HPC Image Rendering

**Contents**

[Overview 2](#_Toc369946862)

[Getting Started 3](#_Toc369946863)

[Task 1 - Inspecting the AzureBlobCopy Project 3](#_Toc369946864)

[Task 2 – Inspecting the Run.cmd File 5](#_Toc369946865)

[Task 3 - Inspecting the RenderCmd Client Application 6](#_Toc369946866)

[Deployment 9](#_Toc369946867)

[Task 1 – Deploying the Parametric Sweep Application to Windows Azure Nodes 9](#_Toc369946868)

[Running the Client 10](#_Toc369946869)

[Task 1 – Verifying Cluster State 10](#_Toc369946870)

[Task 2 - Manually Uploading Instruction Files to the Blob Storage 11](#_Toc369946871)

[Task 3 – Submitting the Job Manually Through the HPC Job Scheduler 12](#_Toc369946872)

[Task 4 – Running the Client Application 17](#_Toc369946873)

[Summary 19](#_Toc369946874)

Overview

* 1. The HPC Image Rendering sample demonstrates how to run a parametric sweep application that renders images according to instruction files. The application demonstrates how to download and upload files from Windows Azure blob storage, and how to create a parametric sweep job from .NET code.
  2. This sample uses the [Aqsis](http://www.aqsis.org/) application that renders 3D images according to the RenderMan® standard.
  3. **Note:** To use this sample application, [download](http://www.microsoft.com/downloads/en/details.aspx?FamilyID=acde41c6-153a-4181-912e-78024fcc86da) the sample RenderMan data files and extract them to the labs folder.

# Key Features

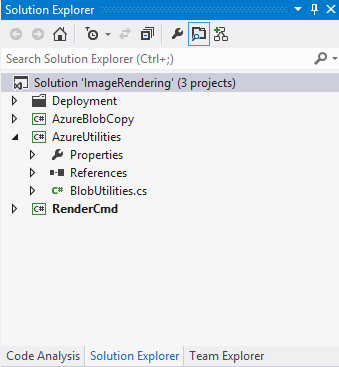
* 1. This sample demonstrates the following:
  + Uploading a parametric sweep application package to Windows Azure nodes.
  + Uploading and downloading files to Windows Azure blob storage.
  + Creating and running a parametric sweep job in a Windows Server 2012 Datacenter with HPC Pack 2012.

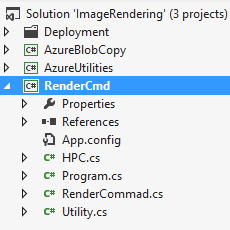
Getting Started

* 1. To run this sample, you should install [the HPC Pack 2012 Client Utilities Redistributable Package with Service Pack 1](http://www.microsoft.com/en-us/download/details.aspx?id=39959) (or higher), the [HPC Pack 2012 SDK with Service Pack 1](http://www.microsoft.com/en-us/download/details.aspx?id=39967) (or higher), and the [Windows Azure SDK and Windows Azure Tools for Microsoft Visual Studio (March 2011)](http://www.microsoft.com/download/en/details.aspx?id=15658). In addition, you need to have administrative access to your HPC cluster’s head node.
  2. To run the image rendering application in Windows Azure nodes, you must have a valid Windows Azure account, a Windows Azure worker node template defined in your head node, and several Windows Azure worker nodes in the HPC cluster that are started and online. Please refer to 1. windows-hpc-install in the training material.

Task 1 - Inspecting the AzureBlobCopy Project

In this task, you will inspect the AzureBlobCopy utility project to see how to download and upload files from Windows Azure blob storage. The parametric sweep job uses the AzureBlobCopy utility to copy a rendering instruction file from a blob container, and copy the rendered image to a different blob container.

