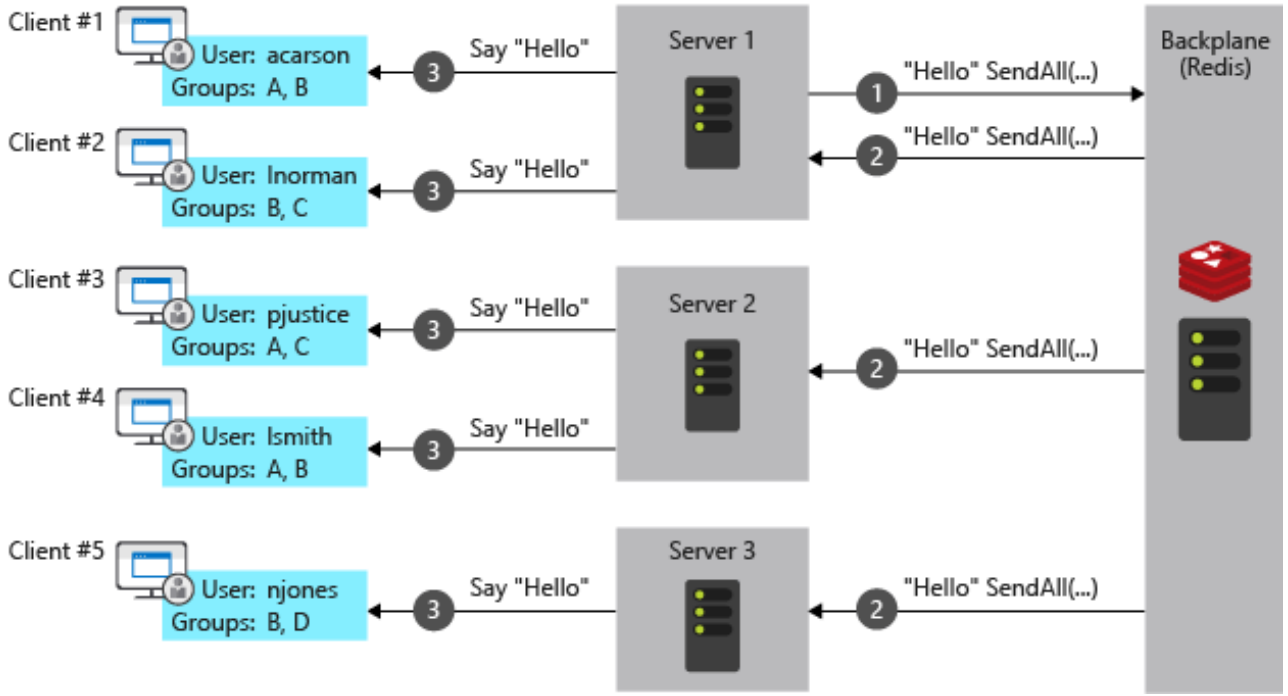
To solve this the servers need a way to **share data**.

Redis backplane

Redis is an in-memory key-value store that supports a messaging system with a publish/subscribe model. The SignalR Redis backplane uses the pub/sub feature to forward messages to other servers.

When a client makes a connection, the connection information is passed to the backplane. When a server wants to send a message to all clients, it sends to the backplane.

The backplane knows all connected clients and which servers they're on. It sends the message to all clients via their respective servers.



Azure SignalR Service

Azure SignalR Service

Azure SignalR Service is designed for large-scale real-time applications.

Azure SignalR Service allows multiple instances to work together to scale to millions of client connections. The service also supports multiple global regions for sharding, high availability, or disaster recovery purposes.

Azure SignalR Service

Azure SignalR Service is designed for large-scale real-time applications.

It is common to scale SignalR with SQL Server, Azure Service Bus, or Azure Cache for Redis. Azure SignalR Service handles the scaling approach for you.

The performance and cost is comparable to these approaches without the complexity of dealing with these other services.

All you have to do is update the unit count for your service. Each unit supports up to 1000 client connections.

