Using block storage in IBM PureApplication System pattern workloads

Bhadri Madapusi
Advisory Software Developer
IBM

IBM PureApplication System 2.0 supports a new kind of storage volume called block storage. Unlike storage support in previous versions, block storage decouples storage from the lifecycle of the workload encapsulated in the patterns. The storage persists after the workload is deleted, and can be attached to a different workload. This tutorial shows you how to use block and shared block storage in PureApplication System pattern workloads.

Introduction

IBM® PureApplication® System 2.0 introduces block storage, which decouples storage from the lifecycle of the workload and can be shared by multiple virtual machines. A block storage add-on lets you take advantage of block storage in your pattern workloads, but you need to use a REST API to exploit shared block storage. This tutorial describes block storage features and shows specifically how you can use them in your pattern workloads.

Block and shared block storage volume

Block storage volumes are raw volumes of storage with no formatting or partitions. On PureApplication System Intel® racks, block storage uses VMWare® Physical Mode Raw Device Mapping (pRDM) to bypass the Virtual Machine File System (VMFS) and map a storage Logical Unit Number (LUN) directly to a virtual machine (VM). In pRDM, all SCSI commands from the VM are passed directly to the LUN and storage size can be over 2 TB. On PureApplication System Power racks, a block storage volume is treated like RAW storage. On both PureApplication System Intel and Power racks, block storage volumes can be as large as 8TB.

You create block storage by using PureApplication System storage infrastructure (internal block storage), or by using existing external storage infrastructure (external block storage). You can attach block storage to and detach it from pattern workloads. You can also replicate internal block storage across racks. On PureApplication System Intel racks, you can attach block storage to multiple VMs, and this shareable block storage is called shared block storage.

The advantage of using block and shared block storage in PureApplication System is that the storage lifecycle is decoupled from the pattern workloads that use of storage. You can create a
block or shared block storage independent of the workload, and attach it to the workload during or after the workload is deployed. You can also detach it from the workload while the workload is running. The storage persists when the workload is deleted, even if the workload is deleted while the storage is attached to the workload.

Here are some typical use cases for block storage:

- Reuse the data in the storage by activating a new workload when the existing workload goes down and is not recoverable
- Reuse the data in the storage on a different workload
- Share the data in the storage (shared block) among multiple workloads at the same time

### Using block storage in pattern workloads

PureApplication System 2.0 provides a block storage add-on called Default attach block disk for AIX® and Linux® workloads, and Default windows attach block disk for Microsoft® Windows® workloads. You can use this add-on to attach an existing block storage to a workload during deployment, or to create a new block storage and attach it to the workload during deployment. In addition to attaching or detaching storage, the add-on can discover (cleanup) devices, and format and mount or unmount storage to or from an input mount point. Post deployment, you can detach the attached block storage, and reattach a different new or existing block storage. The add-on does not support shared block storage, and therefore you must use the block storage REST API discussed below to work with shared block storage.

**Figure 1. Default attach block disk add-on**

![Default attach block disk add-on](image)

Figure 1 shows the Default attach block disk add-on as found in the add-on catalog. The add-on has following input fields (environment variables get translated to input fields when added to the workload):

- **VOLUME_ID**
  
  UUID of an existing block storage. It is called the runtimeid in the block storage REST API. Required when attaching an existing disk.
Using block storage in IBM PureApplication System pattern workloads

You can add the block storage add-on to Virtual System Pattern (VSP) workloads and Virtual System Pattern Classic (VSP Classic) workloads. With the Virtual Application Pattern (VAP), you must use the block storage REST API discussed below to attach and detach the block storage. The block storage add-on support for VSP workload is much more robust than for VSP classic. The following sections describe the add-on support in each of the pattern workloads.