* 1. Open Microsoft Visual Studio 2012 from **Start** | **All Programs** | **Microsoft Visual Studio 2012** | **Microsoft Visual Studio 2012**.
  2. Open the ***ImageRendering.sln*** solution file located in the ***ImageRendering\Source*** folder.
  3. In the **Solution Explorer** window, expand the ***AzureUtilities*** project node, as shown in Figure 1:
     1. 
     2. Figure 1
     3. The AzureUtilities project
  4. Open the **BlobUtilities.cs** file and inspect the **BlobUtilities** class. The class contains several methods for working with Windows Azure blob storage:
     1. **InitializeStorage**. The method validates the existence of a required blob container, and, if necessary, creates such a container and sets its permissions.
     2. **DownloadFile**. The method downloads a file from a block blob and saves it to a specified path.
     3. **UploadFile**. The method uploads a file to a block blob. The **CloudBlob.UploadFile** method divides large content into several blocks, and the blocks are uploaded in parallel, according to the minimum number of I/O threads in the thread pool. If you want to change the number of parallel uploads, you can set the **CloudBlobContainer.ParallelOperationThreadCount** property, as shown in the following example:
        1. C#
        2. public void UploadFile(
        3. string containerName, string path, string fileName)
        4. {
        5. var blobStorage = InitializeStorage(containerName);
        7. if (blobStorage != null)
        8. {
        9. CloudBlobContainer container =
        10. blobStorage.GetContainerReference(containerName);
        11. blobStorage.ParallelOperationThreadCount = 8;
        12. CloudBlob blob = container.GetBlockBlobReference(fileName);
        13. blob.UploadFile(Path.Combine(path, fileName));
        14. }
        15. }
     4. **DeleteFile**. The method deletes a file from a block blob.
  5. In the **Solution Explorer** window, locate the ***AzureBlobCopy*** project, and inspect the contents of the ***Program.cs*** file.  **AzureBlobCopy** is a console application that supports copying files to and from Windows Azure blob storage. For example, the following command line downloads a file named ***frame-1.zip*** from the ***input*** blob storage to a local folder:
     1. CMD
     2. AzureBlobCopy.exe -Action Download -BlobContainer input -LocalDir c:\frames\ -FileName frame-1.zip
  6. Open the **AzureBlobCopy** project’s settings and inspect the following tabs:
     1. **Build**. The ***aqsis*** folder contains the aqsis rendering application that runs on each compute node; this is the only folder that is copied to the compute nodes (on-premises and Windows Azure). Before running the aqsis application, the AzureBlobCopy utility is called in order to download the rendering instruction file from the blob. The output path of the project is therefore set to the ***aqsis\bin*** folder.
     2. **Build Events**. The **Post-build** event runs the hpcpack command to pack the aqsis application and the AzureBlobCopy utility to a compressed zip file for deployment to Windows Azure compute nodes.
  7. Open the ***App.config*** file from the **AzureBlobCopy** project. The **appSettings** section contains two keys, ***StorageAccountName*** and ***StorageKey***.
  8. Change the storage account name and the storage primary key to match your Windows Azure storage account settings.
  9. Build the **AzureBlobCopy** project.

1. Task 2 – Inspecting the Run.cmd File
   1. In this task, you will inspect the contents of the run.cmd file to see which commands run when the parametric sweep job executes in each compute node.
   2. In the **Solution Explorer** window, right-click the solution and select **Open Folder in Windows Explorer**.
   3. In the **Windows Explorer** window navigate to the ***aqsis\bin*** folder, locate the **run.cmd** file, and view the file’s contents:
      1. RUN.CMD
      2. REM Use the input parameter as a frame index.
      3. set frame=%1
      4. REM Setup the executable, input, and output folders.
      5. set root=%CCP\_PACKAGE\_ROOT%\Aqsis
      6. set inputdir=%CCP\_WORKDIR%\%CCP\_JOBID%\%CCP\_TASKID%\input
      7. set outputdir=%CCP\_WORKDIR%\%CCP\_JOBID%\%CCP\_TASKID%\output
      8. if not exist %inputdir% mkdir %inputdir%
      9. if not exist %outputdir% mkdir %outputdir%
      10. REM Pull input data from blob storage.
      11. %root%\bin\AzureBlobCopy.exe -Action Download -BlobContainer input -LocalDir %inputdir% -FileName %frame%.zip
      12. REM Unzip the input file, run the executable, and create output data.
      13. %root%\bin\rar.exe e -y %inputdir%\%frame%.zip %outputdir%
      14. cd %outputdir%
      15. %root%\bin\aqsis.exe -shaders:"%root%\displacement:%root%\shaders\imager:%root%\shaders\light:%root%\shaders\surface:%root%\shaders\volume" -displays="%root%\bin" %outputdir%\%frame%.rib
      16. REM Upload the output files to blob storage.
      17. %root%\bin\AzureBlobCopy.exe -Action Upload -BlobContainer output -LocalDir %outputdir% -FileName %frame%.tif
      18. REM remove local files
      19. del /Q %inputdir%\%frame%.zip
      20. del /Q %outputdir%\%frame%.rib
      21. del /Q %outputdir%\%frame%.tif
   4. The run.cmd file executes the following commands:
      1. Runs the AzureBlobCopy utility to download the current rendering instruction file from the input blob and decompress it.
      2. Runs the aqsis 3D rendering application.
         * 1. **Note:** This application uses the **root** environment variable, which is set according to the CCP\_PACKAGE\_ROOT environment variable mentioned in Task 2 in this section.
      3. Runs the AzureBlobCopy utility to upload the generated .tiff image to the output blob.
2. Task 3 - Inspecting the RenderCmd Client Application
   1. In this task, you will explore the RenderCmd client application that uploads the instruction files to the Windows Azure blob storage, submits the parametric sweep job, and downloads the rendered images from the output blob.
   2. In the **Solution Explorer** window, expand the ***RenderCmd*** project node.
      1. 
      2. Figure 2
      3. The RenderCmd project
   3. In the **RenderCmd** project, open the **RenderCommand.cs** file, and inspect the contents of the **RenderCommand** class.
      1. **UploadFiles**. This method iterates the files in the input folder and uploads all the render instruction files to a blob.
      2. **DownloadFiles**. This method downloads the contents of the rendered images’ output blob to a local folder.
      3. **Run**. This method calls the **UploadFiles** method, submits the parametric sweep job, and then calls the **DownloadFiles** method to download the rendered images created by the parametric sweep application.
   4. In the **RenderCmd** project, open the **HPC.cs** file and inspect the ***CreateJob*** method.
      1. The method first connects to the job scheduler and creates a new job, as shown in the following code snippet:
         1. C#
         2. Scheduler scheduler = new Scheduler();
         3. scheduler.Connect(headnode);
         5. // Define job settings
         6. ISchedulerJob job = scheduler.CreateJob();
         7. job.Name = "Aqsis on Azure";
         8. job.MinimumNumberOfCores = 1;
         9. job.MaximumNumberOfCores = 1;
         10. job.UnitType = JobUnitType.Core;
         11. // Let the scheduler calculate the required resources for the job
         12. job.AutoCalculateMax = true;
         13. job.NodeGroups.Add(targetNodes);
      2. After creating the job settings, the method adds a parametric sweep task to the new job and submits it to the scheduler as follows:
         1. C#
         2. // Create a parametric sweep task
         3. ISchedulerTask task = job.CreateTask();
         4. task.Type = TaskType.ParametricSweep;
         5. task.StartValue = 0;
         6. task.EndValue = endValue;
         7. task.IncrementValue = 1;
         8. // Run the aqsis command to render the images
         9. // The (\*) wildcard is used as a placeholder for the current index value
         10. task.CommandLine = @"%CCP\_PACKAGE\_ROOT%\Aqsis\bin\run.cmd frame-\*";
         11. task.WorkDirectory = "%CCP\_PACKAGE\_ROOT%";
         13. Console.WriteLine("Running job");
         14. job.AddTask(task);
         15. scheduler.SubmitJob(job, username: null, password: null);

