Quickstart: Use .NET to create a Batch pool and run a job

- Article
- 04/28/2023
- 8 contributors

In this article

This quickstart shows you how to get started with Azure Batch by running a C# app that uses the Azure Batch .NET API. The .NET app:

- Uploads several input data files to an Azure Storage blob container to use for Batch task processing.
- Creates a pool of two virtual machines (VMs), or compute nodes, running Windows Server.
- Creates a job that runs tasks on the nodes to process each input file by using a Windows command line.
- Displays the output files that the tasks return.

After you complete this quickstart, you understand the key concepts of the Batch service and are ready to use Batch with more realistic, larger scale workloads.

Prerequisites

- An Azure account with an active subscription. If you don't have one, create an account for free.
- A Batch account with a linked Azure Storage account. You can create the accounts by using any of the following methods: Azure CLI | Azure portal | Bicep | ARM template | Terraform.
- Visual Studio 2019 or later, or .NET 6.0 or later, for Linux or Windows.

Run the app

To complete this quickstart, you download or clone the app, provide your account values, build and run the app, and verify the output.

Download or clone the app

Download or clone the Azure Batch .NET Quickstart app from GitHub. Use the following command to clone the app repo with a Git client:

Windows Command PromptCopy

```
git clone https://github.com/Azure-Samples/batch-dotnet-quickstart.git
```

Provide your account information

The app needs to use your Batch and Storage account names, account key values, and Batch account endpoint. You can get this information from the Azure portal, Azure APIs, or command-line tools.

To get your account information from the Azure portal:

1. From the Azure Search bar, search for and select your Batch account name.
2. On your Batch account page, select **Keys** from the left navigation.
3. On the **Keys** page, copy the following values:

- **Batch account**
- **Account endpoint**
- **Primary access key**
- **Storage account name**
- **Key1**

Navigate to your downloaded *batch-dotnet-quickstart* folder and edit the credential strings in *Program.cs* to provide the values you copied:

C#Copy

```
// Batch account credentials
private const string BatchAccountName = "<batch account>";
private const string BatchAccountKey  = "<primary access key>";
private const string BatchAccountUrl  = "<account endpoint>";

// Storage account credentials
private const string StorageAccountName = "<storage account name>";
private const string StorageAccountKey  = "<key1>
```

Important

Exposing account keys in the app source isn't recommended for Production usage. You should restrict access to credentials and refer to them in your code by using variables or a configuration file. It's best to store Batch and Storage account keys in Azure Key Vault.

Build and run the app and view output

To see the Batch workflow in action, build and run the application in Visual Studio. You can also use the command line `dotnet build` and `dotnet run` commands.

In Visual Studio:

1. Open the *BatchDotNetQuickstart.sln* file, right-click the solution in **Solution Explorer**, and select **Build**. If prompted, use NuGet Package Manager to update or restore NuGet packages.
2. Once the build completes, select **BatchDotNetQuickstart** in the top menu bar to run the app.

Typical run time with the default configuration is approximately five minutes. Initial pool node setup takes the most time. To rerun the job, delete the job from the previous run, but don't delete the pool. On a preconfigured pool, the job completes in a few seconds.

The app returns output similar to the following example:

OutputCopy

```
Sample start: 11/16/2022 4:02:54 PM

Container [input] created.
Uploading file taskdata0.txt to container [input]...
Uploading file taskdata1.txt to container [input]...
Uploading file taskdata2.txt to container [input]...
Creating pool [DotNetQuickstartPool]...
Creating job [DotNetQuickstartJob]...
Adding 3 tasks to job [DotNetQuickstartJob]...
Monitoring all tasks for 'Completed' state, timeout in 00:30:00...
```

There's a pause at `Monitoring all tasks for 'Completed' state, timeout in 00:30:00...` while the pool's compute nodes start. As tasks are created, Batch queues them to run on the pool. As soon as the first compute node is available, the first task runs on the node. You can monitor node, task, and job status from your Batch account page in the Azure portal.

After each task completes, you see output similar to the following example:

OutputCopy

```
Printing task output.
Task: Task0
Node: tvm-2850684224_3-20171205t000401z
Standard out:
Batch processing began with mainframe computers and punch cards. Today it still plays
a central role...
stderr:
...
```

Review the code

Review the code to understand the steps in the Azure Batch .NET Quickstart.

Create service clients and upload resource files

1. To interact with the storage account, the app uses the Azure Storage Blobs client library for .NET to create a BlobServiceClient.

   C#Copy

   ```
   var sharedKeyCredential = new
   StorageSharedKeyCredential(storageAccountName, storageAccountKey);
   string blobUri = "https://" + storageAccountName +
   ".blob.core.windows.net";

   var blobServiceClient = new BlobServiceClient(new Uri(blobUri),
   sharedKeyCredential);
   return blobServiceClient;
   ```

2. The app uses the `blobServiceClient` reference to create a container in the storage account and upload data files to the container. The files in storage are defined as Batch ResourceFile objects that Batch can later download to the compute nodes.