Using block storage add-on in VSP workloads

The block storage add-on can be added to a VSP using the Pattern Builder. Figure 2 shows VSP with block storage add-on:
Figure 2. VSP workload with default attach block disk add-on

When the block storage add-on is added to a VSP in the Pattern Builder, you can input the value for mount point and file system type. The rest of the add-on values will be entered during the distribution stage of deployment, since they depend on the placement location of the workload. Figure 3 shows this process. During deployment, you can either create and attach a new block storage, or attach an existing block storage:
During deployment, if a value is provided for mount point, the add-on formats the storage if it is not already formatted, and then mounts it to the appropriate mount point in the VM. For the VSP deployed in Figure 3, the add-on would format the storage to ext3 format and mount the storage to the directory /data, as shown in Figure 4. If the mount point value is empty, then the storage is attached to the VM node, but it will not be discovered and mounted to the node -- you will have to connect to the VM and mount it yourself.
Once deployed, you can view the attached block storage in the VSP instances page: Navigate to the VSP instances page, select the deployed instance, expand the VM perspective, and then expand the VM node to which the block storage is attached. From the instance page, you can detach the block storage and reattach a different block storage. Here again you have an opportunity to create a new block storage and attach it. Figure 5 shows post-deployment storage operations for the VSP deployed in Figure 3. Using post-deployment attach or a Create and attach operation, you can only reattach storage and mount it to those mount points that were defined using block storage add-on at pattern deployment time. You cannot create an ad-hoc mount point from the deployed instance page and attach and mount a storage to this ad-hoc mount point.

Using block storage add-on in VSP Classic workloads

As with VSP, the block storage add-on can be added to the VSP Classic part using the VSP Classic Pattern Editor. As with VSP, you can enter the mount point and file format value in the Pattern Editor. The rest of the values must be entered while you deploy the VSP Classic workload.
At deployment time you can choose to select an existing block storage or create a new block storage and attach them. One key difference between VSP and VSP classic is that in VSP Classic, there are no post-deployment operations on block storage. You cannot detach or reattach the block storage after it is deployed. With VSP Classic, once attached, block storage behaves like RAW storage. Figure 6 shows the VSP Classic workload creation and deployment with block storage add-on.

**Figure 6. VSP Classic workload with Default attach block disk add-on**

Using block storage in VAP workloads

As mentioned before, to exploit block storage in your VAP workload, you must use the block storage REST API, which are explained in the next section. Use of the REST APIs is explained using script packages. You can use the same technique in your VAP lifecycle python scripts to attach block storage during VAP workload deployment, and also in the operation python scripts to detach and reattach block storage from the VAP workload instance console.

**Important:** In IBM PureApplication Software, you cannot use the create and attach option in VSP and VSP Classic. IBM PureApplication Software is an on-premises version of PureApplication System, announced in Feb. 2015.
Block storage REST API

Block storage REST API provides support to:

- List PureApplication System block and shared block storage
- Attach and detach block and shared block storage
- Monitor status of attach and detach operations
- List storage attached to a workload node

As mentioned above, a typical use case for the block storage REST API is to attach or detach block storage to a VAP workload, or attach or detach shared block storage to a VAP or VSP workload. Here are the requirements and restrictions on using block storage REST calls:

- Code executing the REST API must have access to the maestro python module.
- The REST request sent to the server must have the appropriate security header, which can be constructed using the maestro module API. Listing 1 shows how to construct and execute a cURL call with the necessary headers using the maestro module API.
- You can only attach volumes to or detach volumes from workloads in the deployment that is executing the REST call.

For more information on the maestro python module, see Plug-in development kit and Plug-in development guide in the PureApplication System Knowledge Center.

Listing 1. Executing REST call using maestro module

```python
import maestro
...
#Get user token
user_token=None
try:
    if maestro.operation:
        user_token=maestro.operation.get('user_token')
except:
    logger.debug('executeREST no user_token')
#Get security header
securityHeader=None
if(user_token != None):
    securityHeader = maestro.get_authn_header(user_token)
else:
    securityHeader = maestro.get_authn_header()
#Get the pem file from SSL options
sslOpts = os.environ.get("CURL_SSL_OPTION")
garb, sslOpts = sslOpts.split()
#Output file to save the response.
outputFile = '/tmp/resp_file'
cmd = ["-X", str(method), "-H", '%s' %securityHeader,
"-H", "Content-Type:application/json",
"-H", "-v", "-o", outputFile, "--url", str(url), "--cacert", sslOpts]
#Execute cURL command using maestro API
rc = maestro.pcurl.main(cmd)
```

Table 1 below summarizes the REST resources that you can call to list all PureApplication System block storage, attach PureApplication System block storage, detach PureApplication System block storage, monitor the status of attach or detach operations, and retrieve the storages used by a VM workloads.
workload node. For a detailed description of these resources, see Block Storage REST API in the PureApplication System Knowledge Center.

