CA Release Automation

CDE Plugin Creation

Best Practices Guide

Author : Walter Guerrero

Version: 1.2

Filename: CA-Release-Automation-CDE-Plugin-Creation-Best-Practices-GuideV1.0.docx

Date: 6/2/2016

Table of Contents

Overview

Customers in their Continuous Delivery journey that is part of their Agile maturity have a new tool called Release Automation Continuous Delivery Edition, which permits customers the ability to build “releases”; and these releases can be part of a sprint, program increments, or milestones.

These releases are composed of multiple applications’ contents (which is what is going to be delivered at the end of the sprint or program increment). It becomes important as part of these releases that the different tools being used by the customer can be called directly by tasks that make up the RA CD Edition release.

Release Automation CDE provides the customers with the ability of creating additional plugins.

This document will take you thru the process that it will take for the creation of a RA CDE plugin based off the Hook.io micro services and utilizing GitHub as the provider of the files needed to interface with the Hook.io micro service. The flow for the creation of RA CDE plugin is as follows:

- Design the plugin by selecting the proper Restful API calls to be used.
- Test the Restful API calls independently.
- Setup the Github user implementation, where the gh-pages is configured and ready to run.
- Implement the RA CDE plugin JSON manifest
- Implement the RA CDE plugin JavaScript
- Create a hook.io micro service and copy the RA CDE plugin JavaScript that is part of the GitHub implementation.
- Add the plugin to the RA CDE implementation by importing the JSON manifest.
- Add an endpoint pointing to the Bamboo implementation.
- Add a task to a RA CDE release.

RA CDE Plugin Framework

The RA CDE plugin implementation follows the steps listed above, now we are going to start the detailed explanation of this process.

RA CDE —Bamboo Plugin

As part of the process in how to create a RA CDE plugin, we are going to utilize the Atlassian Bamboo tool as part of the method to follow.

To accomplish this integration, we are going to be performing the following restful calls dealing with the build processes initially.

- http://<bamboo-host>:<port>/rest/api/latest/plan?expand → this call was selected for you to get familiar with the Bamboo Restful calls. This call will provide you with all the plans that are presently available in the Bamboo server.
- http://<bamboo-host>:<port>/rest/api/queue/{planKey} → this is the call that we are going to use for the execution of builds in a Bamboo server.
    - Where <bamboo-host> → The hostname where the Bamboo server is running
    - Where <port> → the port number where the Bamboo server is listening on, and by default is 8085.

If you want to learn more about the Bamboo restful API, you can get that information at the following URL location.

- https://docs.atlassian.com/bamboo/REST/5.10.0

Calling Bamboo RESTful API

To successfully use the Bamboo RESTful API, the following pre-requisites are needed:

- HTTP basic authentication.
- Add the following header entry: "X-Atlassian-Token" with a value of "nocheck" to be able to execute the job build.
- Generate a Base64 string for the username/password combination. This can be generated by the postman application or you can use an online encoder/decoder to obtain this value.
- You can also use the Base64 API directly in your application.
- Define the HTML header in the micro service or JavaScript function.

The recommendation is to use the Postman utility to conduct most of your testing of the Bamboo RESTful API that you would need to use. Not only will postman help you build the correct RESTful API call, but it will also build the "curl" options as well.

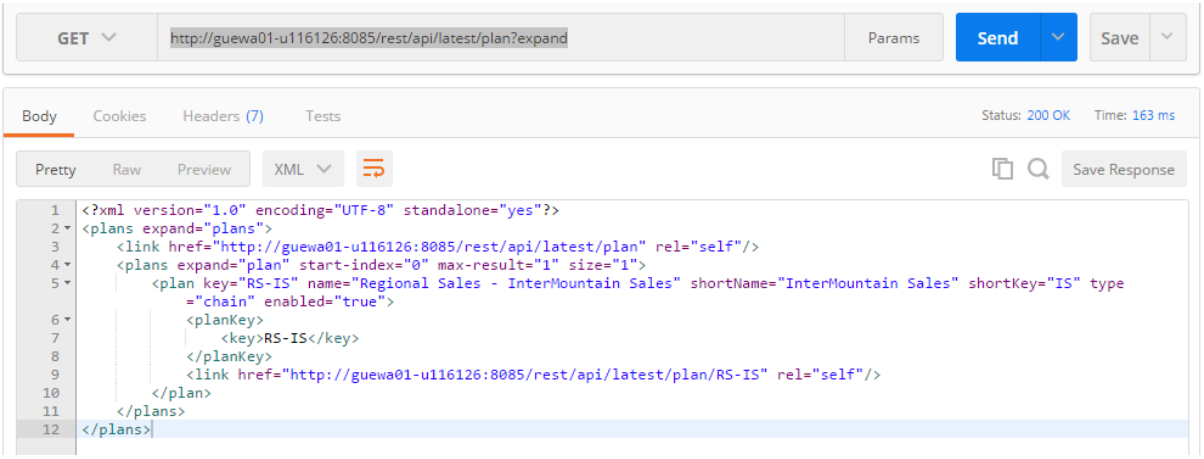

Figure 1: Typical Postman session

In the above figure, you see how the "/plan" call is built and the corresponding results in the postman session.

You will also need to take into account how the how the different parameters being used by the different GET/POST calls, these parameters can either be inline URL call, for example: /plan?expand

Setting up Hook.io micro service

The available Release Automation CDE plugin framework is based off the creation of the necessary micro-services as defined in the Hook.io (www.hook.io) implementation. This provides us with benefits of taking a very complex implementation and creating discrete, independent processes.

Hook.io (www.hook.io) support several languages, but we are going to concentrate on the creation of the necessary RA CDE plugins based off JavaScript language.

You can learn and download Hook.io from the following GitHub location (https://github.com/bigcompany/hook.io).

The Hook.io micro service is composed of three sections:

- GitHub Repository
    - Plugin manifest file
    - Plugin JavaScript function(s)
- The micro service contents

We are going to start by explaining how the manifest file is setup and where it would need to be located.

GitHub Repository

To make the creation of a successful RA CDE plugin, you will need to create a new GitHub repository (recommended).

