Technical Report

Storage as a Service Blueprint for IT Providers

Simplifying Delivery of On-Demand Self-Service

Rich O'Connor and Owen Hughes, NetApp
January 2018 | TR-4654

Abstract

This document describes how NetApp's Engineering IT team uses NetApp® solutions and the ServiceNow application to implement a web-based self-service portal to deliver on-demand storage provisioning that dramatically improves time to closure and requires no human interaction.

NetApp®

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 Solution Overview

NetApp's Engineering IT team supports teams who design and implement NetApp products and solutions. Engineers on these teams have an ongoing need to store department data, project files, and so on. DevOps teams who are rapidly prototyping and iterating continuously create new applications that require storage.

Traditionally, NetApp engineers requested storage by opening help tickets in the Engineering IT team's ServiceNow platform. These requests were prioritized and put in a storage administrator's queue. The storage administrator would find a storage location, provision the storage, send an e-mail to the requester, and perform other tasks before closing the ticket.

To improve the agility of the Engineering organization, the team implemented an on-demand, self-service solution called storage as a service (StaaS), which enables engineers to provision their own file-based storage when they need it and without human interaction. With the StaaS solution, an engineer can fill out a simple web form in ServiceNow. Within a few minutes of submitting the request, the requester is notified that storage is provisioned and ready to use.

In the Engineering IT team's StaaS implementation, there are two use cases:

- General purpose file shares (Dept Shares), which are accessible from UNIX (NFS) and Windows (SMB/CIFS) clients.
- A variant of the first use case (Testbeds), customized for the QA community, which has specific requirements on directory/path naming conventions and locations.

1.1 Key Features

The StaaS solution provides the following key features:

- Provisions storage based on a user's requirements for capacity and performance.
- Provides three service level tiers—value, performance, and extreme—tailored to high-capacity, database, and latency-sensitive applications.
- Supports NFS and SMB/CIFS shares for access across UNIX and Windows environments.
- Dynamically allocates new volumes based on an understanding of existing headroom, capacity, and underlying drive types of the NetApp ONTAP® system.

1.2 Solution Components

Table 1 lists the components for the StaaS solution.

Table 1) StaaS solution components.