Azure SignalR Service

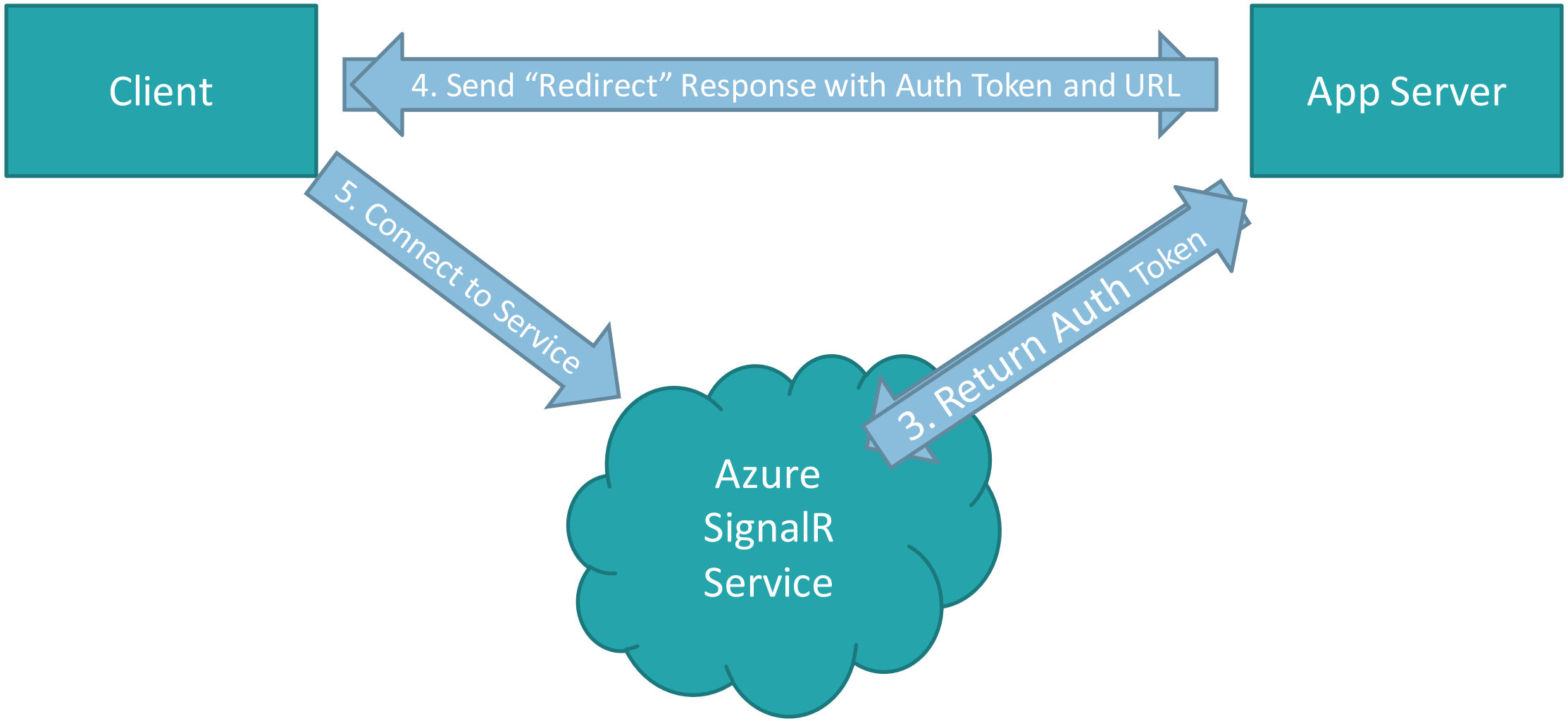
One of the key reasons to use the Azure SignalR Service is simplicity.

