11. CODING AND IMPLEMENTATION

11.1 Cloud Platform Coding Examples on Visual Basic.net

There are millions of examples for cloud platform coding. Here, we will discuss about how to use trace in Windows Azure based cloud applications.

This chapter will revolve around the basic workaround of how to use Trace in the Windows Azure. Take any Windows Azure app. For instance, we see that most of them utilize a Trace API. At the beginning of creating a cloud app for Windows Azure in Visual Studio, the app contains a Trace Code as depicted below:

```csharp
Trace.WriteLine("Working", "Information");
```

There are lots of instances when it comes to coding for cloud enabled apps in the VB.NET platform. We choose this Trace feature because it is a highly claimed program. Taking the Windows Azure template in hand, we already have a generated code which has a partially configured trace diagnostic's listener. Here a fully configured tracing in apps built for Windows Azure is shown.

11.1.1 Persisting Trace

First, it is important to use respective codes for handling errors and information in tracing. Normally, we use Trace.TraceError as an error handler and Trace.TraceInformation as the information handler. Data persisting in Windows Azure requires more than calling the Trace API alone. We need a fully configured diagnostic monitor for persisting trace. A diagnostic monitor can be configured by following the code mentioned below.

```csharp
private void ConfigDiagnostics()
{
    DiagnosticMonitorConfiguration config =
        DiagnosticMonitor.GetDefaultInitialConfiguration();

    config.ConfigurationChangePollInterval = TimeSpan.FromMinutes(1d);
}
```
config.Logs.BufferQuotaInMB = 500;

config.Logs.ScheduledTransferLogLevelFilter = LogLevel.Verbose;

config.Logs.ScheduledTransferPeriod = TimeSpan.FromMinutes(1);

DiagnosticMonitor.Start(


    config);

}

11.1.2 Save Transaction Cost
As the discussion in the above-mentioned code that the transfer rate is set to one minute, it is the minimum value that can be set in this trace program. Any value less than one minute will be regarded as one minute. In case of the cloud application for which this trace program is created; it generates trace data in a very slow manner. Suppose there is one event per minute, that time only one row will be uploaded for every agent in the single minute gap in a separate transaction. We can easily save on the transaction cost by setting up the upload rate to at least sixty minutes. In this case, about sixty rows will be packaged in a single batch, and the batch will be uploaded in only one transaction. However, there is a drawback for this large transfer period; a delay will occur in the storage data.

If there is a pile of data, then using this trick will be troublesome as the upload will certainly be bulky, and it can cause data throttling.

The above sample code directs the ScheduledTransferLogLevelFilter to Verbose; therefore, all the trace reports are captured.

11.1.3 Calling up the ConfigDiagnostics Handler
In order to use the code in OnStart() scheme for every web role and worker role, we need to call ConfigDiagnostics as shown below:

public override bool OnStart()
220

```csharp
{  
  ConfigDiagnostics();  
  Trace.TraceInformation("Initializing storage account")  
  // Code removed for clarity.  
}
```

The written instructions above will only work in Windows Azure Cloud Applications. This code is for using on the Windows Azure based web sites.

![Figure 11.1 Azure Storage Explorers](image)

It is always better to use two separate storage account for production data and storage data respectively. The following XML sample shows the actual Trace configuration.

```xml
<?xml version="1.0" encoding="utf-8"?>

<ServiceConfiguration serviceName="AzureEmailService" xmlns="http://schemas.microsoft.com/ServiceHosting/2008/10/ServiceConfiguration" osFamily="1" osVersion="*" schemaVersion="2012-05.1.7">
```

220
<Role name="MvcWebRole">
    <Instances count="1" />
    <ConfigurationSettings>
            value="DefaultEndpointsProtocol=https;
            AccountName=[TraceAccount];
            AccountKey=[Key]" />
        <Setting name="StorageConnectionString"
            value="DefaultEndpointsProtocol=https;
            AccountName=[DataAccount];
            AccountKey=[Key2]" />
    </ConfigurationSettings>
</Role>

<Role name="WorkerRoleA">
    <Instances count="1" />
    <ConfigurationSettings>
            value="DefaultEndpointsProtocol=https;
            AccountName=[TraceAccount];
            AccountKey=[Key]" />
    </ConfigurationSettings>
</Role>
AccountKey=[Key]" />

<Setting name="StorageConnectionString"
value="DefaultEndpointsProtocol=https;
AccountName=[DataAccount];
AccountKey=[Key2]" />

</ConfigurationSettings>

</Role>

</ServiceConfiguration>

Tracing offers a stable workaround for monitoring Windows Azure applications. It provides good information that is necessary to debug the problems.