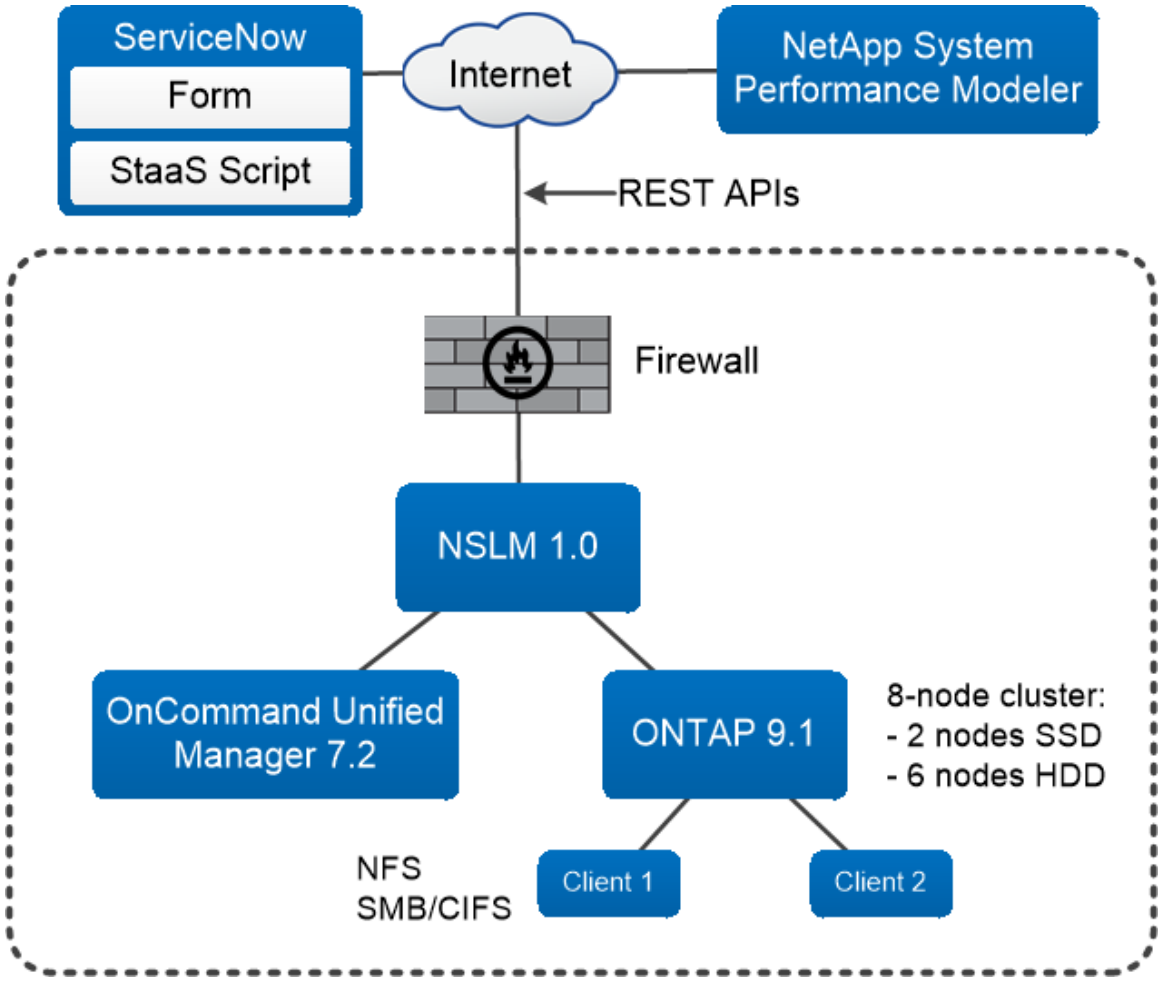
Component	Purpose	NetApp Solution	Your solution
ONTAP 8.3.2 and later	Storage	ONTAP 9.1 (FAS, AFF)	Your ONTAP version
Service portal	Hosts user interface	ServiceNow	Any service portal compatible with REST APIs
NetApp Service Level Manager (NSLM) 1.0 and later	Service level management	VM: • RHEL 7.2 • 12GB RAM • 200GB disk space	Physical or virtual server: • Minimum 8GB RAM, recommended 12GB • Minimum 50GB disk space, recommended 120GB
NetApp OnCommand® Unified Manager 7.2 and later	Performance management	VM: • RHEL 7.2	Physical or virtual server:

Component	Purpose	NetApp Solution	Your solution
		• 12GB RAM • 200GB disk space	• Minimum 8GB RAM, recommended 12GB • Disk space depends on the server type
Internet	Access to NetApp System Performance Modeler (SPM), which provides headroom calculations	Port 443	Port 443

1.3 Storage as a Service Solution Configuration

Figure 1 shows the StaaS solution configuration.

Figure 1) StaaS solution configuration.



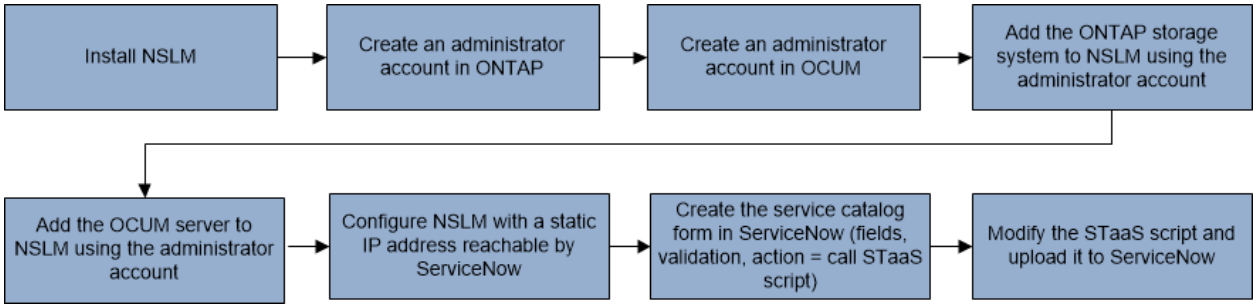
2 Implementation Blueprint

By leveraging NetApp's Engineering IT team implementation, including the script workflows and the sample script code, you can implement StaaS in your environment.

