INRULE

InRule for Salesforce

Deployment Guide

Document Updated against InRule v5.8.1

Document Updated against InRule for Salesforce v2.19.0

InRule does not upgrade this document after each Integration Framework release, please see release notes for individual versions if the version that you are using does not match the versions listed above.

If you are working with earlier versions of any of the above products, the information in this document may not apply to you. Please check to see if earlier documentation is available to cover your needs.

Table of Contents

1 Introducing InRule for Salesforce

InRule for Salesforce enables rich rule integration with Salesforce. The solution is comprised of:

- **irAuthor with irX for Salesforce** – extension to irAuthor, InRule's premium desktop rule authoring tool
- **InRule App for Salesforce**– available from the Salesforce AppExchange
- **Rule Execution Services for Salesforce** - InRule SaaS, self-hosted in Azure, or other

This guide focuses on the deployment of the Rule Execution Services for Salesforce and corresponding InRule Salesforce App to your environment. The InRule Salesforce App may be deployed directly from the Salesforce AppExchange or from the Integration Framework zip file downloaded from the InRule Support Site.

Before beginning this guide, you may first want to familiarize yourself with the *irX for Salesforce* product by reading the *irX for Salesforce Help Documentation*. This irAuthor extension will allow you to author rules against Salesforce entities and become familiar with the types of rules-driven processes that can be implemented. After testing locally from your desktop using irVerify, the rules will be ready for execution from Salesforce. At this point, this guide will become highly relevant for deploying the InRule solution and services to establish the selected integration patterns.

This document also provides an addendum, Appendix E: *Methods for Executing Rules from Salesforce* that discusses the different options available to you for running rules beyond what this Deployment Guide covers. It is a good next step for implementers who are looking for advanced options for running rules.

Additional material is available on the Downloads section of our support website. Please see Appendix A: Additional Resources for support website detail. To find what's new, see Appendix J: New Releases and Upgrading Versions.

This guide assumes that the reader has basic familiarity with Salesforce, irAuthor and irX for Salesforce.

Important: Not all Salesforce environments support API access. However, InRule for Salesforce requires a Salesforce instance that supports the API. If you attempt to connect with an instance that does not support this access, you will receive an error when connecting that contains information such as "The REST API is not enabled for this Organization." The edition types that do NOT include API access are Contact Edition, Group Edition and Professional Edition. Please refer to official Salesforce documentation for up-to-date information as this list is subject to change.

2 Understanding your options

Salesforce is available only as an online SaaS offering, but the Rule Execution Service and Catalog Service components of InRule for Salesforce can be hosted anywhere an IIS site can be deployed. This guide assumes deployment in Azure, but an on-prem or VM environment can also be used if required.

The following sections describe which deployment scenario will be the most applicable depending on your needs:

- 2.1 Salesforce – InRule SaaS
- 2.2 Salesforce – Self-hosted in Microsoft Azure
- 2.3 Salesforce – IIS On-Premises Rule Execution Service

The primary consideration for InRule SaaS vs self-hosted Azure generally depends on if your organization is already setup to manage an Azure subscription and services.

- **If your organization does not have an Azure subscription**, this can be compelling rationale to go with InRule SaaS and forgo the overhead of Azure management and deployment.
- **If your organization has an Azure subscription**, it does not exclude you from choosing InRule SaaS, but it may indicate that your IT department has requirements or preferences for self-managed Azure solutions.

Either way, this guide will provide you with the information to help determine which InRule configuration will best suit your needs, including Government Cloud and other considerations.

2.1 Salesforce – InRule SaaS

InRule for Salesforce is now available via InRule SaaS, in which case many of the deployment steps in this document are not applicable and will be managed for you. This is the most stream-lined deployment available to both qualified Trial users and licensed customers.

Important: If you are new to InRule and do not have an InRule SaaS subscription, you can request a free trial here and specify that you would like to integrate it with your Salesforce instance.

With InRule SaaS, the deployment steps are significantly reduced by eliminating the need to install the InRule's App Services in Azure. This narrows the deployment focus down to irAuthor with irX for Salesforce and the InRule for Salesforce App from the Salesforce AppExchange. The main deployment steps in this scenario are summarized as follows:

Deployment Step	Installation Type
1. irAuthor with irX for Salesforce	InRule installer via the InRule Support Site
2. Rule Execution Services for Salesforce	*managed by InRule SaaS Support
3. InRule for Salesforce App	Salesforce AppExchange

All information to set up the connectivity between Rule Execution Services hosted by InRule SaaS and your organization's Salesforce instance can be managed through the InRule SaaS Portal. The following configuration is what will be managed through the Portal. More information can be found in Appendix G: InRule SaaS Portal Configuration.

Entered by users in the SaaS Portal – configuration information for the Rule Execution App Services

- Service account credentials for Salesforce, including that account's API security token – used to allow the InRule SaaS-hosted Rule Execution Service to connect to Salesforce.
- Salesforce Connected App Consumer Key and Consumer Secret information -- used to allow the InRule SaaS-hosted Rule Execution Service to connect to Salesforce.

Provided to users in the SaaS Portal – configuration information for the InRule Salesforce Package:

- Rule Execution App Service endpoint URL – used for allowing Salesforce to make requests out to the Rule Execution App Service
- Rule Execution App Service access credentials -- used for allowing Salesforce to make requests out to the Rule Execution App Service

The net effect of an InRule SaaS deployment is that after the above pre-requisites are met, you get to ***skip ahead*** to Section 3: Configuring the InRule for Salesforce App. Beyond this initial setup the remainder of the guide will be beneficial for detailing advanced rule execution methods and troubleshooting.

2.2 Salesforce – Self-hosted in Microsoft Azure

The suggested setup is to install InRule using a Platform-As-A-Service (PaaS) model on Microsoft Azure. This document discusses the following three components:

1. Catalog App Service and Azure SQL Database
2. Rule Execution App Service for Salesforce
3. InRule Decision Client package for Salesforce

Section 3 of this document, Performing the Installation: In Azure, provides a complete walkthrough for setting up each of these components.

Appendix B: Anatomy of a Request for Execution of Rules Diagram contains a more complete diagram depicting how a standard request navigates through components.

2.3 Salesforce - IIS On-Premises Rule Execution Service

If for any reason you cannot use the Azure App Service, the Rule Execution Service and Catalog Service can also be installed in an on-prem environment, or in any IaaS server configured with IIS. The following three components are required:

1. Catalog Web Service and Database
2. Rule Execution Web Service for Salesforce
3. InRule Decision Client package for Salesforce

Installation Documentation and InRule Installer, which will be used to install the Catalog Service, are available on our support website's downloads section and provide instructions for setting up the catalog.

Detailed installation steps are not included with this guide, but the InRule.Salesforce.WebService.zip package in the RuleExecutionAzureService folder can also be deployed in an on-prem environment. This is a Web Deploy package, so it can be deployed via MSDeploy or by copying the contents to an IIS Site. Alternatively, an MSI is also provided, located in the same file directory as the aforementioned Web

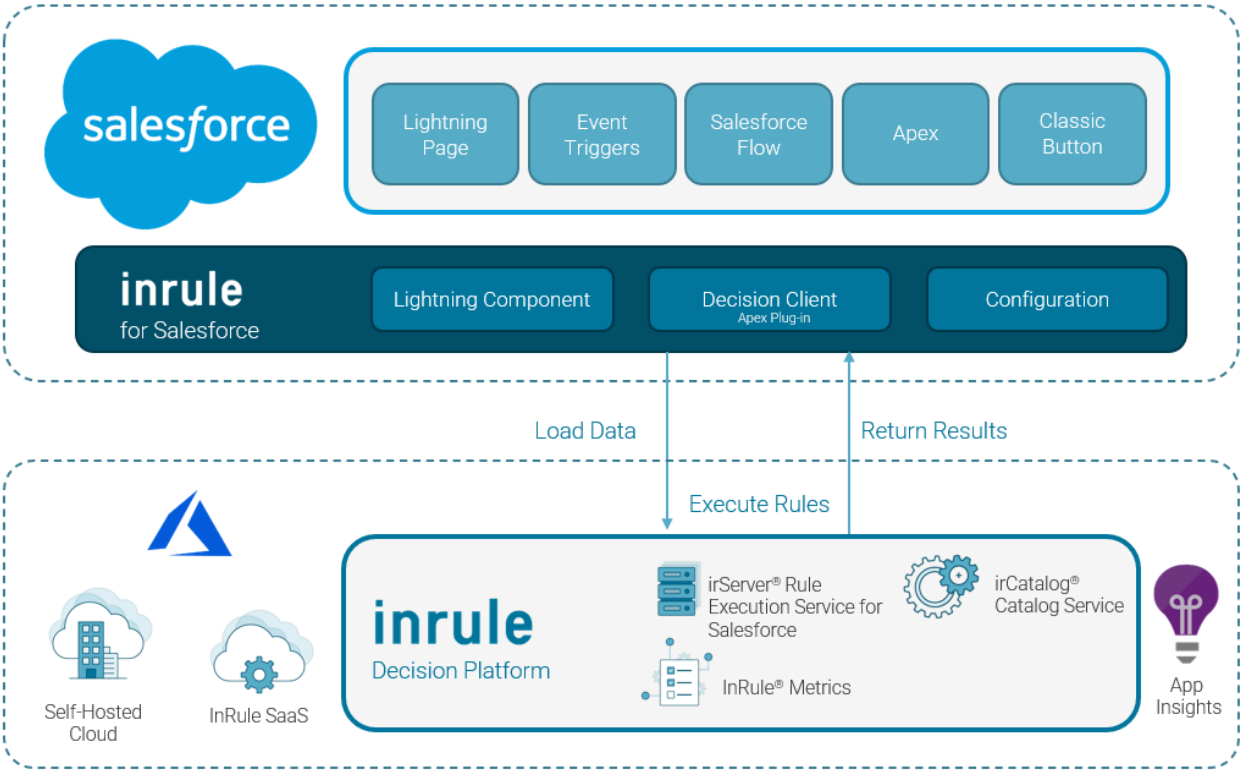
Deploy package. This can provide a more streamlined installation process for on-prem Rule Execution services.

