عنوان: **استفاده از درایوها در Window Azure Storage جهت استفاده در RavenDB**
نویسنده: سام ناصری
تاریخ: ۱۳۹۲/۰۱/۲۶ ۲:۵
آدرس: www.dotnettips.info
برچسب‌ها: RavenDB

در تلاش برای راه اندازی دیتابیس RavenDB بر روی Windows Azure چند مقاله ای خوندم که گاهی خیلی گیج کننده بود. الان تقریباً به نتایجی رسیده‌ام و دوست دارم در این مقاله نکاتی رو که به نظرم دانستن آنها بایسته است را مطرح کنم. باشد که مفید واقع شود.

پیش زمینه 1، یکی دیگر از روشهای راه اندازی RavenDB:

راه اندازی سرویس، نصب بر روی IIS و استفاده به صورت توکار، روش‌هایی هستند که در خود مستندات نچندان کامل RavenDB در حال حاضر مطرح شده است. راه دیگری که برای راه اندازی RavenDB می‌تواند مورد استفاده قرار گیرد، از طریق برنامه نویسی است. یعنی سرور RavenDB را با اجرای کد بالا می‌آوریم. نگران نباشید، این کار خیلی سخت نیست و به سادگی از طریق نمونه سازی از کلاس HttpServer و ارائه پارامترهای پیکره‌بندی و فراخوانی یک و یا دو متود می‌تواند صورت گیرد. مزیت این روش در پویایی و انعطاف پذیری آن است. شما می‌توانید هر تعداد سرور را با هر پیکره‌بندی پویایی، بالا بیاورید.

به کلمه HttpServer خوب دقت کنید. بله، درست است؛ این یک سرور کامل است و تمام درخواست‌های Http را طبق قواعد RavenDB و البته HTTP پاسخ می‌دهد. حتی studio ی RavenDB, که یک برنامه Silverlight است, نیز سرو میشود. (برنامه Silverlight در ریسورسهای RavenDB.Database.dll توکار(embed) شده است.)

کد مینیمالیست نمونه، یک RavenDB http server در قالب یک برنامه Console Application:

```
static void Main(string[] args)
{
    var configuration = new Raven.Database.Config.RavenConfiguration() {
        AccessControlAllowMethods = "All",
        AnonymousUserAccessMode = Raven.Database.Server.AnonymousUserAccessMode.All,
        DataDirectory = @"C:\Sam\labs\HttpServerData",
        Port = 8071,
    };
    var database = new Raven.Database.DocumentDatabase(configuration);
    var server = new Raven.Database.Server.HttpServer(configuration, database);
    database.SpinBackgroundWorkers();
    server.StartListening();

    Console.WriteLine("RavenDB http server is running ...");
    Console.ReadLine();
}
```

با اجرای برنامه فوق، پایگاه داده شما در پورت 8071 ماشین، فعال است و آماده پاسخگویی. استودیوی RavenDB نیز از طریق مسیر http://127.0.0.1:8071 قابل دسترسی است.

چرا این مطلب را گفتم، چون برای راه اندازی RavenDB در Azure می‌خواهیم از این روش استفاده کنیم. در یک worker role دیگر ما نه IIS داریم و نه یک virtual machine در اختیار داریم تا یک service را بر روی آن نصب کنیم. پس بهترین گزینه برای ما راه اندازی سرور RavenDB از طریق برنامه نویسی است.

پیش زمینه 2، چندساکنی در RavenDB و مسیر داده ها:(Multi Tenancy)

یک سرور RavenDB می‌تواند چندین پایگاه داده را میزبانی کند. هر چند به طور پیش فرض تک ساکنی برگزیده شده است. اما شما می‌توانید پایگاه‌های داده جدید را به سیستم اضافه کنید. مشکلی که من با مستندات RavenDB دارم این است که به طور پیش فرض درباره زمانی مصداق پیدا می‌کنند که RavenDB در حالت تک ساکنی مورد استفاده قرار میگیرد.

مهم است که بدانید مسیری که به عنوان مسیر داده‌ها در هنگام راه اندازی سرور ارائه می‌دهید برای پایگاه داده پیش فرض مورد استفاده قرار میگیرد و باید مسیرهای جداگانه مستقلی برای پایگاه داده‌های بعدی تنظیم کنید.

توجه داشته باشید که در RavenDB اگر در هنگام ساخت پایگاه داده، مسیری را مطرح نکنید، مسیر پیش فرض انتخاب خواهد شد. همچنین در حالت چندساکنی هم هیچ ارتباطی بین پایگاه‌های داده بعدی با پایگاه داده <system> وجود ندارد و همواره مسیر پیش

فرض به صورت ~/Databases/dbName خواهد بود که dbName نام پایگاه داده مورد نظر شما است. مهم است که بدانید که ~ در مسیر فوق دارای تعریف رسمی ای نیست و آنچه از کد بر می‌آید ~ مسیر BaseDirectory برای AppDomain جاری است. پس با توجه اینکه نوع برنامه میزبان سرور چیست (IIS, Windows Service, Worker Role) مقدار آن می‌تواند متفاوت باشد.