2.1 Implementation Workflow

The workflow in Figure 2 shows the tasks you must complete to implement StaaS. The workflow assumes ONTAP, OnCommand Unified Manager, and ServiceNow are installed, and that your ONTAP system is sending AutoSupport messages to NetApp.

Figure 2) StaaS implementation workflow.



2.2 User Interface

For this solution, the Engineering IT team used the ServiceNow interface builder to create a simple web-based request form in the ServiceNow application.

Though most of the fields are self-evident, the following fields require explanation:

- **File Share Type** determines the type of file share:
  - TestBed, with support for NFS file shares only
  - Dept Shares, with support for both NFS and CIFS file shares.
- **Service Level** determines the service level: value, performance, or extreme. The service levels are tailored, respectively, to high-capacity, database, and latency-sensitive applications. Table 2 describes the differences between each level.

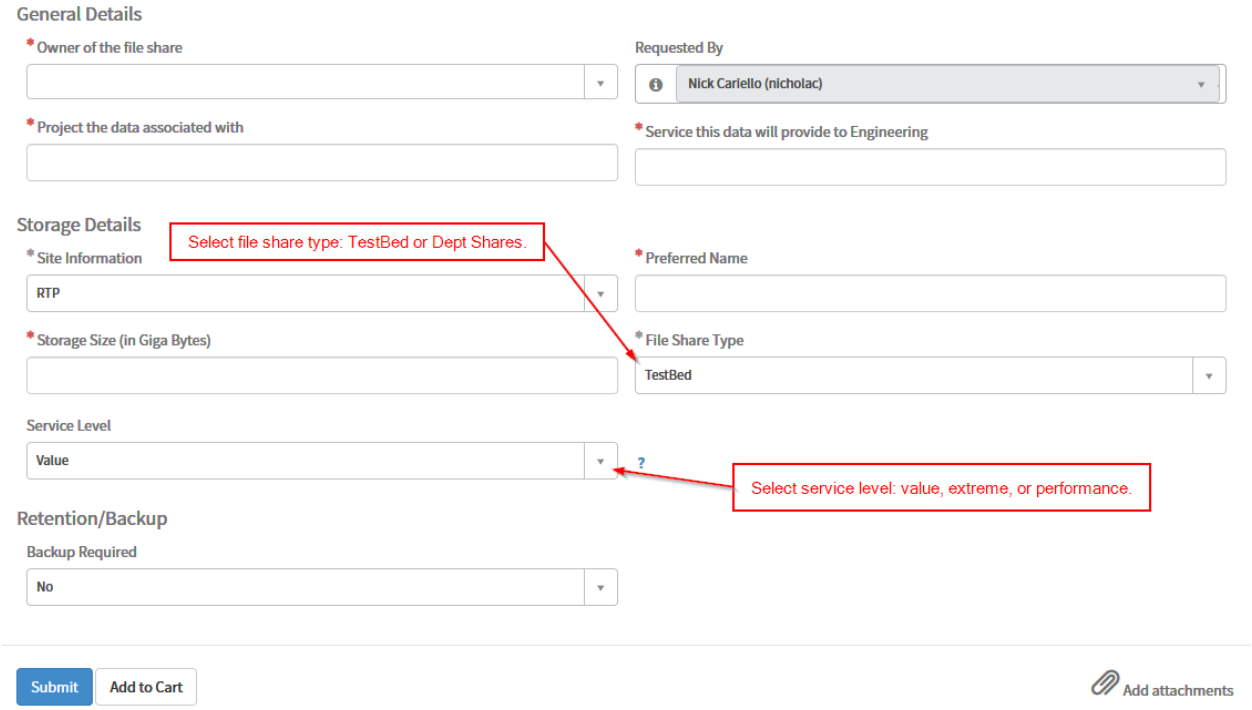
Table 2) Service levels on the ServiceNow application.

