# Generic Worker quickstart document

## Setup environment

### Development tools

Azure SDK, Visual Studio Express + SQL Server Express (needed for the azure storage emulator):  
<http://www.microsoft.com/windowsazure/getstarted/>

WIF SDK:  
<http://www.microsoft.com/downloads/en/details.aspx?FamilyID=c148b2df-c7af-46bb-9162-2c9422208504>  
Please install only this one:  WindowsIdentityFoundation-SDK-4.0.msi

**Windows Identity Foundation Runtime needed:**

http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=17331

Hg (Mercurial) extension for Visual Studio:  
<http://visualstudiogallery.msdn.microsoft.com/9bc074fa-9e1f-4ce2-a75d-b90e65f7475a?SRC=VSIDE>

ASP.Net MVC3:  
<http://download.microsoft.com/download/3/4/A/34A8A203-BD4B-44A2-AF8B-CA2CFCB311CC/AspNetMVC3Setup.exe>

NuGet:  
<http://download.codeplex.com/Project/Download/FileDownload.aspx?ProjectName=nuget&DownloadId=222685&FileTime=129461818428230000&Build=17748> **(Link does not work)**

Other useful tools (Optional):  
<http://azurestorageexplorer.codeplex.com/> allows you to explore and manipulate the azure storage

### Repository

* Download mercurial GUI/commandline tool from: <http://mercurial.selenic.com/>
* Clone the GW repository on your machine:

$ hg clone http://projects.venus-c.eu/hg/venus/venusinfra genericworker

$ cd genericworker

* Doubleclick the solution file “*OGF.BES.Sample.sln”* in “*dev\OGF\”*

### Configuration (adapting the GW setting to your Deployment

* Adapt the ServiceConfiguration.cscfg file:
  + Replace the host name my.genericworker.net with the hostname for your service  
    e.g. <SERVICE-NAME>.cloudapp.net
  + Replace the certificate information
  + Point the connectionstrings to your cloudstorage
* Adapt the app.config files of your client applications:
  + Same as for the ServiceConfiguration.cscfg file

## Deployment

To manage azure deployments you need to create a local certificate on your pc and register it to the azure portal.

### Create a self signed certificate:

Using visual studio command prompt (run as administrator):

makecert -r -pe -n "CN=AzureMgmt" -a sha1 -len 2048 –sr localmachine -ss My "AzureMgmt.cer"

Using the GUI, use the certificate manager: Start->Run->certmgr.msc

Register the certificate on [www.azure.com](http://www.azure.com)

**To import an existing certificate use (also in the Visual Studio CMD):**

certmgr –add "*<Location>\*cert.crt" –s –r localMachine

### The actual deployment can be done in alternative ways

#### In Visual Studio:

Use publish in the context menu of the cloud icon in the solution explorer

#### In Powershell by using Azure Management Commandlets

##### Install the Azure Powershell Commandlets:

Set-alias installutil

$env:windir\Microsoft.NET\Framework\v2.0.50727\installutil.exe

Installutil -i Microsoft.Samples.AzureManagementTools.PowerShell.dll

(use a 32-bit/x86 version of powershell to initiate this command)

**Or:** <http://wappowershell.codeplex.com/> **download and install**

##### Check if the commandlets were installed properly:

Get-pssnapin –registered. If not please re-install them before go to next steps.

##### Add the Snapins to your Powershell session

Add-PSSnapin AzureManagementToolsSnapIn

##### Using the Azure Management Commandlets to manage your deployment

**A list of all Azure Management Commandlets** :

Get-Command -PSSnapin AzureManagementToolsSnapIn

**To simplify the usage of these commandlets we recommend you to set some useful variables in your powershell profile settings:**   
<USERFOLDER>\Documents\WindowsPowerShell\**Microsoft.PowerShell\_profile.ps1** (create the path if it does not exist)

There you should set:

$thumbprintInUpperCase = "<YOUR CERTIFICATES THUMBPRINT>" **#(from Step 2.1)**

$serviceName = "<THE GENERIC WORKER SERVICE NAME>"

$subid = "<YOUR SUBSCRIPTION ID>"

$cert = (get-item cert:\**CurrentUser**\My\$thumbprintInUpperCase) **#Depending on the location of the certificate**

$pkg = "<CSPKG\_FILE>" #build the pkgfile by using the cspack tool that comes with the azure SDK

$cscfg="<CSCFG\_FILE>"

When you now start powershell(x86) then these variables are already set.