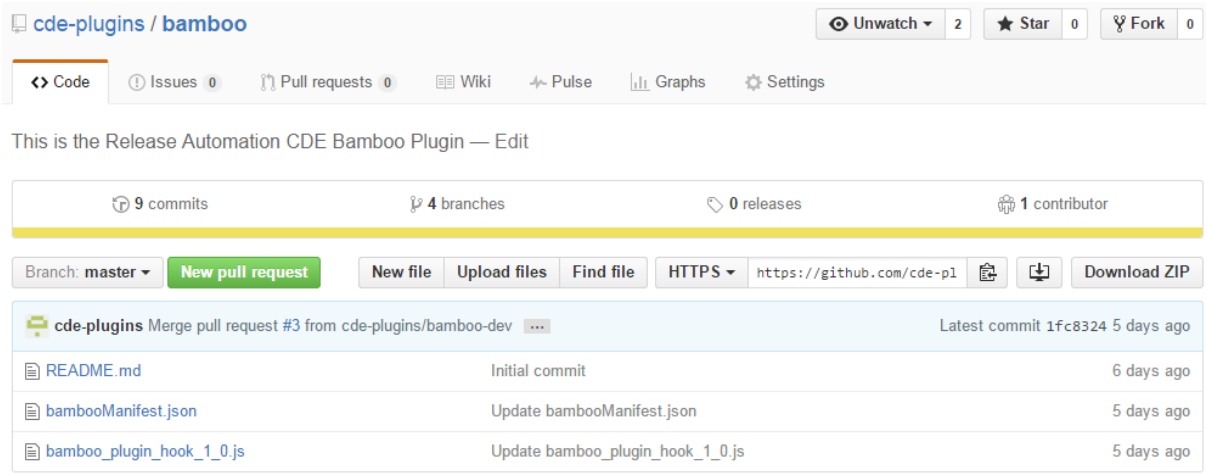

Figure 2: A GitHub Reposatory

After you have created the GitHub repository, it is recommended that you create the static GitHub pages by clicking in the “Launch automatic page generator” in the settings tab.

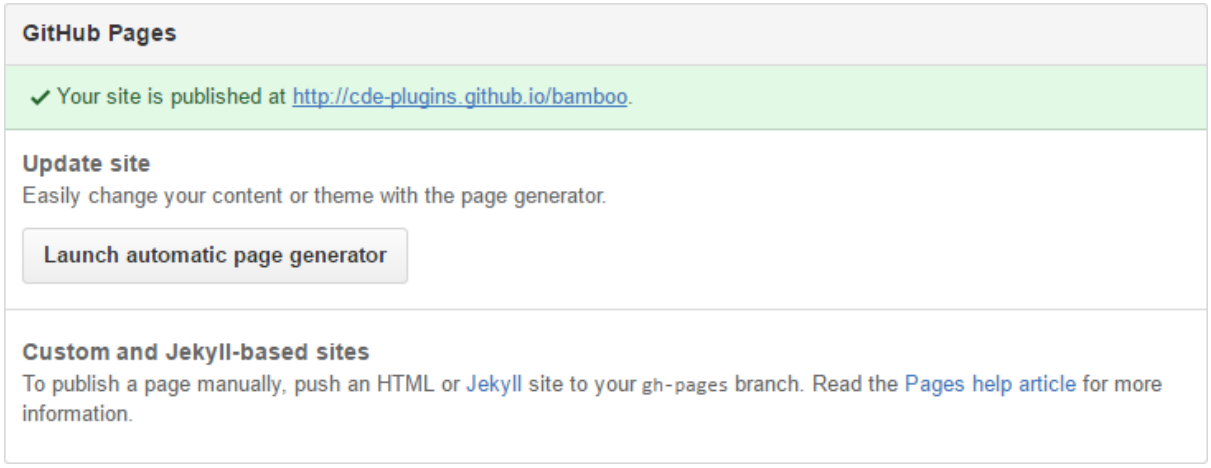

Figure 3: Adding GitHub Pages

Once the GitHub pages have been created, you can place the manifest and JavaScript function files in the *gh-pages* branch

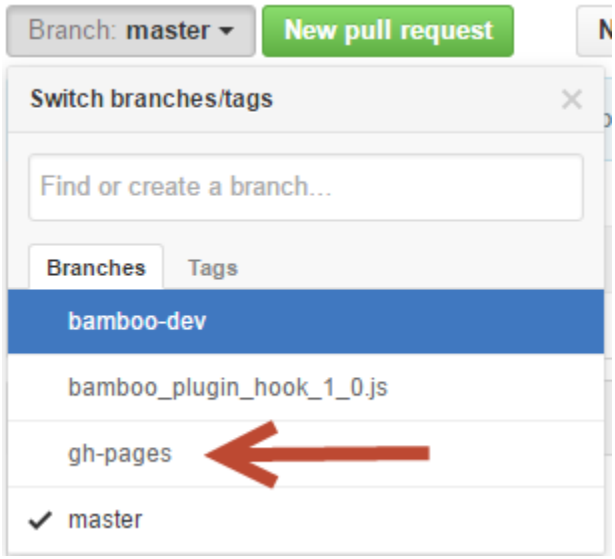

Figure 4: Selecting gh-pages branch

Since GitHub pages are static in nature and for you to be able to get to the manifest and JavaScript function files, you will need to place copies of these files in the "*gh-pages*" branch.