3 Performing the Installation

This section discusses the steps needed to integrate InRule with Salesforce. Using this scenario InRule components are hosted on Microsoft Azure.

3.1 Solution Overview

This guide will provide the instructions for setting up all the components below:



Catalog App Service and Azure SQL Database

A Catalog service will be used to store Rule Apps that will be consumed by the Rule Execution App Service. This Catalog Service will be hosted as an Azure App Service. The back end of the Catalog Service utilizes an Azure SQL Database for retrieval and persistence of Rule Applications.

Rule Execution App Service for Salesforce

The Rule Execution App Service is responsible for loading Salesforce entity data, executing rules against loaded data, and responding to Salesforce with rule execution results.

InRule for Salesforce App

The InRule for Salesforce App contains Apex classes, as well as custom settings and objects. It is configured to communicate with the execution service over REST.

3.2 Gathering prerequisites

This section reviews what you will want to have prepared before you begin with the integration steps in the next section.

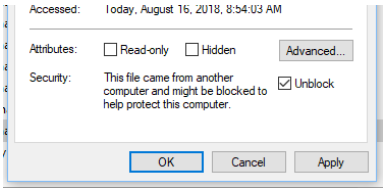
Optional Resource Files

The following file can be downloaded from our support website's downloads section:

- `InRule for Salesforce.zip`

This zip file contains several resources that can be used in advanced deployments. A typical installation does not require the use of these resources, as all the required deployment assets are available online. However, this zip file provides an alternative means of accessing the InRule for Salesforce assets.

After you have downloaded this file, but before extracting, make sure that you go to the file properties for the zip and select **Unblock**. If the zip file is not unblocked before extracting, the deployment scripts will not be able to execute successfully.



After unblocking the zip file, extract the contents to a working folder. When you are finished, you should have a directory structure that looks like this:

```
|   InRule for Salesforce.zip
|
└──InRule for Salesforce
    |   readme.txt
    |
    ├──RuleApplications
    |       SalesforceRules.ruleappx
    |
    ├──RuleExecutionAzureService
    |       azuredeploy.json
    |       azuredeploy.parameters.json
    |       InRule.Salesforce.WebService.zip
    |
    └──RuleHelperDeployment
            InRule.Salesforce.RuleHelper.dll
            ...(many other supporting files)
```

Rule Authoring Environment

A rule authoring environment is used to upload a Rule Application to your Catalog Service. A rule authoring environment is a machine or virtual machine where irAuthor has been installed with the irX for Salesforce extension. If you followed the instructions outlined in *irX for Salesforce Help Documentation,* then you should already have a rule authoring environment available to you.

We have made it a point to call out the rule authoring environment separately because it is important to be aware of the licensing implications of this step. You will need to utilize an irAuthor license and an irX for Salesforce license to activate the related components in the authoring environment. If you are a system administrator who does not intend to perform rule authoring activities after the deployment is up and running, you can either chose to borrow an environment from someone who will use a rule authoring environment, or you will want to be sure to deactivate your license when you're finished with your deployment responsibilities.

Administrative Accounts

Salesforce Service Account

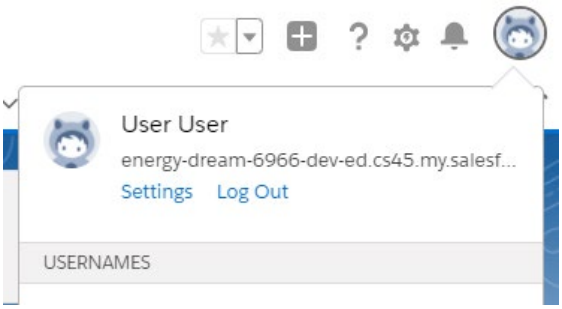
You will need an existing or new Salesforce account to use as a service account to connect to the Salesforce API. This account will be used by the rule execution service to load and save data from Salesforce required by rules. This account does not need to be a System Administrator account, but it will need permissions to use the Salesforce API and interact with entities used by rules.

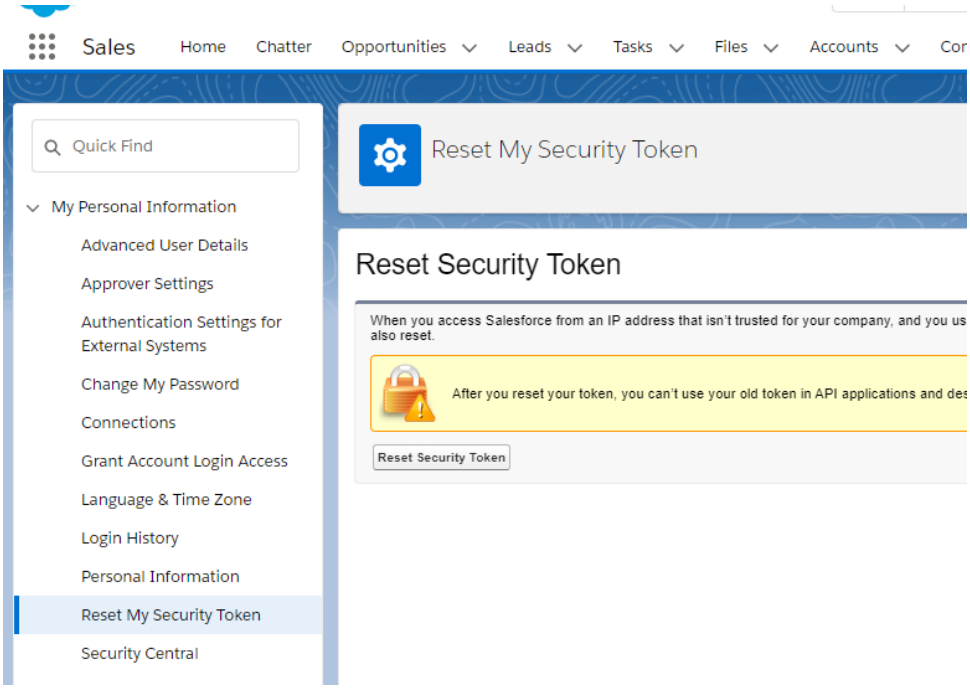
Security Token

If the account does not already have an API token, then log into the Salesforce portal and go to the user Settings from the avatar menu in the top-right. On the sidebar that appears, navigate to My Personal Information → Reset My Security Token. On the page that appears, click the 'Reset Security Token', and you should receive an email within a few minutes that contains an API token.

Important: Security tokens are, in most scenarios, **required.** They are only optional in scenarios when the account attempting to authenticate is connecting from a Trusted IP address. Trusted IPs can be set globally, meaning any user that connects from an IP in the trusted range will be regarded as trusted, or on the Profile level, meaning that only users assigned to the given Profile will be regarded as trusted when connecting from an IP that falls in the trusted range. But regardless of how Trusted IPs are configured, if the account in question does not fall in a configured Trusted IP range, it **must** have a security token set.

If trusted IP ranges are enabled for this account, you will not see the 'Reset My Security Token' button. If this is the case, simply leave this value blank where required later.