**Retrieve the hosted service:**

$hostedservice = Get-HostedService -ServiceName "$serviceName" -SubscriptionId $subid -Certificate $cert

**Deploy on azure:**

New-Deployment -subscriptionId $subid -certificate $cert -serviceName $serviceName -slot staging -package $pkg -configuration $cscfg -name TestDeploy -label TestLabelStaging

It is recommended to first deploy to the staging slot (the service runs on a randomly picked URL of the form <some-guid>.cloudapp.net ) and switch the deployment to the production slot (URL: servicename.cloudapp.net ) once everything is running smooth.

**Swap staging to production:**

echo $hostedservice | Get-Deployment -slot staging | Move-Deployment | Get-OperationStatus –WaitToComplete

**Retrieve the deployment:**

$deployment = echo $hostedservice | Get-Deployment production

**Run the deployment:**

echo $deployment | Set-DeploymentStatus running | Get-OperationStatus –WaitToComplete

**Stop the deployment: ?**

echo $deployment | Set-DeplaymentStatus suspended | Get-OperationStatus –WaitToComplete

**Remove the deployment:**

echo $deployment | Remove-Deployment

## Use the Generic Worker programatically

### There are two main steps

* + 1. Installing the application:

Provide an application description (an URI and a description of the commandline arguments)

string splitterAppIdentificationURI=  <http://www.upvbio.eu/demo/gw/UPVBIO/Splitter>

VENUSApplicationDescription splitterAppDesc = new VENUSApplicationDescription()

{

ApplicationIdentificationURI = splitterAppIdentificationURI,

CommandTemplate = new VENUSCommandTemplate()

{

Path = string.Empty,

Executable = "splitter.exe",

Args = new List<CommandLineArgument>()

{

new CommandLineArgument(){

Name = "inputfile",

FormatString = "{0}",

Required = Required.Mandatory,

CommandLineArgType = CommandLineArgType.SingleReferenceInputArgument

},

new CommandLineArgument()

{

Name = "numfragments",

FormatString = "{0}",

Required = Required.Mandatory,

CommandLineArgType = CommandLineArgType.SingleLiteralArgument

},

new CommandLineArgument()

{

Name = "startfragment",

FormatString = "{0}",

Required = Required.Mandatory,

CommandLineArgType = CommandLineArgType.SingleLiteralArgument

},

new CommandLineArgument()

{

Name="fragmentfiles",

FormatString = "", //\*.sqf

Required = Required.Mandatory,

CommandLineArgType = CommandLineArgType.MultipleReferenceOutputArgument

}

}

};

Package all needed files in one zip file (except the input files of course )

Func<string[], MemoryStream> createZippedAppPkg = (filenameArray) =>

{

MemoryStream AppZipBytes = new MemoryStream();

using (var zip = new ZipFile())

{

foreach (string file in filenameArray)

zip.AddFile(file, "");

zip.Save(AppZipBytes);

}

AppZipBytes.Seek(0L, SeekOrigin.Begin);

return AppZipBytes;

};

string[] splitterFiles = { @".\Splitter.exe" };

MemoryStream splitterAppZipBytes = createZippedAppPkg(splitterFiles);

Upload the package to the AppStore and register the application:

// Retrieve a token which allows us to stream the application ZIP file directly into the blob store of the Application Repository.

Action<string, MemoryStream, VENUSApplicationDescription> uploadApp = (appURI, zipBytes, appDesc) =>

{

var appStore = new AppStoreClient(

new BasicHttpBinding() { TransferMode = TransferMode.Streamed },

new EndpointAddress(ApplicationStoreUrl));

var token = appStore.GetUploadToken(appURI);

var binaddress = string.Format("{0}{1}", token.BlobEndpoint, token.Filepath);

var cred = new StorageCredentialsSharedAccessSignature(token.SAS);

var appblob = new CloudBlob(binaddress, cred);

appblob.UploadFromStream(zipBytes);

print(ConsoleColor.Green, string.Format("Uploaded {0} bytes",zipBytes.Length));

// After the bits are uploaded, provide the missing description

appStore.RegisterExistingApplication(appDesc);

};

uploadApp(splitterAppIdentificationURI, splitterAppZipBytes , splitterAppDesc );

* + 1. Upload/reference input data and submit jobs

Access your cloud storage and create a container if necessary and setup access permissions:

var account = CloudStorageAccount.Parse(UserDataStoreConnectionString);

var blobClient = account.CreateCloudBlobClient();

var userDataContainer = blobClient.GetContainerReference("userdatacontainer");

userDataContainer.CreateIfNotExist();

