**High Performance Computing in the Public Cloud with Microsoft Azure Batch**

**0.1: Abstract:**

Azure Batch is a first-party embedded service on Microsoft Azure that performs 2 major functions:

* Provision and management of compute resources
* Task submission execution and management

Azure Batch uses the concept of Pools when managing compute resources, and Jobs when managing collections of tasks. Compute resources within a pool are defined to exacting standards. Jobs specify a pool on which the tasks are to run, and allow for these tasks to be created either individually or in an automated fashion.

The following Lab & associated presentation will describe the features and capabilities of Azure Batch in greater detail and walk through the potential implementations of a workflow. Different Azure characteristics will be used to demonstrate the various capabilities, and do not necessarily represent a reference architecture or set of best practices.

Azure Batch is described fully on the [Microsoft Azure documentation site](https://docs.microsoft.com/en-us/azure/batch/).

The application that will be used in this Lab is [EnergyPlus](https://energyplus.net/), an open-source project developed by the US Department of Energy (DoE). EnergyPlus™ is a whole building energy simulation program that engineers, architects, and researchers use to model both energy consumption—for heating, cooling, ventilation, lighting and plug and process loads—and water use in buildings.

**0.2: Lab structure:**

Each participant for this Lab should have an Azure subscription created with sufficient regional VM SKU core quota to submit a job to cores across multiple compute node. The default Batch quota limits are 20 Dedicated & 20 LowPriority cores. This Lab guide will suggest the use of *Standard\_F4* VM size, using 5 nodes of both Dedicated & LowPriority to aggregate a total of 40 cores for task execution.

The Lab can be re-run using any Azure subscription, though attention will need to be paid to the intended scale of the jobs to ensure sufficient VM SKU core quota exists.

EnergyPlus is officially supported on Ubuntu 16.04, which is what this Lab will be based on.

There is an associated *lab-setup.sh* Bash script that automates the initial setup & configuration (steps 1.0 through to 2.0).

**1.0: Required artifacts:**

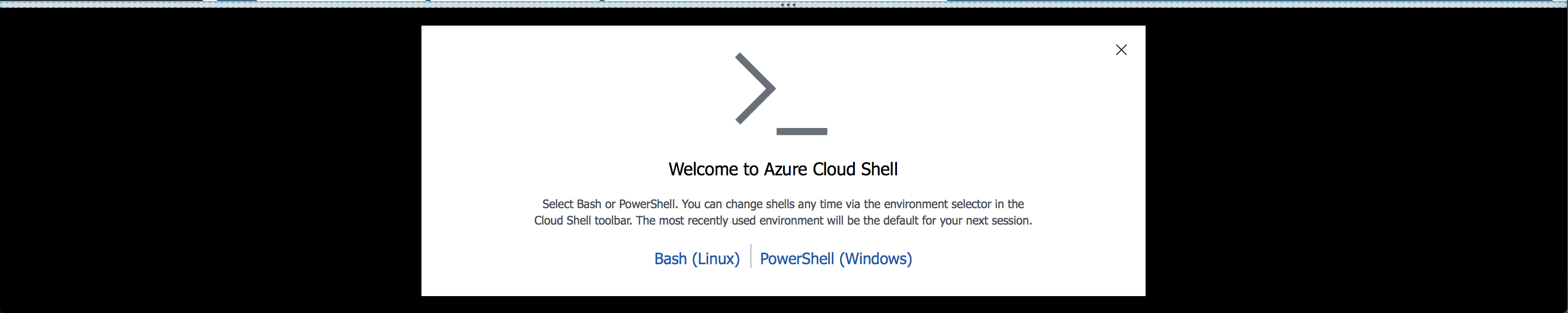
The workflow requires certain templates and configuration scripts. Such artifacts are available on the public GitHub repository (repo): <https://github.com/grandparoach/azure-batch/tree/master/EnergyPlus>

Open a web browser and navigate to <https://portal.azure.com>. Login using either the credentials provided or those associated with an Azure subscription.

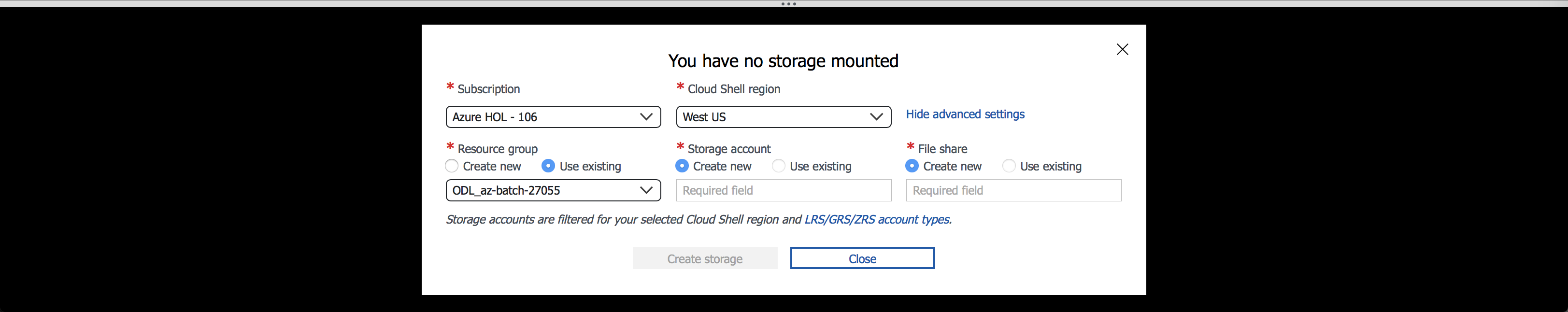
1.0.1:

The Azure Portal has a built-in feature called the [Azure Cloud Shell](https://docs.microsoft.com/en-us/azure/cloud-shell/overview), which offers a native CLI interface for both Bash and PowerShell in the portal browser window and is maintained by Microsoft.

The Azure Cloud Shell requires some initial configuration after logging into the Azure Portal for the first time. After opening the Cloud Shell, users will be prompted to setup the environment. For the purposes of this Lab, select Bash as the shell of choice:



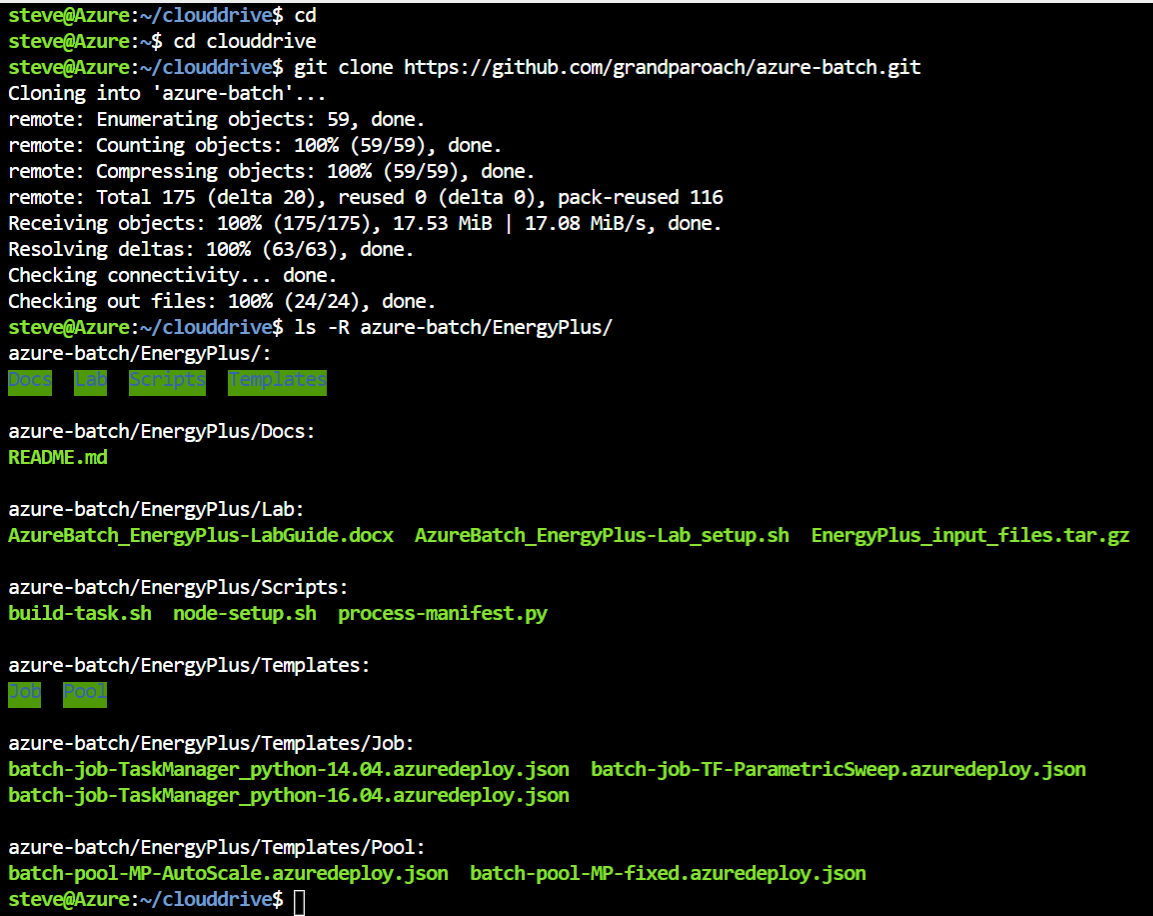
Subsequently, click the ‘*Show advanced settings*’ which will allow users to specify details for the Resource Group (RG), Storage Account (SA) and File share:



* + - The Resource Group (RG) field can be left to use the default existing RG
    - Specify a globally unique name for the Storage Account (SA)
    - Specify a name for the File share

1.0.2:

With the Cloud Shell environment initialized, the source *azure-batch* repository that will be used for this Lab can be cloned using Git:



In addition to setting up the environment, inputs are required to use the EnergyPlus application. Inputs, schedules, weather and IDF files along with a manifest are contained in a blob tarball which is contained in the repo clone.