Other Credentials

Administrative Password to use for Catalog Service: You should decide what username and password you want to use for administrative privileges within Catalog. You will use this password when following the referenced catalog setup guide and will need to provide it when deploying the Execution Service.

*** This walkthrough utilizes the default login of 'admin' and password of 'password'. It will be up to the reader to go through the process of utilizing the Catalog Manager to change these credentials to be more secure.*

API Key for the Rule Service: The rule execution service is protected using an API key. For SaaS deployments, your API key can be found in the InRule Portal by navigating to "Provisioned Resources" and finding the "Execution" resource. For Self-hosted deployments, you can generate your own key, but it is important to save this key as it will be used when configuring Salesforce later.

Administrative Password to use for SQL Server: You should decide what username and password you want to use for administrative privileges on the SQL Server. You will use this password when following the referenced catalog setup guide.

*** This walkthrough will utilize the above administrative login and password for the Catalog Service to connect to the SQL Server Database. In a more secure environment, a separate SQL User should be created that only has access to the single database needed by the catalog. It is up to the reader of this document to go this more secure route.*

Administrative Login and Password for Microsoft Azure: You must have a username and password that will be used to perform administrative tasks within Microsoft Azure.

InRule Azure License File

You will need a special .xml file used for licensing InRule in an Azure cloud environment. This may have been provided with your InRule Welcome package. You can contact support@InRule.com if you have questions about where to get your license file.

Deciding resource names

The following worksheet can be used to decide what to name Azure resources as you go through this Guide.

Many of these resources must have names that are unique in the world; they are hosted on Microsoft Azure and are given domain names that match. We recommend creating a “Base” name that does not exceed 14 characters. We recommend encoding an organization name, an application name, and an environment name into this ‘Base’ name. For Example:

`{ApplicationAbbreviation}{OrganizationAbreviation}{EnvironmentAbreviation}`

`MyAppInRuleDev`
`12345678901234`

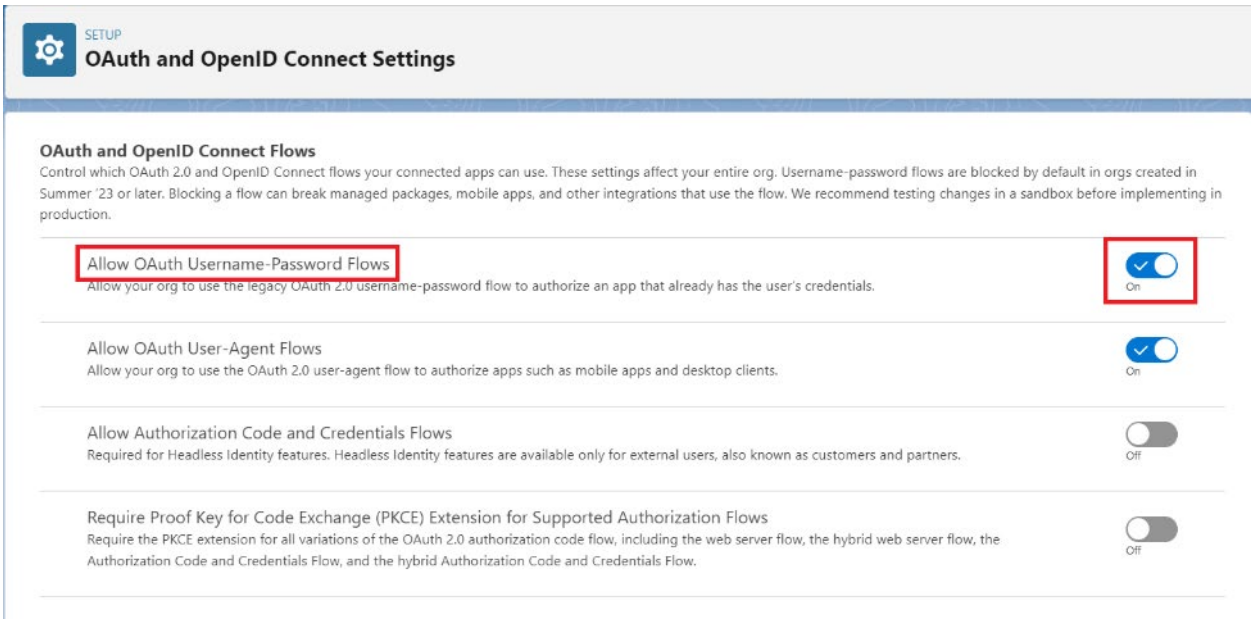
You can choose to follow this convention or invent your own.

Resource and Description	Example Name
Base Name	MyAppInRuleDev
Azure Resource Group Name	MyAppInRuleDevResourceGroup
Azure SQL Server Name *must be lower case*	myappinruledevsqlserver
Catalog Database Name	MyAppInRuleDevCatalogDb
Catalog App Service Name	MyAppInRuleDevCatalogService
Rule Execution App Service Plan Name	MyAppInRuleDevRuleExecutionAppServicePlan
Rule Execution App Service Name	MyAppInRuleDevRuleExecutionAppService

Enabling OAuth Username-Password Flow

Starting with the Summer ’23 Salesforce update, new Salesforce instances now have the Allow OAuth Username-Password Flows setting disabled. In order for InRule to authenticate back to Salesforce, you will need to verify that this setting is enabled.

To begin, navigate to the OAuth and OpenID Connect Settings page by using the quick find search box. Once there, find the Allow OAuth Username-Password Flows toggle and enable it.



3.3 Deploying and Configuring Components

There are two primary paths for deploying the required InRule components:

1. **App Stores**: Azure Marketplace and Salesforce AppExchange - *(Recommended)* Provides a straightforward, UI-driven process that simplifies deployment and eliminates the need to individually deploy both the Catalog App Service and the Rule Execution App Service. Both the Azure Marketplace (for InRule App Services) and the Salesforce AppExchange (for the InRule for Salesforce App) must be utilized to deploy their respective InRule Apps (see matrix below).

2. **Manual Deployment using ARM Templates and Scripts**: Manually deploy Azure Resource Management (ARM) Templates through either the Azure Portal, PowerShell, or Azure CLI; then deploy the InRule for Salesforce App via a versioned package link to your Salesforce environment.

InRule components by deployment path:

InRule Component	App Stores	Manual Deployment
Catalog App Service	Azure Marketplace	ARM Template
Rule Execution App Service for Salesforce		ARM Template
InRule for Salesforce App	Salesforce AppExchange	Versioned Package Link

While deploying from Azure Marketplace and AppExchange is generally recommended for initial environments; in advanced scenarios the manual deployment resources can be leveraged for multi-environment DevOps automation. Also note, choosing either path does not require you to stay on the same path across all components; as the deployment options can be mixed and matched as situations warrant.

The following sections are primarily focused on the manual deployment path for the Catalog App Service and Rule Execution App Service for Salesforce, while focusing on the AppExchange deployment path for the InRule for Salesforce App. However, there are still relevant config and testing steps which apply to both paths. For Marketplace deployments, you may skip the manual deployment steps and go directly to the relevant sub-sections for license management, configuration, and setup verification.

3.3.1 Catalog App Service

Installing the Catalog App Service

The first major objective for a Salesforce implementation is the deployment of a Catalog service to Microsoft Azure.

During this process, you will be creating:

- An Azure SQL Server Database
- An Azure App Service to host the InRule Catalog in the Azure cloud

When you are finished, you should be able to connect to this app service from a locally installed copy of irAuthor, and successfully save a RuleApp to the catalog.

The full process of installing the Catalog in Microsoft Azure is outlined in the documentation found on the InRule AzureAppServices GitHub, which can be found here:

https://github.com/InRule/AzureAppServices/blob/master/README.md#ircatalog-and-ircatalog-manager

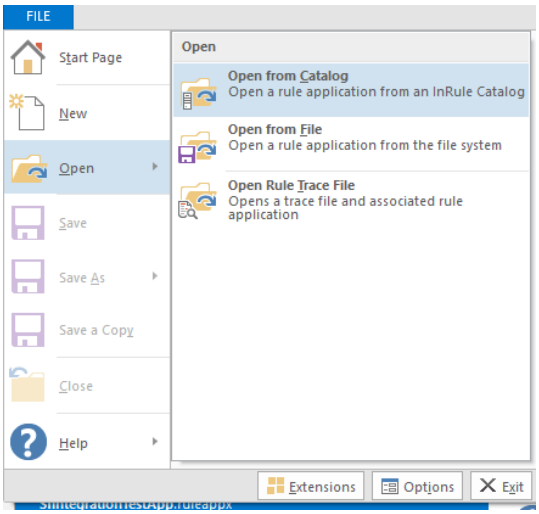
Please ensure you are installing the Catalog service, not the irServer Rule Execution Service, which is found on the same page and is not compatible with Salesforce.