Service Level	Peak Latency	Peak IOPS/TB (max)	Expected IOPS/TB (min)
Value	17	512	128
Performance	2	4096	2048
Extreme	1	12288	6144

StaaS automatically scales throughput to volume size, maintaining the ratio of input/output operations per second (IOPS) to terabytes (TBs)/gigabytes (GBs) as the size of the volume changes. Users are warned when a volume is noncompliant with the current service level.

Figure 3 shows the service portal form after a user selects the TestBed file share type.

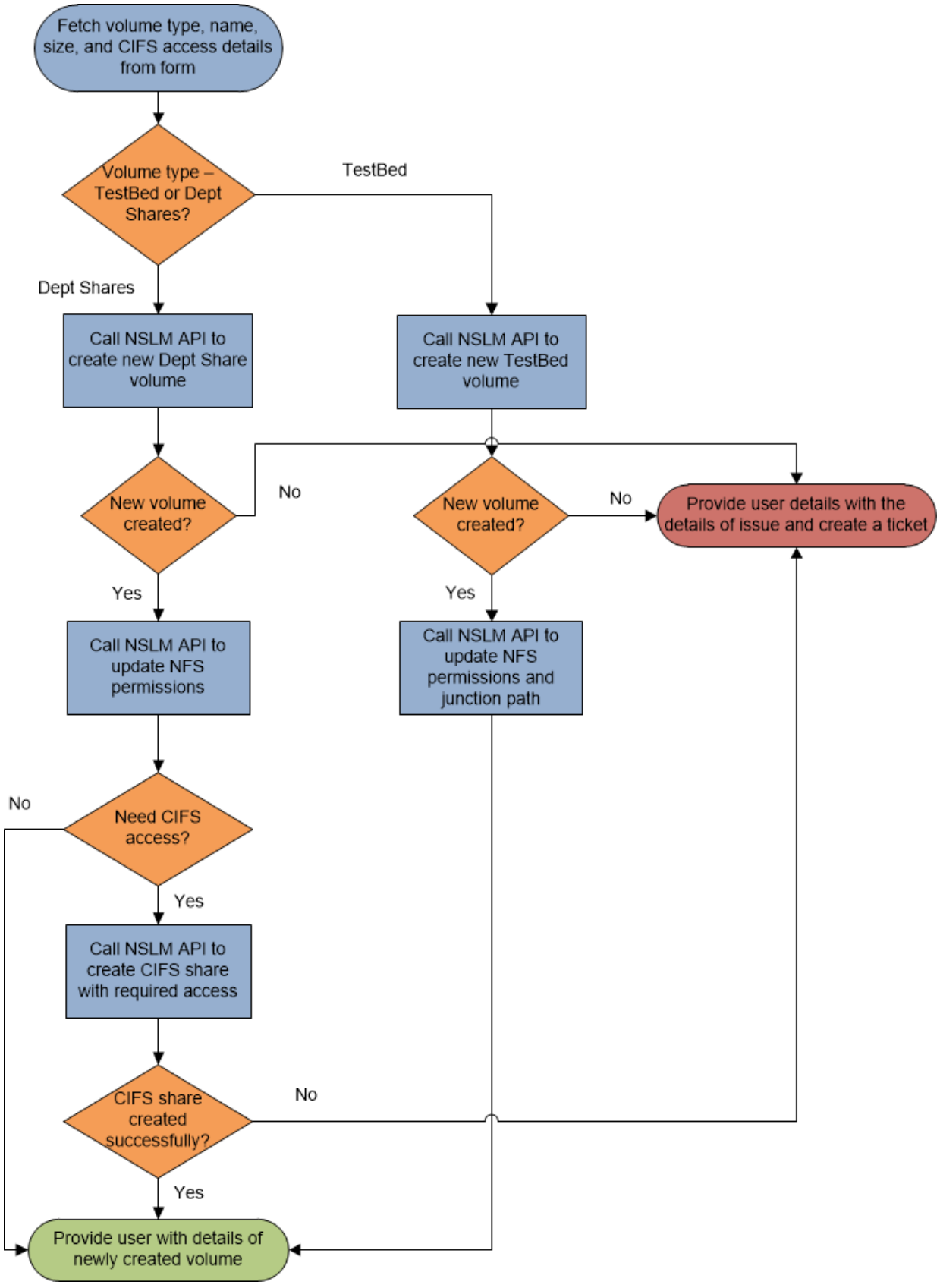
Figure 3) Service portal form.