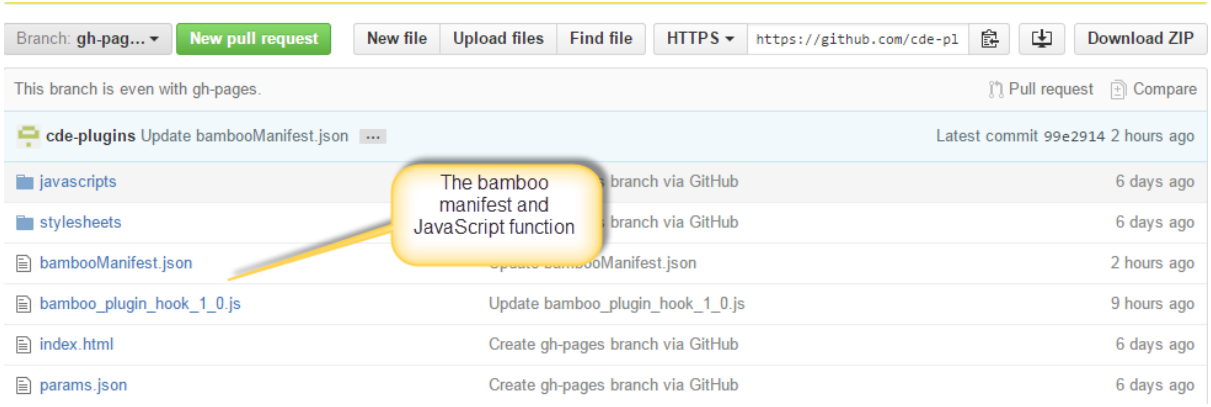

Figure 5: Adding files to gh-pages branchr

JSON Manifest File

The manifest file will describe the connection between Release Automation CDE and the plugin. This file contains the available tasks and the necessary endpoint template information.

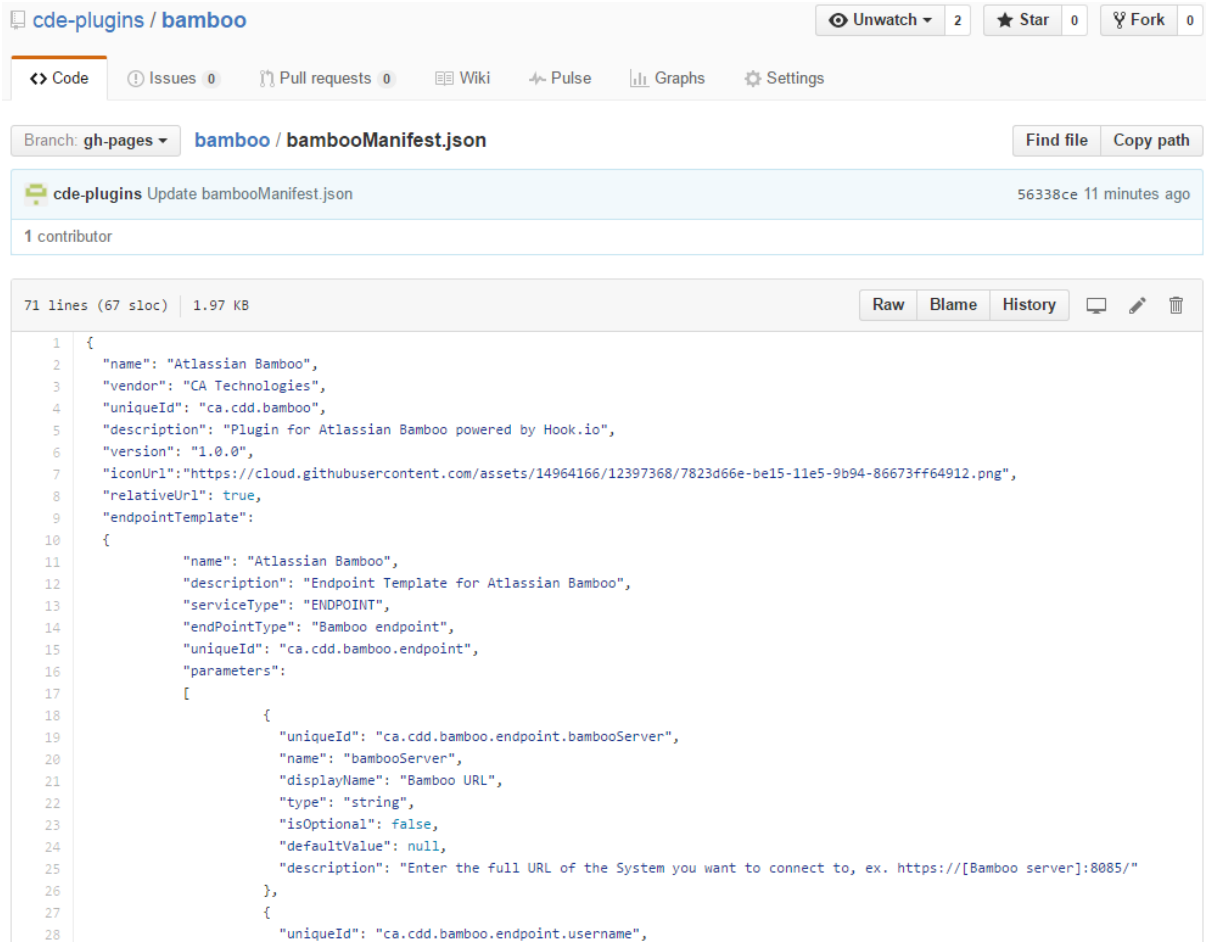

Figure 6: Manifest file

Contents of the proposed Bamboo-plugin manifest file

```
{
  "name": "Atlassian Bamboo",
  "vendor": "CA Technologies",
  "uniqueId": "ca.cdd.bamboo",
  "description": "Plugin for Atlassian Bamboo powered by Hook.io",
  "version": "1.0.0",
  "iconUrl":"https://cloud.githubusercontent.com/assets/14964166/12397368/7823d66e-be15-11e5-
9b94-86673ff64912.png",
  "relativeUrl": true,
  "endpointTemplate":
  {
            "name": "Atlassian Bamboo",
            "description": "Endpoint Template for Atlassian Bamboo",
            "serviceType": "ENDPOINT",
            "endPointType": "Bamboo endpoint",
            "uniqueId": "ca.cdd.bamboo.endpoint",
            "parameters":
            [
                     {
                       "uniqueId": "ca.cdd.bamboo.endpoint.bambooServer",
                       "name": "bambooServer",
                       "displayName": "Bamboo URL",
                       "type": "string",
                       "isOptional": false,
                       "defaultValue": null,
                       "description": "Enter the full URL of the System you want to connect
to, ex. https://[Bamboo server]:8085/"
                     },
                     {
                       "uniqueId": "ca.cdd.bamboo.endpoint.username",
                       "name": "username",
                       "displayName": "Bamboo Username",
                       "type": "string",
                       "isOptional": false,
                       "defaultValue": null,
```

```
                    "description": "Enter the Bamboo username"
                },
                {
                    "uniqueId": "ca.cdd.bamboo.endpoint.password",
                    "name": "password",
                    "displayName": "Bamboo User Password",
                    "type": "password",
                    "isOptional": false,
                    "defaultValue": null,
                    "description": "Enter the Bamboo user's password"
                }

            ]
    },
        "services":
        [
                {
                    "name": "Bamboo Run Build",
                    "uniqueId": "ca.cdd.bamboo.task.run_build",
                    "description": "Use this task to run a Bamboo build",
                    "serviceType": "TASK",
                    "url":"rest/api/latest/queue",
                    "parameters":
                    [
                            {
                                    "name": "planKey",
                                    "uniqueId": "ca.cdd.bamboo.task.run_build.planKey",
                                    "displayName": "Bamboo Plan Key",
                                    "type": "string",
                                    "isOptional": false
                            }
                    ]
                }
        ]


}
```