**Note:** The CCP\_PACKAGE\_ROOT environment variable is defined in Windows Azure nodes and points to the folder in which packaged HPC applications are deployed.

* + 1. After submitting the job, the client application waits until the job completes, periodically checking the job’s status.
  1. In the **RenderCmd** project, open the ***app.config*** file and inspect the ***appSettings*** section. This section contains four settings that you need to change in accordance with your HPC cluster definitions and Windows Azure storage account settings:
     1. **HeadNodeName**. The name of your head node machine.
     2. **NodeGroup**. The group of compute nodes that will run the parametric sweep application.
     3. **StorageAccountName**. The name of the blob storage in which the input and output files will be stored.
     4. **StorageKey**. The primary storage key of the storage specified in **StorageAccountName**.
  2. Build the **RenderCmd** project.

Deployment

* 1. In order to run this sample, you will need to deploy the parametric sweep application to your compute nodes.

1. Task 1 – Deploying the Parametric Sweep Application to Windows Azure Nodes
   1. In this task, you will deploy the aqsis parametric sweep application to your Windows Azure nodes.
   2. Open the command prompt window from **Start** | **All Programs** | **Microsoft Visual Studio 2012** | **Visual Studio Tools** | **Visual Studio Command Prompt (2012)**.
   3. Make sure you run the command prompt as Administrator.
   4. Navigate to the **ImageRendering** labs folder, and run the following command to upload the deployment package to the Windows Azure package storage:
      1. CMD
         1. hpcpack upload Source\aqsis.zip /nodetemplate:"Default AzureNode Template" /relativePath:aqsis
      2. **Note:** Change the value of the **nodetemplate** parameter to the name of your Windows Azure node template.
   5. If you already have Windows Azure nodes started in the HPC cluster, you need to copy the new package to them. To sync the Windows Azure nodes with the new packages stored in the application packages blob, run the following command:
      1. CMD
         1. clusrun /nodegroup:AzureNodes hpcsync

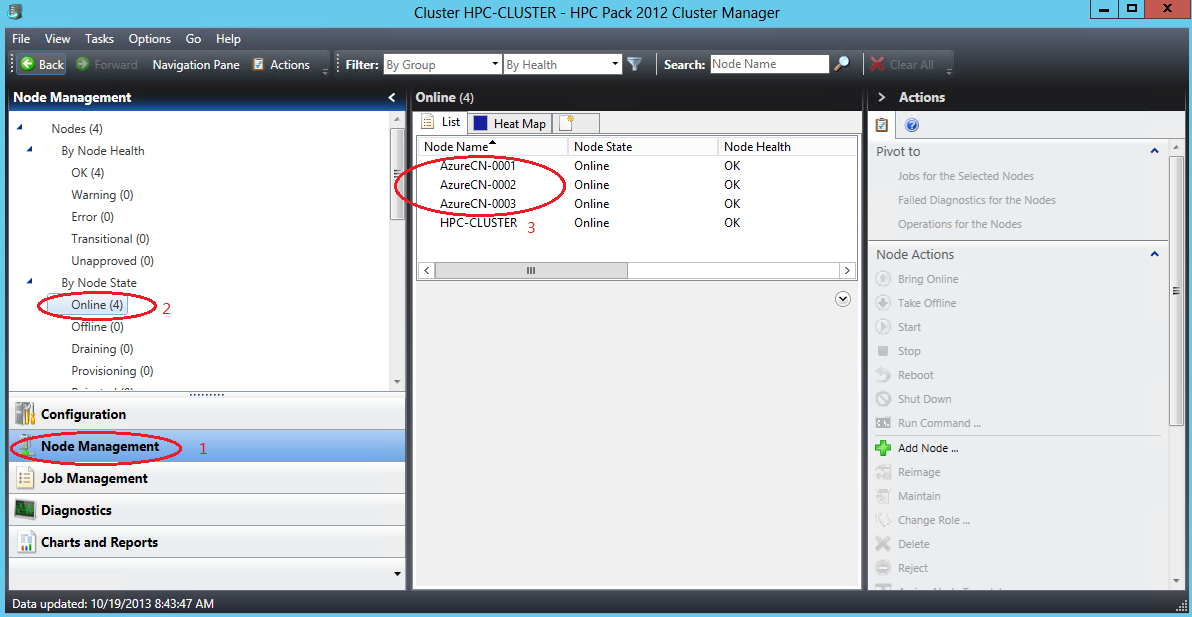
You can just check the file **deploy.cmd** under the deployment folder and update related your configuration in the code.

