SGCWEBSOCKETS · TECHNICAL DOCUMENT

Multi-node Clustering

TsgcWSCluster — scale your sgcWebSockets servers across multiple nodes. A publish or Presence event on one node reaches subscribers connected to every other node, through a zero-infrastructure mesh backplane or a Redis Pub/Sub backplane.

eSeGeCe

WWW.ESEGECE.COM

Overview

Put two or more WebSocket servers behind a load balancer and a message published on one node still reaches subscribers connected to the others. `TsgcWSCluster` adds a pub-sub backplane between your nodes, so channels, broadcasts and Presence work cluster-wide. Drop the component next to your server, point the nodes at each other, and your existing `Publish`, channel and Presence code stays unchanged.

At a glance

COMPONENT CLASS
`TsgcWSCluster`

BACKPLANE
Mesh or Redis

BACKPLANE ENGINES
Mesh (zero-infrastructure) or Redis Pub/Sub

PLATFORMS
Delphi 7–13 (Win32/Win64, Linux64, macOS, Android, iOS) and .NET

FRAMEWORKS
VCL, FireMonkey, Lazarus / FPC, .NET

EDITION
Enterprise

Features

- **Two backplane engines.** Published property `EngineType`: `clusterMesh` (no external infrastructure) or `clusterRedis` (Redis Pub/Sub).
- **Mesh, zero-infrastructure.** Each node listens on `ClusterPort` and connects directly to the peers listed in `Peers` — nothing extra to install.
- **Redis Pub/Sub.** Properties `RedisHost`, `RedisPort` and `RedisChannel` route messages through Redis for larger deployments.
- **Attach / Detach.** Cluster the `sgc` protocol and the Presence protocol with one call each; `Start` and `Stop` control the backplane.
- **Cluster-wide Presence.** The member roster becomes the union across all nodes; a node that drops has its members purged, so there are no ghost members.

- **Readiness signals.** Property `Ready`, `ConnectedPeerCount` and events `OnPeerConnected`, `OnPeerDisconnected` and `OnClusterMessage` let you observe cluster health.
- **Loop prevention.** Every message carries an origin node id so it is never echoed back to the node that produced it.
- **Drop-in.** Your existing `Publish` and channel/subscription code is unchanged — clustering is transparent to the protocol layer.

Technical specification

Standards & specs	Redis Pub/Sub · WebSocket Protocol — RFC 6455
Component class	`TsgcWSCluster` (unit `sgcWebSocket_Cluster`)
Backplane engines	Mesh (zero-infrastructure) or Redis Pub/Sub
Frameworks	VCL, FireMonkey, Lazarus / FPC, .NET
Platforms	Delphi 7–13 (Win32/Win64, Linux64, macOS, Android, iOS) and .NET
Edition	Enterprise

Main properties

The principal published / public properties used to configure and drive the component. Consult the online help for the full list.

`EngineType`	Selects the backplane engine: `clusterMesh` for the zero-infrastructure peer mesh, or `clusterRedis` for a Redis Pub/Sub backplane.
`ClusterPort`	TCP port this node listens on for the mesh backplane; peers connect to this port to exchange cluster messages.
`Peers`	List of the other nodes in `host:port` form that this node connects to when the mesh engine is active.
`RedisHost`	Hostname or IP address of the Redis server used as the backplane when `EngineType` is `clusterRedis`.
`RedisPort`	TCP port of the Redis server used as the backplane.
`RedisChannel`	Redis Pub/Sub channel name that all cluster nodes publish to and subscribe from.
`Ready`	Indicates whether the cluster backplane is up and this node is ready to relay messages to and from its peers.
`ConnectedPeerCount`	Number of peer nodes currently connected to this node through the backplane.

Main methods

The principal public methods exposed by the component.