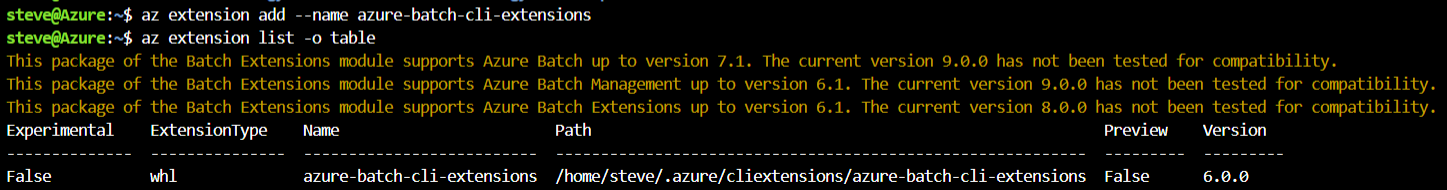
Extract the contents from the tarball to the local directory as indicated by running this tar command:



1.0.4:

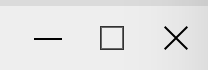
This Azure CLI is pre-installed on the Azure Cloud Shell and includes commands to create and manage all Azure services, e.g. Azure Batch.

The advanced features of Azure Batch are enabled through the use of [Azure Batch CLI Extensions](https://github.com/Azure/azure-batch-cli-extensions). These extensions, however, are not pre-installed on the Azure Cloud Shell, thus it’s a necessary prerequisite:



(Do not worry about the warnings in yellow)

**1.1: Environment preparation:**

Minimize the Azure Cloud shell (top-right of the Cloud Shell window pane): 

1.1.1:

Within the Azure portal, complete the following tasks using unique naming:

* Create a Resource Group (RG)
* Create a Storage Account (SA), within the previously created RG (as above)

The following attributes are acceptable for this Lab:

* + - **Deployment model:** Resource Manager
    - **Kind:** General Purpose v2
    - **Replication:** Locally Redundant Storage (LRS)
    - **Performance:** Standard
    - **Access tier:** Hot
    - **Secure transfer:** Disabled
    - **Virtual Networks:** Disabled

The following [Quickstart guide](https://docs.microsoft.com/en-us/azure/storage/common/storage-quickstart-create-account?tabs=portal) walks through the process of doing both steps. All Azure resources can be created and managed via the portal or CLI. This Lab will use both, although it is simply user preference as to which method is used at any stage.

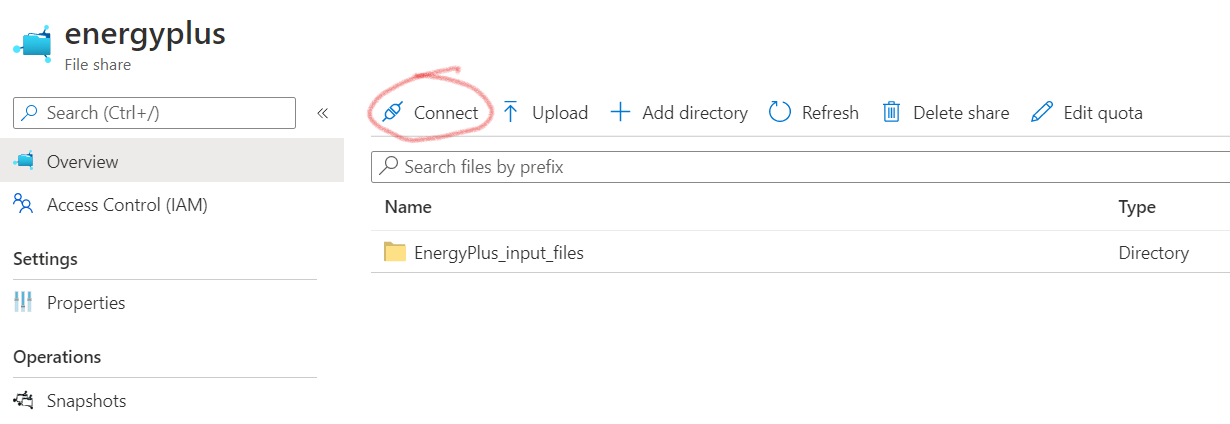
1.1.2:

Once the deployment succeeds, navigate to the SA blade and select ‘Files’:



Within the File service, create a File share using a unique name, and set the quota to 1024GB. [This article](https://docs.microsoft.com/en-us/azure/storage/files/storage-how-to-create-file-share#create-file-share-through-the-azure-portal) walks through the steps of creating a File share in the Azure portal (steps 1-3 are required only).

N.B. There is a ‘*Connect*’ button within the file share that opens a blade with the command to mount the share to a node:



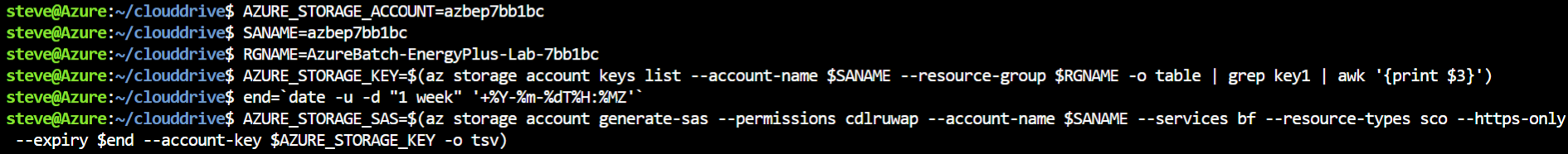


Re-open the Azure Cloud Shell, which can be done by clicking on the **>\_** icon again (in the top-right navigation pane).

1.1.3:

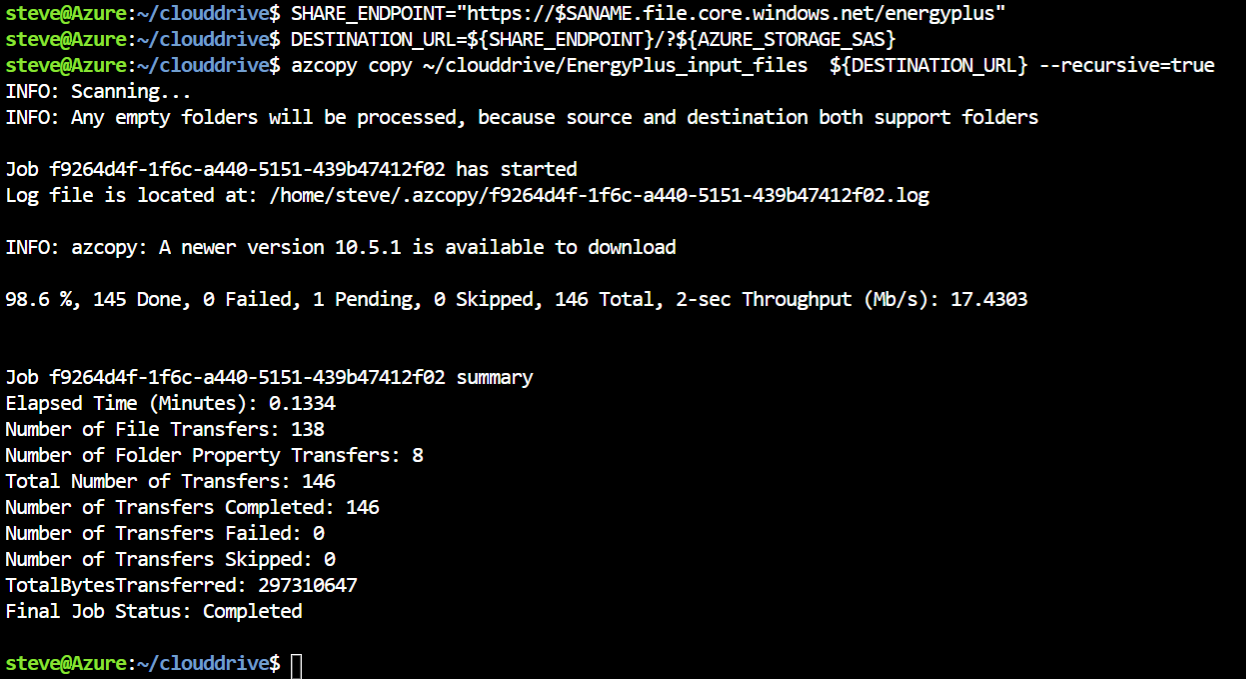
We will use the Azure CLI to upload the input files (downloaded to the Azure Cloud Shell in step 1.0.2) to the Azure File Share (created in step 1.1.2) using [AzCopy](https://docs.microsoft.com/en-us/azure/storage/common/storage-use-azcopy-linux).

Initially, we need to retrieve the Storage Account keys, and then generate a SAS token which can be done using the Azure CLI:



1.1.4:

Using AzCopy to upload the files from the *EnergyPlus\_input\_files/* directory to the newly created Azure Files directory:



1.1.5:

Using the keys listed in step 1.1.3, create a SA Blob container using the Azure CLI:



This container will be used to store:

* *node-setup.sh* --- a script that will be run as a Start Task on each compute node that joins a Pool
* *build-task.sh* --- a script that will run as part of the Job to create the tasks to be executed

Compute nodes provisioned will automatically download and run *node-setup.sh* as part of a *startTask* command. When a Job runs, each compute node will automatically download *build-task.sh* to a shared directory which will be used during task execution.