JavaScript Function

The implementation of the hook.io micro service that will carry out the task defined in the JSON manifest file will be look something like the following entries.

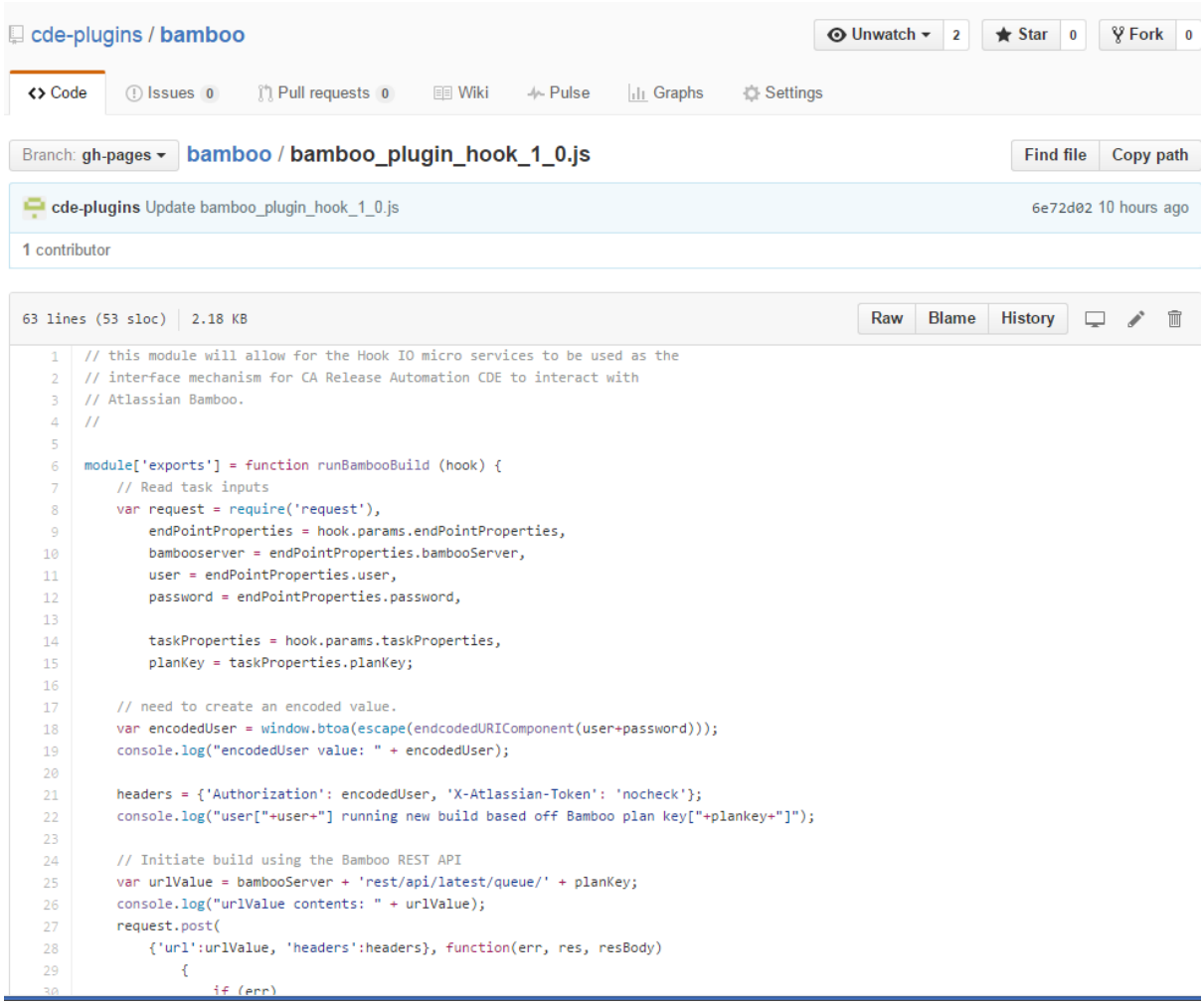

Figure 7: Implementation of Hook.io micro service

Here is the contents of a typical JavaScript function that will implement the Hook.io micro service.

```
// this module will allow for the Hook IO micro services to be used as the
// interface mechanism for CA Release Automation CDE to interact with
// Atlassian Bamboo.
//
module['exports'] = function runBambooBuild (hook) {
    // Read task inputs
    var request = require('request'),
        endPointProperties = hook.params.endPointProperties,
        bambooserver = endPointProperties.bambooServer,
        user = endPointProperties.user,
        password = endPointProperties.password,

        taskProperties = hook.params.taskProperties,
        planKey = taskProperties.planKey;

    // need to create an encoded value.
    var encodedUser = window.btoa(escape(endcodedURIComponent(user+password)));
    console.log("encodedUser value: " + encodedUser);

    headers = {'Authorization': encodedUser, 'X-Atlassian-Token': 'nocheck'};
    console.log("user["+user+"] running new build based off Bamboo plan key["+plankey+"]");

    // Initiate build using the Bamboo REST API
    var urlValue = bambooServer + 'rest/api/latest/queue/' + planKey;
    console.log("urlValue contents: " + urlValue);
    request.post(
        {'url':urlValue, 'headers':headers}, function(err, res, resBody)
```

```
            {
                if (err)
                {
                    return hook.res.end(err.messsage);
                }

            // Build response
            hook.res.setHeader("Content-Type", "application/xml");