`Attach()`	Clusters a protocol — the `sgc` protocol or the Presence protocol — so its publish/subscribe traffic is shared across all nodes.
`Detach()`	Stops clustering a previously attached protocol, returning it to single-node behaviour.
`Start()`	Starts the cluster backplane: opens the mesh listener and connects to the configured peers, or connects to Redis.
`Stop()`	Stops the cluster backplane, closing peer connections (or the Redis connection) while leaving the WebSocket server running.

Public events

The component exposes the following published events; consult the online help for full event-handler signatures.

`OnPeerConnected`	Fires when a peer node connects to this node through the cluster backplane.
`OnPeerDisconnected`	Fires when a peer node disconnects from this node; its Presence members are purged from the union roster.
`OnClusterMessage`	Fires when a message arrives from another node over the backplane, so the application can observe inter-node traffic.

Quick Start

Drop the component next to your `TsgcWebSocketServer` and the `sgc` protocol, point the nodes at each other and start it. A client on one node can **Publish** to a channel and clients on the other nodes receive it, with no sticky sessions.

> **About this scenario.** Two or more nodes sit behind a load balancer. Each node runs a `TsgcWebSocketServer` with the `sgc` protocol attached to a `TsgcWSCluster` using the mesh backplane. Your existing Publish, channel and Presence code stays unchanged.

Delphi (VCL / FireMonkey)

```
uses
  sgcWebSocket, sgcWebSocket_Protocols, sgcWebSocket_Cluster;
var
  Server: TsgcWebSocketServer;
  Protocol: TsgcWSPServer_sgc;
  Cluster: TsgcWSCluster;
begin
  Server := TsgcWebSocketServer.Create(nil);
  Server.Port := 8080;

  Protocol := TsgcWSPServer_sgc.Create(nil);
  Protocol.Server := Server;

  Cluster := TsgcWSCluster.Create(nil);
  Cluster.EngineType  := clusterMesh;               // zero-infrastructure backplane
  Cluster.ClusterPort := 5410;                      // this node's mesh listener
  Cluster.Peers.Add('192.168.1.101:5410');          // the other nodes
  Cluster.Peers.Add('192.168.1.102:5410');
  Cluster.Attach(Protocol);                         // cluster this protocol's pub/sub
  Cluster.Start;

  Server.Active := True;
end;
```

C++ Builder

```
#include "sgcWebSocket.hpp"
#include "sgcWebSocket_Protocols.hpp"
#include "sgcWebSocket_Cluster.hpp"

TsgcWebSocketServer *Server = new TsgcWebSocketServer(NULL);
Server->Port = 8080;

TsgcWSPServer_sgc *Protocol = new TsgcWSPServer_sgc(NULL);
Protocol->Server = Server;

TsgcWSCluster *Cluster = new TsgcWSCluster(NULL);
Cluster->EngineType  = clusterMesh;                // zero-infrastructure backplane
Cluster->ClusterPort = 5410;                       // this node's mesh listener
Cluster->Peers->Add("192.168.1.101:5410");          // the other nodes
Cluster->Peers->Add("192.168.1.102:5410");
Cluster->Attach(Protocol);                          // cluster this protocol's pub/sub
Cluster->Start();

Server->Active = true;
```

.NET (C#)

```
var server = new TsgcWSServer { Port = 8080 };
var protocol = new TsgcWSPServer_sgc { Server = server };

var cluster = new TsgcWSCluster {
    EngineType  = ClusterEngineType.Mesh,   // zero-infrastructure backplane
    ClusterPort = 5414,
    Protocol    = protocol
};
cluster.Peers.Add("192.168.1.101:5414");
cluster.Peers.Add("192.168.1.102:5414");
cluster.Start();

server.Active = true;
```

Common scenarios

The following scenarios show the configuration and method calls needed to drive the component through a specific real-world flow.

1 · Redis Pub/Sub backplane

For larger deployments, switch the engine from mesh to Redis. Every node publishes to and subscribes from the same Redis channel, so you do not need to maintain a peer list.