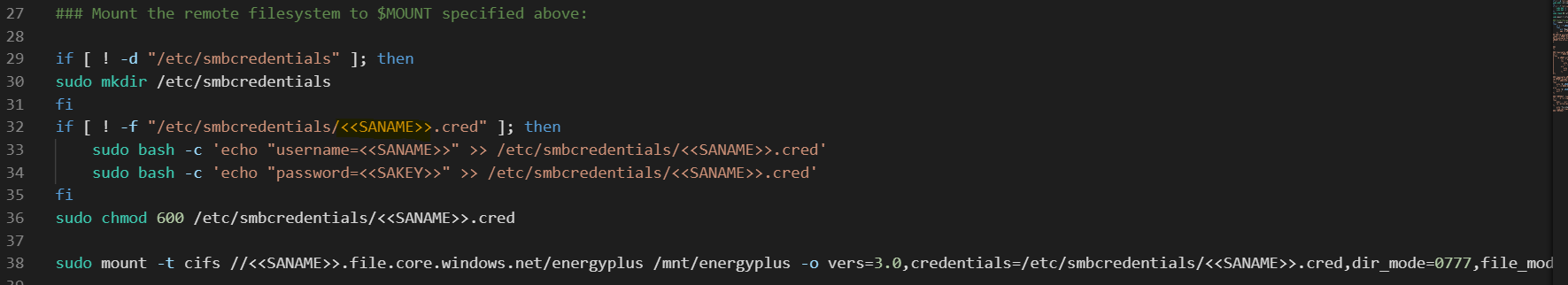
1.1.6:

Prior to uploading the setup files, the *node-setup.sh* requires one line of update to include the mount command for the file share that will be used to read the input data (as created in step 1.1.2).

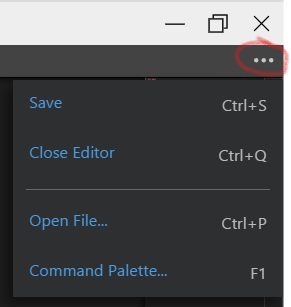
To update the script, open the file using a text editor (Visual Studio Code is included in the Cloud Shell, so it will be used in this Lab, although any preferred editor can be used):



In the section where the remote filesystem is mounted (lines 27 through 39) all of the <<SANAME>> tags need to be replace with the Storage Account name, and the <<SAKEY>> tag needs to be replaced with the Storage Account Key:

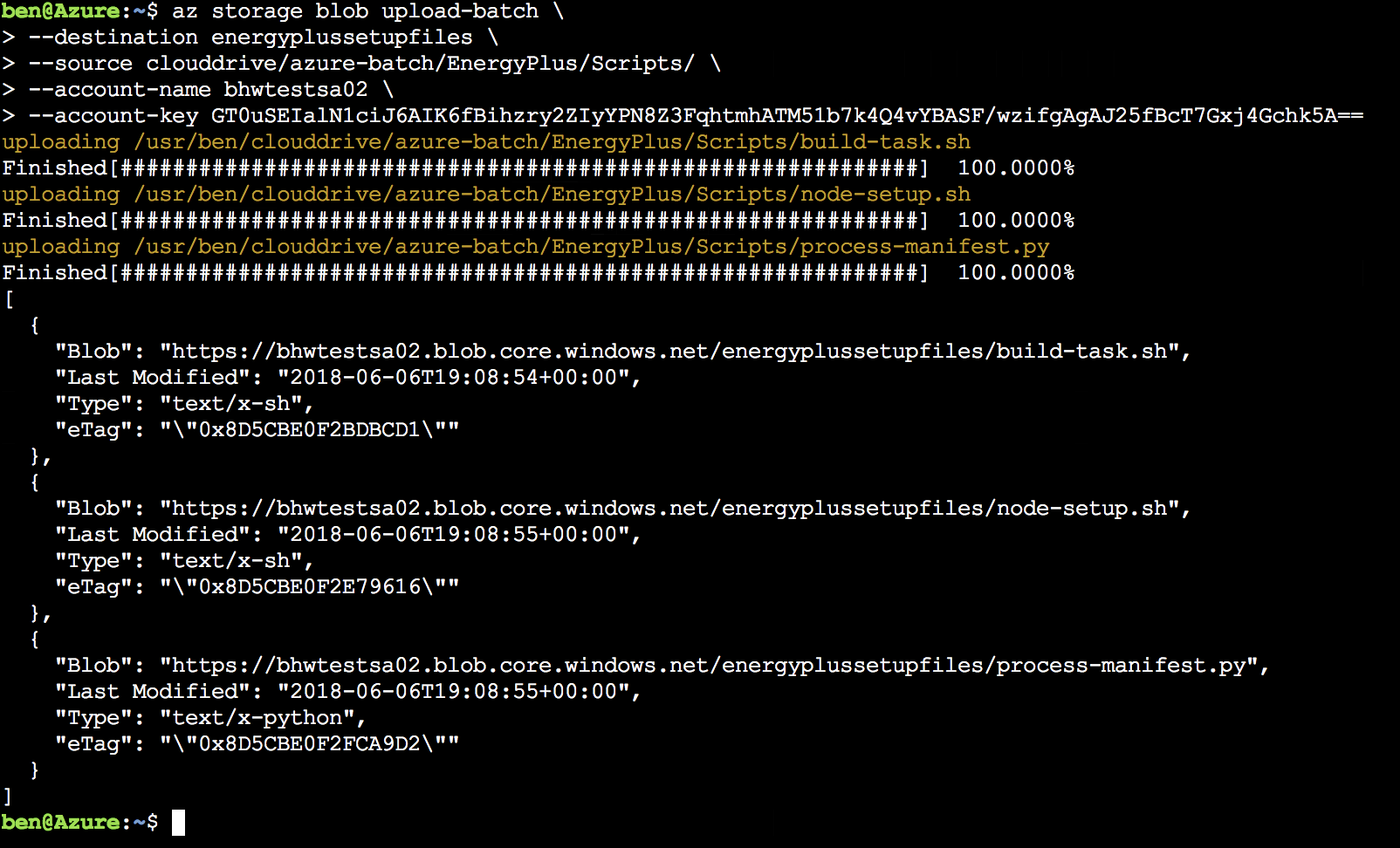


Finally, to save the file, click on the “…” in the top right corner, and select “Save”



1.1.7:

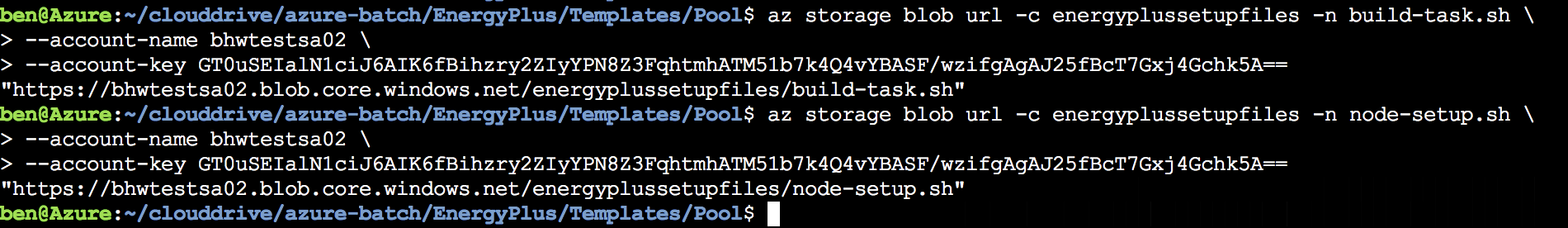
Upload these setup files from the *azure-batch/EnergyPlus/Scripts* directory to the newly created Blob container:



Once uploaded to the SA container, the files now exist as blobs and are accessible to download for our compute nodes.

1.1.8:

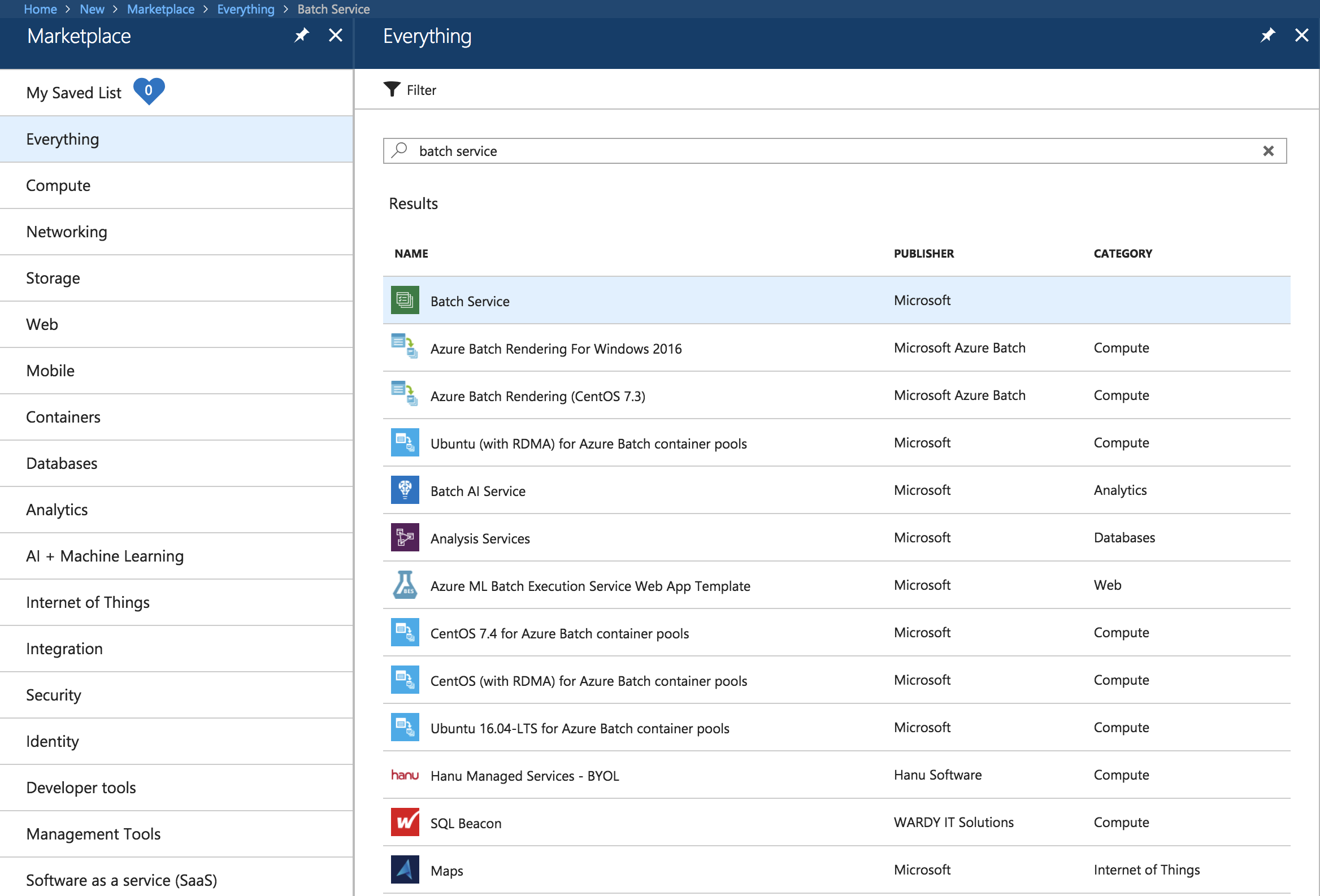
List the blob URLs for the files recently uploaded to the SA blob container:



1.1.9:

To use Azure Batch, an Azure Batch account needs to be created.