BlobContainerPermissions resultPermissions = userDataContainer.GetPermissions();

resultPermissions.PublicAccess = BlobContainerPublicAccessType.Container;

userDataContainer.SetPermissions(resultPermissions);

Upload inputfiles from your filesystem to your storage account:

string inputfile = <SOME FILENAME>;

string addressInputFile = null;

Func<string, string, string> uploadFile = ((blobname, filename) =>

{

var blob = userDataContainer.GetBlobReference(blobname);

blob.UploadFile(filename);

var blobAddress = blob.Uri.AbsoluteUri;

print(ConsoleColor.Blue, string.Format("Uploaded \"{0}\" to {1}", new FileInfo(filename).FullName, blobAddress));

print(ConsoleColor.Blue, "");

return blobAddress;

});

addressInputFile = uploadFile(inputFile, inputFile);

If the inputfile is already stored in the cloud (for example when the input file was created by some other job) you can omit the upload and just compute the URI of the file:

Func<string, string> computeName = ((name) =>

{

var result = userDataContainer.GetBlobReference(name).Uri.AbsoluteUri;

print(ConsoleColor.Blue, string.Format("Expecting result at {0}", result));

return result;

});

In this case you would then do:

addressInputFile = computeName  (inputFile);

For result files you can also use computeName to retrieve the URI for the location on the storage where you want to store the resultfile.

Provide a job description: This is really similar to the application description

var splitterJob = new VENUSJobDescription()

{

ApplicationIdentificationURI = splitterAppIdentificationURI,

CustomerJobID = "UPVBIO Splitter Job " + DateTime.Now.ToLocalTime().ToString(),

JobName = "some job name",

JobArgs = new ArgumentCollection()

{

new SingleReference

{

Name="inputfile",

IsDownload= DataSetType.Input,

Ref= new Reference()

{

LocalFileName=inputFile,

ProviderSpecificReference = new AzureBlobReference

{

DataAddress = addressInputFile,

ConnectionString = UserDataStoreConnectionString,

}

}

},

new LiteralArgument

{

Name = "numfragments",

LiteralValue = "2"

},

new LiteralArgument

{

Name = "startfragment",

LiteralValue = "0"

},

new ReferenceArray

{

Name = "fragmentfiles",

IsDownload= DataSetType.Result,

References = new ReferenceCollection()

{

new Reference(){

LocalFileName="seqfile0.sqf",

ProviderSpecificReference = new AzureBlobReference

{

DataAddress = computeName("seqfile0.sqf"),

ConnectionString = UserDataStoreConnectionString,

}

},

new Reference(){

LocalFileName="seqfile1.sqf",

ProviderSpecificReference = new AzureBlobReference

{

DataAddress = computeName("seqfile1.sqf"),

ConnectionString = UserDataStoreConnectionString,

}

},

}

}

}

};

## Main locations in the storage

### Tables:

GWJobDetails: The status and output of the jobs

GWJobIndex: A list of jobs that are active (have not terminated) in the system

Appstoretable: A list of the installed Apps

### Blobs:

Appstoreblob: Contains a zip and a xml file for each installed application

## Appendix

### Deployment in the DevFabric:

In order to speed up the development process you can also deploy on your own machine. The GW Web Role then runs in the compute emulator on your machine. Now you can continue to use the cloud storage or move even the storage to your own machine. In this case the cloud storage is emulated by the storage emulator. To use the storage emulator you only need to set the DataConnectionString "UseDevelopmentStorage=true"

When the deployment is running in the DevFabric the GW retrieves its settings from your registry. For this reason you have setup your registry.

#### Setup the registry:

* + Open dev\OGF\commons\DeveloperSettings.reg in an editor
  + Enter your username and password for your domain
  + Replace "developername" with your name/alias (please use only characters or digits [a-zA-Z0-9])  
    The developername is only used for having separate tables and containers for researchers that are using the same storage account in the cloud.
  + Replace “AppStoreOnAzureCertificateThumbprint” Value with the generated certificate thumbprint (created in 2.1.), to do so run “ls cert:\LocalMachine\My” in Powershell and look for “CN=AzureMgmt”
  + Set the HostSwitch to DevelopmentFabric, Premises or Cloud
  + Save the file in a folder which is not version controlled
  + Doubleclick that file to install/update the keys in your registry

You can now remove all CURRENT\_USER setting from the file.

For switching between deployment in the cloud and deployment on your machine you can use the HostSwitch registry files in dev\OGF\commons .

## Start the project in VisualStudio 2010

To start the project with F5 you have to set the “StartUp Project” in the Solution Explorer:

