Christopher Bulmer
Alexander Evans

Implementing a RESTful API to the IBM Storwize Family

The IBM® SAN Volume Controller and Storwize® family currently allow administration through a web-based graphical user interface (GUI), a command-line interface (CLI), and by using Storage Management Initiative Specification (SMI-S). However, certain environments are better suited for a RESTful (related to Representational State Transfer (REST) application programming interfaces (APIs)). RESTful APIs use the HTTP verbs to act on a resource. For example, a GET request to /volumes lists all volumes and DELETE /volumes/1 deletes the volume with the ID 1. This paper describes an approach to developing a RESTful API to the SAN Volume Controller and Storwize family.

Most of the CLI commands that are available on the IBM Storwize family map to RESTful actions, aiding the development of an API. For example, many **`view`** commands give both a concise view that lists all of a particular item, and when given an ID or a resource name, display a more detailed list of that particular item. These commands map perfectly to GET /resource and GET /resource/id (where resource might be a volume or a host). The same mapping applies to the task commands where most resources have create (**`mk*`**), update (**`ch*`**), and delete (**`rm*`**) commands. In REST terms, these commands map to the following HTTP actions: POST /resource, PUT (or PATCH) /resource/id, and DELETE /resource/id.

Although our implementation used Node.js and Express framework, the source code snippets that are shown later are intended to illustrate the required steps to create the API that can be implemented in most programming languages. In the interest of simplicity, this paper focuses on only one resource, volumes. However, with a little extra work, further resources can be added easily because the approach is identical.

Design

When we designed the API, three main stages were apparent in the processing of an API request for Storwize.

The first stage is a method that is invoked when a certain URL is requested; this method gathers the arguments that are passed by the client and converts them into a command to be executed on the SAN Volume Controller system CLI. The second stage is a way of executing the command, obtaining the response, and checking for errors. The final stage is parsing the output of the command and converting it into a suitable response format, such as JavaScript Object Notation (JSON) or XML so that it can be returned to the client. Figure 1 shows how a request can be processed. In our implementation, we used only a small subset of HTTP status codes to keep it simple.

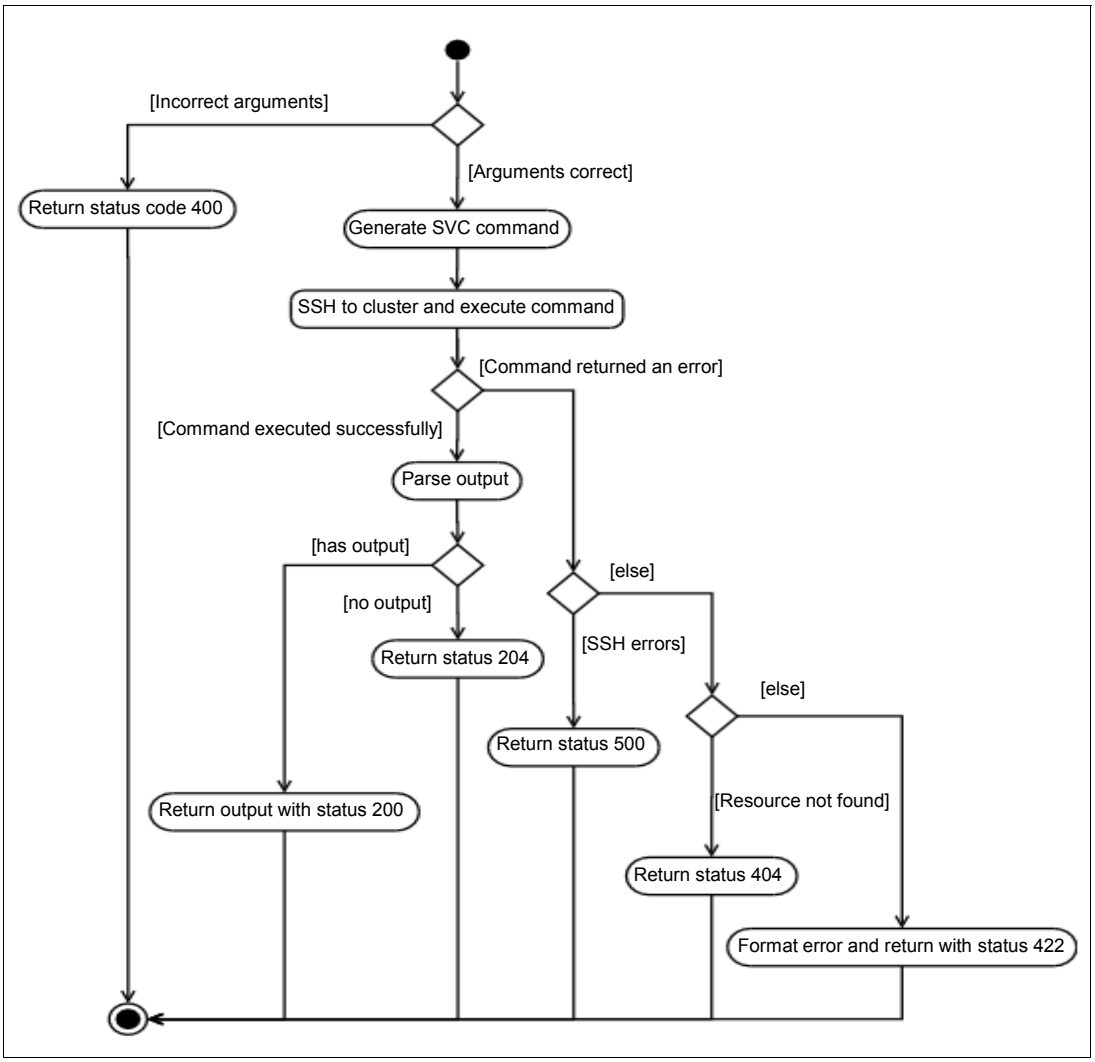

Figure 1 Activity diagram of an API request

The three main areas were each split into their own modules (Figure 2 on page 3). The Resource module provides methods to invoke when an API request is received. These methods contain the logic for generating the commands to execute on the SAN Volume Controller system. These methods take the input parameters that are passed to the API, run basic validation on the parameters, and construct the command. The Resource module invokes the Core module with the command and the Parser to use. The Core module is responsible for executing the command and coordinating the response. After the Core module executes the command and captures the response, it invokes a Parser to format the response correctly. The Core module returns the formatted response with the correct HTTP status. The Parsers are responsible for formatting the output of the command that was executed on the SAN Volume Controller system into a format that is suitable to return to the requester.

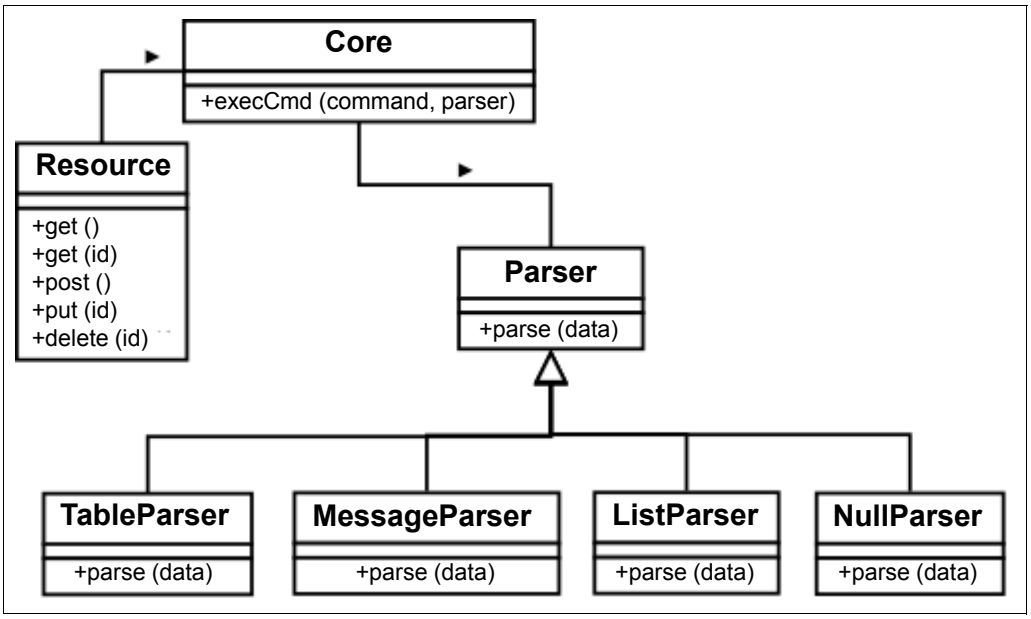

Figure 2 Class diagram for the API

With this design, the only requirement to add further functionality is the implementation of additional resources. Furthermore, the complexity of the resource needs to be only as complex as the use case. For example, if you do not use volume mirroring in your environment, you can skip the implementation parsing arguments for volume mirroring on POST /volume and PUT /volume/id. In the example that is presented later in this paper, we included only the minimum that is required to create a volume.

Core module

The Core module is responsible for executing commands that are requested by the user and coordinating the response. A resource class invokes the Core module after it constructs a command-line command that the Core module will execute, capture the output, invoke a parser, and return it to the requester with the correct HTTP status code. We set up public key authentication on the SAN Volume Controller system to enable the API to run commands without requiring a password. However, HTTP authentication can be added, and the username and password that are provided can be used to authenticate with the SAN Volume Controller system.