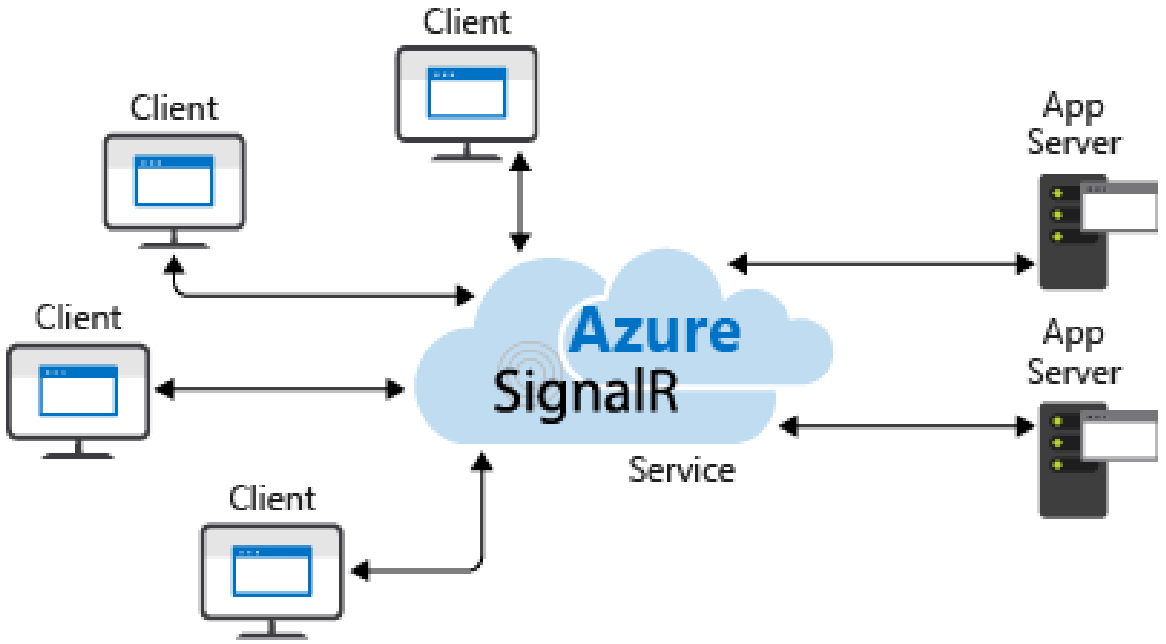
With Azure SignalR Service, you don't need to handle problems like performance, scalability, availability. These issues are handled for you with a 99.9% service-level agreement.

Usage of SignalR Service in Web App

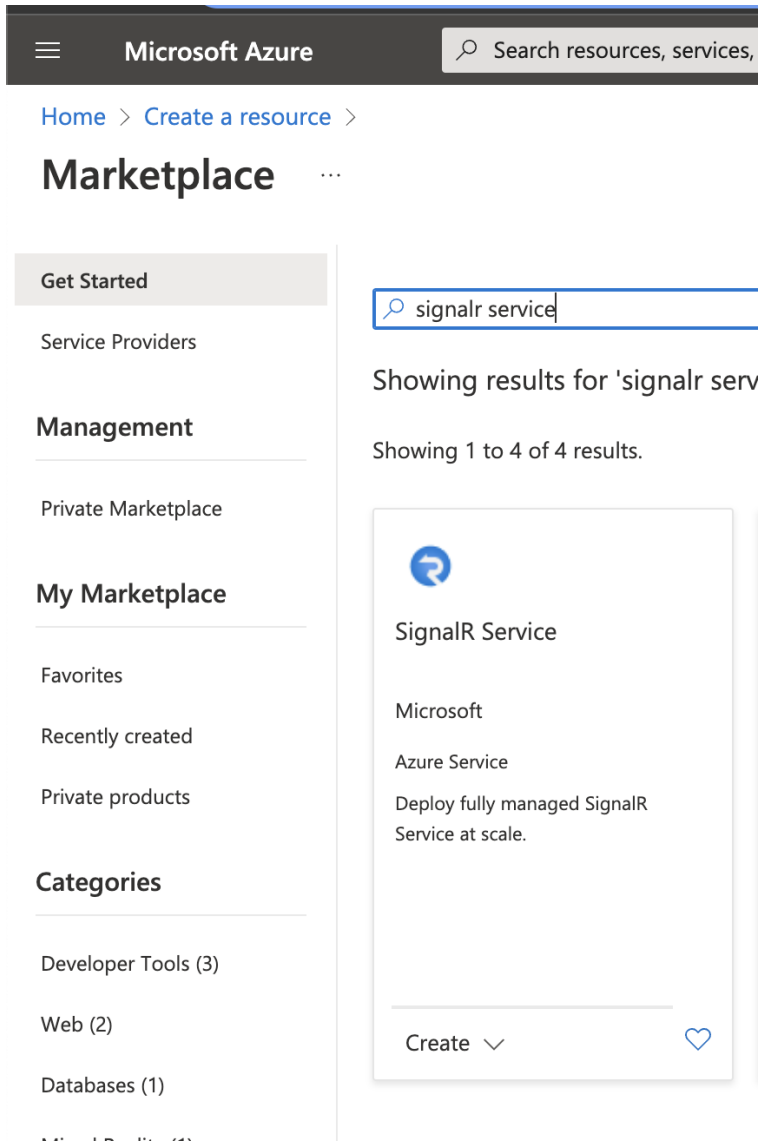
```
var builder = WebApplication.CreateBuilder(args);
builder.Services
      .AddSignalR()
      .AddAzureSignalR();

var app = builder.Build();

app.UseDefaultFiles();
app.UseRouting();
app.UseStaticFiles();
app.MapHub<ChatSampleHub>("/chat");
app.Run();
```