Delphi (VCL / FireMonkey)

```
  Cluster := TsgcWSCluster.Create(nil);
  Cluster.EngineType   := clusterRedis;            // Redis Pub/Sub backplane
  Cluster.RedisHost    := '127.0.0.1';
  Cluster.RedisPort    := 6379;
  Cluster.RedisChannel := 'sgc-cluster';
  Cluster.Attach(Protocol);
  Cluster.Start;
```

C++ Builder

```
Cluster = new TsgcWSCluster(NULL);
Cluster->EngineType   = clusterRedis;            // Redis Pub/Sub backplane
Cluster->RedisHost    = "127.0.0.1";
Cluster->RedisPort    = 6379;
Cluster->RedisChannel = "sgc-cluster";
Cluster->Attach(Protocol);
Cluster->Start();
```

.NET (C#)

```
var cluster = new TsgcWSCluster {
    EngineType   = ClusterEngineType.Redis,      // Redis Pub/Sub backplane
    RedisHost    = "127.0.0.1",
    RedisPort    = 6379,
    RedisChannel = "sgc-cluster",
    Protocol     = protocol
};
cluster.Start();
```

2 · Cluster-wide Presence

Attach the Presence protocol to the cluster as well. The member roster becomes the union across all nodes, and when a node drops its members are purged so there are no ghost members.

Delphi (VCL / FireMonkey)

```
  Cluster.Attach(Protocol);          // the sgc protocol
  Cluster.Attach(Presence);          // the Presence protocol
  Cluster.Start;
  // Presence.Members now reflects every node in the cluster
```

C++ Builder

```
Cluster→Attach(Protocol);          // the sgc protocol
Cluster→Attach(Presence);          // the Presence protocol
Cluster→Start();
// Presence→Members now reflects every node in the cluster
```

.NET (C#)

```
cluster.Attach(protocol);          // the sgc protocol
cluster.Attach(presence);          // the Presence protocol
cluster.Start();
// presence.Members now reflects every node in the cluster
```

3 · Readiness signals

Watch the backplane come up and observe peer connections through the readiness property and events.

Delphi (VCL / FireMonkey)

```
procedure OnPeerConnected(Sender: TObject; const aPeer: string);
begin
  // a peer node joined the cluster
  ShowStatus('Peers: ' + IntToStr(Cluster.ConnectedPeerCount));
end;

if Cluster.Ready then
  // the backplane is up and relaying messages;
```

C++ Builder

```
void __fastcall OnPeerConnected(TObject *Sender, const String aPeer)
{
  // a peer node joined the cluster
  ShowStatus("Peers: " + IntToStr(Cluster->ConnectedPeerCount));
}

if (Cluster->Ready)
{
  // the backplane is up and relaying messages
}
```

.NET (C#)

```
cluster.OnPeerConnected += (sender, peer) =>
{
    // a peer node joined the cluster
    ShowStatus($"Peers: {cluster.ConnectedPeerCount}");
};

if (cluster.Ready)
{
    // the backplane is up and relaying messages
}
```

Sources used to build this document

Every external claim links back to a primary source. The online-help references decode the canonical deep-link the company maintains for this component.

Primary standard / spec — Redis Pub/Sub	redis.io/docs/latest/develop/interact/pubsub
Primary standard / spec — WebSocket Protocol — RFC 6455	datatracker.ietf.org/doc/html/rfc6455
Blog — how the backplane works	www.esegece.com/community/blog/sgcwebsockets-clustering
Delphi demo project (in the sgcWebSockets package)	`Demos\02.WebSocket_Protocols\14.MultiNode_Clustering`
Component page	www.esegece.com/products/websockets/components/scaling/cluster/
Product page	www.esegece.com/products/websockets/

> **Document scope.** This document covers the publicly-documented surface of the Multi-node Clustering component (TsgcWSCluster) shipped with sgcWebSockets. For full property, method and event reference consult the online help linked above.