Consolidated “Active Hostname in DB” Approach (ServiceName = NVARCHAR(100))

# 1) Intent & Outcome

Ensure only one instance of each legacy .NET Framework Windows Service executes work in an active-active server setup, without heavy code changes. We use a small DB-backed leasing mechanism that records the active hostname and enforces single-active execution.

# 2) Schema & Stored Procedures

Use NVARCHAR(100) for ServiceName (app-level identifier), not sysname.

## 2.1 Table

```sql  
CREATE TABLE dbo.ServiceLease (  
 ServiceName nvarchar(100) NOT NULL PRIMARY KEY,  
 ActiveHostName nvarchar(255) NULL,  
 LeaseId uniqueidentifier NULL,  
 LeaseExpiresUtc datetime2(3) NULL,  
 LastHeartbeatUtc datetime2(3) NULL,  
 RowVersion rowversion NOT NULL  
);  
```

## 2.2 Acquire (Create or Steal Expired Lease)

```sql  
CREATE OR ALTER PROCEDURE dbo.ServiceLease\_Acquire  
 @ServiceName nvarchar(100),  
 @HostName nvarchar(255),  
 @NowUtc datetime2(3),  
 @LeaseSeconds int,  
 @LeaseId uniqueidentifier OUTPUT  
AS  
BEGIN  
 SET NOCOUNT ON;  
 SET XACT\_ABORT ON;  
 DECLARE @newId uniqueidentifier = NEWID();  
  
 BEGIN TRAN;  
  
 IF EXISTS (SELECT 1 FROM dbo.ServiceLease WITH (UPDLOCK, HOLDLOCK)  
 WHERE ServiceName = @ServiceName)  
 BEGIN  
 UPDATE dbo.ServiceLease  
 SET ActiveHostName = CASE WHEN LeaseExpiresUtc <= @NowUtc THEN @HostName ELSE ActiveHostName END,  
 LeaseId = CASE WHEN LeaseExpiresUtc <= @NowUtc THEN @newId ELSE LeaseId END,  
 LeaseExpiresUtc= CASE WHEN LeaseExpiresUtc <= @NowUtc THEN DATEADD(SECOND, @LeaseSeconds, @NowUtc) ELSE LeaseExpiresUtc END,  
 LastHeartbeatUtc = CASE WHEN LeaseExpiresUtc <= @NowUtc THEN @NowUtc ELSE LastHeartbeatUtc END  
 WHERE ServiceName = @ServiceName  
 AND LeaseExpiresUtc <= @NowUtc;  
 IF @@ROWCOUNT = 1  
 BEGIN  
 SET @LeaseId = @newId; COMMIT; RETURN 0;  
 END  
 END  
 ELSE  
 BEGIN  
 INSERT dbo.ServiceLease(ServiceName, ActiveHostName, LeaseId, LeaseExpiresUtc, LastHeartbeatUtc)  
 VALUES (@ServiceName, @HostName, @newId, DATEADD(SECOND, @LeaseSeconds, @NowUtc), @NowUtc);  
 SET @LeaseId = @newId; COMMIT; RETURN 0;  
 END  
  
 ROLLBACK; RETURN 1;  
END  
```

## 2.3 Heartbeat (Renew if Still Owner)

```sql  
CREATE OR ALTER PROCEDURE dbo.ServiceLease\_Heartbeat  
 @ServiceName nvarchar(100),  
 @LeaseId uniqueidentifier,  
 @NowUtc datetime2(3),  
 @LeaseSeconds int  
AS  
BEGIN  
 SET NOCOUNT ON;  
 UPDATE dbo.ServiceLease  
 SET LastHeartbeatUtc = @NowUtc,  
 LeaseExpiresUtc = DATEADD(SECOND, @LeaseSeconds, @NowUtc)  
 WHERE ServiceName = @ServiceName  
 AND LeaseId = @LeaseId;  
 RETURN CASE WHEN @@ROWCOUNT = 1 THEN 0 ELSE 1 END;  
END  
```

## 2.4 Release (Graceful Stop)

```sql  
CREATE OR ALTER PROCEDURE dbo.ServiceLease\_Release  
 @ServiceName nvarchar(100),  
 @LeaseId uniqueidentifier  
AS  
BEGIN  
 SET NOCOUNT ON;  
 UPDATE dbo.ServiceLease  
 SET LeaseExpiresUtc = DATEADD(SECOND, -1, SYSUTCDATETIME())  
 WHERE ServiceName = @ServiceName  
 AND LeaseId = @LeaseId;  
END  
```

# 3) Service-Side Drop‑in Library (C#)

```csharp  
using System;  
using System.Data.SqlClient;  
using System.Threading;  
using System.Threading.Tasks;  
  
public sealed class LeaseGate : IDisposable  
{  
 private volatile bool \_isLeader;  
 private string \_serviceName;  
 private SqlConnection \_conn;  
 private System.Threading.Timer \_timer;  
 private Guid? \_leaseId;  
  
 public bool IsLeader => Volatile.Read(ref \_isLeader);  
 public string LeaseId => \_leaseId?.ToString();  
  
 public async Task StartAsync(string serviceName,  
 string connectionString,  
 TimeSpan heartbeatPeriod,  
 TimeSpan leaseTtl)  
 {  
 \_serviceName = serviceName;  
 \_conn = new SqlConnection(connectionString);  
 await \_conn.OpenAsync();  
  
 \_timer = new System.Threading.Timer(async \_ => await Tick(), null, TimeSpan.Zero, heartbeatPeriod);  
  
 async Task Tick()  
 {  
 try  
 {  
 if (\_leaseId == null)  
 {  
 var result = await SqlAcquire(\_serviceName, Environment.MachineName, leaseTtl);  
 if (result.ok) { \_leaseId = result.id; \_isLeader = true; }  
 else { \_isLeader = false; }  
 }  
 else  
 {  
 var renewed = await SqlHeartbeat(\_serviceName, \_leaseId.Value, leaseTtl);  
 if (!renewed) { \_leaseId = null; \_isLeader = false; }  
 }  
 }  
 catch  
 {  
 \_isLeader = false; \_leaseId = null; // fail-safe  
 }  
 }  
 }  
  
 public async Task StopAsync()  
 {  
 if (\_leaseId != null)  
 await SqlRelease(\_serviceName, \_leaseId.Value);  
 \_timer?.Dispose();  
 \_conn?.Dispose();  
 }  
  
 public void Dispose() => \_ = StopAsync();  
  
 // --- SQL helpers ---  
 private async Task<(bool ok, Guid id)> SqlAcquire(string service, string host, TimeSpan ttl)  
 {  
 using (var cmd = \_conn.CreateCommand())  
 {  
 cmd.CommandText = "dbo.ServiceLease\_Acquire";  
 cmd.CommandType = System.Data.CommandType.StoredProcedure;  
 cmd.Parameters.AddWithValue("@ServiceName", service);  
 cmd.Parameters.AddWithValue("@HostName", host);  
 cmd.Parameters.AddWithValue("@NowUtc", DateTime.UtcNow);  
 cmd.Parameters.AddWithValue("@LeaseSeconds", (int)ttl.TotalSeconds);  
 var leaseIdParam = cmd.Parameters.Add("@LeaseId", System.Data.SqlDbType.UniqueIdentifier);  
 leaseIdParam.Direction = System.Data.ParameterDirection.Output;  
 var rc = await cmd.ExecuteNonQueryAsync();  
 // When using RETURN code pattern, switch to ExecuteScalar/parameter capture as needed.  
 var id = (Guid)leaseIdParam.Value;  
 return (id != Guid.Empty, id);  
 }  
 }  
  
 private async Task<bool> SqlHeartbeat(string service, Guid leaseId, TimeSpan ttl)  
 {  
 using (var cmd = \_conn.CreateCommand())  
 {  
 cmd.CommandText = "dbo.ServiceLease\_Heartbeat";  
 cmd.CommandType = System.Data.CommandType.StoredProcedure;  
 cmd.Parameters.AddWithValue("@ServiceName", service);  
 cmd.Parameters.AddWithValue("@LeaseId", leaseId);  
 cmd.Parameters.AddWithValue("@NowUtc", DateTime.UtcNow);  
 cmd.Parameters.AddWithValue("@LeaseSeconds", 45);  
 var rc = await cmd.ExecuteNonQueryAsync();  
 return rc >= 0; // adjust to check RETURN if you wire it  
 }  
 }  
  
 private async Task SqlRelease(string service, Guid leaseId)  
 {  
 using (var cmd = \_conn.CreateCommand())  
 {  
 cmd.CommandText = "dbo.ServiceLease\_Release";  
 cmd.CommandType = System.Data.CommandType.StoredProcedure;  
 cmd.Parameters.AddWithValue("@ServiceName", service);  
 cmd.Parameters.AddWithValue("@LeaseId", leaseId);  
 await cmd.ExecuteNonQueryAsync();  
 }  
 }  
}  
```

## Integration Example

```csharp  
private LeaseGate \_lease;  
private CancellationTokenSource \_workerCts;  
  
protected override void OnStart(string[] args)  
{  
 \_lease = new LeaseGate();  
 \_lease.StartAsync(  
 serviceName: "Country.FileIn",  
 connectionString: System.Configuration.ConfigurationManager.ConnectionStrings["LeaseDb"].ConnectionString,  
 heartbeatPeriod: TimeSpan.FromSeconds(10),  
 leaseTtl: TimeSpan.FromSeconds(45)  
 ).GetAwaiter().GetResult();  
  
 \_workerCts = new CancellationTokenSource();  
 \_ = Task.Run(() => WorkerLoop(\_workerCts.Token));  
}  
  
private async Task WorkerLoop(CancellationToken ct)  
{  
 while (!ct.IsCancellationRequested)  
 {  
 if (\_lease.IsLeader)  
 {  
 await ProcessNextBatchAsync(ct);  
 }  
 else  
 {  
 await Task.Delay(TimeSpan.FromSeconds(5), ct);  
 }  
 }  
}  
  
protected override void OnStop()  
{  
 \_workerCts.Cancel();  
 \_lease?.StopAsync().GetAwaiter().GetResult();  
}  
```

# 4) Rollout Steps

1. Assign stable ServiceName values (≤100 chars) per Windows Service.  
2. Deploy table & procs. Grant EXEC to service account.  
3. Add the LeaseGate library to each service; wire Start/Stop and guard the worker loop with IsLeader.  
4. Configure: Heartbeat=10s, TTL=45s (tune as needed).  
5. Test: run two nodes; kill leader; observe takeover within TTL.  
6. Ops playbook: manual failover by expiring LeaseExpiresUtc or calling Release.

# 5) Notes & Guardrails

• Split-brain protection via row locks and TTL. Use UTC timestamps from SQL Server.  
• DB outage ⇒ instances self-demote (safe). Processing resumes when DB returns.  
• Prefer idempotent processing at item level for retries.