Azure SignalR Service

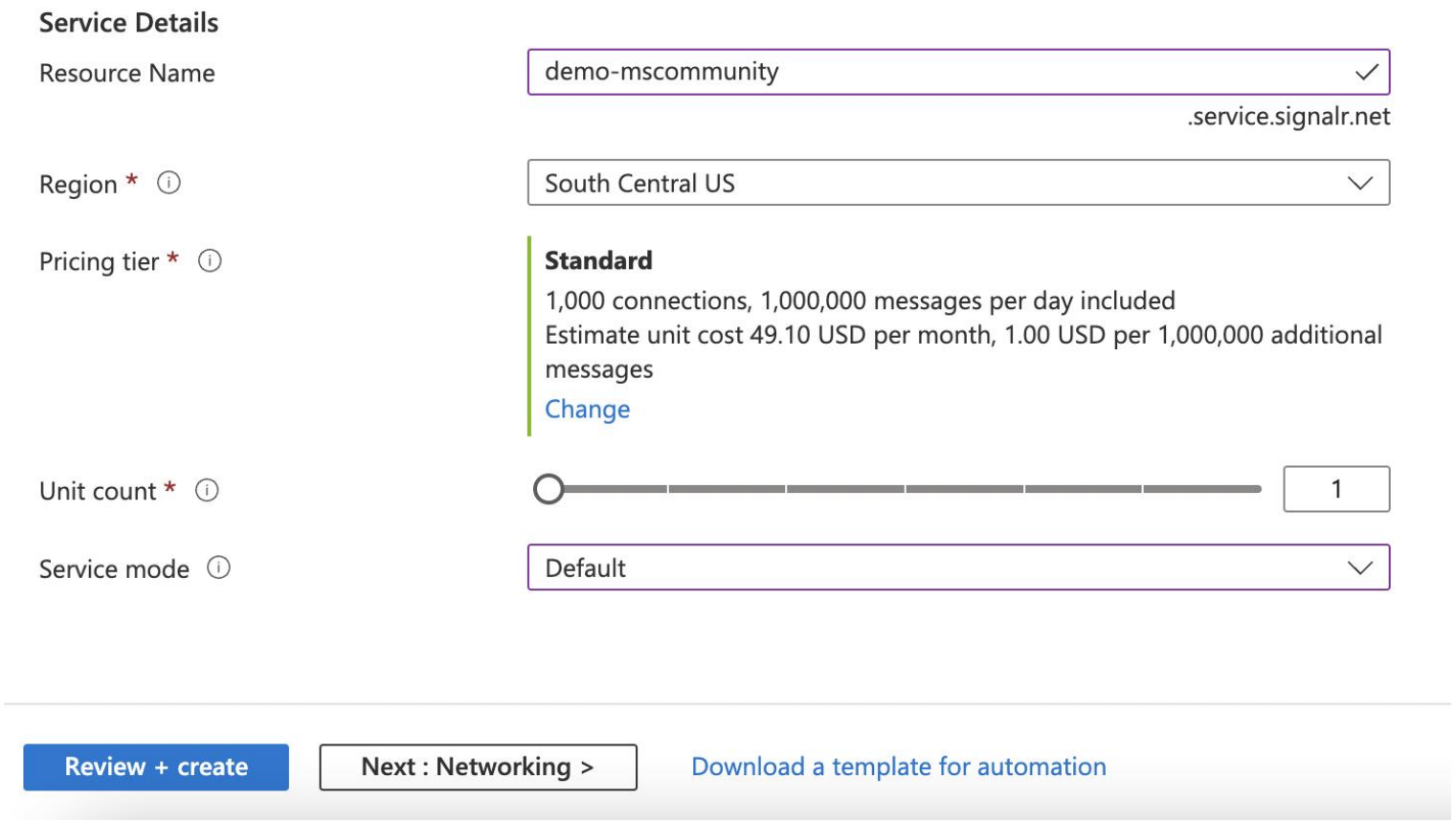



Client
Client
Client
Client
Client
App
Server
App
Server
Azure
SignalR
Service

Getting started with SignalR Service



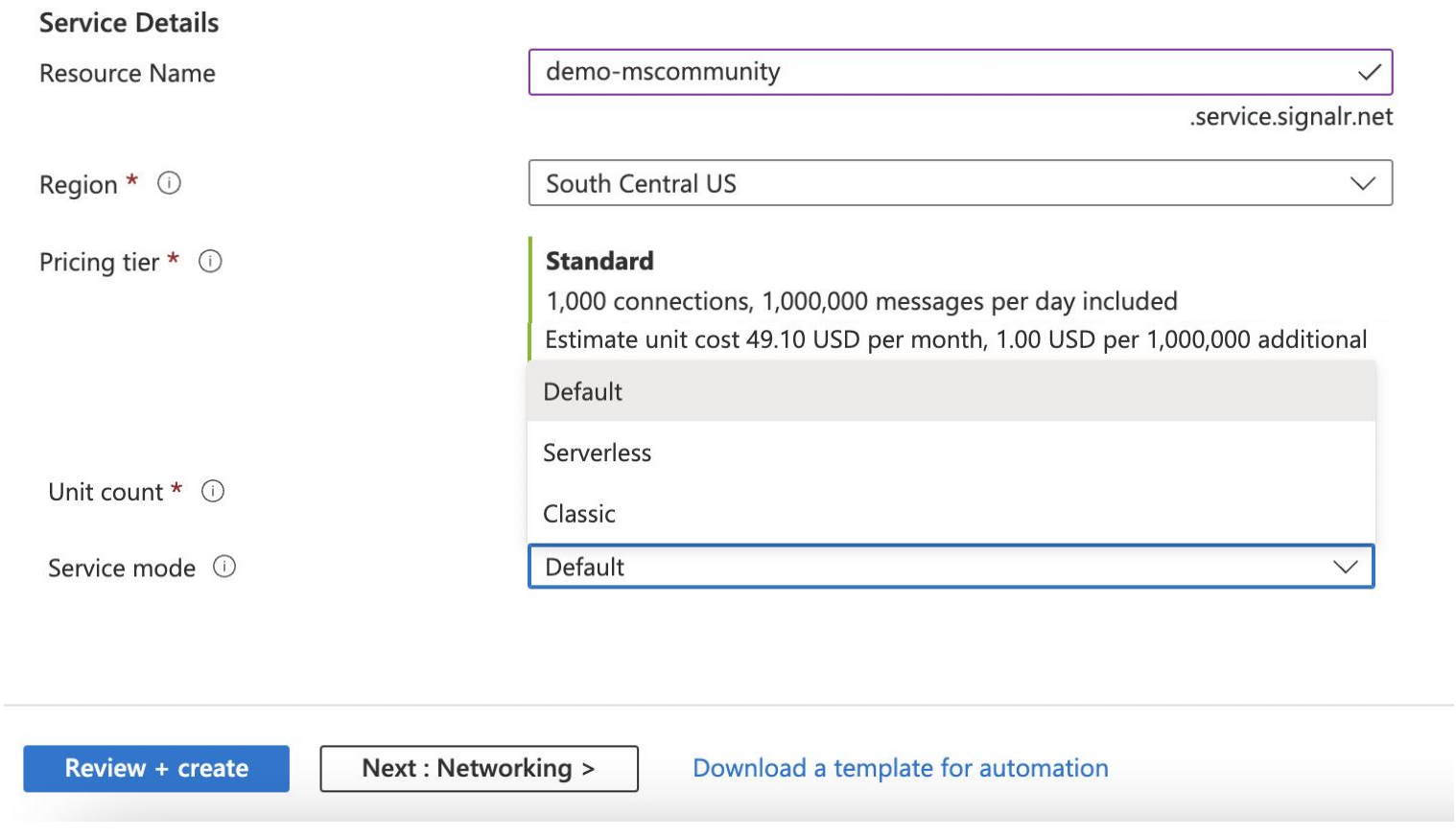
Getting started with SignalR Service



A unit is a sub-instance that processes your SignalR messages.

Units are used to increase the performance and connections count.

Getting started with SignalR Service



Service Mode

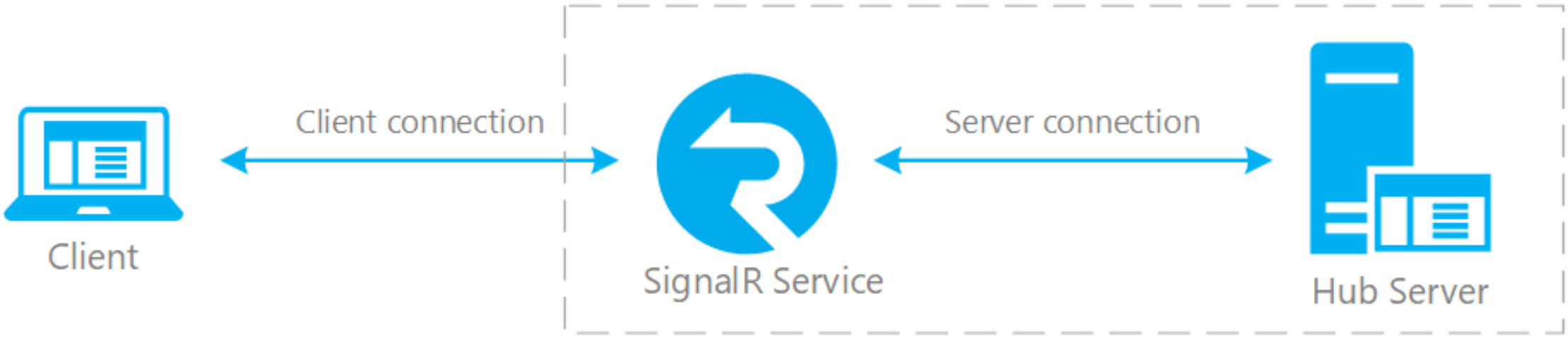
- Default
- Serverless

Service Mode - Default

- Default mode is the default value for service mode when you create a new SignalR resource. In this mode, your application works as a typical ASP.NET Core (or ASP.NET) SignalR application, where you have a web server that hosts a hub (called hub server hereinafter) and clients can have duplex real-time communication with the hub server.

  The only difference is instead of connecting client and server directly, client and server both connect to SignalR service and use the service as a proxy.

Service Mode - Default



Service Mode - Serverless (no upstream)

Serverless mode, as its name implies, is a mode that you cannot have any hub server.

Comparing to default mode, in this mode client doesn't require hub server to get connected.

All connections are connected to service in a "serverless" mode and service is responsible for maintaining client connections like handling client pings (in default mode this is handled by hub servers).