Improving security

The Core module executes the commands by wrapping the command in a Secure Shell (SSH) request that is sent to the SAN Volume Controller system for execution. To eliminate many security concerns, a regular expression is applied on the command to remove certain characters that allow for extra commands to be added instead of a parameter by a malicious user. The following line shows an example list of characters to replace. Although this approach is far from perfect, it illustrates a possible method to protect against malicious commands:

```
command = command.replace(/\`|\||\&|<|>|;/g, '');
```

HTTP status codes

In our implementation, the number of HTTP status codes that are used is limited to keep the approach simple. The status code is based on the return code from the SAN Volume Controller system after it executes a command. If the return code is a success (0), an HTTP success (2*xx*) code is issued. A quick check to see whether the SAN Volume Controller system returned any content determines which code to use. If no content exists, the `NO CONTENT (204)` status code is returned; otherwise, the `SUCCESS (200)` code is used. An improvement here is to use other HTTP status codes, such as `CREATED`; however, this approach requires additional complexity.

The error status codes that are returned are generated by parsing the command-line error output (STDERR) for certain keywords. If the phrase "`does not exist`" is matched in the output, a `RESOURCE NOT FOUND (404)` status response is returned. Any other errors from the SAN Volume Controller system result in the error code `UNPROCESSIBLE ENTITY (422)` being returned with the error message from the SAN Volume Controller system. If the command does not execute for reasons such as SSH failures, the status code `INTERNAL SERVER ERROR (500)` is returned to show that the command was not executed instead of the command failing.

Example code

Although this approach is basic, it returns the correct status codes in most cases. Any further status codes are straightforward to add. The following two functions are an example of how this component of the API might be implemented:

```
   function sendResponse(status, data, response){
      if (!data && status == 200){
         response.status(204);
      } else {
         response.writeHead(status, {'Content-Type':'application/json'});
      }
         response.end(data);
      }

   function execCmd(command, parser, response){
      stdout = '';
      stderr = '';
      error = false;
      command = command.replace(/\`|\||\&|<|>|;/g, '');
      cmd = exec('ssh', ['-q', 'superuser@' + cluster_address, command]);

      cmd.stdout.on('data', function(data){
         stdout += data;
      });

      cmd.stderr.on('data', function(data){
         error = true;
         stderr += data;
      });

      cmd.on('close', function(code){
         if (error){
// If the resource(s) wasn't found, we should return a 404 status code and not 500
            if (stderr.match(/does not exist/)){
```

```
                sendResponse(404, '', response);
             } else {
                stderr = JSON.stringify({'error': stderr});
                sendResponse(422, stderr, response);
             }
          } else if (code != '0'){
             sendResponse(500, '', response);
          } else {
// Parse the output and then send the response
             stdout = parser.parse(stdout);
             statusCode = 200;

             sendResponse(statusCode, JSON.stringify(stdout), response);
          }
       });
    }
```

Output parsers

The parsers are used to convert the text output from the SAN Volume Controller system into JSON or another format. Our implementation used JSON; however, by modifying the parsers, any format can be returned. We used four parsers, each with a specific use:

- Message Parser
- List Parser
- Table Parser
- Null Parser

All of the parsers provide the same interface, which consists of a single function: `parse(data)`.

Message Parser

The Message Parser is responsible for taking message output from the SAN Volume Controller system and converting it into a JSON to display to the user. This parser is used with commands, such as **`mkvdisk`**, which return a message that contains the ID of the volume that was created:

```
function parse (data) {
   var json= {};
   json['Message:'] = data;
   return json;
}
```

List Parser

The List Parser is responsible for taking a list output from the SAN Volume Controller system and converting it into a JSON array to display to the user. This approach splits the data that is received from the SAN Volume Controller system by the newline character and then by white space. The first part of each line before the space is used as the identifier and everything after the space is used as the value. The data is rebuilt as a JSON array to return to the user. Typically, this parser is used with the **`ls`** commands, which return the detailed view that consists a list of entries, such as **`lsvdisk ID`** (for example, **`lsvdisk`** 0):

```
function parse (data) {
   var lines = data.split("\n");
   var JSONArray = {};
   for(var i in lines) {
      var entry = lines[i].split(" ");
      JSONArray[entry[0]] = entry[1];
   }
   return JSONArray;
}
```

