StupidSimplePatchServer Documentation

Release 1.0

Bryson Tyrrell

Feb 16, 2018

Deploy and Setup

The Supid Simple Patch Server (SSPS) is a serverless application to provide a patch server for Jamf Pro administrators.

With the SSPS you can host your own software title patch definitions for custom patch policies. The SSPS also allows you to *subscribe to other administrators' patch definitions* and automate the management of your patch definitions using it's *REST API*.

CHAPTER 1

Deploy on AWS

How to deploy the patch server in your AWS account.

1.1 Prerequisites

You will need the AWS command line utility to deploy a copy of this application to your AWS account. You can get instructions for installing the awscli here.

Your AWS `IAM User` will require permissions for creating resources in your AWS account including:

- API Gateway
- Lambda Functions
- S3 Buckets
- IAM Roles/Permissions

Here is an example `IAM Policy` you can use for your `IAM User`:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "apigateway:DELETE",
                "apigateway:GET",
                "apigateway:PATCH",
                "apigateway:POST",
                "cloudformation:CreateChangeSet",
                "cloudformation:DescribeChangeSet",
                "cloudformation:DescribeStacks",
                "cloudformation:ExecuteChangeSet",
                "cloudformation:ListChangeSets",
```

```
                "dynamodb:CreateTable",
                "dynamodb:DeleteTable",
                "dynamodb:DescribeTable",
                "iam:AttachRolePolicy",
                "iam:CreateRole",
                "iam:DeleteRole",
                "iam:DeleteRolePolicy",
                "iam:DetachRolePolicy",
                "iam:GetRole",
                "iam:PassRole",
                "iam:PutRolePolicy",
                "lambda:AddPermission",
                "lambda:CreateFunction",
                "lambda:DeleteFunction",
                "lambda:GetFunctionConfiguration",
                "lambda:ListTags",
                "lambda:RemovePermission",
                "lambda:TagResource",
                "lambda:UntagResource",
                "lambda:UpdateFunctionCode",
                "s3:CreateBucket",
                "s3:DeleteBucket",
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Resource": "*"
        }
    ]
}
```

1.2 Deploy the Patch Server

Clone this repo to your computer and go to it in your `Terminal`.

```
$ cd /path/to/StupidSimplePatchServer
```

Using the AWS CLI, package the application for `CloudFormation`:

```
$ aws cloudformation package --template-file template.yaml --s3-bucket <Your-S3-
↪Bucket> --output-template-file deployment.yaml
```

Note: If the S3 bucket specified for `aws cloudformation package` does not exist, you can create it from the CLI with the following command: `aws s3 mb s3://<Your-S3-Bucket>`

Use the created `deployment.yaml` file to create the application in `CloudFormation` (you can change the `--stack-name` value to whatever you prefer):

```
$ aws cloudformation deploy --template-file deployment.yaml --stack-name ssps --
↪capabilities CAPABILITY_IAM
```

You should see the following output on your screen:

```
Waiting for changeset to be created..
Waiting for stack create/update to complete
Successfully created/updated stack - ssps
```

1.3 Access Your Patch Server

Once complete, go to the `AWS Console` in your browser and go to the `CloudFormation` page (be sure you are in the correct region).

You should see in the list the stack name used in the `deploy` command. Select it and click on the `Resources` tab. This will show you all of the resources that were created for the application.

To get the URL for your Patch Server, go to the `API Gateway` page in the `AWS Console`.

Select the Patch Server (it will have the same name as the stack), go to `Stages` in the sidebar, and click on `Prod`. You should see a URL string similar to this:

```
**https://<API-GATEWAY-ID>.execute-api.<REGION>.amazonaws.com/Prod**
```

1.4 About AWS Costs

This application is created and deployed within your AWS account. While you are responsible for the costs of running the service, it is highly likely that this will fall within AWS's Free Tier.

Refer to AWS's pricing guides for more information:

- API Gateway: https://aws.amazon.com/api-gateway/pricing
- Lambda: https://aws.amazon.com/lambda/pricing
- S3: https://aws.amazon.com/s3/pricing
- DynamoDB: https://aws.amazon.com/dynamodb/pricing

CHAPTER 2

Setup in Jamf Pro

How to configure the patch server as an External Patch Source in Jamf Pro.

Note: "External Patch Sources" is a feature of Jamf Pro v10.2+.

To add your Patch Server as a `Patch External Source` in Jamf Pro, go to:

Settings > Computer Management > Patch Management

1. Click the `+ New` button next to `Patch External Source`.
2. Give the Patch Server a name.
3. Enter the URL without the schema (i.e. `https://`) in the `SERVER AND PORT` field (e.g. `<API-GATEWAY-ID>.execute-api.<REGION>.amazonaws.com/Prod/`) and 443 for the `PORT`.
4. Check the `Use SSL` box.

The Patch Server will now be available to subscribe to when adding new titles under `Patch Management`:

Computers > Patch Management

CHAPTER 3

Managing Patch Definitions

Manage your patch definition files for your patch server.

3.1 Populate the S3 Bucket in AWS Console

To make patch titles available on your Patch Server, upload the JSON file of the full patch definition into the root of the S3 bucket created for the application.

Note: The JSON filename must match the software title ID of the patch.

Warning: Manually managing your patch definitions requires you to validate the JSON of the patch definitions on your own before uploading the files. It is recommended to use the *REST API* for managing your patch definition files as it will perform JSON schema validation on all requests.

Examples taken from Jamf's official patch server would be saved to the S3 bucket as:

```
AdobeAcrobatProDC.json
Composer.json
GoogleChrome.json
JavaSEDevelopmentKit8.json
macOS.json
MicrosoftWord2016.json
```

To update your patch definitions, replace the existing file in S3 with the new, updated file.

CHAPTER 4

Patch Definition Subscriptions

How to subscribe to someone else's patch definition.

4.1 Subscribe to a Patch Definition

To subscribe to the URL of a patch definition, make a POST request to the `/subscribe` endpoint with a JSON payload containing the software title ID (`'id'`) and the URL to the JSON file (`'json_url'`):

```
POST https://<patch-server-url>/subscribe
Content-Type: application/json
Body: {"id": "<patch-title-name>", "json_url": "<url-to-JSON-file>"}
```

Successful response:

```
Status Code: 201
Content-Type: application/json
Body: {"success": ""PatchServer has subscribed to title '<patch-title-name>' at <url-
↪to-JSON-file>"}
```

Note: The Patch Server performs validations on all patch definitions it retrieves. If the validation fails, the subscription request will be rejected.

4.2 Patch Definition Syncing

Every five (5) minutes, the Patch Server will sync all subscribed patch definitions. It will download the source from the provided URL and write the definition into the S3 bucket, overwriting the last version of the patch definition.

Note: The Patch Server will perform validation on the patch definitions that it is syncing. If the validation fails, the downloaded patch definition will not be written to the S3 bucket and the existing version will be preserved.

4.3 Unsubscribe from a Patch Definition

To unsubscribe from a patch definition, make a POST request to the */unsubscribe* endpoint with the name of the software title:

```
POST https://<patch-server-url>/unsubscribe/<patch-title-name>
```

This will remove the patch definition from syncing and delete the patch definition file from the S3 bucket.

Successful response:

```
Status Code: 200
Content-Type: application/json
Body: {"success": "PatchServer has unsubscribed from the title '<patch-title-name>'"}
```

CHAPTER 5

About Patch Endpoints

All about the patch endpoints that deliver the patch definitions to Jamf Pro.

The following endpoints are exposed for this service for your Jamf Pro server to view and subscribe avaialable patch titles:

- `/software` : Lists all available patch titles that are hosted on your Patch Server. They will be returned in the following JSON format:

```
[
    {
        "name": "string",
        "publisher": "string",
        "lastMondified": "ISO date string",
        "currentVersion": "string",
        "id": "TitleName1"
    },
    {
        "name": "string",
        "publisher": "string",
        "lastMondified": "ISO date string",
        "currentVersion": "string",
        "id": "TitleName2"
    },
    {
        "name": "string",
        "publisher": "string",
        "lastMondified": "ISO date string",
        "currentVersion": "string",
        "id": "TitleName3"
    }
]
```

- `/software/TitleName1,TitleName2` : Returns a subset of patch titles. The titles must have their IDs passed in a commma separated string as shown. The returned data is the same as the `/sofware` endpoint.

```
[
    {
        "name": "string",
        "publisher": "string",
        "lastMondified": "ISO date string",
        "currentVersion": "string",
        "id": "TitleName1"
    },
    {
        "name": "string",
        "publisher": "string",
        "lastMondified": "ISO date string",
        "currentVersion": "string",
        "id": "TitleName2"
    }
]
```

- `/patch/TitleName` : Returns the full JSON patch definition for the provided patch title ID (see Jamf's documentation for more details on the patch title schema).

```
{
    "name": "string",
    "publisher": "string",
    "appName": "string",
    "bundleId": "string",
    "lastModified": "ISO date string",
    "currentVersion": "string",
    "requirements": ["Array of Requirement Objects"],
    "patches": ["Array of Patch Objects"],
    "extensionAttributes": ["Array of Extension Attribute Objects"],
    "id": "TitleName"
}
```

Each endpoint's full URL would be entered into your browser as:

```
https://<API-GATEWAY-ID>.execute-api.<REGION>.amazonaws.com/Prod/software
https://<API-GATEWAY-ID>.execute-api.<REGION>.amazonaws.com/Prod/software/TitleName1,
↪TitleName2
https://<API-GATEWAY-ID>.execute-api.<REGION>.amazonaws.com/Prod/patch/TitleName
```

CHAPTER 6

REST API

How to programmatically manage your patch definitions on your patch server.

Note: JSON Schema Validation:

All API endpoints use JSON schema validation to ensure the submitted JSON is valid for patch definitions and versions. Invalid definitions can cause issues with Jamf Pro when it retrieves data from an external patch source. The API will provide feedback on where a validation error occurred to assist you in troubleshooting your definitions. Because the API performs this validation, it is considered a safer method to loading your patch definitions than manually uploading the files to S3.

6.1 POST `/api/title`

Create a new patch definition for a software title on the Patch Server. The new patch title name will be taken from the `id` of the submitted definition.

Request:

```
POST https://<patch-server-url>/api/title
Content-Type: application/json
Body: {"<Patch-Definition-JSON"}
```

Successful response:

```
Status Code: 201
Content-Type: application/json
Body: {"success": "Successfully created patch definition for '<patch-title-name>'"}
```

6.2 PUT /api/title/<patch-title-name>

Replace an existing patch definition for a software title on the Patch Server that is not a subscription. This action does not perform any logic for merging changes between the current and submitted patch definitions.

Request:

```
POST https://<patch-server-url>/api/title/<patch-title-name>
Content-Type: application/json
Body: {"<Patch-Definition-JSON"}
```

Successful response:

```
Status Code: 200
Content-Type: application/json
Body: {"success": "Successfully updated the patch definition for '<patch-title-name>'
↪"}
```

6.3 POST /api/title/<patch-title-name>/version

Add a new patch version to an existing patch definition that is not a subscription. The `lastModified` and `version` values of the patch definition will be updated as a part of this operation.

Request:

```
POST https://<patch-server-url>/api/title/<patch-title-name>/version
Content-Type: application/json
Body: {"<Patch-Definition-JSON"}
```

Successful response:

```
Status Code: 201
Content-Type: application/json
Body: {"success": "Successfully updated the version for patch definition '<patch-
↪title-name>'"}
```