            // now processing the response
            var resBuild = "/s:restQueueBuild/buildResultKey";
            var responseNode = XML.getNode(hook.resBody, resBuild);

            hook.res.end(JSON.stringify(
                        {
                            'bamboo build' : "+responseNode"
                        }
                    )
                );

            hook.res.end(JSON.stringify(
                {
                  'externalTaskExecutionStatus' : 'FINISHED',
                  'executionContext' : {},
                  'taskState' : "Bamboo build has been issued"+responseNode,
                  'detailedInfo': "Bamboo build for plan key " +planKey+ "has been issued, build
number: " +responseNode,
                  'progress' : 100,
                  'delayTillNextPoll' : 0
                }
              )
            );
        }
)
};
```

Hook.io Micro Service

We need to define the micro service that will be used to make the CA Release Automation CDE plugin work correctly. First, you will need to login to http://hook.io

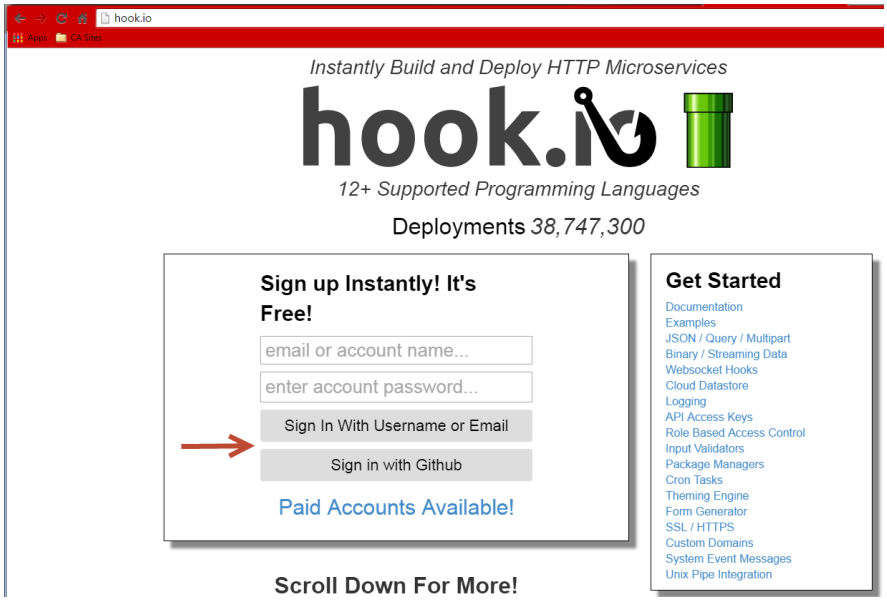

Figure 8: Landing page for hook.io

Login to Hook IO either by using your email or GitHub account (it is recommended that the GitHub account be used).

Once you have login to Hook IO, you will see the following screen:

Figure 9: Your Hook IO view

Select the "Create Hook" option, and enter the following information: "Service Name" initially, we are going to be discussing the additional entries and logic shortly.

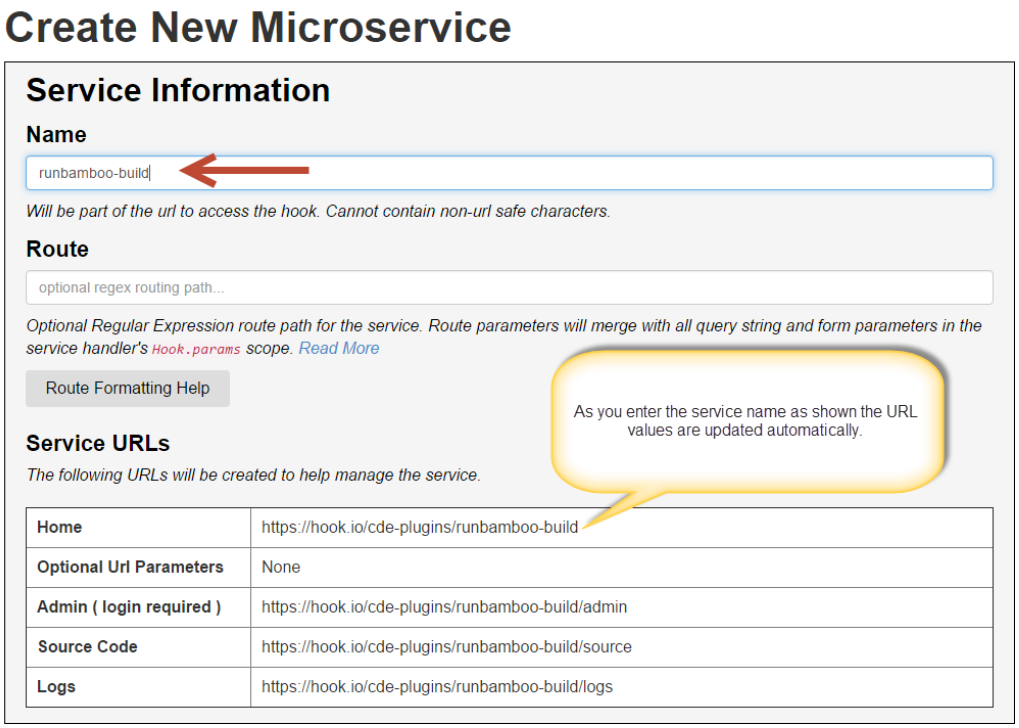

Figure 10: Hook IO Micro service definition

In the above example, we are defining a service name "runbamboo-build", which will eventually execute a Bamboo defined build utilizing the Bamboo Restful API.