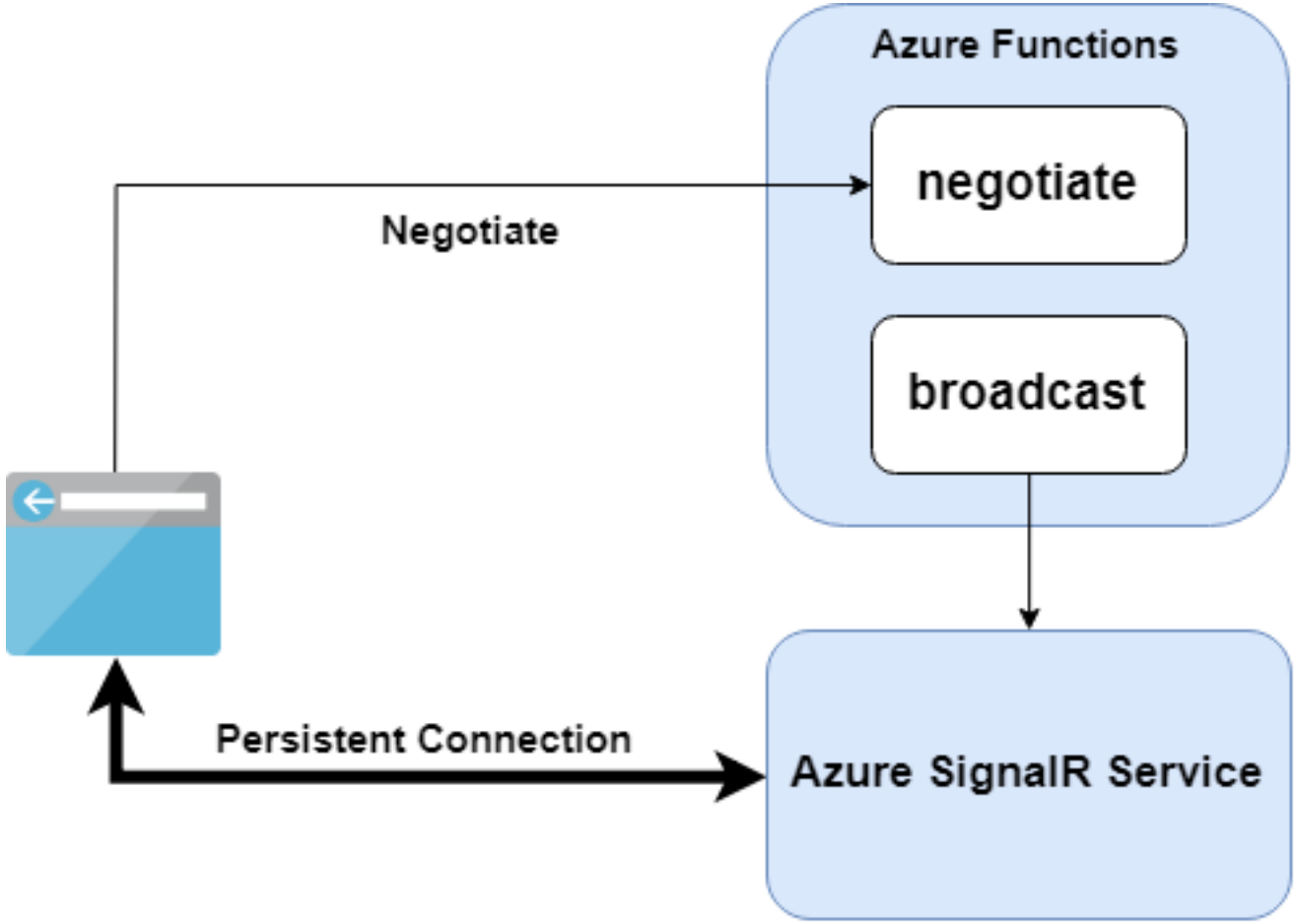
Service Mode - Serverless (no upstream)

Therefore there is also no connection routing and server-client stickiness.