   C#Copy

   ```
   List<string> inputFilePaths = new()
   {
       "taskdata0.txt",
       "taskdata1.txt",
       "taskdata2.txt"
   };

   var inputFiles = new List<ResourceFile>();
   ```

```
foreach (var filePath in inputFilePaths)
{
    inputFiles.Add(UploadFileToContainer(containerClient,
inputContainerName, filePath));
}
```

3. The app creates a BatchClient object to create and manage Batch pools, jobs, and tasks. The Batch client uses shared key authentication. Batch also supports Azure Active Directory (Azure AD) authentication.

   C#Copy
   ```
   var cred = new BatchSharedKeyCredentials(BatchAccountUrl,
   BatchAccountName, BatchAccountKey);

    using BatchClient batchClient = BatchClient.Open(cred);
   ...
   ```

Create a pool of compute nodes

To create a Batch pool, the app uses the BatchClient.PoolOperations.CreatePool method to set the number of nodes, VM size, and pool configuration. The following VirtualMachineConfiguration object specifies an ImageReference to a Windows Server Marketplace image. Batch supports a wide range of Windows Server and Linux Marketplace OS images, and also supports custom VM images.

The `PoolNodeCount` and VM size `PoolVMSize` are defined constants. The app creates a pool of two Standard_A1_v2 nodes. This size offers a good balance of performance versus cost for this quickstart.

The Commit method submits the pool to the Batch service.

C#Copy

```
private static VirtualMachineConfiguration
CreateVirtualMachineConfiguration(ImageReference imageReference)
{
    return new VirtualMachineConfiguration(
        imageReference: imageReference,
        nodeAgentSkuId: "batch.node.windows amd64");
}

private static ImageReference CreateImageReference()
{
    return new ImageReference(
        publisher: "MicrosoftWindowsServer",
        offer: "WindowsServer",
        sku: "2016-datacenter-smalldisk",
```

```
        version: "latest");
}

private static void CreateBatchPool(BatchClient batchClient,
VirtualMachineConfiguration vmConfiguration)
{
    try
    {
        CloudPool pool = batchClient.PoolOperations.CreatePool(
            poolId: PoolId,
            targetDedicatedComputeNodes: PoolNodeCount,
            virtualMachineSize: PoolVMSize,
            virtualMachineConfiguration: vmConfiguration);

        pool.Commit();
    }
...
```

Create a Batch job

A Batch job is a logical grouping of one or more tasks. The job includes settings common to the tasks, such as priority and the pool to run tasks on.

The app uses the BatchClient.JobOperations.CreateJob method to create a job on your pool. The Commit method submits the job to the Batch service. Initially the job has no tasks.

C#Copy

```
try
{
    CloudJob job = batchClient.JobOperations.CreateJob();
    job.Id = JobId;
    job.PoolInformation = new PoolInformation { PoolId = PoolId };

    job.Commit();
}
...
```

Create tasks

Batch provides several ways to deploy apps and scripts to compute nodes. This app creates a list of CloudTask input `ResourceFile` objects. Each task processes an input file by using a CommandLine property. The Batch command line is where you specify your app or script.

The command line in the following code runs the Windows `type` command to display the input files. Then, the app adds each task to the job with the AddTask method, which queues the task to run on the compute nodes.

C#Copy
```
for (int i = 0; i < inputFiles.Count; i++)
{
    string taskId = String.Format("Task{0}", i);
    string inputFilename = inputFiles[i].FilePath;
    string taskCommandLine = String.Format("cmd /c type {0}", inputFilename);

    var task = new CloudTask(taskId, taskCommandLine)
    {
        ResourceFiles = new List<ResourceFile> { inputFiles[i] }
    };
    tasks.Add(task);
}

batchClient.JobOperations.AddTask(JobId, tasks);
```

View task output

The app creates a TaskStateMonitor to monitor the tasks and make sure they complete. When each task runs successfully, its output writes to *stdout.txt*. The app then uses the CloudTask.ComputeNodeInformation property to display the *stdout.txt* file for each completed task.

C#Copy
```
foreach (CloudTask task in completedtasks)
{
    string nodeId = String.Format(task.ComputeNodeInformation.ComputeNodeId);
    Console.WriteLine("Task: {0}", task.Id);
    Console.WriteLine("Node: {0}", nodeId);
    Console.WriteLine("Standard out:");

Console.WriteLine(task.GetNodeFile(Constants.StandardOutFileName).ReadAsString());
}
```

Clean up resources

The app automatically deletes the storage container it creates, and gives you the option to delete the Batch pool and job. Pools and nodes incur charges while the nodes are running, even if they aren't running jobs. If you no longer need the pool, delete it.

When you no longer need your Batch account and storage account, you can delete the resource group that contains them. In the Azure portal, select **Delete resource group** at the top of the resource group page. On the **Delete a resource group** screen, enter the resource group name, and then select **Delete**.

Next steps

In this quickstart, you ran an app that uses the Batch .NET API to create a Batch pool, nodes, job, and tasks. The job uploaded resource files to a storage container, ran tasks on the nodes, and displayed output from the nodes.

Now that you understand the key concepts of the Batch service, you're ready to use Batch with more realistic, larger scale workloads. To learn more about Azure Batch and walk through a parallel workload with a real-world application, continue to the Batch .NET tutorial.