Table Parser

The Table Parser is responsible for parsing a table of data from the SAN Volume Controller system and converting it to the correct format. The approach that is used splits the data by the newline character to get each resource and the first line is used as the identifiers. Each line is then split by whitespace to get each value and it is reconstructed into a JSON array. This parser is used with **`view`** commands that return the concise view, such as **`lsvolume`** (with no arguments):

```
function parse (data) {
   var lines = data.split("\n");
   var headers = lines[0].split(" ");
   var JSONArray = {};
   for(var i=1; i<lines.length; i++) {
      var content = lines[i].split(" ");
      if(!content.length == 0) {
         var column = {};
         for(var j in headers) {
            content[headers[j]] = content[j];
         }
         JSONArray[i-1] = column;
      }
   }
   return JSONArray;
}
```

Null Parser

The Null Parser returns no content when it is called. It is used in situations when an SAN Volume Controller command does not output any content. This parser is not strictly needed; however, it was implemented so that a consistent flow through the API exists for each type of request and a consistent interface exists for every type of command.

Resources

Resources are the part of the API that knows the domain. Each class contains the functions to invoke when an action on the resource is requested. The function performs validation on the parameters and assembles the command to run on the SAN Volume Controller system. Because it knows the domain, it knows the correct type of parser for the command. Therefore, when it invokes **`execCmd`** on the core module, it also passes the correct parser for the request. In our implementation, the response object is also passed to the Core module, which allows it to set the status code and return the data directly from the Core module instead of requiring the Resource class to send the response.

The following examples illustrate how GET /volume and GET /volume/id can be implemented. The commands are constructed and the correct parsers are selected:

```
Get('/volume', function(request, response){
   core.execCmd("lsvolume", core.tableParser, response);
});

Get('/volume/:id', function (request, response){
   core.execCmd("lsvolume "
                + request.params.id, core.listParser, response);
});
```

A more interesting example is creating a volume. The following example shows the implementation by using the required information only. However, you can add more arguments, as required. This example uses the function that is defined (parameter) in the Core module to check whether a parameter was passed as part of the request. If the request does not include all of the required parameters, a `BAD REQUEST (400)` HTTP status code is returned. This example can also be implemented by using the sendResponse function in the core class. Also, this example illustrates how HTTP verbs are used to determine the action to perform:

```
Post('/volume', function(request, response){
   if(core.defined(request.body.iogrp) &&
      core.defined(request.body.mdiskgrp) &&
      core.defined(request.body.size) &&
      core.defined(request.body.unit))
   {
      cmd = "mkvolume -mdiskgrp " + request.body.mdiskgrp;
      cmd += " -iogrp " + request.body.iogrp;
      cmd += " -size " + request.body.size;
      cmd += " -unit " + request.body.unit;
      core.execCmd(cmd, core.messageParser, response);
   } else {
      core.endResponse(400, '', response);
   }
});
```

The following example shows how HTTP verbs can be used to change the function to invoke. The following example issues an **`rmvolume`** command on the SAN Volume Controller system. Because that command does not provide any output, the Null Parser is used:

```
Delete('/volume/:id', function(request, response){
   core.execCmd("rmvolume "
                + request.params.id, core.nullParser, response);
});
```

Further work

This paper shows the minimum effort that is required to develop a RESTful API for IBM Storwize and SAN Volume Controller; however, many additions are possible. Several of them are outlined next.

Additional resources

Many other resources can be implemented, such as hosts or host-volume mappings, to increase the usefulness of the API. As well as more resources, the number of parameters that can be passed can also be increased to create objects, such as compressed or mirrored volumes.

Authentication and authorization

Stateless authentication, such as HTTP basic, can be added to protect the API against unauthorized access. Furthermore, different levels of authorization can be added by using different roles on the SAN Volume Controller system. If the authentication or authorization fails, a `FORBIDDEN (403)` status is returned.

Authors

This paper was produced by a team of specialists from around the world working at the International Technical Support Organization, Hursley Center, UK.

Christopher Bulmer is a Software Engineer and Development Team Leader of the Host Interface for the IBM SAN Volume Controller and Storwize family based at Hursley Park, UK. Before joining IBM in 2013, he studied for a Masters in Computer Science at the University of Southampton.

Alexander Evans is a Software Engineer at the Systems and Technology Group in Hursley Park, UK. Before joining IBM in 2013, he studied for a Masters in Computer Science at the University of Birmingham.

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

`ibm.com/redbooks/residencies.html`

Stay connected to IBM Redbooks

- Find us on Facebook:

  `http://www.facebook.com/IBMRedbooks`
- Follow us on Twitter:

  `http://twitter.com/ibmredbooks`

- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806

- Explore new IBM Redbooks® publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm

- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

This document REDP-5167-00 was created or updated on January 22, 2015.

Send us your comments in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:
  ibm.com/redbooks
- Send your comments in an email to:
  redbooks@us.ibm.com
- Mail your comments to:
  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400 U.S.A.

Trademarks