The clients have persistent connections to Azure SignalR Service. Since there is no application server to handle traffic, clients are in **LISTEN** mode, which means they can only receive messages but can't send messages.

SignalR Service will disconnect any client who sends messages because it is an invalid operation

Service Mode - Serverless (no upstream)



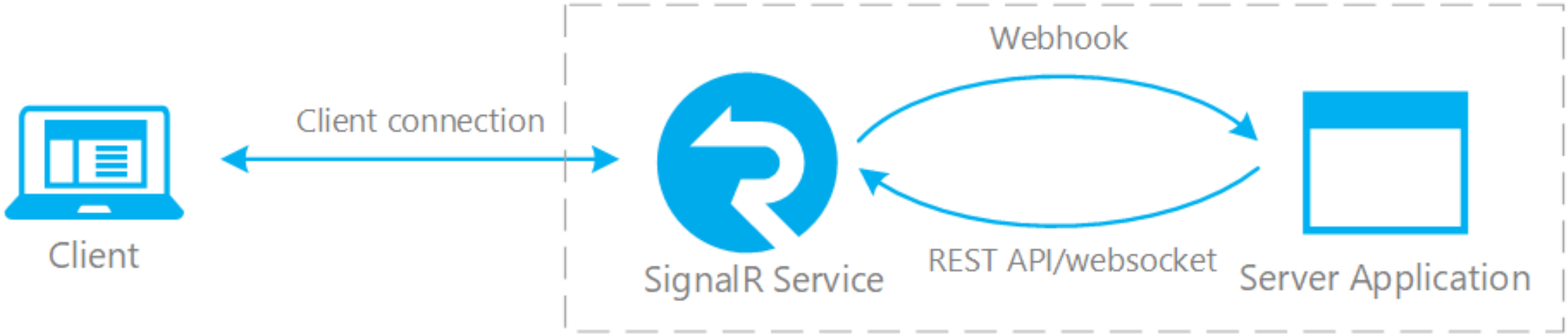
Service Mode - Serverless (upstream)

Therefore there is also no connection routing and server-client stickiness.

The clients have persistent connections to Azure SignalR Service. Since there is no application server to handle traffic, clients are in **LISTEN** mode, which means they can only receive messages but can't send messages.

SignalR Service will disconnect any client who sends messages because it is an invalid operation

Service Mode - Serverless



Can I send message from client in serverless mode?

- You can send message from client if you configure upstream in your SignalR instance. Upstream is a set of endpoints that can receive messages and connection events from SignalR service.

  If no upstream is configured, messages from client will be ignored.

- For more information about upstream, see Upstream settings.
- Upstream is currently in public preview.

Azure SignalR Service – Serverless (no upstream)

- Pros:
  - The client connects to the SignalR Serivce directly
- Cons:
  - The client can only receives messages, and can’t send message to the SignalR Service.

Azure SignalR Service – Serverless (upstream)

- Pros:
  - As the serverless mode, the client connection connects to the SignalR Service directly.
  - With the Upstream, the client can also send messages to the SignalR Service.
- Cons:
  - Compare with the default mode, the upstream server is used to receive the message, but not to store the client's status. The upstream server is stateless.

For homework

- http://dontcodetired.com/blog/post/Using-the-Azure-SignalR-Service-Bindings-in-Azure-Functions-to-Create-Real-time-Serverless-Applications

Thank you!

ευχαριστώ Salamat Po متشكرم شكراً Grazie

благодаря ありがとうございます Kiitos Teşekkürler 谢谢

ขอบคุณครับ Obrigado شكريه Terima Kasih Dziękuję

Hvala Köszönöm Tak Dank u wel дякую Tack

Mulţumesc спасибо Danke Cám ơn Gracias

多謝晒 Ďakujem תודה நன்றி Děkuji 감사합니다

Almir Vuk

almirvuk@outlook.com
almirvuk.com | @almirvuk

Thank you!

ευχαριστώ Salamat Po متشكرم شكراً Grazie

благодаря ありがとうございます Kiitos Teşekkürler 谢谢

ขอบคุณครับ Obrigado شكريه Terima Kasih Dziękuję