تعریف Worker Role برای RavenDB

در واقع مطلب اصلی درباره نحوه استفاده از CloudDrive در Web Role یا Worker Role است. همانطور که میدانید Web Role و Worker Role هر دو برای ذخیره سازی داده‌ها مناسب نیستند. در واقع بایستی با این رویکرد به آنها نگاه کنید که فقط کدهای اجرایی بر روی آنها قرار بگیرند و نه چیز دیگری. در مورد استفاده پایگاه داده RavenDB در Windows Azure می‌توانید آن را به صورت یک Worker Role تعریف کنید. اما برای اینکه داده‌ها را ذخیره کنید بایستی از یک Cloud Drive استفاده کنید.

خوب، در ابتد لازم است که کمی درباره‌ی CloudDrive بدانیم؛ خواندن این مطلب درباره‌ی اولین انتشار Windows Azure Drive خالی از لطف نیست.
حالا برای اینکه RavenDB را راه بیاندازیم باید نخست Wroker Role را بسازیم و سپس قطعه کدی بنویسیم تا درایو مجزا و مختصی را برای اینکه RavenDB اطلاعات را در آن بریزد بسازد. در آخر باید Worker Role را تنظیم کنیم تا درایو ساخته شده را در خود mount کند.

برای ساختن درایو قطعه کد زیر آن را انجام میدهد:

```
CloudStorageAccount storageAccount = CloudStorageAccount.FromConfigurationSetting(connectionString);
// here is when later on you may add code for inititalizing CloudDrive chache
CloudBlobClient blobClient = storageAccount.CreateCloudBlobClient();
blobClient.GetContainerReference("drives").CreateIfNotExist();

CloudDrive cloudDrive = storageAccount.CreateCloudDrive(
    blobClient
    .GetContainerReference("drives")
    .GetPageBlobReference("ravendb4.vhd")
    .Uri.ToString()
);

try
{
    // create a 1GB Virtual Hard Drive
    cloudDrive.Create(1024);
}
catch (CloudDriveException /*ex*/ )
{
    // the most likely exception here is ERROR_BLOB_ALREADY_EXISTS
    // exception is also thrown if the drive already exists
}
```

در کد فوق نامهای drives و ravendb.vhd کاملاً اختیاری هستند. اما باید از قواعد نامگذاری container پیروی کنند.
برای سوار کردن درایو قطعه کد زیر آن را انجام میدهد:

```
string driveLetter = cloudDrive.Mount(25, DriveMountOptions.Force);
```

توجه داشته باشید که کد سوار کردن درایو، قاعدتاً، بایستی در Worker Role صورت بگیرد و همچنین باید قبل از راه اندازی RavenDB باشد.
این یک ایراد طراحی Windows Azure است که شما نمیتوانید حرف درایو را خودتان انتخاب کنید، بلکه خروجی متود Mount مشخص میکند که درایو در چه حرف درایوی سوار شده است. و شما محدود هستند که کدهای خود را به گونه ای بنویسید که مسیر ذخیره سازی اطلاعات در Cloud Drive را ثابت فرض نکند و ارجاعات به این مسیرها شامل حرف درایو نباشد.