* + 1. CMD
       1. cluscfg setenvs "CCP\_PACKAGE\_ROOT=c:\apps"
       2. clusrun /nodegroup:AzureNodes xcopy \\hpc-cluster\apps\aqsis\\*.\* c:\apps\aqsis\ /YE
       3. hpcpack upload ..\aqsis.zip /nodetemplate:"Default AzureNode Template" /relativePath:aqsis
       4. clusrun /nodegroup:AzureNodes hpcsync

Running the Client

* 1. There are two options you can choose from in order to run the Image Rendering sample:
  2. Run the client application. The *RenderCmd* client application uploads the input files to Windows Azure storage, submits the parametric sweep job, and downloads the outputted files from the storage. To use this technique refer to Tasks 1 and 4.
  3. Submit the parametric sweep job manually. This option requires manually uploading the input files to the storage, submitting the parametric sweep job in the HPC Job Scheduler, and then viewing the outputted images from the storage. To use this technique refer to Tasks 1, 2, and 3.

1. Task 1 – Verifying Cluster State
   1. In this task, you will verify that the nodes in your cluster are online.
   2. Open the HPC 2012 Cluster Manager application from **Start** | **All Programs** | **Microsoft HPC Pack 2012 R2** | **HPC Cluster Manager**.
   3. In the Cluster Manager application, enter the **Node Management** section and verify that the Windows Azure nodes in the cluster are online, as shown in Figure 3:

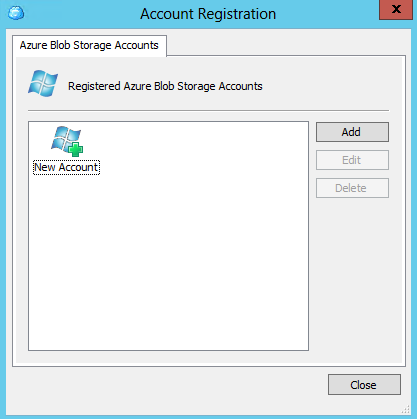
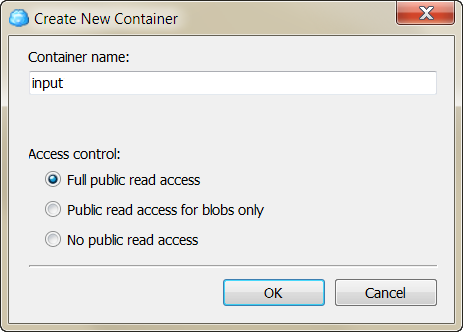


* + 1. Figure 3
    2. Verifying the State of the Windows Azure Nodes

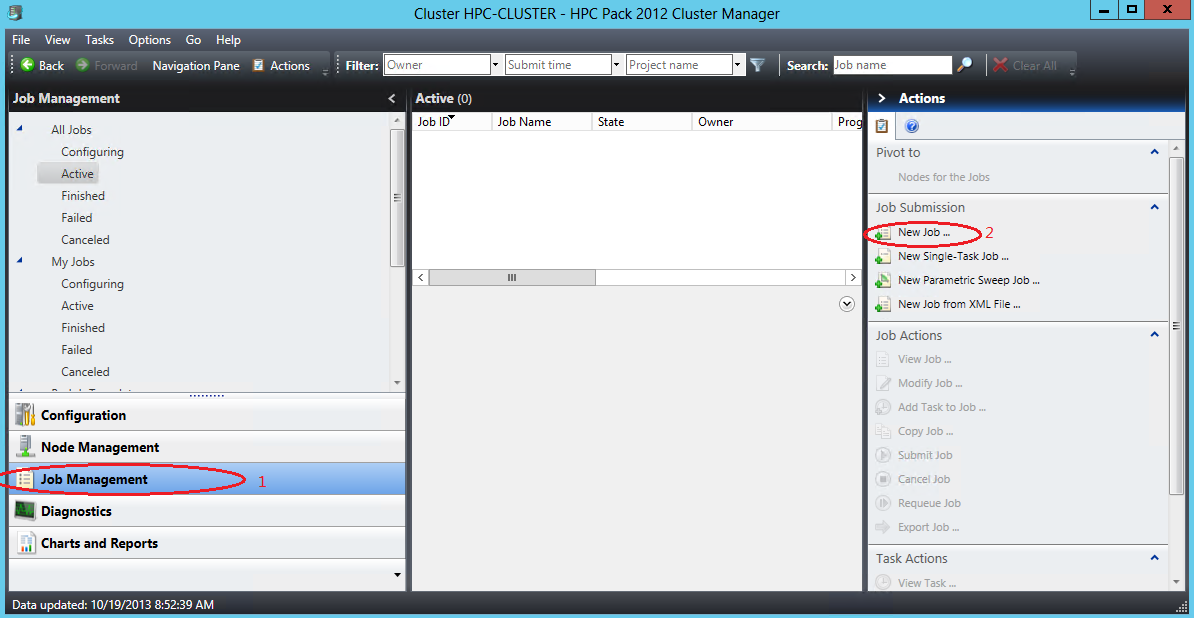
Task 2 - Manually Uploading Instruction Files to the Blob Storage