2.3 Storage as a Service Script Workflow

The StaaS script runs inside the ServiceNow application. The script collects user input from the service catalog form, processes it according to environment requirements, and invokes the appropriate REST API calls to NSLM. The workflow shown in Figure 4 describes the StaaS script in schematic form.

Figure 4) StaaS script workflow.



2.4 Storage as a Service Sample Code

The following example shows the code for the StaaS script:

```
// function to create a file share
function create_fileshare(name,size){
	try {
// here we build the body of the request to create the share via API and execute it
		var r = new sn_ws.RESTMessageV2('Test API', 'create file_share');
		var
body="{\"storage_service_level_key\":\""+SSL_VALUE_KEY+"\",\"name\":\""+name+"\",\"size\":\""+siz
e+"\",\"storage_vm_key\":\""+SVM_KEY+"\"}";
		r.setRequestBody(body);
		var response = r.execute();
		var responseBody = response.getBody();
		gs.log("NSLM create_fileshare response:"+response.getBody);
		var httpStatus = response.getStatusCode();
// simple error check to invoke failure on anything other than success
		if (httpStatus!=202){
			var obj = JSON.parse(responseBody);
			output="Failed to invoke file share creation:\nStatus code-
"+httpStatus+"\nCause-"+obj.status.error.message+"\nReason:"+obj.status.error.reason;
			return output;
		}
		else{
			var location = response.getHeader('Location');
			gs.log("NSLM job location:"+location);
			output = wait_for_job(location, 300);
			return output;
		}
	}
	catch(ex) {
		var message = ex.getMessage();
		gs.log("NSLM ERROR:"+message);
		return message;
	}
}
//wait for job to complete after calling an API
function wait_for_job(job_uri, max_time_wait){
	time_to_wait = max_time_wait;
	while (time_to_wait > 0){
// building/executing the body of request to get status of job (share create)
		var r = new sn_ws.RESTMessageV2('Test API', 'Job status');
		r.setEndpoint(job_uri);
		var response = r.execute();
		var responseBody = response.getBody();
		var httpStatus = response.getStatusCode();
		if (httpStatus == 200){
			var obj = JSON.parse(responseBody);
			status = obj.result.records[0].status;
			if (status == "FAILED"){
				err_msg = obj.result.records[0].error_message;
				return err_msg;
			}
			else if (status == "COMPLETED"){
				return 0;
			}
			else{
				gs.sleep(5000);
				time_to_wait -= 5;
			}
		}
		else{
			return "FAILED";
		}
	}
	return "FAILED";
}
// get key of the selected resource
// this allows us to get the unique key that applies to volume/share/cifs
```

```
// we must have the key of the resource before we modify attributes of it
// such as creating a cifs share for the volume we just created
function get_key(name,resource){
        var r = new sn_ws.RESTMessageV2('Test API', 'Default GET');
        if (resource=='volumes'){
                r.setEndpoint("https://"+NSLM_INSTANCE_IP+"/api/2.0/ontap/volumes/?name="+name);
        }
        else if (resource=='file_shares'){
                r.setEndpoint("https://"+NSLM_INSTANCE_IP+"/api/1.0/slo/file-shares/?name="+name);
        }
        else if (resource=='cifs'){
                r.setEndpoint("https://"+NSLM_INSTANCE_IP+"/api/2.0/ontap/cifs-
shares?name="+name);
        }
        else{
                return "INVALID Request";
        }
        var response = r.execute();
        var responseBody = response.getBody();
        var httpStatus = response.getStatusCode();
        if (httpStatus == 200){
                var obj = JSON.parse(responseBody);
                temp_key = obj.result.records[0].key;
        }
        else{
                gs.log("NSLM - Get Key Request could not be processed");
        }
        if(temp_key){
                return temp_key;
        }
        else{
                return ("No "+resource+" found with name "+name);
        }
}
// function to create CIFS share
// we get the key of the fileshare created earlier
// then we create the cifs share for it
function create_cifs(name){
        file_shares_key=get_key(name,"file_shares");
        var r = new sn_ws.RESTMessageV2('Test API', 'create cifs_share');
        var body="{\"file_share_key\":\""+file_shares_key+"\",\"name\":\""+name+"\"}";
        r.setRequestBody(body);
        var response = r.execute();
        var responseBody = response.getBody();
        var httpStatus = response.getStatusCode();