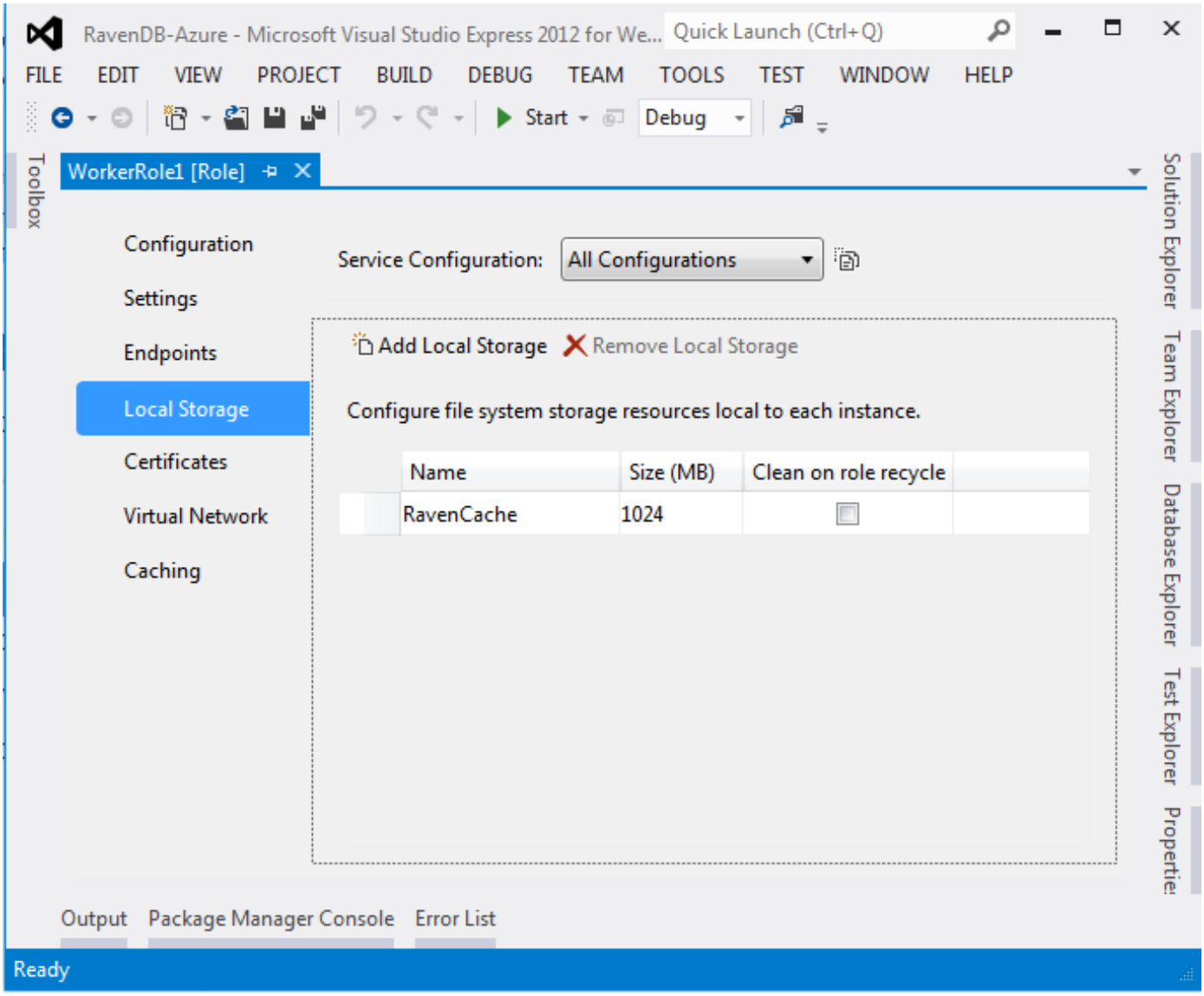
رفع مشکل کندی درایو در Windows Azure با تعریف کش:

کد فوق برای راه اندازی درایو مورد نظر ما کافی است. اما هنوز دارای یک مشکل اساسی و مهم است و آن اینست که بسیار کند عمل خواهد کرد.

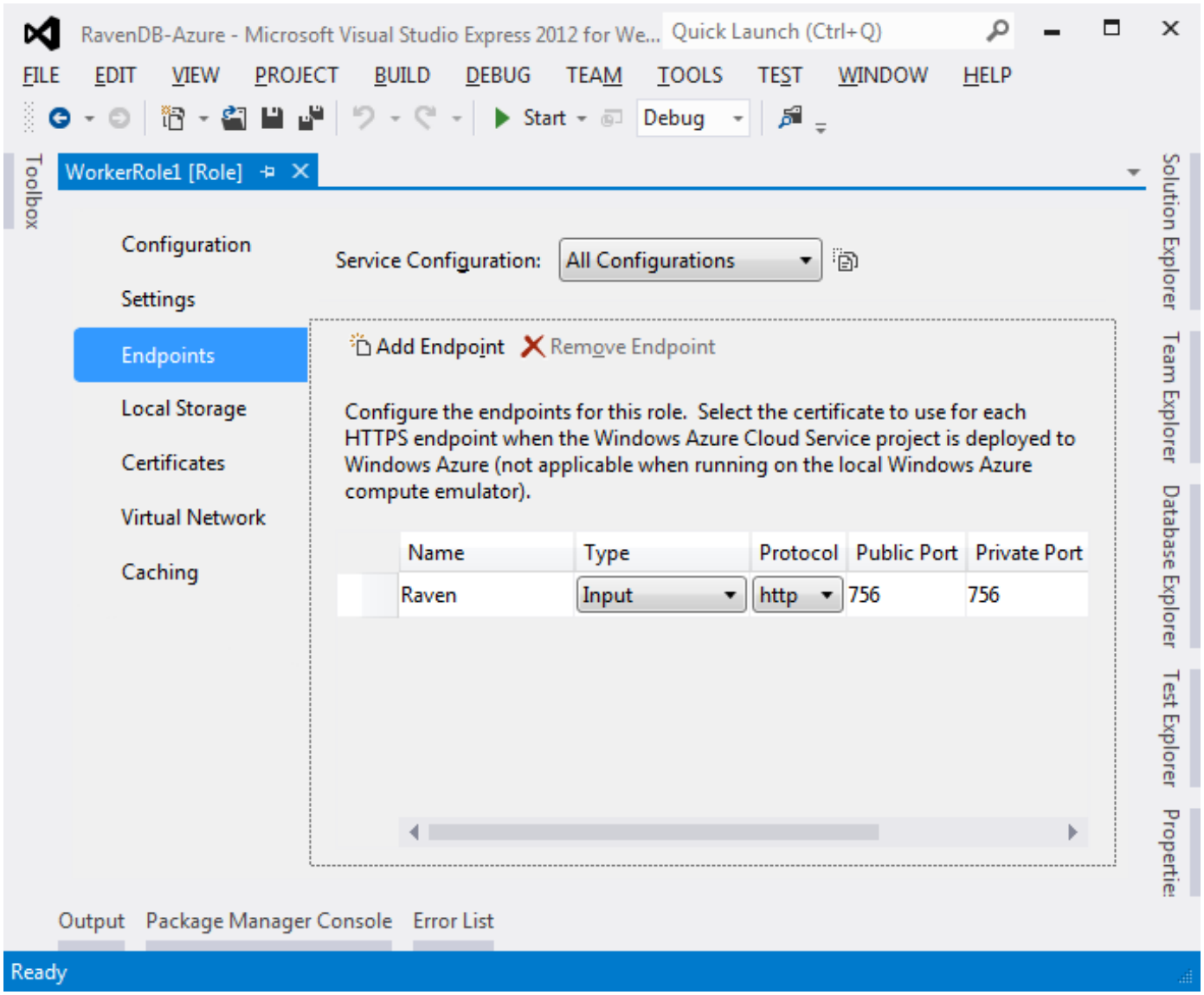
با فراخوانی متود CloudDrive.InitializeCache این متود به طور اتوماتیک برای تمام درایوهای mount شده یک کش محلی فراهم میکند و در نتیجه network I/O کمتری صورت خواهد گرفت. توجه داشته باشید که در صورت استفاده از این متود بایستی کش را برای Worker Role تعریف کنید. در صورت عدم استفاده از این متود کارائی پایگاه داده شما به شدت افت میکند. کد زیر را قبل از تعریف هر نوع درایوی قرار دهید.

```
LocalResource localCache = RoleEnvironment.GetLocalResource("RavenCache");
CloudDrive.InitializeCache(localCache.RootPath, localCache.MaximumSizeInMegabytes);
```

در کد فوق RavenCache نام یک Local Storage است که شما در تنظیمات Worker Role تعریف میکنید.(نام آن اختیاری است.) برای تعریف Local Storage بایستی در قسمت تنظیمات Worker Role رفته و آنگاه زبانه Local Storage رفته و سپس یک Local Storage را به مانند تصویر زیر اضافه کنید. نام که میتواند هر نامی باشد. اندازه را به اندازه مجموع درایوهایی که میخواهید در Worker Role تعریف کنید قرار دهید(در مثال برنامه ما در اینجا مقدار 1024) و گزینه Clean on role recycle را آنتیک کنید.



حال که درایو مورد نیاز ما آماده است قدم دیگر این است که پورتی را که RavenDB میخواهد در آن فعال شود را تعریف کنیم.
برای اینکار بایستی در قسمت تنظیمات Worker Role در زبانه Endpoints رفته و یک endpoint جدید به آن مطابق تصویر زیر ارائه کنیم.



حال که پورت هم تنظیم شده است میتوانیم RavenDB را در Worker Role راه بیاندازیم:

```
var config = new RavenConfiguration
{
    DataDirectory = driveLetter,
    AnonymousUserAccessMode = AnonymousUserAccessMode.All,
    HttpCompression = true,
    DefaultStorageTypeName = "munin",
    Port = RoleEnvironment.CurrentRoleInstance.InstanceEndpoints["Raven"].IPEndpoint.Port,
    PluginsDirectory = "plugins"
};

try
{
    documentDatabase = new DocumentDatabase(config);
    documentDatabase.SpinBackgroundWorkers();
    httpServer = new HttpServer(config, documentDatabase);
```

```
    try
    {
        httpServer.StartListening();
    }
    catch (Exception ex)
    {
        Trace.WriteLine("StartRaven Error: " + ex.ToString(), "Error");

        if (httpServer != null)
        {
            httpServer.Dispose();
            httpServer = null;
        }
    }
}
catch (Exception ex)
{
    Trace.WriteLine("StartRaven Error: " + ex.ToString(), "Error");

    if (documentDatabase != null)
    {
        documentDatabase.Dispose();
        documentDatabase = null;
    }
}
```