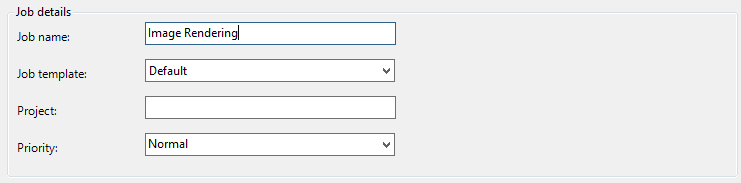
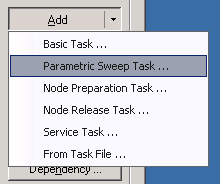
In this task, you will upload the instruction files required by the parametric sweep application to a blob in your Windows Azure storage account.

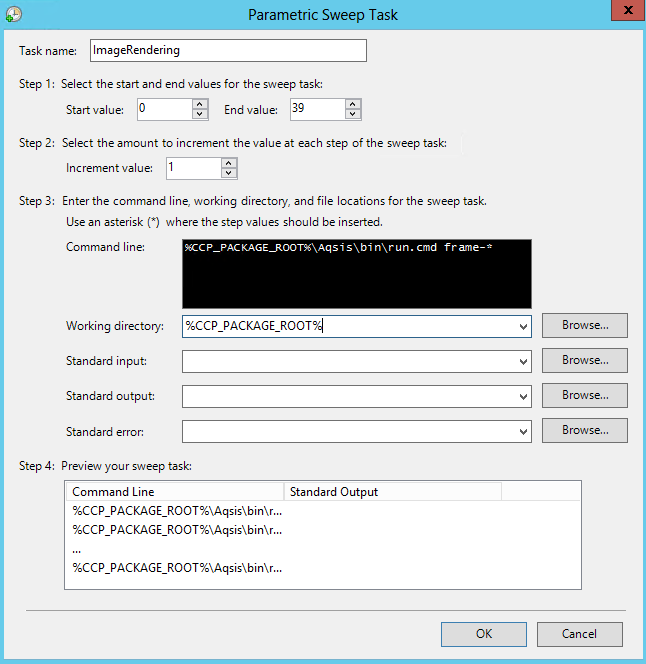
Uploading, downloading, and browsing files in blobs is an easy task if you install one of the blob storage browsing applications, such as [CloudBerry Explorer for Azure Blob Storage](http://cloudberrylab.com/?page=explorer-azure), or the [Azure Storage Explorer](http://azurestorageexplorer.codeplex.com). The following steps are for the CloudBerry Explorer application; you can use the same techniques with Azure Storage Explorer, but the steps may differ.