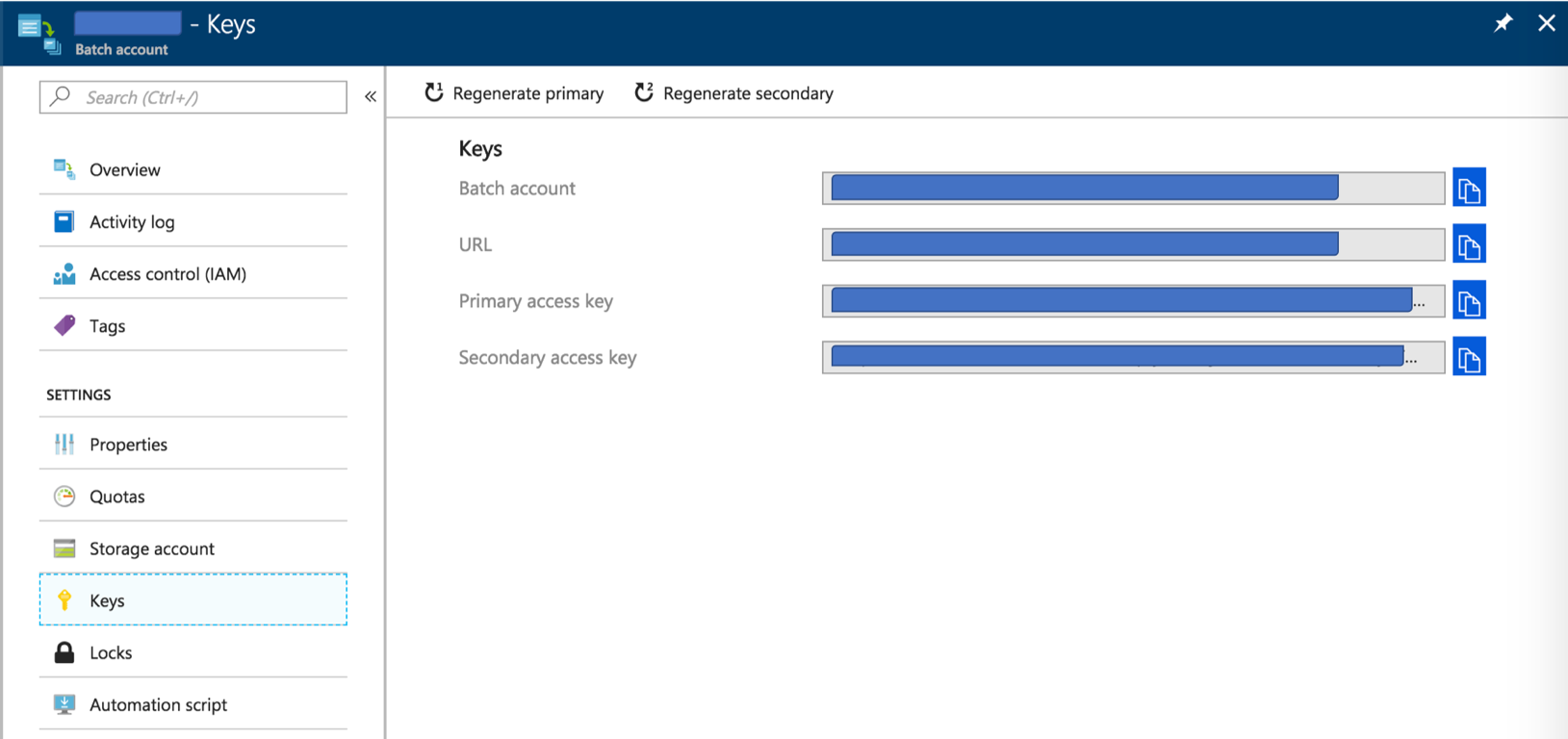
In the Azure portal, click ‘Create a Resource’, and type ‘Batch service’ into the Marketplace search:



Highlight the Azure Batch selection (as per the screengrab above) and click ‘Create’ on the adjacent blade as it opens. Provide values and inputs for the required fields (use RG & SA resources as created in step 1.1.1):

* + - **Account Name:** Specify a unique value
    - **Subscription:** [Use default selection]
    - **Resource Group:** Use previously created RG
    - **Location:** Choose the same location as used for the RG and SA
    - **Storage account:** Choose the SA previously created
    - **Pool allocation mode:** Batch service

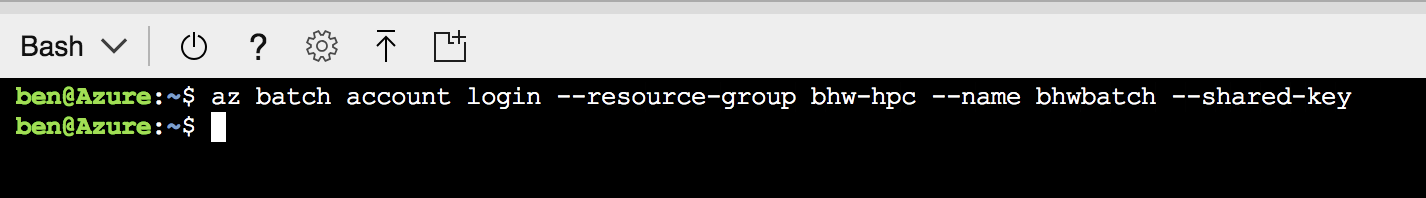
Once the Azure Batch account deployment succeeds, open the resource and pay reference the Batch account keys which are required to authenticate to the service:



1.1.10:

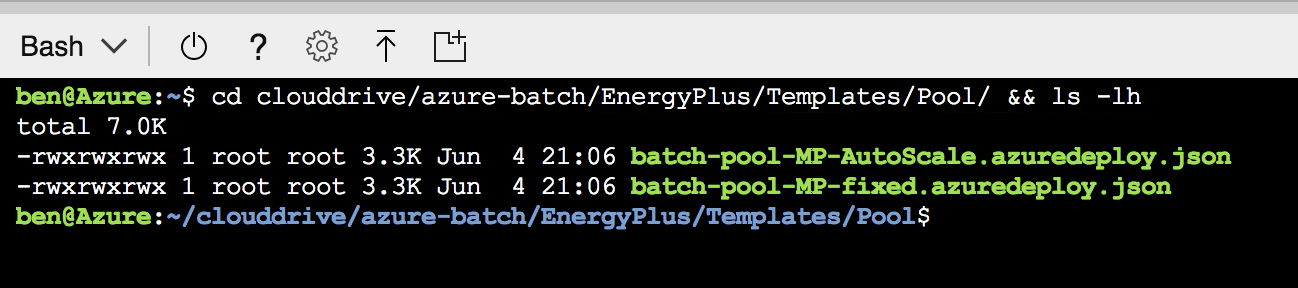
In order to effectively use the Azure Batch extensions, it’s necessary to perform an Azure Batch service login specifying the Azure Batch account previously created in step 1.1.7.

Open the Azure Cloud Shell, and enter:



**2.0: Creating the Azure Batch resources**

Within the cloned GitHub repository, navigate to the directory that contains the Azure json templates for Azure Batch Pool creation:



The templates contained here both create a Pool of compute nodes within an Azure Batch account. The only difference between templates is whether the Pool created will have a fixed number of compute nodes provisioned, or whether an [Autoscale formula](https://docs.microsoft.com/en-us/azure/batch/batch-automatic-scaling) will provision nodes as required. This Lab will use the template with a fixed number of nodes for simplicity, given the number of tasks being run in known in advance.

The templates contained within the cloned GitHub repository require completion before use. Azure templates are JSON files and can be edited with any text editor. This Lab will continue to assume the use of *vi*.