Please keep in mind that the default service security is going to be "Public Service", you can setup this security to be private in nature, but there are costs associated. Please review Hook.io "paid account" for additional information.

Copy the code from the JavaScript that is part of the GitHub repository implementation, for example the “bamboo_plugin_hook_1_0.js” in the Hook source window.

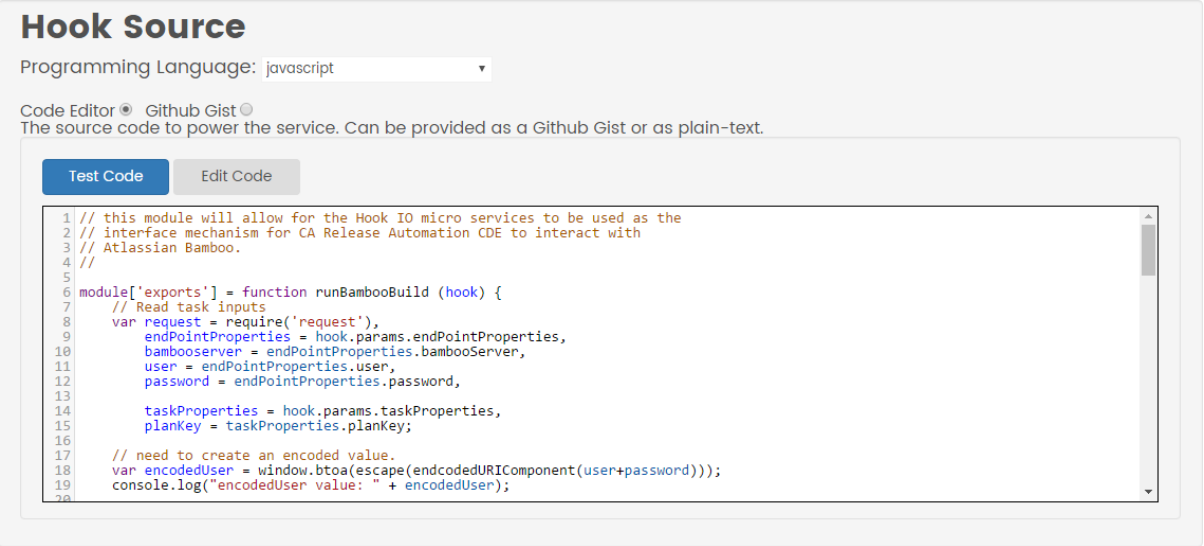

Figure 11: Source Code for runbamboo-build hook

After you have enter the necessary source code for the hook, you can click the “Create new Hook” button for the creation of the Hook.io micro service.

You will get the following message stating that the hook has been created and ready for you to use.

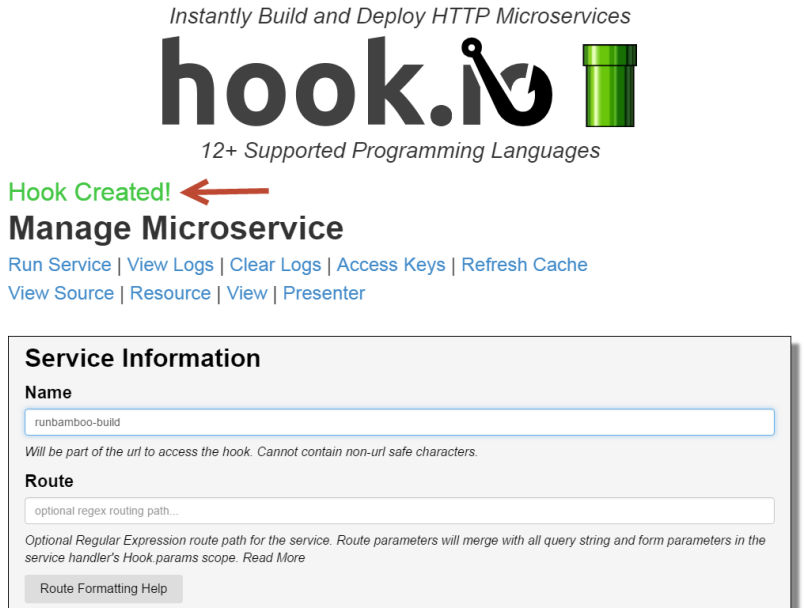

Figure 12: The Hook has been created message

Adding Plugin to RA CDE

After you have completed testing the access to the different hook.io methods, you will need to define the following in Release Automation CDE:

- Register the plugin
- Add the bamboo endpoint based off the registered plugin
- Add a task to a given release's phase

The above steps are accomplished by following the steps below:

Register the plugin with Release Automation CDE, in this scenario that plugin will be Bamboo as described in the prior sections. To get to plugins, select Administration→Plug-ins in RA CDE.

Figure 13: Registering the plugin

Now we are going to register the Bamboo plugin, you will take the complete URL for the manifest file that describes how the plugin will interface with RA CDE, click the "Register" button to complete the plugin registration.

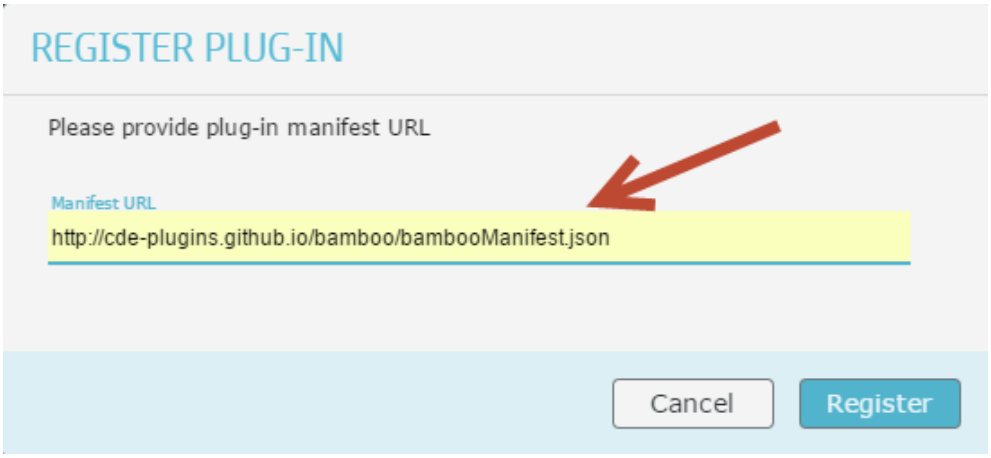

Figure 14: Enter manifest URLr

Once the Bamboo plugin has registered successfully, you will see it listed as one of the available plugins and its available services.