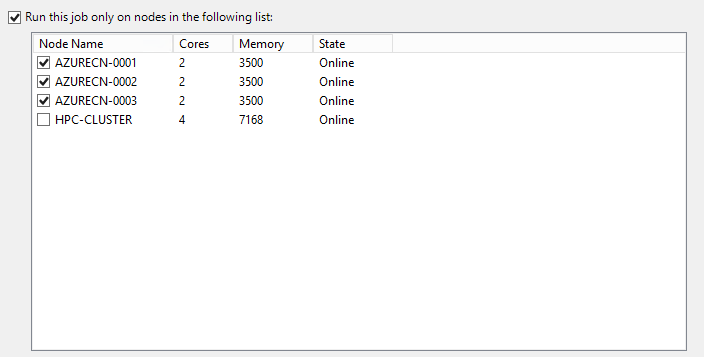
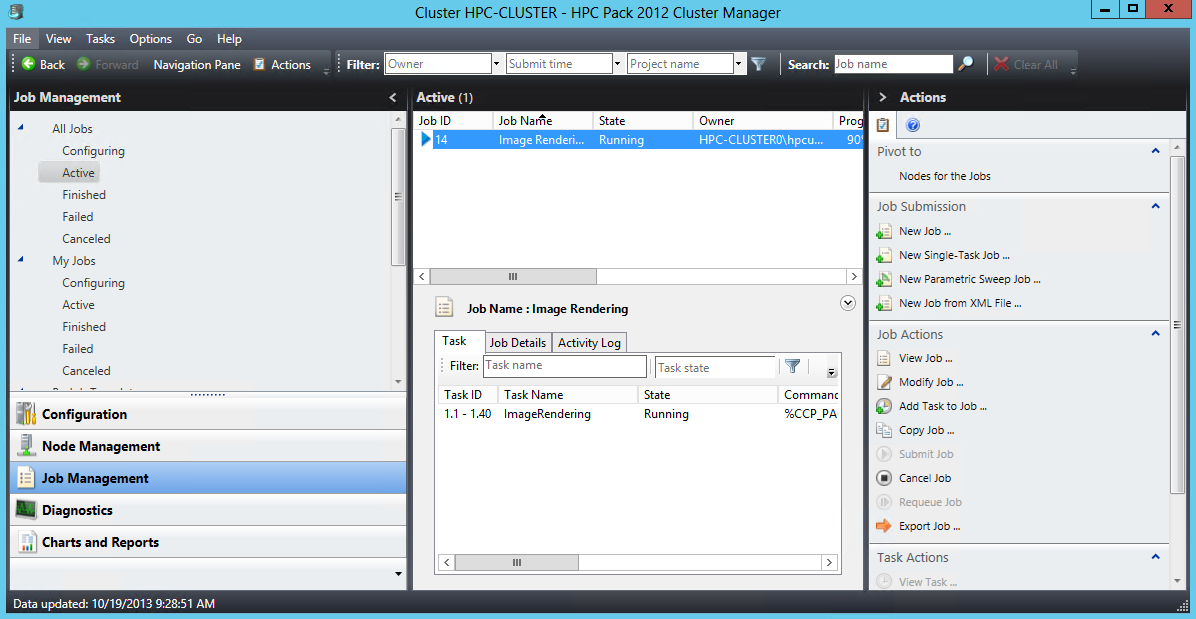
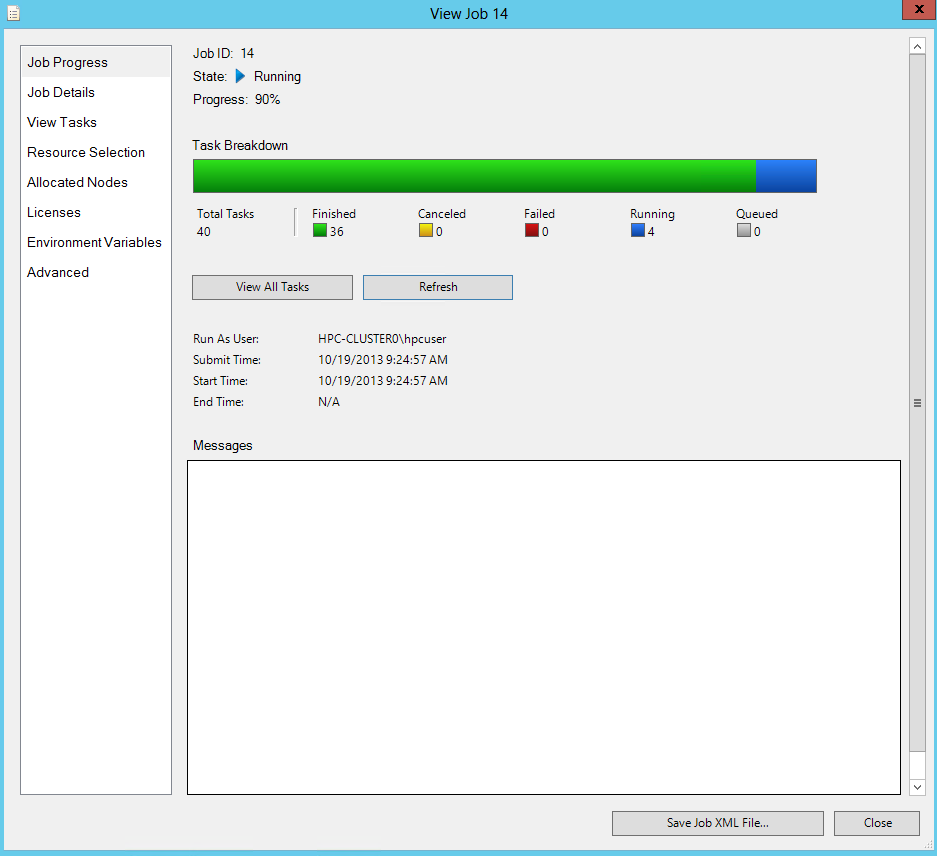
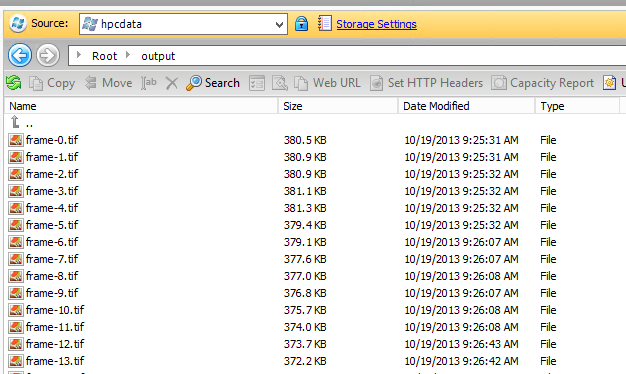
* 1. Download and install CloudBerry Explorer for Azure Blob Storage.
  2. Open CloudBerry Explorer for Azure Blob Storage from **Start** | **All Programs** | **CloudBerryLab** | **CloudBerry Explorer for Azure Blob Storage** | **CloudBerry Explorer for Azure Blob Storage**.
  3. Open the **File** menu and select **Azure Blob Storage Accounts**. The **Account Registration** dialog will appear, as shown in Figure 4:
     1. 
     2. Figure 4
     3. The Account Registration dialog
  4. Click the **Add** button and input the display name of the storage account, the storage account name, and the account’s shared key (primary access key). Use the same storage account settings you used in Task 1 for the **AzureBlobCopy** project.
  5. Click the **Test Connection** button and wait for the approval message. Close the approval message, click OK to add the storage account, and then close the **Azure Blob Storage Accounts** dialog.
  6. You should now see your blob storage in the left pane of the application, and your machine (“My Computer”) in the right pane.
  7. Create a new container in the blob by clicking on the **New Container** button in the left pane, as shown in Figure 5:
     1. 
     2. Figure 5
     3. Creating a new blob container
  8. In the **Create New Container** dialog, set the container name to *input*, select the **Full public read access** option from the Access control options as shown in Figure 6, and click **OK**.
     1. 
     2. Figure 6
     3. The Create New Container dialog
     4. **Note:** If the container already exists, it is preferable that you delete it and create it again using the above settings.
  9. Locate the newly created container in the list of containers and double-click its name to see its contents (it should be empty for now).
  10. In the right pane, navigate to the labs folder, and locate the **RenderMan\SampleInput** folder.
  11. Select all the files (40 .zip files) from the **SampleInput** folder and click the **Copy** button. Click **Yes** in the confirmation message that appears, and then wait for the copy procedure to complete.
      1. **Note:** The content of the folder is about 76MB, so this may take a couple of minutes, depending on your network bandwidth.
  12. After the upload completes, close the CloudBerry Explorer application.