        gs.log("NSLM response create cifs- "+responseBody);
        var location = response.getHeader('Location');
    gs.log("NSLM location - "+location);
        output = wait_for_job(location, 300);
    return output;
}
// function to create cifs acls
// we get the key of the cifs share here
// then apply the ACLs to control the access
function create_cifs_acls(name,access,user_data){
        gs.log("NSLM acls func user_data-"+user_data);
        CIFS_SHARE_KEY = get_key(name,"cifs");
        users=user_data.split(",");
        acls_created=[];
        acls_failed=[];
        user_name="";
        for(i=0;i<users.length;i++){
                if(i>0 && users[i]==users[0])
                        continue;
                else if(users[i]){
                        if(users[i].indexOf("ng") == -1 && users[i].length==32){
                                var user_rec = new GlideRecord('sys_user');
                                user_rec.addQuery('sys_id',users[i]);
                                user_rec.query();
                                if(user_rec.next()){
                                        user_name=user_rec.user_name;
```

```
                                }
                        }
                        else{
                                user_name=users[i];
                        }
                        if(acls_created.indexOf(user_name) == -1){
                                var r = new sn_ws.RESTMessageV2('Test API', 'create cifs_share');
                                r.setEndpoint("https://"+NSLM_INSTANCE_IP+"/api/2.0/ontap/cifs-
share-acls");
                                var
body="{\"cifs_share_key\":\""+CIFS_SHARE_KEY+"\",\"permission\":\""+access+"\",\"user_or_group\":
\"netapp\\\\"+user_name+"\"}";
                                r.setRequestBody(body);
                                var response = r.execute();
                                var responseBody = response.getBody();
                                var httpStatus = response.getStatusCode();
                                gs.log("NSLM response for acls - "+responseBody);
                                var location = response.getHeader('Location');
                                gs.log("NSLM location acls - "+location);
                                output = wait_for_job(location, 300);
                                if (output==0){
                                        acls_created.push(user_name);
                                }
                                else{
                                        acls_failed.push(user_name+":"+output);
                                }
                        }
                }
        }
        return (acls_created+("@")+acls_failed);
}
// function to update permissions of the specified file share
// since we are using unix-style security by default here, this then applies
// whatever unix permissions and user/group ownership we want
function update_permission(name){
        volume_key=get_key(name,"volumes");
        var r = new sn_ws.RESTMessageV2('Test API', 'update permissions');
        r.setEndpoint("https://"+NSLM_INSTANCE_IP+"/api/2.0/ontap/volumes/"+volume_key);
        var body="{\"security_style\":\"mixed\" , \"security_permissions\":\"0777\",
\"security_user_id\":\"0\", \"security_group_id\":\"30\"}";
        r.setRequestBody(body);
        var response = r.execute();
        var responseBody = response.getBody();
        var httpStatus = response.getStatusCode();
        gs.log("NSLM response for permission update - "+responseBody);
        var location = response.getHeader('Location');
    gs.log("NSLM location for permission update- "+location);
        output = wait_for_job(location, 300);
    return output;
}
// function to unmount the junction path
// use this to unmount the volume we created
function umount_jpath(name){
        volume_key=get_key(name,"volumes");
        var r = new sn_ws.RESTMessageV2('Test API', 'create file_share');
        r.setEndpoint("https://"+NSLM_INSTANCE_IP+"/api/2.0/ontap/volumes/"+volume_key+"/jobs/unm
ount");
        var response = r.execute();
        var responseBody = response.getBody();
        var httpStatus = response.getStatusCode();
        gs.log("NSLM response for umount - "+responseBody);
        var location = response.getHeader('Location');
    gs.log("NSLM location for umount- "+location);