Testing the Catalog App Service

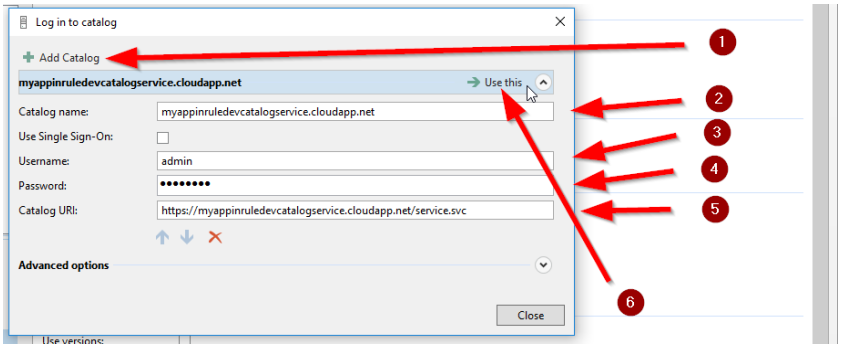
At the conclusion of the installation process outlined above you should have a Catalog URI, Username, and Password to use to connect to the catalog service. We'll now verify that we can appropriately connect to it.

To begin, open up irAuthor. By default, the Catalog App Service is deployed out with a test rule app already available on it. We can attempt to access this rule app to verify that everything is working correctly.

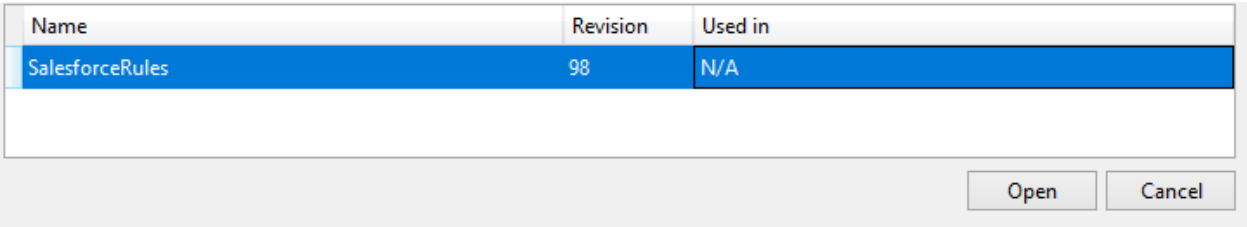
1: From the irAuthor launch screen, navigate to File -> Open -> Open from Catalog



2: Choose `Add Catalog`, enter connection information for the Catalog Server that you deployed, and then select `Use This`.



3: Select the SalesforceRules rule app and hit Open

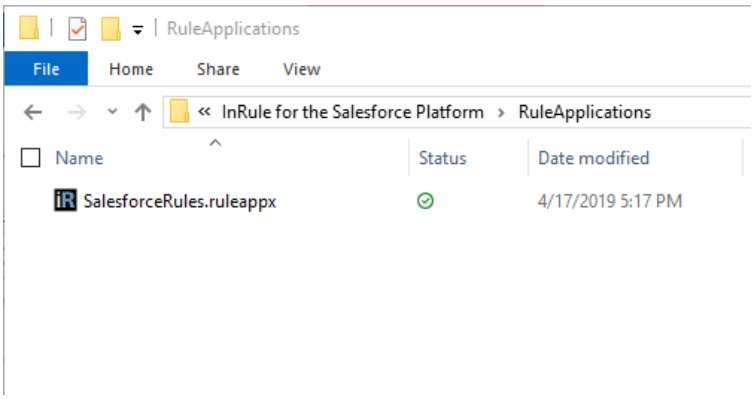


If the SalesforceRules rule app is there and opens without issue, then your catalog app service is good to go.

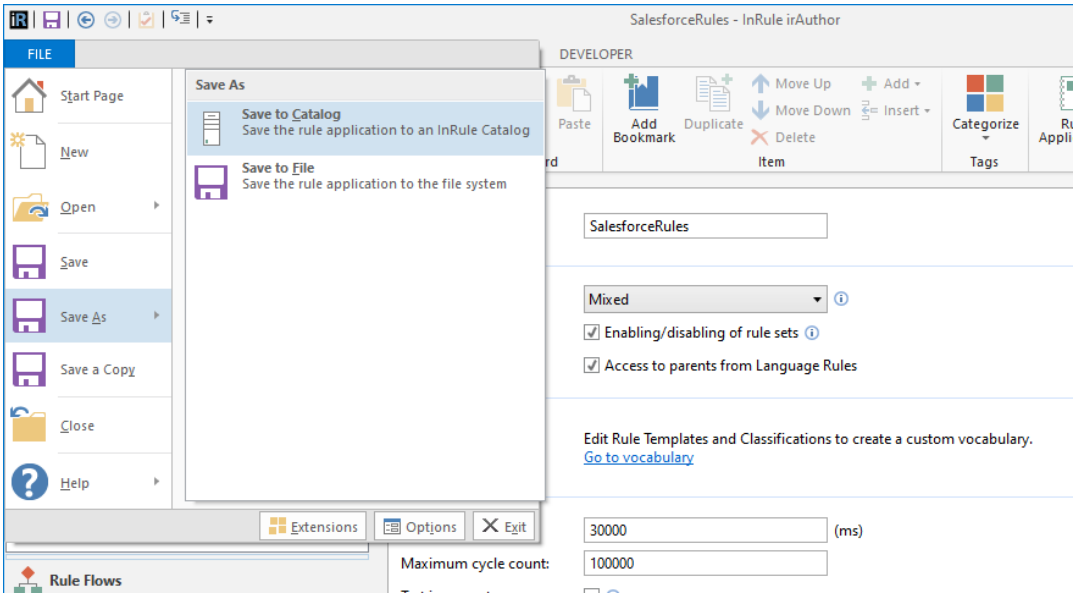
If the rule app does not appear as an option to open after connecting to the catalog, you can try the below steps as an alternative method of testing:

4: Download the SalesforceRules sample rule app.

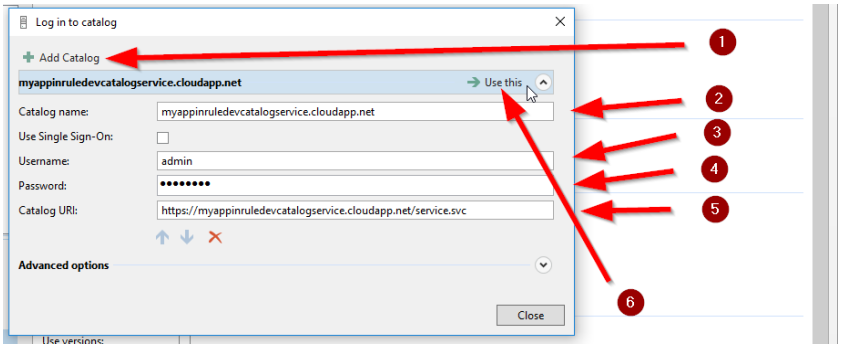
5: Navigate to this file in your rule authoring environment and double click on it, this will open the file with irAuthor.



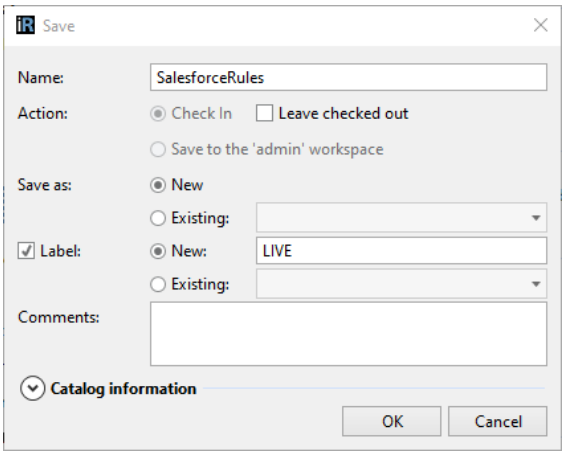
3: Save the Rule Application by choosing `File` → `Save As` →`Save to Catalog`



6: Choose `Add Catalog`, enter connection information for the Catalog Server that you deployed, and then select `Use This`.



7: Save with the name `SalesforceRules`



Save the Rule Application to the Catalog using the name of *SalesforceRules*. You must also be sure to provide the label, which needs to match the label configured on the Catalog Service.

At this point, if you can click OK without an error, you have successfully saved the SalesforceRules Rule App to the new Catalog that you have created.