Table 1. Block Storage REST resource

<table>
<thead>
<tr>
<th>Resource</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET operation on /services/resources/sharedDisks</td>
<td>List PureApplication System block storage.</td>
</tr>
<tr>
<td>PUT operation on /services/resources/sharedDisks</td>
<td>Attach/Detach block storage to/from a VM workload node in current deployment.</td>
</tr>
<tr>
<td>GET operation on /services/resources/tasks/{taskId}/deployments/{depUUID}/virtualMachines/{vmNodeId}/sharedDisks/{ sharedDiskId }</td>
<td>Monitor block storage attach/detach operation status.</td>
</tr>
<tr>
<td>GET operation on /services/resources/virtualMachines/{vmNodeId}/deployments/{depUUID}/sharedDisks</td>
<td>List block storages attached to a VM workload node.</td>
</tr>
</tbody>
</table>

Using shared block storage in workloads

This section shows you how to put together the REST API listed in Table 1 to attach (detach) shared block storage to (from) pattern workloads. The same procedure can be used for block storage. The flow diagram in Figure 7 shows the REST API flow to attach and detach block storage:

Figure 7. Flow diagram to attach or detach block and shared block storage

Attach storage to workload node flow

As shown in Figure 7, the attach flow has the following steps:

1. Get the name of the storage to be attached to the workload node.
2. Check if the storage exists (GET operation on services/resources/sharedDisks).
3. Attach the storage to the workload node (PUT operation on services/resources/sharedDisks).
4. Monitor attach operation status (GET operation on /services/resources/tasks/{taskId}/deployments/{depUUID}/virtualMachines/{vmNodeId}/sharedDisks/{ sharedDiskId }).
The attach flow is centered around Step 3 -- attach PureApplication System block storage to a workload node. The required input for this request (PUT operation on services/resources/sharedDisks) is:

runtimeid
Unique identifier of the storage. This value can be retrieved using GET operation on services/resources/sharedDisks to retrieve all the storages and then filter the result based on the input storage name.

deploymentId
Current deployment identifier. This information can be obtained using the maestro module API in Listing 2.

vmId
Current workload VM node identifier. This information can also be obtained using the maestro module API in Listing 2.

opType
Operation type value attach.

**Listing 2. Code to retrieve current deployment ID and VM node ID**

```bash
# Get current deployment identifier
deployment_id=maestro.node['deployment.id']
# Get current VM node identifier
vmId = maestro.node['id']
```

Listing 3 shows a sample input for attach REST request:

**Listing 3. Input for attach REST request**

```json
{
    "deploymentId": "d-b3daf24c-5f08-45a1-aa97-5a3536ec55c3",
    "vmIds": ["OS_Node_1.1142654931727"],
    "runtimeid": "0b21e087-0d4e-4494-a5c2-be07e0d7f7b8",
    "opType": "attach"
}
```

The response JSON for attach REST request contains a unique identifier for the attach operation (taskId) and a unique identifier for the storage (sharedDiskId). Listing 4 shows a sample response for attach REST request:

**Listing 4. Response for attach operation**

```json
{
    "taskId": 4650,
    "sharedDiskId": 49,
    "diskId": 141,
    "vmIds": ["storehouseupload2.11402609366120"],
    "deploymentId": "d-ad1459a9-82d4-4121-84f8-a2715aa2ab8f"
}
```