        output = wait_for_job(location, 300);
    return output;
}
// function to mount the junction path
// we are using this right now to make a specific change to junction path
// specifically for testbeds, but this will expand later
function mount_jpath(name){
        volume_key=get_key(name,"volumes");
```



```
        var r = new sn_ws.RESTMessageV2('Test API', 'create file_share');
        r.setEndpoint("https://"+NSLM_INSTANCE_IP+"/api/2.0/ontap/volumes/"+volume_key+"/jobs/mou
nt");
// using a custom junction path here
        var body="{\"junction_path\":\"/testbedN/"+name+"\" }";
    r.setRequestBody(body);
        gs.log("https://"+NSLM_INSTANCE_IP+"/api/2.0/ontap/volumes/"+volume_key+"/jobs/unmount");
        var response = r.execute();
        var responseBody = response.getBody();
        var httpStatus = response.getStatusCode();
        gs.log("NSLM response for mount- "+responseBody);
        var location = response.getHeader('Location');
    gs.log("NSLM location for mount- "+location);
        output = wait_for_job(location, 300);
    return output;
}
//init variables, current.variables.* is to fetch input from form
// NSLM_INSTANCE_IP = IP address to reach NSLM API
// SSL_VALUE_KEY
var NSLM_INSTANCE_IP="10.240.22.52";
var SSL_VALUE_KEY="46844bf5-705b-4c5d-8b50-163e7c8e3788";
// populate variable sizegb from the form (e.g. 10)
var sizegb = current.variables.size_data;
// convert the gb value to bytes
var size = parseInt(parseFloat(sizegb)*1073741824);//gb to bytes converstion
// name of the share to create
var name = current.variables.vname_data;
// owner of the share
var owner = current.variables.owner;
// type of volume from the form
var vol_type = current.variables.vol_type;
var output_msg="";
// initialize the basic SVM info
var SVM_NAME="";
var SVM_DATA_LIF="";
var SVM_KEY="";
// based on the volume type, we change the target SVM/SVM_KEY/etc.
// later we will base this on site as well
// this could be based on tenant per SVM
// SVM_DATA_LIF = FQDN to access share
// SVM_NAME = name of SVM to deploy on
// SVM_KEY = SVM key taken from NSLM API - needed to validate correct SVM to use
if (vol_type=="testbed"){
        SVM_NAME="rtptestbeds";
        SVM_DATA_LIF="testbedN.rtp.eng.netapp.com";
        SVM_KEY="6649d442-0be7-11e1-9918-123478563412";
}
if (vol_type=="deptshares"){
        SVM_NAME="rtpdepts";
        SVM_DATA_LIF="deptshares.rtp.eng.netapp.com";
        SVM_KEY="a4bc94e0-e84a-11e1-bf7b-123478563412";
}
// original test instance data
// var SVM_DATA_LIF="nslmdatatest-rtpcms.rtp.netapp.com";
// var SVM_NAME="nslmtest";
// var SVM_KEY="8ddd2d4a-9a2a-11e7-bfe1-00a0989009e3";
// functions to create the share get called here - main part of script
// create_fileshare is the primary task to NSLM that creates the volume
// with the results of that function, we then populate out output_msg to deliver to
// the user information about the share
output = create_fileshare(name,size);
if (output==0){
        output=("Successfully created File Share: "+name);
        output_msg=output_msg+output+"\n";
        output = update_permission(name);
        if(output==0)
                output="Permissions updated successfully.";
        path=("NFS path: "+SVM_DATA_LIF+":/"+name);
// adding output for deptshares to reflect automount location for user's info
         if(vol_type=="deptshares"){
                path=("Automount path: /x/eng/deptshares/"+name);
```

```
                output_msg=output_msg+path+"\n";
        }
// changed the next to reflect the auto mounted path under testbedN
        if(vol_type=="testbed"){
                output=umount_jpath(name);
                if (output==0){
                        output=mount_jpath(name);
                        if(output==0){
                                path=("Automount path: /x/eng/testbedN/"+name);
                                output_msg=output_msg+path+"\n";
                        }
                }
        }
// commenting this out, as handled by previous conditionals
//      if(vol_type=="testbed" || current.variables.nis_data=="Yes"){
//              output_msg=output_msg+path+"\n";
//      }
// if a departmental share, we allow cifs creation
// if cifs selected, we also apply security via separate call to NSLM
// using the create_cifs function
        if(vol_type=="deptshares" && current.variables.cifs_data=="Yes"){
                output = create_cifs(name);