If you have any trouble getting to this point, it is advised that you resolve any issues with the Catalog before attempting to continue.

Upload License File

Next, you'll need to upload a license file to the web app service in order for the Rule Execution App Service to properly function. The simplest way to upload the license file is via the Azure App Service Editor. Alternatively, you can deploy the license file via FTP. Walkthroughs of both these approaches can be found in Appendix K: License Management.

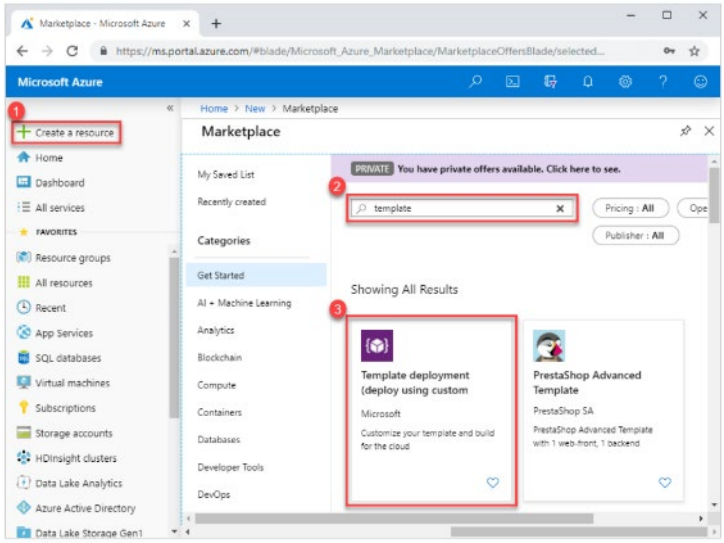
3.3.2 Rule Execution App Service for Salesforce

Next, we will deploy the InRule Rule Execution service as an Azure App Service, along with all its Azure resource dependencies. To make this process easier, we will be using an Azure Resource Manager (ARM) template, which allows us to deploy and configure all the Azure resources the Rule Execution Service relies on.

There are a number of methods for deploying an ARM template; this documentation will detail two: via **Azure CLI** and via **PowerShell**.

Alternatively, while this document does not provide a walkthrough of it, the ARM template provided is configured to work with **Azure Portal** deployment. For an overview of how to leverage ARM deployment through the Azure Portal, reference Microsoft's documentation: Deploy resources with ARM templates and Azure portal and navigate to the section titled "Deploy resources from custom template".

You can navigate directly to the Azure Portal page for deploying an ARM template at this link: Custom deployment - Microsoft Azure.

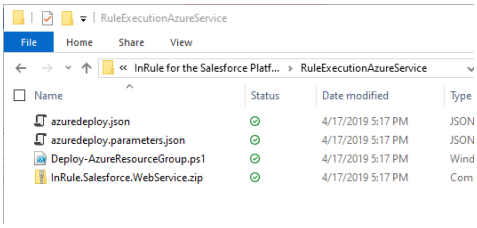


1: Locate azuredeploy.parameters.json

Before deploying the ARM template, we need to define certain parameters.

The required *azuredeploy.json* and *azuredeploy.parameters.json* files can be downloaded here - AzureAppServices/Salesforce at master · InRule/AzureAppServices · GitHub.

Alternatively, they can be located within the *InRule for Salesforce.zip* file downloaded in Section 3.2: Optional Resource Files.



2: Update parameters

Before completing this section, make sure you have the following credentials:

- Salesforce Credentials
- Salesforce API Security Token
- Salesforce Connected App
- InRule Catalog credentials

Open the file with your text editor of choice and edit the parameters listed below

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentParameters.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "appServiceName": {
      "value": ""
    },
    "sfLoginUrl": {
      "value": "https://login.salesforce.com/services/oauth2/token"
    },
    "sfUsername": {
      "value": ""
    },
    "sfPassword": {
      "value": ""
    },
    "sfSecurityToken": {
      "value": ""
    },
    "sfConsumerKey": {
      "value": ""
    },
    "sfConsumerSecret": {
      "value": ""
    },
    "catalogUri": {
      "value": ""
    },
    "catalogUser": {
      "value": "admin"
    },
    "catalogPassword": {
      "value": "password"
    },
    "ruleAppLabel": {
      "value": ""
    },
    "executionServiceApiKey": {
      "value": ""
    },
    "appServicePlanName": {
      "value": ""
    },
    "createOrUpdateAppServicePlan": {
      "value": true
    },
    "inRuleVersion": {
      "value": "5.8.0"
    },
    "appInsightsResourceName": {
      "value": ""
    },
    "appInsightsInstrumentationKey": {
      "value": ""
    },
    "appInsightsConnectionString": {
      "value": ""
    }
  }
}
```

1. appServiceName	Provide a name for the Azure App Service that the Rule Execution Service will run on. Note, by default, an App Service Plan will be created by the ARM template. The App Service Plan will follow the naming convention of the App Service name you provide here, with "Plan" appended to the end (ex. appServiceNamePlan). If you wish to deploy your app service to an already existing App Service Plan rather than create a new one, or for more information on the necessary configurations for the App Service Plan, reference Appendix F: Azure App Service Plan and Application Insights Configuration
2. catalogUri	The URI for the Catalog Service that will be used Example: https://myappinruledevcatalogservice.cloudapp.net/service.svc
3. catalogUser	Username for Catalog Service, default value is 'admin'.
4. catalogPassword	Password for Catalog Service, default is 'password', please change this using the catalog manager!
5. ruleAppLabel	Optionally supply a default label used by the execution service to identify the ruleapp version to run.
6. sfLoginUrl	The default for this is set to https://login.salesforce.com/services/oauth2/token, but you can change this to the relevant URL for your instance if you are deploying to something like a sandbox instance
7. sfUsername	Salesforce username for the service account used to connect from the execution service to the Salesforce API
8. sfPassword	Password for the Salesforce service account
9. sfSecurityToken	API token for the Salesforce service account. Security tokens are **required** unless the service account will be connecting from an IP address that falls in a trusted IP range configured in Salesforce.
10. sfConsumerKey	OAuth consumer key for the connected app
11. sfConsumerSecret	OAuth consumer secret for the connected app
12. executionServiceApiKey	Api key used to secure the rule service. This same api key will need to be provided to Salesforce during configuration
13. inRuleVersion (To deploy most modern version, leave as default value)	This parameter allows the user to configure what version of the InRule Rule Execution Service they wish to deploy. By default, this parameter will be set to the most modern version.
14. appServicePlanName (If you wish to override the default value)	Provide a name for the Azure App Service Plan. If you leave this value blank it will be derived as the App Service name you provide above, with "Plan" appended to the end (ex. appServiceNamePlan) Note, by default, an App Service Plan will be created by the ARM template. If you wish to deploy your app service to an already existing App Service Plan rather than create a new one, or for more information on the necessary configurations for the App Service Plan, reference Appendix F: Azure App Service Plan and Application Insights Configuration
15. createOrUpdateAppServicePlan	By default, this value is set to true, and an App Service Plan will be created by the ARM template. If you wish to deploy your app service to an already existing App Service Plan rather than create a new one, set this value to false and reference Appendix F: Azure App Service Plan and Application Insights Configuration.
16. appInsightsResourceName **(If you wish to create a new AI resource)**	If you want to use Application Insights as a log sink in addition to the app service logging already enabled, but do not already have an Insights resource that you want to use, specify a name for a new resource here. Specifying a value for this parameter will create a new Application insights resource with the given name and populate the instrumentation key app setting on the app service with the key from this new resource. **If you provide a value for this parameter, do not provide a value for appInsightsInstrumentationKey.**
17. appInsightsInstrumentationKey **(If you wish to use an existing AI resource)**	Provide an Instrumentation Key if you have an existing App Insights resource you'd like to use for logging and telemetry. If you are configuring this in a nonstandard azure environment (such as Azure Government), please additionally provide an App Insights Connection String. Otherwise, leave this value blank and provide a value for the 'appInsightsResourceName' parameter, which will create the resource for you. For more information on the logging view Rule Execution Service Event Log.

18. appInsightsConnectionString **(If you wish to use an existing AI resource in a non-standard Azure environment)**	Only override the default value here if you have an existing App Insights resource you'd like to use, AND you need to use a non-standard connection string. If you need to supply your own connection string, be sure to set that value here, as well as providing your instrumentation key in the appInsightsInstrumentationKey parameter. If you want this template to manage the App Insights resource for you, or only need to provide an instrumentation key, leave this value as the default and provide values for appInsightsResourceName or appInsightsInstrumentationKey instead. Additionally, for any consideration about using app insights or setting it up in a non-standard Azure environment view Appendix G: Azure Application Insights Configuration

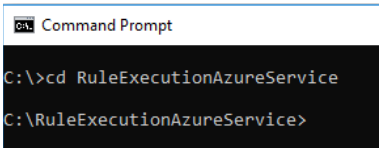
Once you have finished configuring your parameters, save the completed parameters file and keep a spare copy on hand for future upgrades or automation.

3: Option 1: Deploy ARM Template with Azure CLI

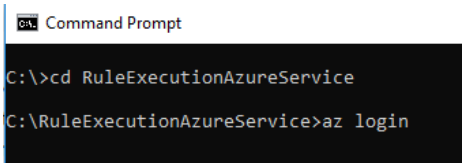
Now that the ARM template is configured, we'll deploy it to get the resources up and running. The following will detail how to use the Azure CLI to deploy the ARM template (Note, this section assumes Azure CLI has already been installed):

3.1 Run Command Prompt or PowerShell

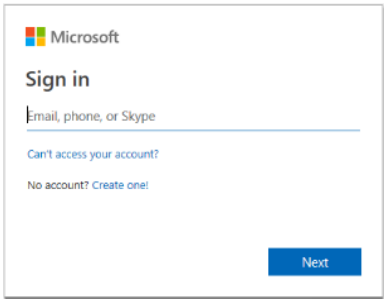
3.2 Navigate to the RuleExecutionAzureService folder



3.3 Enter "az login" to login into Azure



3.4 Enter your Azure admin credentials to login when prompted in the new browser window opened



3.5 Select the appropriate subscription

If your Azure account has access to multiple subscriptions, you will need to set your active subscription to where you create your Azure resources:

```
C:\RuleExecutionAzureService>az account set --subscription SUBSCRIPTION_NAME
```

3.6 Create Resource group

If you have not created a resource group yet, you will need to create one. You will need to define a name and a geographic location for where to host the resource. This example uses Central US:

```
C:\RuleExecutionAzureService>az group create --name ResourceGroupName --location centralus
```

3.7 Execute the following command to deploy the ARM template

Replace “ResourceGroupName” with the name of the Azure Resource Group you want to deploy to

```
C:\>az deployment group create -g ResourceGroupName --template-file
.\azuredeploy.json --parameters .\azuredeploy.parmeters.json
```

Observe that the template deploys with no errors

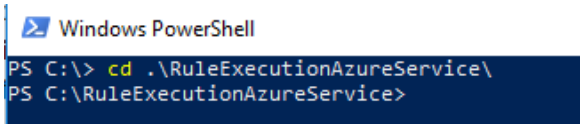
4: Option 2: Deploy ARM Template with PowerShell

(If you have already deployed the ARM template via Azure CLI in the section above, this section is not necessary)

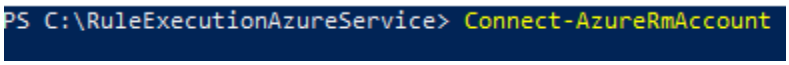
Now that the ARM template is configured, we’ll deploy it to get the resources up and running. The following will detail how to use PowerShell to deploy the ARM template (Note, this section assumes Azure PowerShell has already been installed):

1.1 Run PowerShell

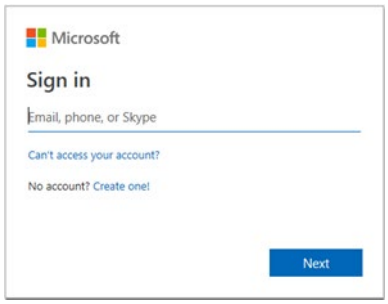
1.2 Navigate to the RuleExecutionAzureService folder



1.3 Enter “Connect-AzureRmAccount” to login into Azure

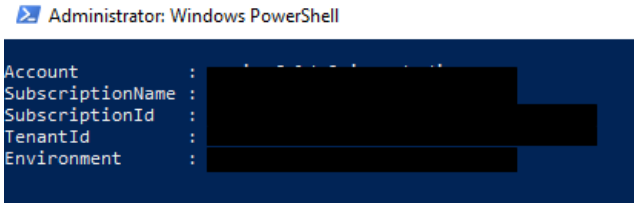


1.4 Enter your Azure admin credentials to login when prompted in the new browser window opened

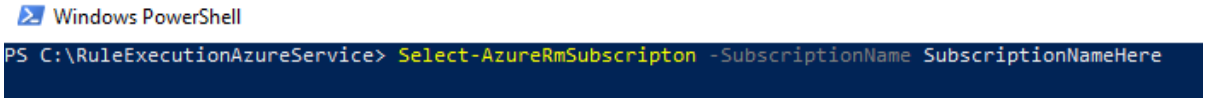


1.5 Select the appropriate subscription

Upon logging in, your default subscription information will be displayed:



If this is not the subscription you want to deploy to, you can use the “Select-AzureRmSubscription” cmdlet to change the targeted subscription. Just replace “SubscriptionNameHere” with the name of the desired subscription:



4.6 Create Resource Group

If you have not created a resource group yet, you will need to create one. You will need to define a name and a geographic location for where to host the resource. This example uses Central US:

```
PS C:\RuleExecutionAzureService> New-AzureRmResourceGroup -Name ResourceGroupName -Location centralus
```

4.7 Execute the following command to deploy the ARM template

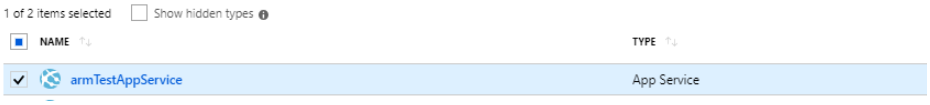
Replace “ResourceGroupName” with the name of the Azure Resource Group you want to deploy to