The attach operation is an asynchronous operation, so its status must be monitored at regular intervals to determine whether it has failed or succeeded. You can monitor attach operation status using the response returned by the monitor attach operation status REST call shown below:
GET operation on /services/resources/tasks/{taskId}/deployments/{depUUID}/virtualMachines/{vmNodeId}/sharedDisks/{sharedDiskId}

The value for deployment identifier (depUUID ) and workload VM node identifier (vmNodeid) can be obtained as shown in Listing 2. The value for taskId and sharedDiskId can be obtained from the response to the attach REST call (Listing 4). The monitor attach operation status call returns one of the following statuses: ATTACHING, ATTACHED, or ATTACH_FAILED. If the status is ATTACHED, it also returns the LUN identifier of the storage volume. If you plan to mount the device to the workload node immediately after the attach operation, you can use this LUN identifier to identify a SCSI drive (/dev/sd*) and mount the device to a mount point on your workload node. Listing 5 shows a sample response for monitor attach operation status:

Listing 5. Response for monitor attach operation status request

```json
{
"currentstatus": "ATTACHED",
"lunId": "60050768028503D3080000000000018B"
}
```

**Detach storage from workload node flow**

As shown in Figure 7, the detach flow has the following steps:

1. Get the name of the storage that is to be detached from the workload node.
2. Check if the storage is attached to the workload (GET operation on /services/resources/virtualMachines/{vmNodeid}/deployments/{depUUID}/sharedDisks?runtimeid={runtimeid}).
3. Detach the storage from the workload node (PUT operation on services/resources/sharedDisks).
4. Monitor detach operation status (GET on operation /services/resources/tasks/{taskId}/deployments/{depUUID}/virtualMachines/ {vmNodeid}/sharedDisks/ { sharedDiskId })

Steps 3 and 4 of the detach flow are similar to those for the attach flow. The difference in Step 3 is that the opType request value for detach operation is detach instead of attach. In Step 4, the difference is that the response value for monitor detach operation status call is either DETACHING, DETACHED, or DETACH_FAILED. A key difference between the attach flow and detach flow is that in the detach flow, you need to verify if the storage is attached to the workload node before detaching it, which you can do using the following REST call:

GET operation on /services/resources/virtualMachines/{vmNodeid}/deployments/{depUUID}/sharedDisks?runtimeid={runtimeid}

The value for deployment identifier (depUUID ) and workload VM node identifier (vmNodeid) can be obtained as shown in Listing 2. The value for the runtimeid can be retrieved using a GET operation on services/resources/sharedDisks to retrieve all the storages and then filter the result based on the input storage name. If the storage is attached to the workload node, then the call to verify if the storage is attached to the workload node returns a response with storage details and currentstatus attribute set to ATTACHED. Listing 6 shows a sample response for this call:
Listing 6. Response for retrieve storages attached to workload node request

```json
[

{
"shared": "T",
"virtualmachineid": 488,
"lunId": "60050768028593088000000000124",
"runtimeid": "e60a2db9-f5ae-4b86-b6bc-9351d3272562",
"type": "ext4",
"currentstatus": "ATTACHED",
"size": "1024",
"id": 110,
"mount": "/test",
"updated": 1402080762629,
"created": 1402080728268,
"name": "NewDisk",
"diskid": "30"
}
]
```

Working with the sample

You can download the sample code at the bottom of this tutorial. It has three script packages: one to attach PureApplication System block storage to a node (BlockStorageAttach-1.0.0.0.zip), one to detach PureApplication System block form a node (BlockStorageDetach-1.0.0.0.zip), and one to list the storages attached to a node (BlockStorageList-1.0.0.0.zip). Each of these script packages contains a cbscript.json file that defines the script package, and a python file (attach.py in BlockStorageAttach-1.0.0.0.zip, detach.py in BlockStorageDetach-1.0.0.0.zip, and list.py in BlockStorageList-1.0.0.0.zip) that drives the block storage REST API calls. To import these samples into PureApplication System, select Catalog => Script Packages => Create New. After they are imported, you can add them to your VSP Pattern node as shown in Figure 8:

Figure 8. VSP Pattern with imported sample script packages

The script package is configured to execute when started manually using the Execute link displayed next to the script name for a VM node, so you can leave the default values for the...
script package input as-is and deploy the pattern. The deployed pattern VM workload node will have these three scripts listed with them, as shown in Figure 9. The execution output of the script package is captured in remote_std_out.log.