                if (output==0){
                        output=("Successfully created CIFS Share: "+name);
                        path=("CIFS path: \\\\"+SVM_DATA_LIF+"\\"+name);
                        output_msg=output_msg+path+"\n";
                        full_access=current.variables.full_access.toString();
                        full_access_ng=current.variables.full_access_ng.toString();
                        full_access = owner+","+full_access+","+full_access_ng;
                        gs.log("NSLM full_access - "+full_access);
                        output = create_cifs_acls(name,"full_control",full_access);
                        result=output.split("@");
                        if(result[0]){
                                msg=("Full control for CIFS given to: "+result[0]);
                                output_msg=output_msg+msg+"\n";
                        }
                        if(result[1]){
                                msg=("Full control could not be provided to: "+result[1]);
                                output_msg=output_msg+msg+"\n";
                        }
                        read_access=current.variables.read_access.toString();
                        read_access_ng=current.variables.read_access_ng.toString();
                        if(read_access_ng){
                                read_access = read_access+","+read_access_ng;
                        }
                        if (read_access){
                                gs.log("NSLM read_access - "+read_access);
                                output = create_cifs_acls(name,"read",read_access);
                                result1=output.split("@");
                                if(result1[0]){
                                        msg = ("Read access for CIFS given to: "+result1[0]);
                                        output_msg=output_msg+msg+"\n";
                                }
                                if(result1[1]){
                                        msg = ("Read access could not be provided to:
"+result1[1]);
                                        output_msg=output_msg+msg+"\n";
                                }
                        }
                }
                else{
                        msg = ("CIFS share creation failed:"+output);
                        output_msg=output_msg+msg+"\n";
                }
        }
// this section will handle backup/retention information
        if(current.variables.backup_data=="Yes"){
                msg = ("Default backup policy applied");
                output_msg=output_msg+msg+"\n";
        }
        current.stage="delivery";
```

```
}
else{
        msg=("File Share creation failed: "+output);
        output_msg=output_msg+msg+"\n";
        // create SN ticket for the issue
        var inc = new GlideRecord("incident");
        inc.initialize();
        inc.applyTemplate("NSLM error");
        inc.short_description="ERROR: New File share creation-"+name;
        inc.caller_id=current.variables.requester;
        inc.opened_by=current.variables.requester;
        inc.description="Request Item: "+current.number+"\n"+output_msg;
        inc.insert();
        output_msg=output_msg+"Ticket-"+inc.number+" has been raised with Engineering support
regarding this issue."+"\n";
}
current.comments=output_msg;
```

3 Where to Find Additional Information

To learn more about the products used in the StaaS solution, refer to the following documents and/or websites:

- ONTAP Resources
  https://mysupport.netapp.com/ontap/resources
- OnCommand Unified Manager Resources
  http://mysupport.netapp.com/unifiedmanager/resources
- NetApp Service Level Manager Documentation
  https://mysupport.netapp.com/documentation/productlibrary/index.html?productID=62384
- ServiceNow Product Documentation
  https://docs.servicenow.com/

You can find additional documentation on the NetApp Support Site.

Refer to the Interoperability Matrix Tool (IMT) on the NetApp Support site to validate that the exact product and feature versions described in this document are supported for your specific environment. The NetApp IMT defines the product components and versions that can be used to construct configurations that are supported by NetApp. Specific results depend on each customer's installation in accordance with published specifications.

TR-4654-0118