Figure 15: Registered Bamboo plugin

After the Bamboo plugin has been registered, we move to the Administration→Endpoints to add an endpoint for the tasks as defined as part of a release's phases can be executed. Click the :Add Endpoint" to start the process.

Figure 16: Add Endpoint

Enter the necessary information as shown below, you will have to select the correct endpoint type. Click the "Add" button to complete adding the Bamboo endpoint.

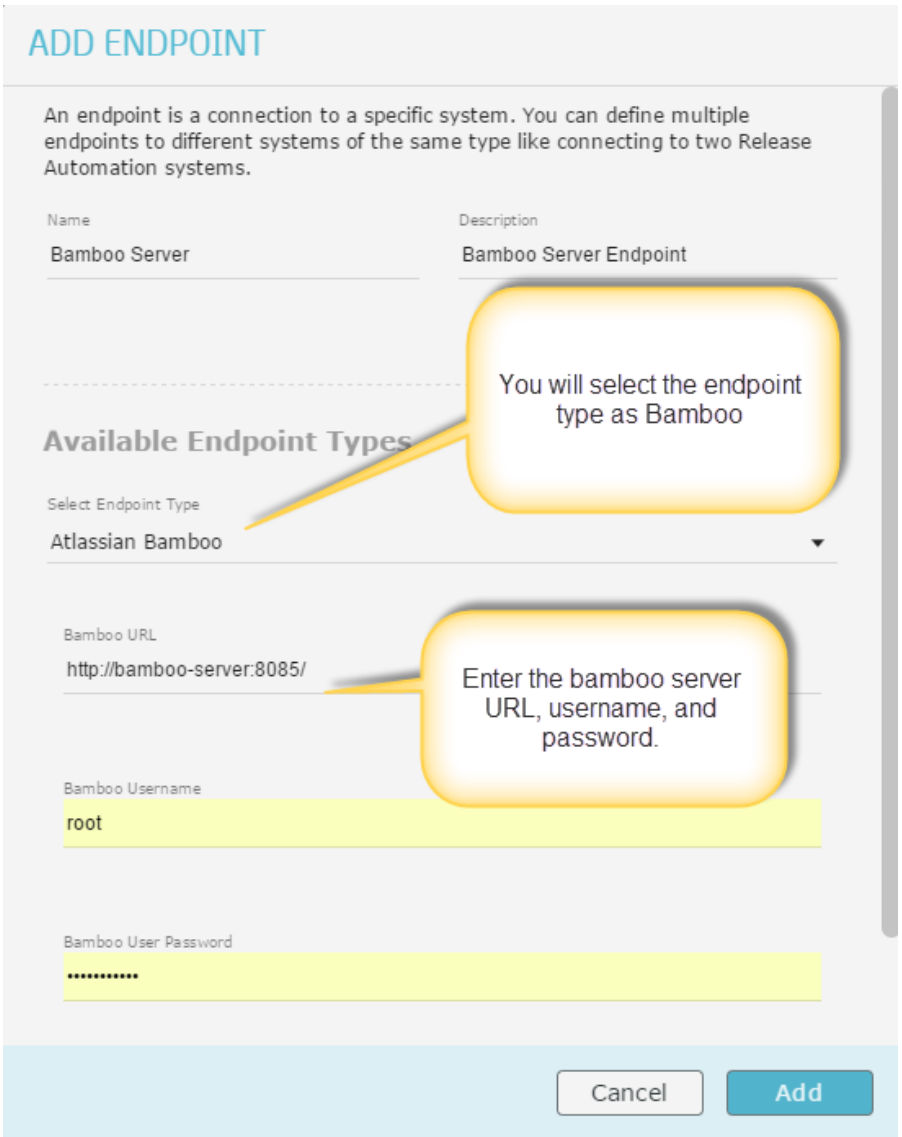

Figure 17: Adding Endpoint

Now we need to go to a release, which already has a phase defined and select “Create a task…” option in one of the phases and add entries as shown below, once you have entered the necessary information, click on the “Create” button.