2.0.1:

Open the template file using:

*$ vi batch-pool-MP-fixed.azuredeploy.json*

N.B. The color scheme used with *vi* in the Azure Cloud Shell is not particularly easy on the eyes. For a more readable color scheme, type in sequence:

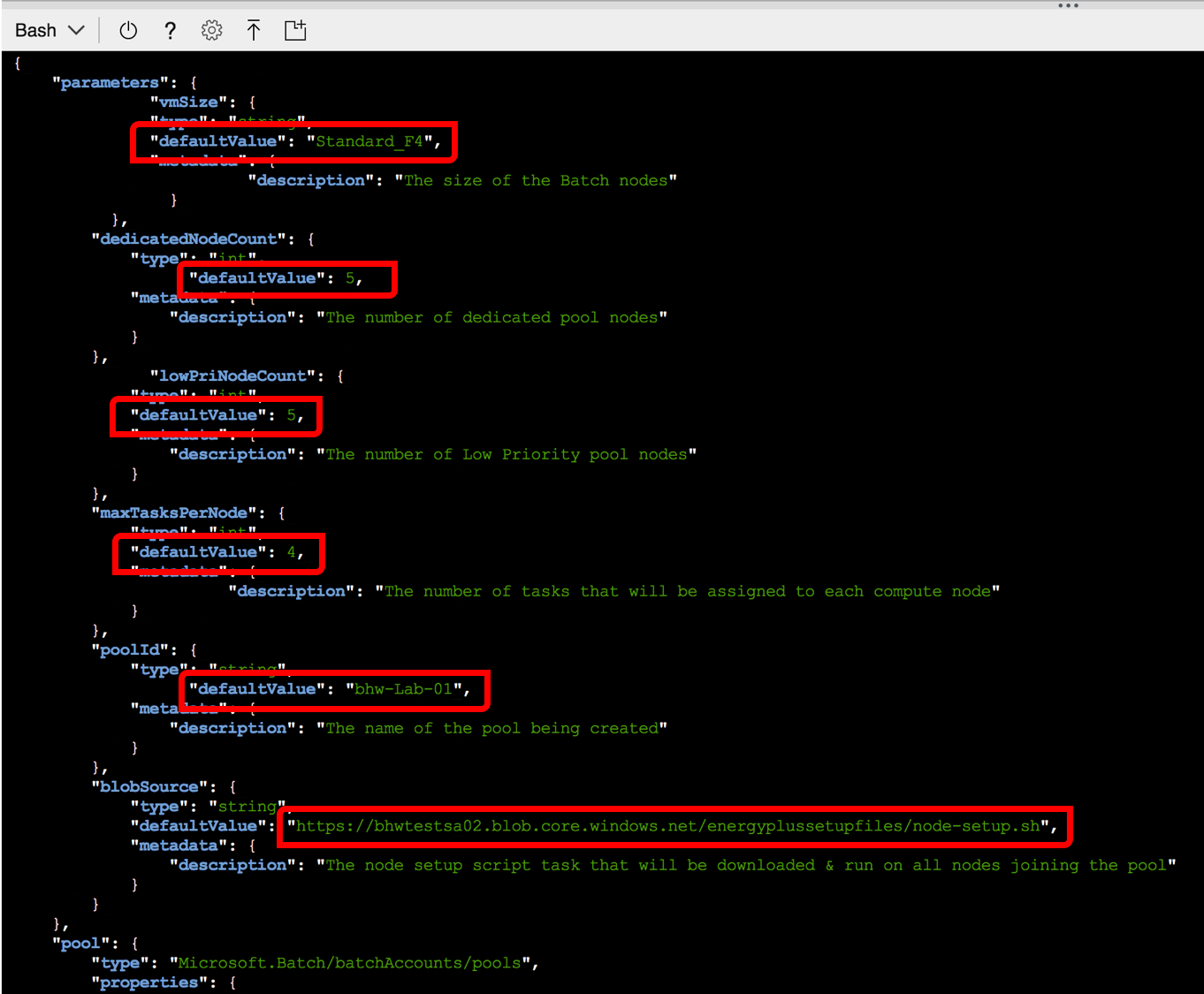
[ESC] *: colo delek*

And pressing ‘enter’ (or ‘return’) on the keyboard once. The color scheme should adjust to the one reflected below.

Amend the subsections of the ‘parameters’ section to reflect the desired values. The image below identifies the sections requiring updates.

For the purposes of the Lab, we will specify:

* + - vmSize: *Standard\_F4*
    - dedicatedNodeCount: 5
    - lowPriNodeCount: 5
    - maxTasksPerNode: 4
    - poolId: A unique name (within the Batch environment)
    - blobSource: The URL of the *node-setup.sh* script, as per step 1.1.7

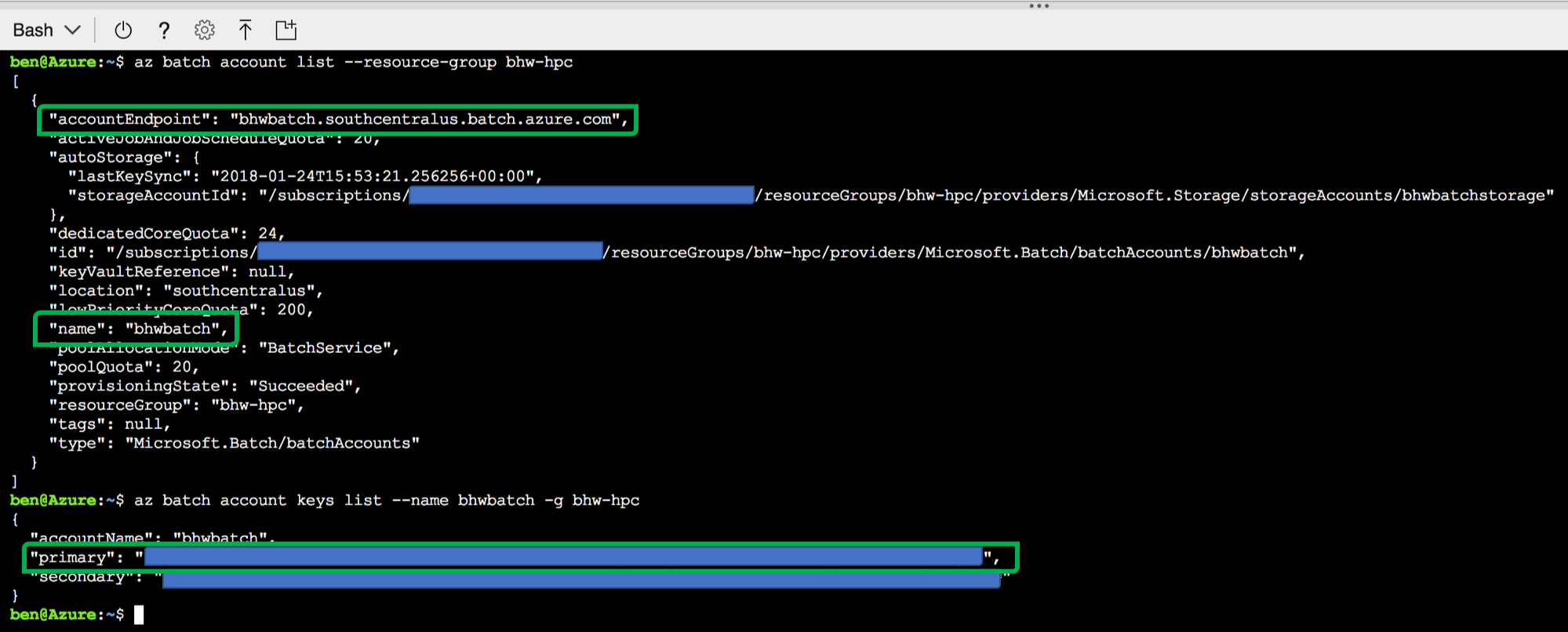


Once the template file has been updated, to save the file (‘w’ = ‘write’) and quit (‘q’ = ‘quit’) enter:

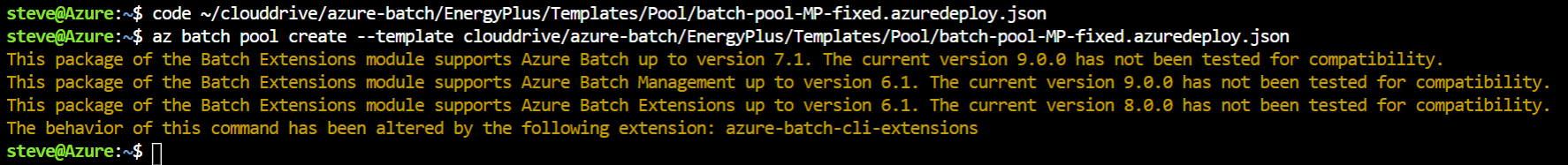
*[ESC] : wq*

2.0.2:

The Azure CLI can expose the Batch account name, Batch account endpoint and Batch account keys:



To create a Pool in the Azure Batch account, the template (recently modified in step 2.0.1) will be used with the Azure Batch CLI extensions feature:



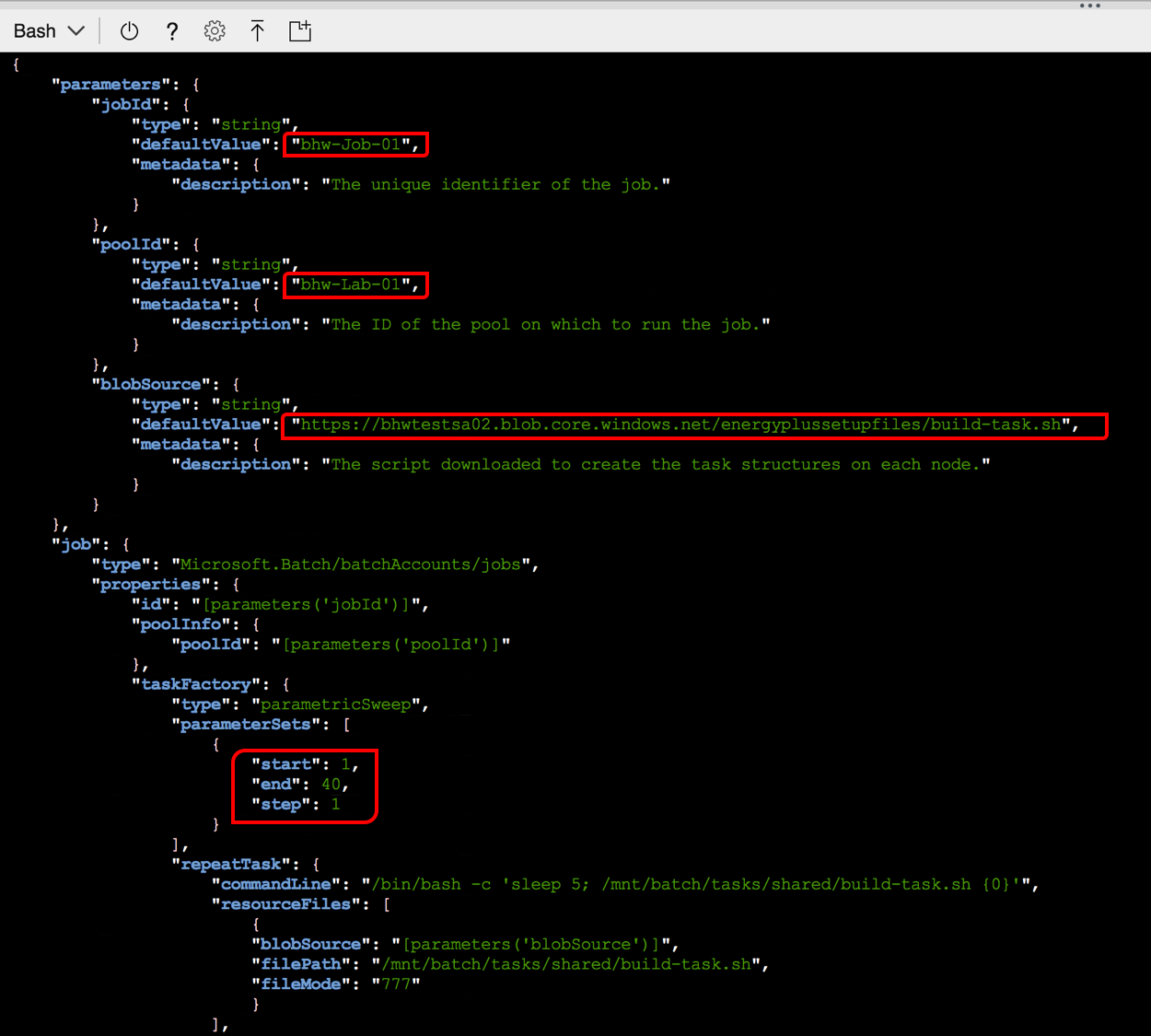
**2.1: Creating an Azure Batch Parametric Sweep Job**