1. Task 3 – Submitting the Job Manually Through the HPC Job Scheduler
   1. In this task, you will create a parametric sweep job for the image rendering application.
   2. In the **Cluster Manager** application, enter the **Job Management** section and click **New Job…** in the **Actions** pane as shown in Figure7:



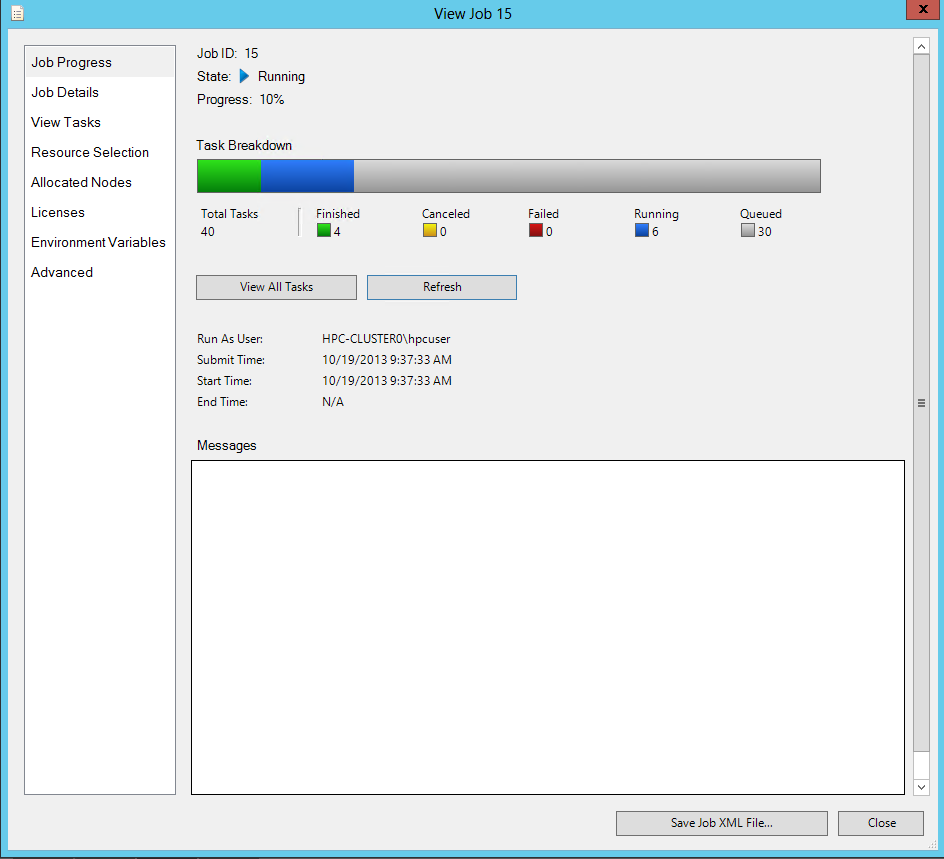
* + 1. Figure 7
    2. Create a New Job
  1. In the **New Job** dialog, set **the Job name** to *Image Rendering*, as shown in Figure 8:
     1. 
     2. Figure 8
     3. Setting the name of the job
  2. Still in the **New Job** dialog, click the **Edit Tasks** option, click the arrow next to the **Add** button, and then click **Parametric Sweep Task**…, as shown in Figure 9:
     1. 
     2. Figure 9
     3. New Parametric Sweep Task
  3. In the **Parametric Sweep Task** dialog, enter the following information and then click OK:
     1. Task name: ImageRendering
     2. Start value: 0
     3. End value: 39
     4. Command line: %CCP\_PACKAGE\_ROOT%\Aqsis\bin\run.cmd frame-\*
     5. Working Directory: %CCP\_PACKAGE\_ROOT%

