InRule® Deployment Brief: Publishing Rule Changes to JavaScript

Use Case

You may be considering the use of InRule® *for JavaScript* within a web application to improve the user experience or within a hybrid mobile application to support disconnected rule execution. If so, you might be wondering how rule changes will ultimately make their way from the rule author to the consuming application.

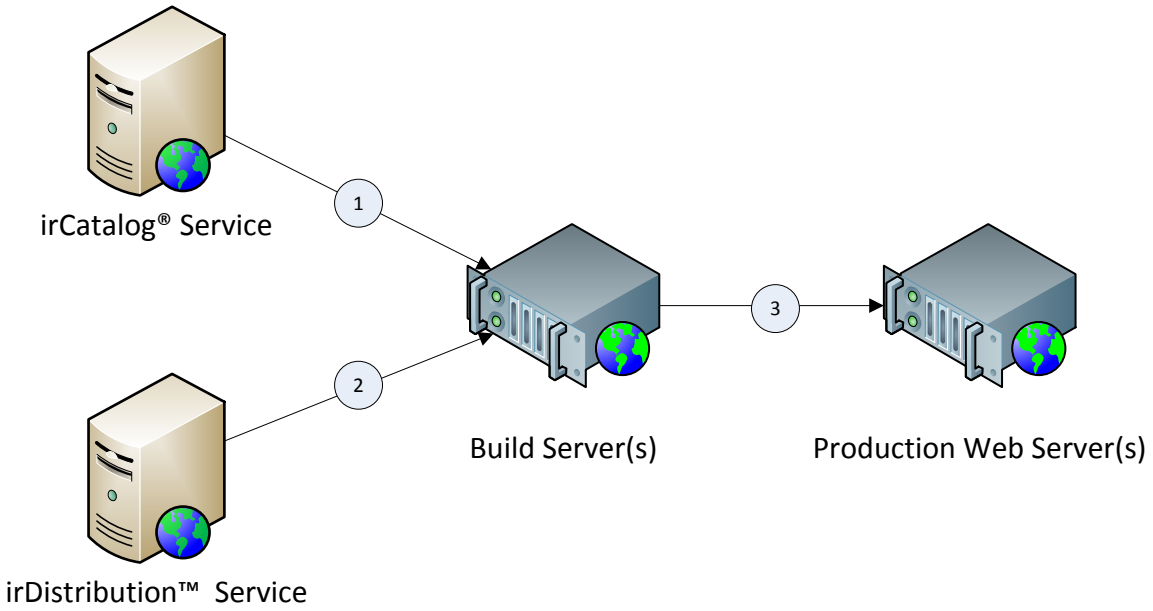
This deployment brief covers the mechanics of how you can include InRule *for JavaScript* in a build and deployment process. Concepts such as governance and approval of rule changes will not be covered.

Before the Build Process Begins

In most enterprises, the rule author saves their changes to an irCatalog® instance through irAuthor®. Multiple irCatalog instances may exist for different environments and are promoted either manually through the irCatalog Manager tool or programmatically using an irSDK® API. Rules can alternatively be saved to a file which is persisted as XML and so made available to the build process as a file or via source control.

Packaging and Deploying an Updated JavaScript File

The following diagram shows the inclusion of two InRule services called during the build process.



Step 1

The build server retrieves the latest version of the rules from the irCatalog service as shown in the following C# code. Note that in this example rules can also be retrieved from a file location. A third option, not shown, is to retrieve the rules as XML from a source control system.

```
        private RuleApplicationDef GetRuleApplication(string ruleAppName, int revision)
        {
            RuleApplicationDef ruleAppDef;
            var useCatalog = true;
            if (useCatalog)
            {
                using (var connection = new RuleCatalogConnection(new
Uri("http://localhost/InRuleCatalogService/service.svc"), new TimeSpan(0, 0, 60),
Settings.Default.CatalogUser, Settings.Default.CatalogPassword))
                {
                    if (revision == 0) // get latest
                    {
                        ruleAppDef = connection.GetLatestRuleAppRevision(ruleAppName);
                    }
                    else
                    {
                        var ruleAppref = connection.GetRuleAppRef(ruleAppName, revision);
                        ruleAppDef =
connection.GetSpecificRuleAppRevision(ruleAppref.Guid, ruleAppref.PublicRevision);
                    }
                }
            }
            else
            {
                var path = AppDomain.CurrentDomain.GetData("DataDirectory") + "\\" +
ruleAppName + ".ruleapp";
                ruleAppDef = RuleApplicationDef.Load(path);
            }

            return ruleAppDef;
        }
```

Step 2

The build server then passes the rule application to the cloud-based irDistribution™ service via a web service call. The following C# code provides an example of how that could be done.

```
        public string GetJavaScriptFromRuleApp(string ruleAppXml, bool turnOnLogging)
        {
            var client = new HttpClient();

            using (var mpfdc = new MultipartFormDataContent())
            {
                byte[] byteArray = System.Text.Encoding.UTF8.GetBytes(ruleAppXml);
                var stream = new MemoryStream(byteArray);
                var httpContent = new StreamContent(stream);
                mpfdc.Add(httpContent, "ruleApplication", "Healthinsurance.ruleapp");
                client.DefaultRequestHeaders.Accept.Add(new
System.Net.Http.Headers.MediaTypeWithQualityHeaderValue("multipart/form-data"));
                string selectedVersion = null;
                var queryString = new System.Text.StringBuilder();
```

```
                queryString.AppendFormat("?logOption=None&subscription-key={0}", "[your
key here]");

                var result =
client.PostAsync("https://irjavascriptpackagingproduction.azure-api.net/" +
selectedVersion + "package" + queryString, mpfdc).Result;


                dynamic resource = result.Content.ReadAsAsync<JObject>().Result;


                var resultDownload =
client.GetAsync((string)resource.PackagedApplicationDownloadUrl + "?subscription-key=" +
"[your key]").Result;

                File.WriteAllText(@"C:\TEMP\jsoutput.txt",
resultDownload.Content.ReadAsStringAsync().Result);

                return resultDownload.Content.ReadAsStringAsync().Result;
            }
        }
```

Step 3

The JavaScript returned by the service is then persisted to a file and included with the rest of the deployment package.

For More Information

For more information on how to integrate the JavaScript rule engine into an application, please see *InRule Integration Brief: Client-side Rule Execution*.

For more information on pushing rule changes to a mobile application, please see *InRule Deployment Brief: Disconnected Mobile Applications*.

For additional information please visit inrule.com, contact us or reach out to your InRule Account Executive to get in touch with a pre-sales engineer.