With the Pool created (per step 2.0.3) and nodes provisioned & sitting in an idle state, a Job can be created to host a collection of Tasks.

This phase of the Lab will focus on using the Parametric Sweep capability within [Azure Batch’s TaskFactory](https://github.com/Azure/azure-batch-cli-extensions/blob/master/doc/taskFactories.md) feature. The Task Factories allows for tasks to be created within a Job based on a specific input, e.g. specified tasks, number of files in a target storage account container, or an incrementing value.

2.1.1:

Open the Job template file using *vi*, and modify the parameter sections outlined as follows:

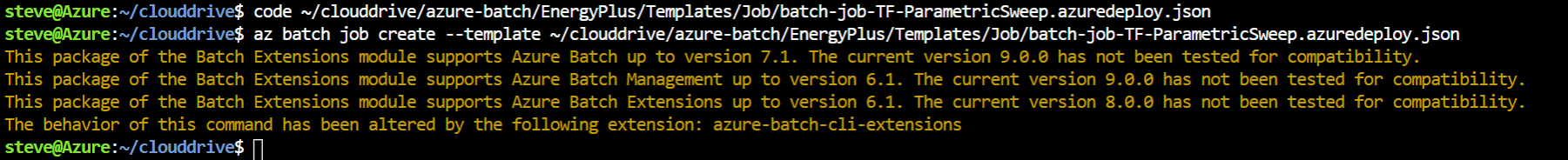


Some commentary on the properties & values in the Job template:

* + - jobId: A unique name (within the Batch environment) for the Job
    - poolId: The name of the Azure Batch Pool, as defined in section 2.0.1
    - blobSource: The URL of the *build-task.sh* script uploaded to blob storage
    - parameterSets: The start, end and step (interval) increments for the sweep
      * start: 1
      * end: 40
      * step: 1

2.2.2:

The modified Job template file can be submitted through the Azure CLI using the Azure Batch CLI extensions:

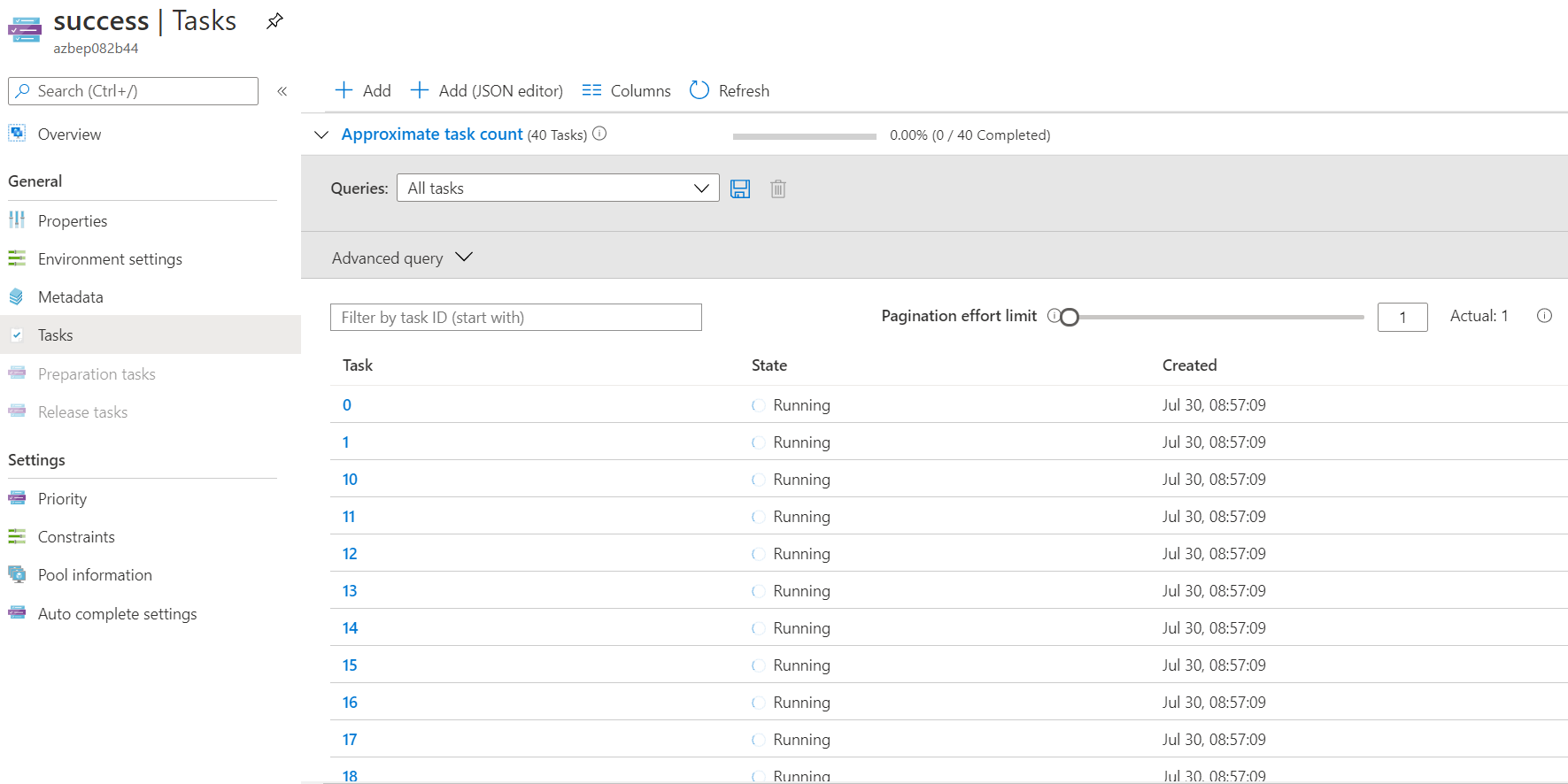


This Job creation command submits the Job to the Azure Batch service, and requests that the TaskFactory generates tasks based on the values contained within the Job template file. For the purposes of this Lab, 40 tasks have been generated (one task per input file in the File Share), where each Task has a command referencing a different line in the *manifest.txt* file centrally stored in the file share (40 lines will be read).

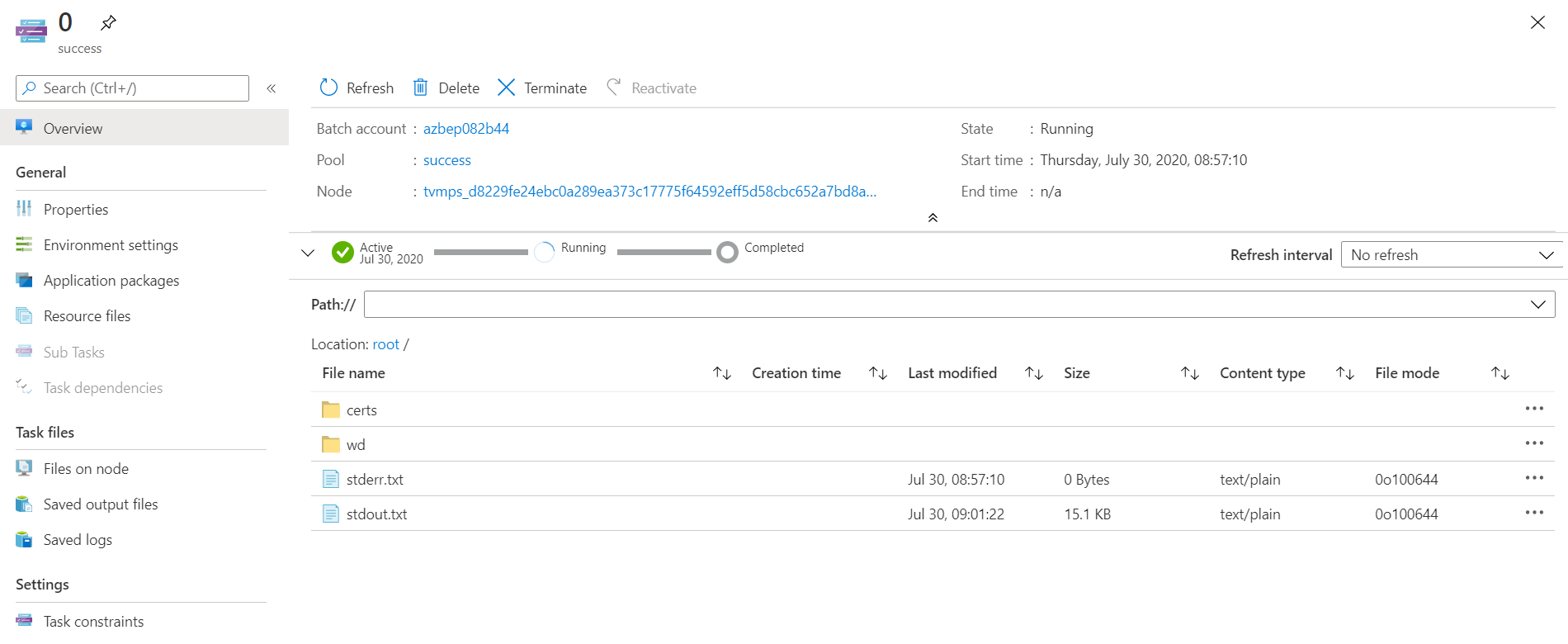
2.2.3:

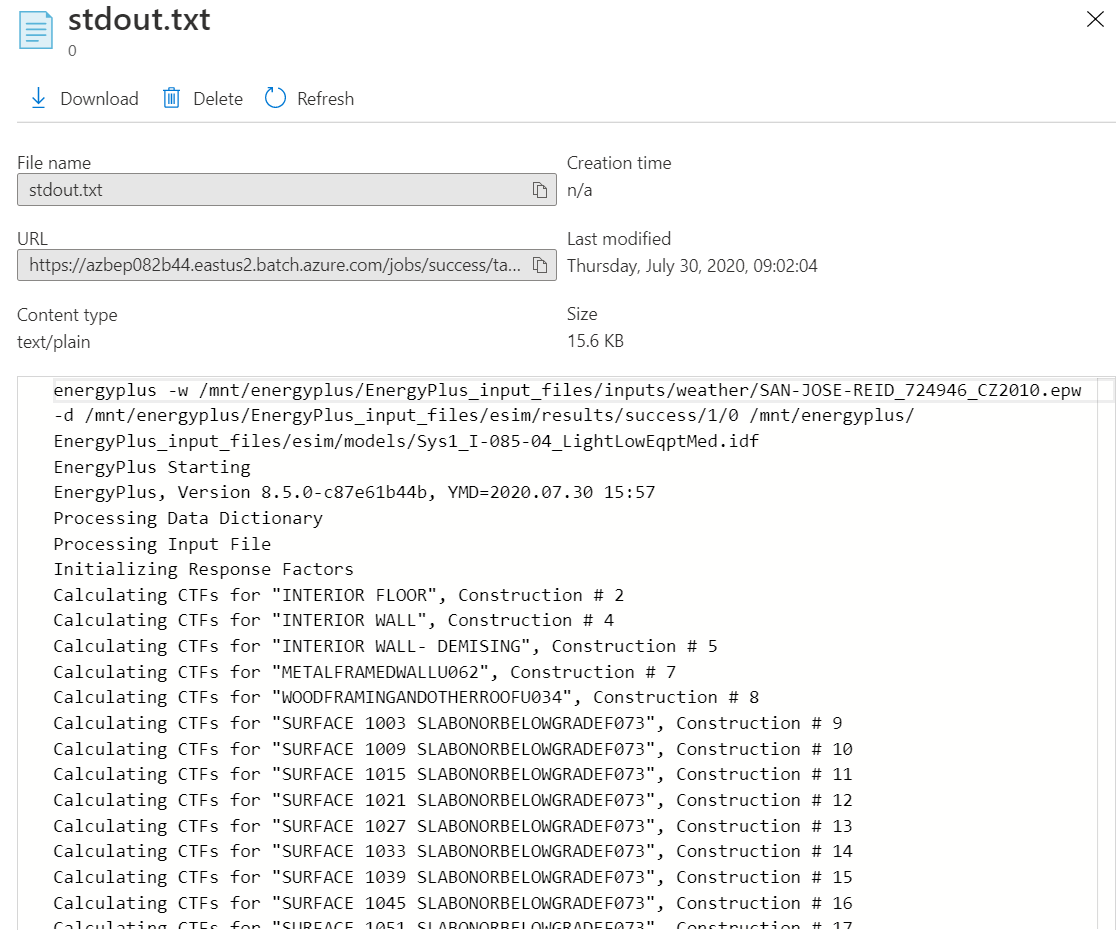
Once the job is underway, Task progress can be seen from the Azure Portal:





Click on the individual tasks to view Task-specific details, including the console output generated:



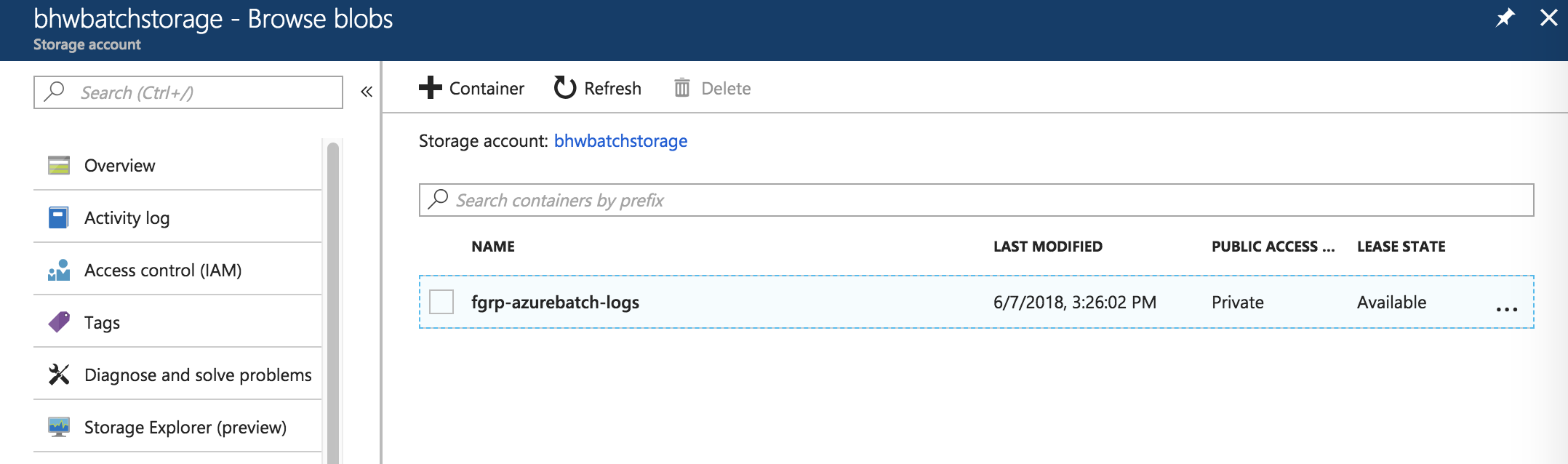


2.2.4:

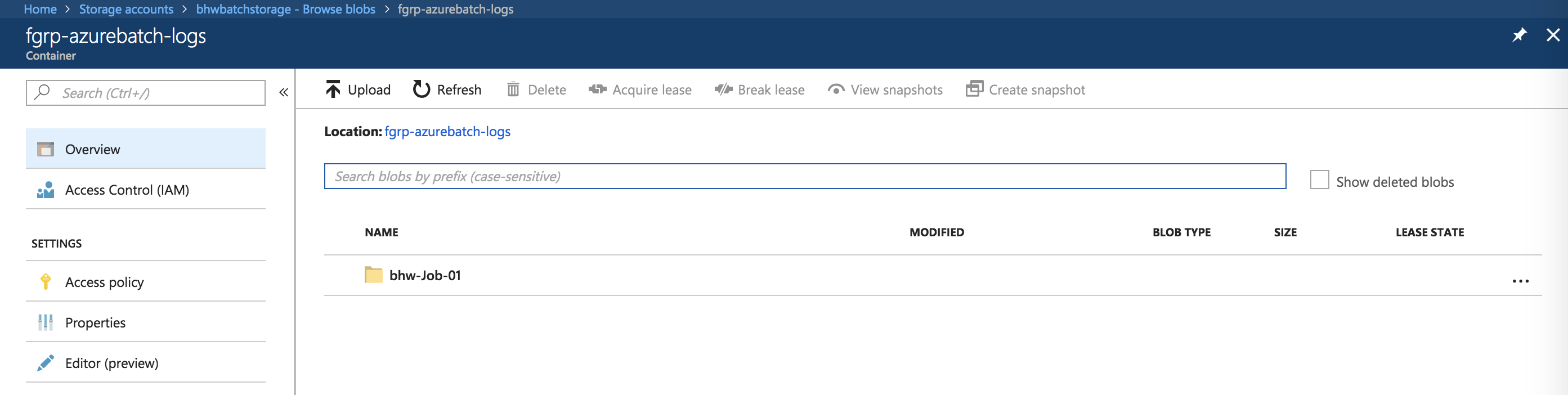
As part of the Azure Batch Job template, the Task logs have been uploaded to Azure Blob storage using the *“outputFiles”:* File Group capability of Batch’s TaskFactory feature:



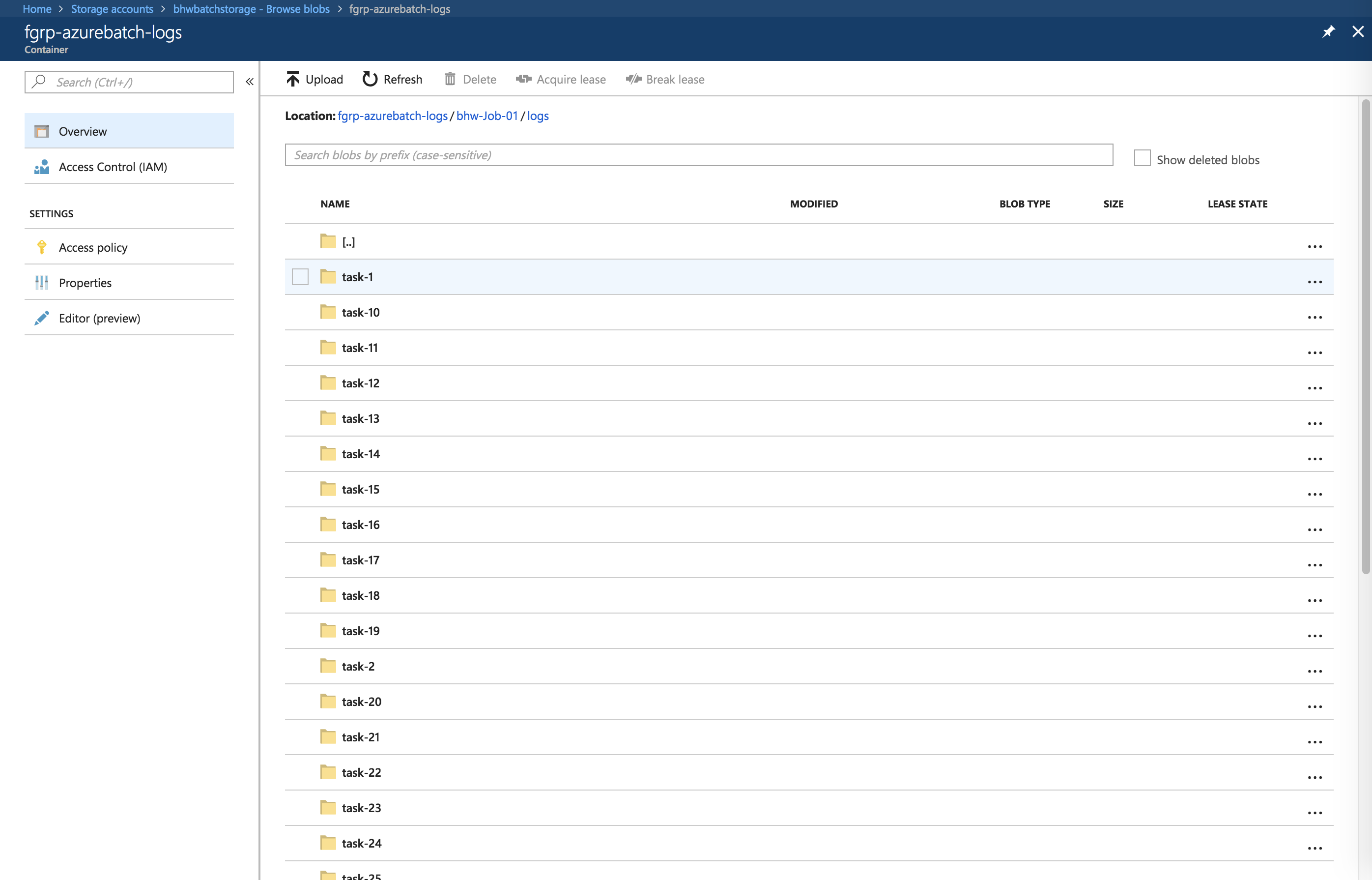
Browsing the Azure Storage account shows the container created:

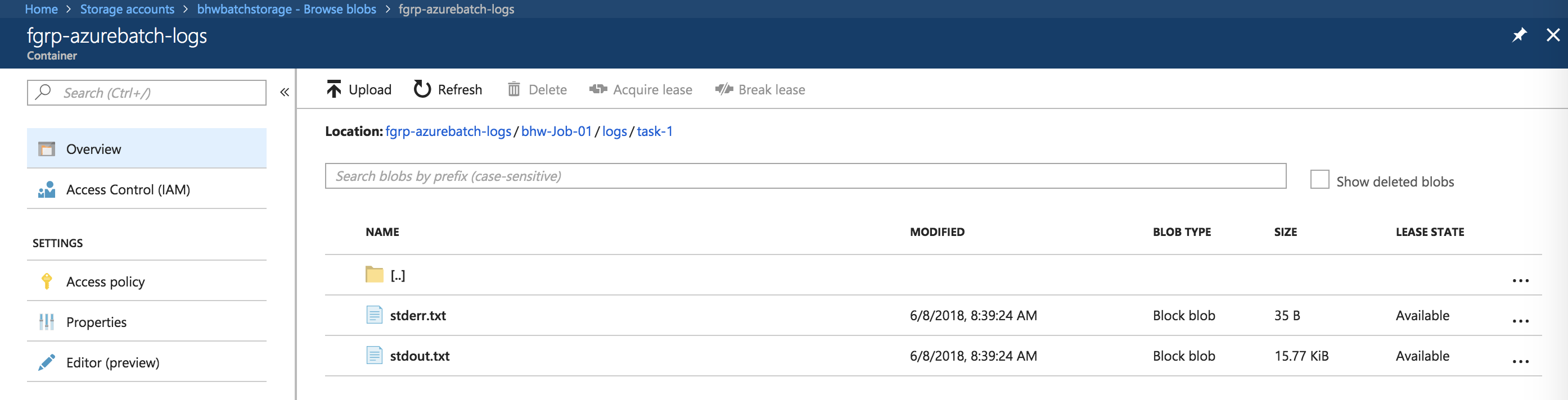


And the subdirectory hierarchy, as defined in the *“path”:* property from the Job template:

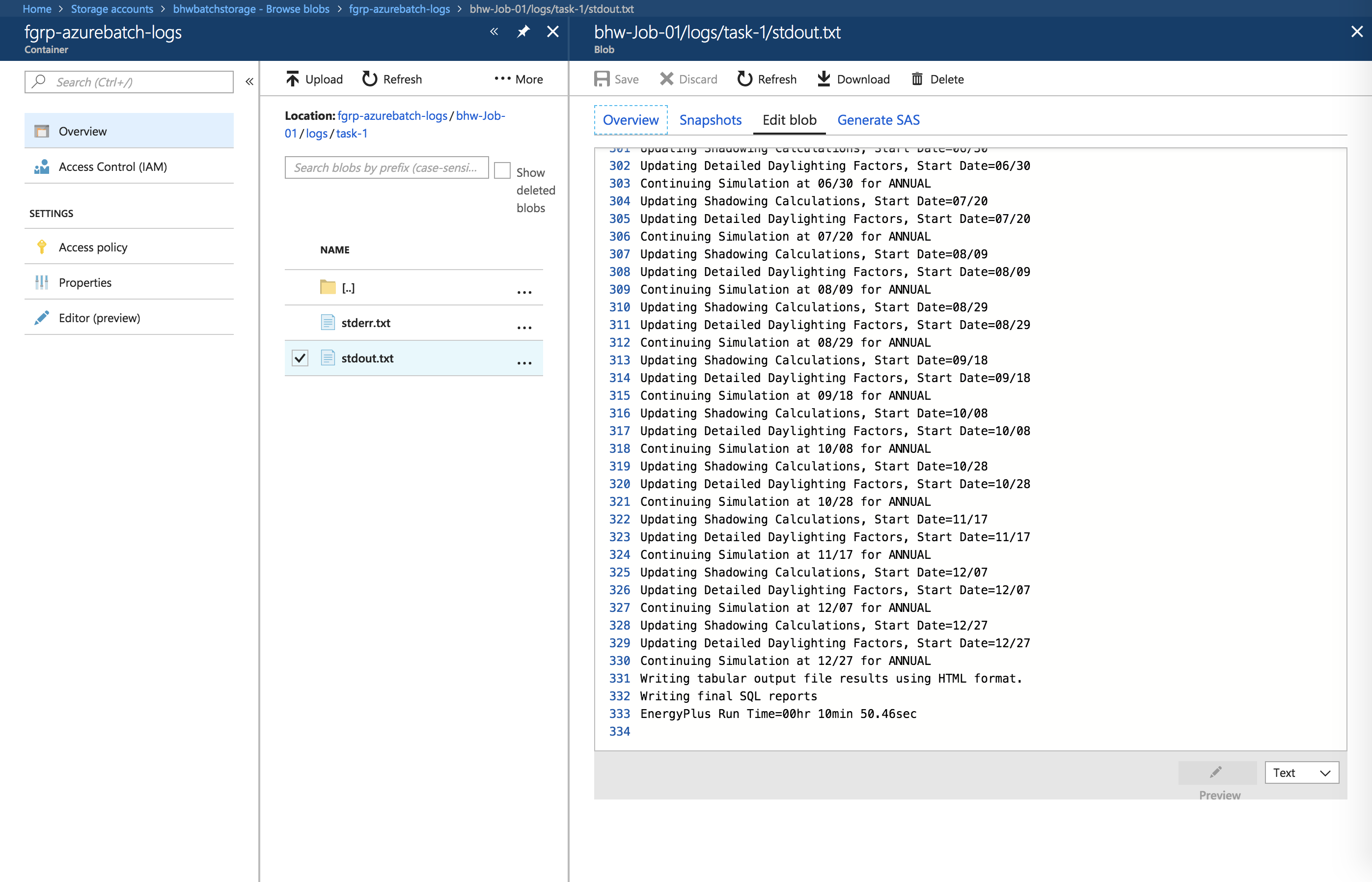








Clicking on the ellipsis (‘…’) on the far-right of the blob allows for blob-level operations. One such option is ‘*View/Edit blob’*, which results in the contents of the text file being displayed in the Azure Portal:



**Appendix:**

BatchExplorer

Batch Explorer is a tool to manage your Azure Batch accounts. The goal is to implement a great user experience that will help you debug, monitor and manage your pools, jobs and tasks. It will also include experimental features such as Batch Templates in the aim to improve your Batch experience.

BatchLabs is updated monthly with new features and bug fixes and can be downloaded for Windows, MacOS and Linux on the website:

<https://github.com/Azure/BatchExplorer>