Figure 9. Deployed VSP Pattern with sample script packages

![Figure 9](image)

Testing the sample script packages

To test the sample script package, click **Execute** next to the script package name. When you execute the BlockStorageAttach script package, you should see the input dialog shown in Figure 10. The BlockStorageAttach script package takes the following input:

**Block Storage Name**
- Name of the block storage to attach to this node. The code in attach.py retrieves the corresponding storage identifier (runtimeid) to be used in the attach REST call.

**File System Type**
- File system format type of the block storage. If provided, this information can be used after attachment to format the storage.

**Mount Point**
- If provided, this information can be used to mount the storage to a mount point on the node.

**Type of storage**
- Value can be Block or Shared Block. This information is used to filter the block storage list that is retrieved to get the storage identifier (runtimeid).

**Attach if the shared block storage is already used**
- True if you want to attach a disk that is currently attached to another node, otherwise false. This value is considered only if the type of storage is Shared Block.
Figure 10 shows the input for attaching a shared block storage named BM03 to the current workload node even if it is already attached to another workload node. The inputs that are provided to the script package are stored in the environment variables. The script attach.py that drives the BlockStorageAttach script package retrieves the input value from the environment variable using the code shown in Listing 7:

Listing 7. Retrieve script package input in python script

```python
def getInput():
    input = {}
    env_data = dict(os.environ)
    for key in ("VOLUME_NAME", "FILESYSTEM_TYPE", "MOUNT_POINT", "TYPE", "INUSE"):
        if key in env_data:
            input[key] = env_data[key]
    return input
```

Since the input storage name needs to be translated to identifier (runtimeid), the script first retrieves the storage using a GET operation on /services/resources/sharedDisks?type=< Type of storage >. The returned response is further filtered based on the input name. The script then constructs the JSON input for the attach operation as shown in Listing 3 and executes the REST call using the code shown in Listing 1. The script monitors the attach operation status until the attach status is either ATTACHED or ATTACH_FAILED. The attach operation status is logged into remote_std_out.log as shown in Figure 11, which also shows the REST call stack for the attach operation:
No script id is provided in the sample code to format the storage to the type specified by the input file system type and to mount to the input mount point. But you can write a standard shell script to perform this operation using the following algorithm:

- Refresh the devices in your system.
- Using the LUN ID returned by the monitor attach status operation call (Listing 5), get the SCSI drive (/dev/sd*) of the storage that was attached.
- Check whether the device is formatted and if not, format it.
- Mount the device to the mount point.

To list the storages attached to a node, execute the BlockStorageList script package. The list will be printed in remote_std_out.log, as shown in Figure 12:

When you execute the BlockStorageDetach script package, you should see the input dialog as shown in Figure 13. The BlockStorageDetach script package takes the following input:

**Block Storage Name**
Name of the block storage to detach from this node. The code in detach.py script retrieves the corresponding storage identifier (runtimeid) to be used in the detach REST call.

**Type of storage**
Value can be Block or Shared Block. This information is used to filter the block storage list that is retrieved to get the storage identifier (runtimeid).

Figure 13 shows the input for detaching a shared block storage named BM03:
Like the attach script, the detach script retrieves the input from environment variables and translates the storage name to runtimeid. The script then checks whether the storage is attached using list block storages attached to a workload node call. If the storage is attached, the script constructs the JSON input for the detach operation similar to the one in Listing 3 and executes the REST call using the code shown in Listing 1. The script monitors the detach operation status until the detach status is either DETACHED or DETACH_FAILED. The status of the detach operation is logged into remote_std_out.log, as shown in Figure 14, which also shows the REST call stack for the detach operation:

**Figure 14. BlockStorageDetach script package execution log**

---

**Conclusion**

In this tutorial, you learned about block storage support in PureApplication System 2.0, and how and when to use the block storage add-on in VSP and VSP Classic workloads. You also learned how to use the block storage REST API to exploit shared block storage in VSP workloads and to exploit block storage and shared block storage in VAP workloads. Using the sample code provided with this article, you should be able provide shared block storage support for your VAP workloads, and be able to reuse the code in the VSP workload lifecycle and operation scripts to provide support for block storage and shared block storage in VAP workloads.
## Downloads

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code sample</td>
<td>BlockStorage.zip</td>
<td>25 KB</td>
</tr>
</tbody>
</table>
Resources

- **IBM PureApplication System resources**
  - **IBM PureApplication System Knowledge Center**
    A single portal to conceptual, task, and reference information on installing, configuring, and using IBM PureApplication System.
  - **IBM PureApplication System W2500 V2 Knowledge Center**
    A single portal to conceptual, task, and reference information on installing, configuring, and using IBM PureApplication System W2500 V2.
  - **IBM PureSystems developer resources**
    Latest trials, downloads, tutorials, and other developer resources for IBM PureSystems products and services.
  - **IBM PureSystems Centre**
    Access PureSystems patterns from IBM and IBM Business Partners, updates to systems and patterns, and expertise for maximizing the benefit of systems and patterns.
  - **IBM PureApplication Systems Plug-in development kit**
    The Plug-in Development Kit (PDK) is designed to help you build your own pattern projects, which include software components, script packages, pattern types, plug-ins. The PDK provides creation, configuration, packaging, deployment, and debug capabilities for these projects.
  - **IBM PureApplication Systems Plug-in development guide**
    Plug-ins define the components, links, and policies that you use in the Pattern Builder to create virtual application patterns, or extend existing virtual application patterns.
  - **IBM PureApplication Systems Block Storage REST API**
    These APIs are intended to be used in the development of software components or script packages that consume block storage or shared block storage.
  - **Implementing high availability and disaster recovery in IBM PureApplication Systems V2**
    This IBM Redbook describes common prescriptive scenarios for setting up disaster recovery for common workloads using WebSphere Application Server, DB2, and WebSphere MQ between two IBM PureApplication System racks using the features in PureApplication System V2.

- **WebSphere resources**
  - **developerWorks WebSphere**
    Technical resources for developers who use WebSphere and related middleware products. Includes downloads, how-to information, support resources, and a free technical library of tutorials, best practices, IBM Redbooks, and product manuals.
  - **Most popular WebSphere trial downloads**
    No-charge trial downloads for key WebSphere products.
  - **WebSphere forums and community**
    Product-specific forums where you can get answers to your technical questions and share your expertise with other WebSphere users.
  - **WebSphere demos**
    Download and watch these self-running demos, and learn how WebSphere products can provide business advantage for your company.
  - **WebSphere-related tutorials on developerWorks**
Nearly 3000 tutorials on WebSphere and related technologies by top practitioners and consultants inside and outside IBM. Search for what you need.

- **WebSphere-related books from IBM Press**
  Convenient online ordering through Barnes & Noble.

- **developerWorks resources**
  - **Trial downloads for IBM software products**
    No-charge trial downloads for selected IBM® DB2®, Lotus®, Rational®, Tivoli®, and WebSphere® products.
  - **Technical events for developers**
    Conferences, seminars, and technical events to deepen your technical expertise and help you succeed in your software projects.
  - **developerWorks podcasts**
    Listen to interesting and offbeat interviews and discussions with software innovators.
  - **developerWorks on Twitter**
    Check out recent Twitter messages and URLs.
  - **IBM Education Assistant**
    A collection of multimedia educational modules that will help you better understand IBM software products and use them more effectively to meet your business requirements.
About the author

Bhadri Madapusi

Bhadri Madapusi is a Advisory Software Developer on the IBM PureApplication System Development Team. Bhadri has over 10 years of software development experience. Previously, he worked on cloud computing, Business Process Management, and WebSphere Commerce. You can contact Bhadri at bhadrim@ca.ibm.com.

© Copyright IBM Corporation 2015
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)