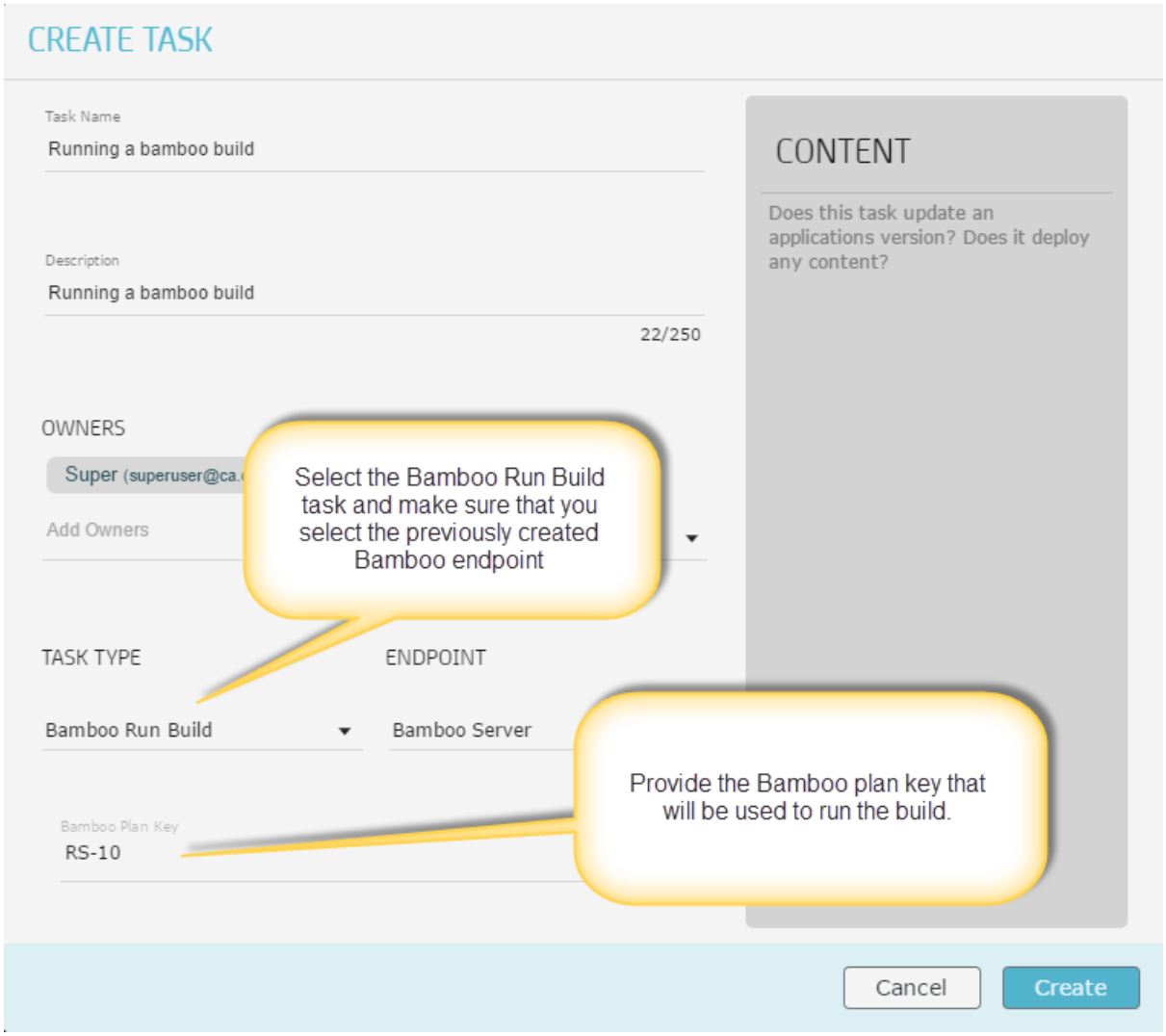

Figure 18: Creating Bamboo task

The newly created Bamboo task will be shown in the phase.

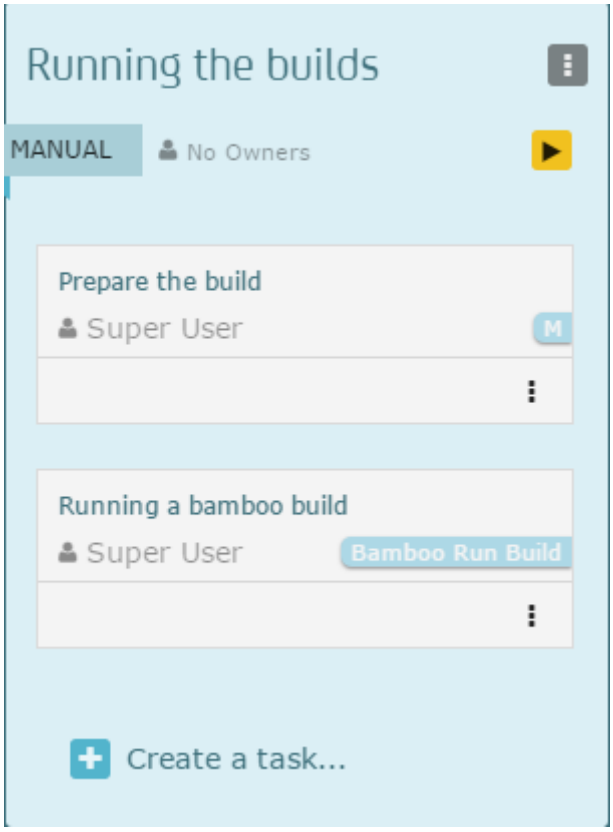

Figure 19: Bamboo task as part of phase

Best Practices

The following best practices will help you in creating a Release Automation CDE plugin, these best practices will take from planning, setting up manual tasks to emulate the new tasks, tools to be used.

Planning

As in any development project, it is important that you plan how the RA CDE plugin will interface and interact with the targeted tool, as well as availability of the plugin's resources.

For the plugin to work correctly, it is important that the following conditions are met.

- A HTTP service that can accept a POST request, instrument the requested operation, and return a response.
- A "manifest.json" file is needed, which contains the details of the plugin's capabilities.
- For offline implementations, you will need to setup a JAVA project via Eclipse or IntelliJ IDEA.
- For online implementations, you need to make sure that you have access to the online service that will be performing the call.

Tools

Prior to starting the development effort of the RA CDE plugin, make sure that the targeted tool is accessible online, as well as making sure that the tool's server can interact with the GitHub implementation of Hook.io.

Using the proper version of the API calls

Some of the API calls can be used as version 1 (v1) or version 2 (v2), and there are other calls that will provide you with different results based on the version being utilized.

Using the correct parameter method

Given that some API calls use URL parameters and other calls are using JSON body parameters, it becomes very crucial that you become aware of the requirements for the API calls that you are going to be using. This is where the tools listed in this document can be of great help to you in determining how build the parameters require for a given API call.

Recommended Tools

To fully test the different Bamboo RESTful API calls prior to creating the necessary Hook.io micro services, the following tools can be used:

- Postman by postman running as a standalone Google Chrome application or plug-in.
- REST Easy extension in FireFox.
- HttpRequester extension in FireFox.
- Curl, which is a command-line utility used extensively to test the different HTTP get/post calls.
- SOAPUI 5.2.x or greater.

Copyright Notice

Useful Links

https://docs.atlassian.com/bamboo/REST/5.10.0/

http://hook.io/

https://docops.ca.com/ca-release-automation-continuous-delivery-edition/6-1/en

http://www.getpostman.com/docs

http://github.com

https://www.base64decode.org/

http://tomcat.apache.org

http://www.eclipse.org

http://www.jetbrains.com/idea/