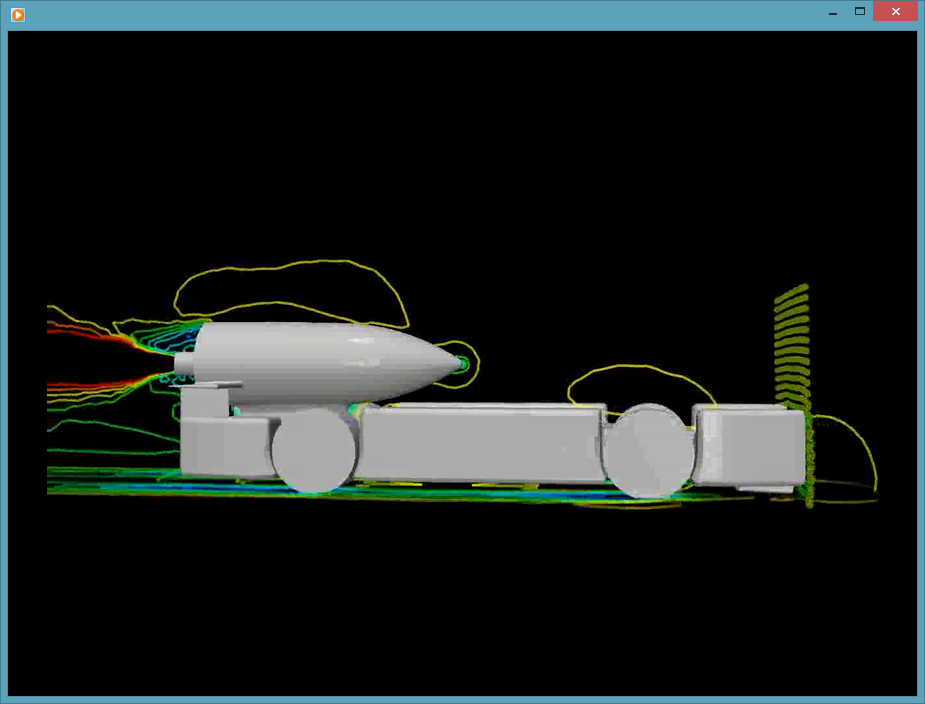


* + 1. Figure 10
    2. Setting the parametric sweep task details
  1. Click the **Resource** **Selection** option, check the **Run this job only on nodes in the following list:** checkbox, and then add the **machines you need,**  as shown in Figure 11:
     1. 
     2. Figure 11
     3. Selecting nodes for the job
  2. Click the **Submit** button to start the parametric sweep job.
  3. While the job is running, enter the **Job Management** section, select the Image Rendering job from the active jobs, and click on **View Job…** in the **Actions** pane, as shown in Figure 12:
     1. 
     2. Figure 12
     3. View running job information
  4. The **View Job** dialog displays the current state of the job, as shown in Figure 13:
     1. 
     2. Figure 13
     3. Running job’s state
  5. Click the **Refresh** button every couple of seconds to see the change in the number of finished tasks.
  6. After the job completes, open CloudBerry Explorer for Azure Blob Storage from **Start** | **All Programs** | **CloudBerryLab** | **CloudBerry Explorer for Azure Blob Storage** | **CloudBerry Explorer for Azure Blob Storage**.
  7. Use the CloudBerry Explorer to view the contents of the **output** blob container, as shown in Figure 14:
     1. 成都。。
     2. Figure 14
     3. Viewing the contents of the output blob container
  8. Double-click any of the .tif files to view the rendered image.

1. Task 4 – Running the Client Application

In this task, you will run the client application that creates the parametric sweep job.

* 1. **Note:** If you prefer to submit the job manually, you can skip this task
  2. Open the command prompt window from **Start** | **All Programs** | **Microsoft Visual Studio 2010** | **Visual Studio Tools** | **Visual Studio Command Prompt (2010)**.
  3. Navigate to the **ImageRendering** labs folder, and from there to **Source\RenderCmd\bin\debug**, and then run the following command to start the parametric sweep job:
     1. CMD
     2. RenderCmd ..\..\..\..\..\RenderMan\SampleInput
  4. While you wait for the application to finish rendering, you can inspect the status of the parametric sweep job using the HPC 2012 Cluster Manager utility, as shown in Figure 15:
     1. 
     2. Figure 15
     3. Status of the Parametric Sweep Rendering Job
  5. Wait a couple of minutes for the application to complete and then navigate to the **RenderMan\SampleInput\output** folder to inspect the rendered .tiff images.
  6. We can run the following command use ffmpeg.ext to merge those frames into a new video.
     1. CMD
     2. ffmpeg -f image2 -r 25 -i frame-%d.tif –pix\_fmt yuv420p movie.mp4



Summary

* 1. After running the Image Rendering sample, you should have learned the following:
  + How to work with Windows Azure blob storage.
  + How to package an application for Windows Azure.
  + How to deploy an application to Windows Azure nodes and to on-premises nodes.
  + How to submit a parametric sweep job from a .NET client application.
  + How to check a running job’s status.