```
PS C:\RuleExecutionAzureService> New-AzResourceGroupDeployment -ResourceGroupName ResourceGroupName
-TemplateFile .\azuredeploy.json -TemplateParameterFile .\azuredeploy.parameters.json
```

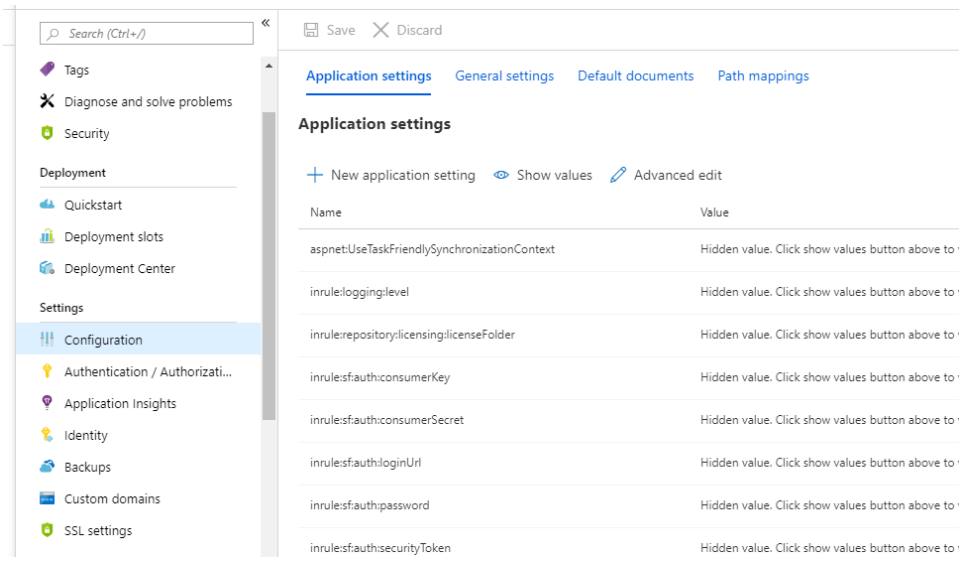
Observe that the template deploys with no errors

5: Verify Setup

Navigate to the Azure portal and locate the deployed App Service



Ensure that all of the app settings are configured correctly for your setup



6: Upload License File

Next, you'll need to upload a license file to the web app service in order for the Rule Execution App Service to properly function. The simplest way to upload the license file is via the Azure App Service Editor. Alternatively, you can deploy the license file via FTP. Walkthroughs of both these approaches can be found in Appendix K: License Management.

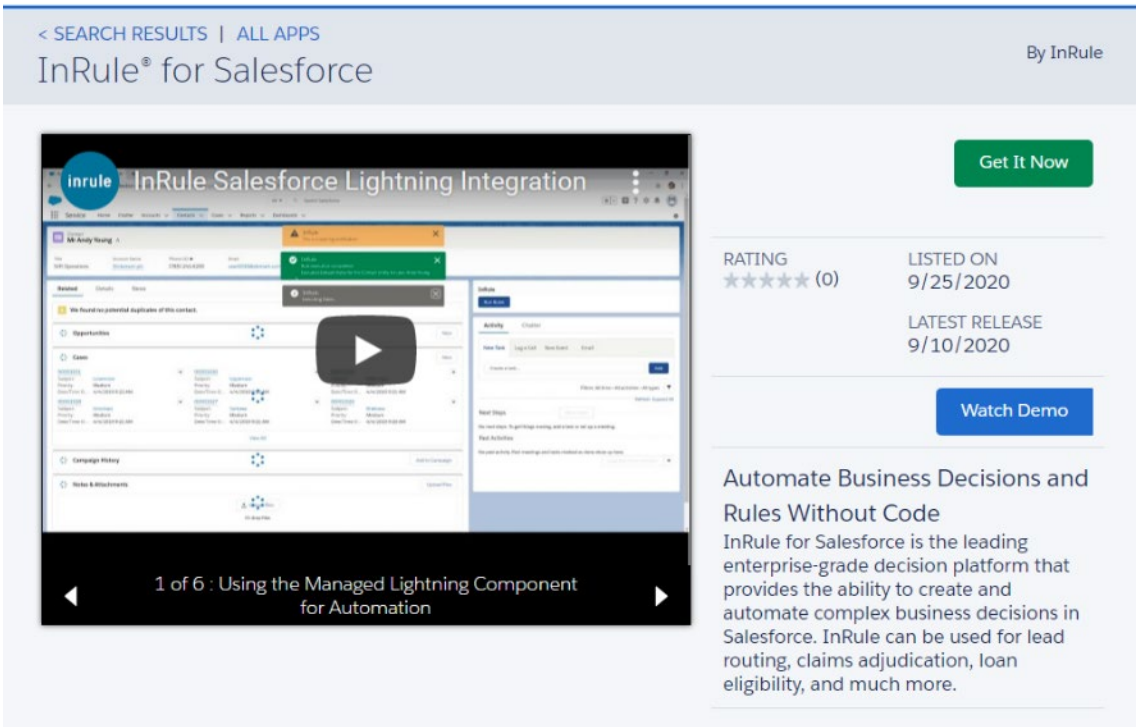
3.3.3 InRule for Salesforce App

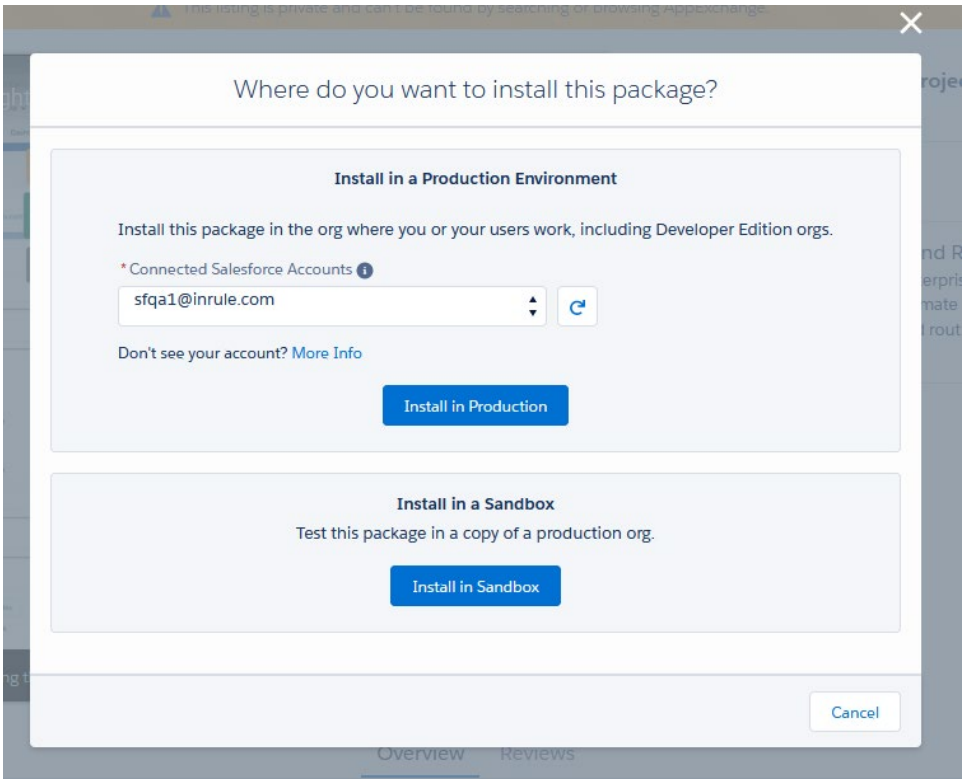
At this point, all the Azure requirements are met. The execution service should be listening for incoming communication from Salesforce. We must now setup the InRule for Salesforce App, which is done with a managed Salesforce package. This can be done via either AppExchange or direct install link:

- Deploy latest version – navigate to the AppExchange, select 'Get It Now'
- Deploy specific version – deploy the specific version associated to this Deployment Guide (refer to cover page for version)

1: Install from AppExchange:

InRule for Salesforce can be installed from the Salesforce AppExchange marketplace. You can either search for 'InRule for Salesforce' or go directly to the listing here: https://login.salesforce.com/packaging/installPackage.apexp?p0=04tHu000003EZBV When you get to the listing page, you'll need to select 'Get It Now' and choose the org you want to install the package in.

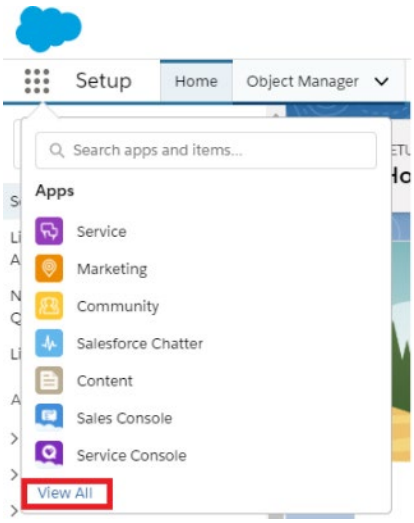




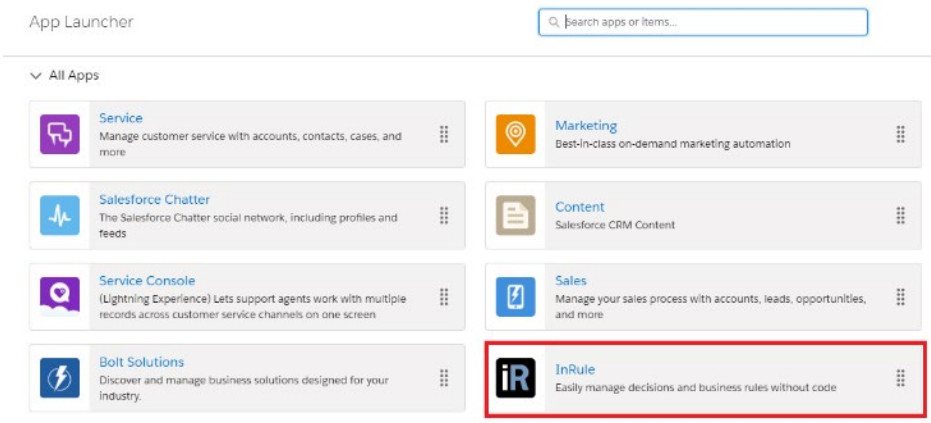
Once you've selected an org and logged in, you'll see a screen prompting you to install the package. Select 'Install for all users' and hit ok. Installing this package installs all the components required for integrating with InRule, including a configuration app, Apex classes, and custom settings and objects. Once the installation is complete, you will need to configure the connection to the execution service.

2: Navigate to the InRule for Salesforce App

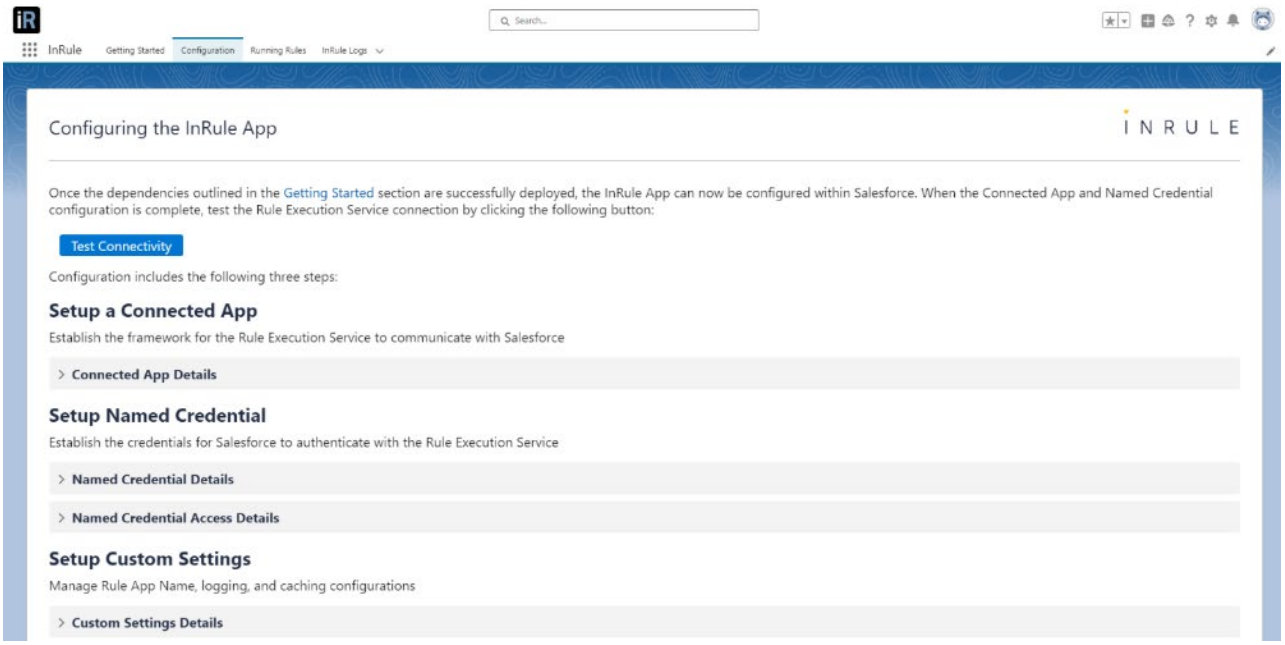
In Lightning view, select the Salesforce App Launcher button in the top right of the home screen and select "View All"



Select the InRule App



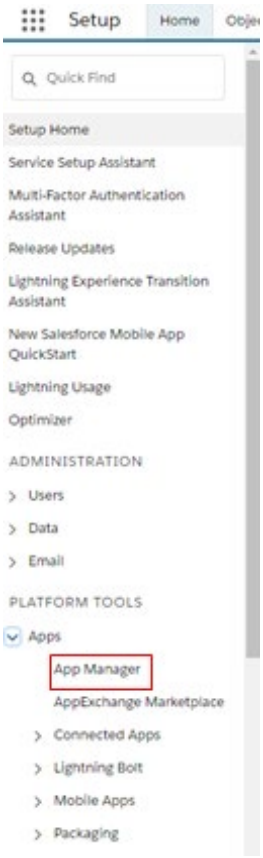
This app provides useful information on how to get started using InRule for Salesforce, as well as quick links to important configuration pages and logging.



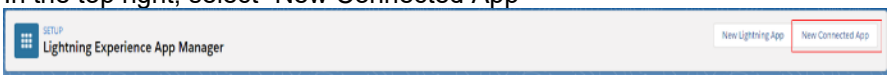
3: Configure the InRule App

Connected App

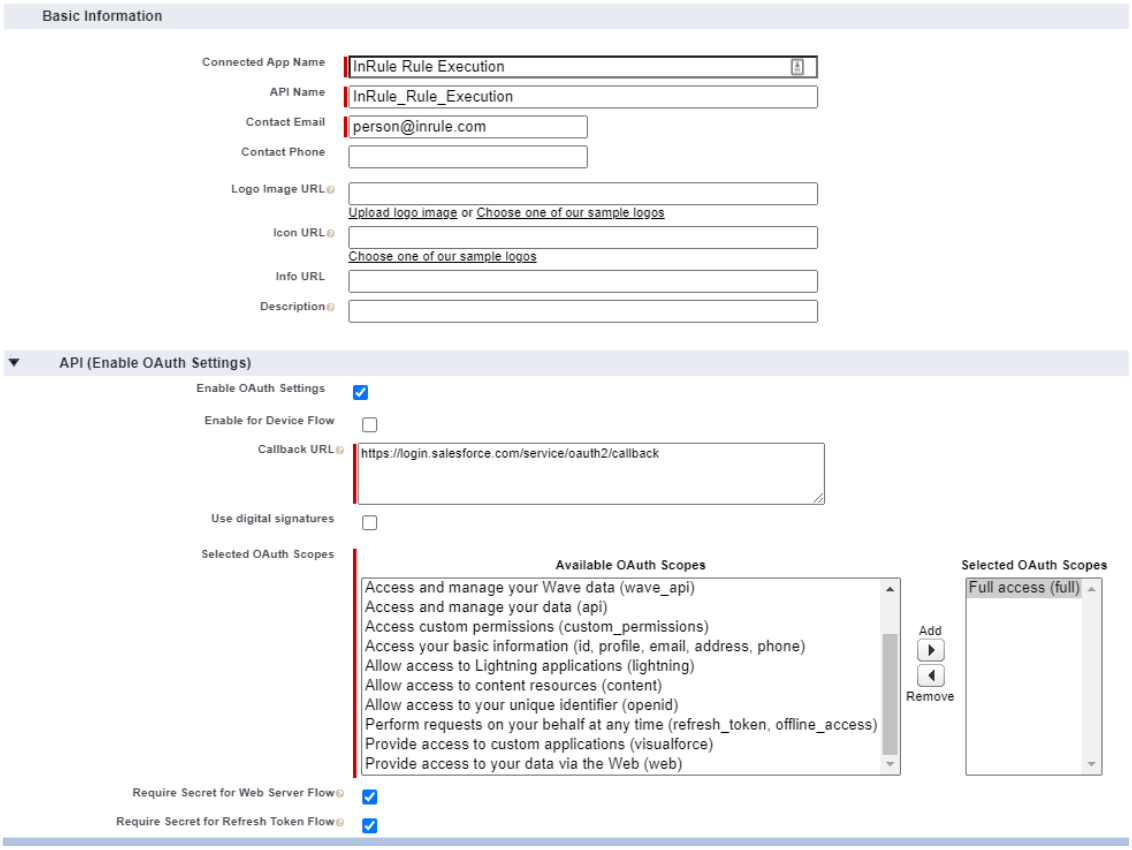
First, create a new Connected App in Salesforce that is configured for OAuth authentication. Navigate to Setup → App Manager -> New Connected App



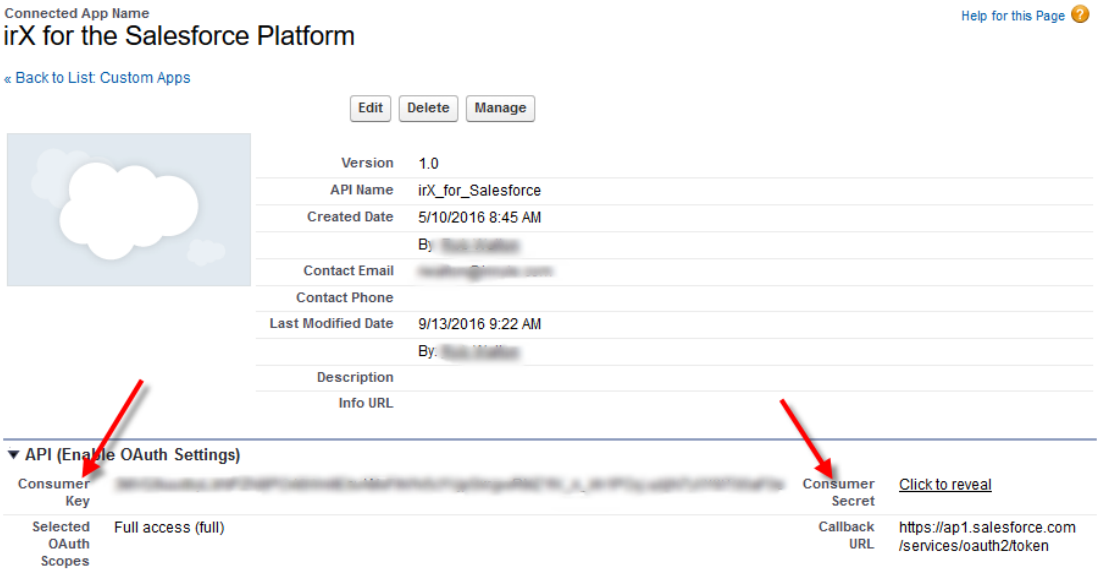
In the top right, select “New Connected App”



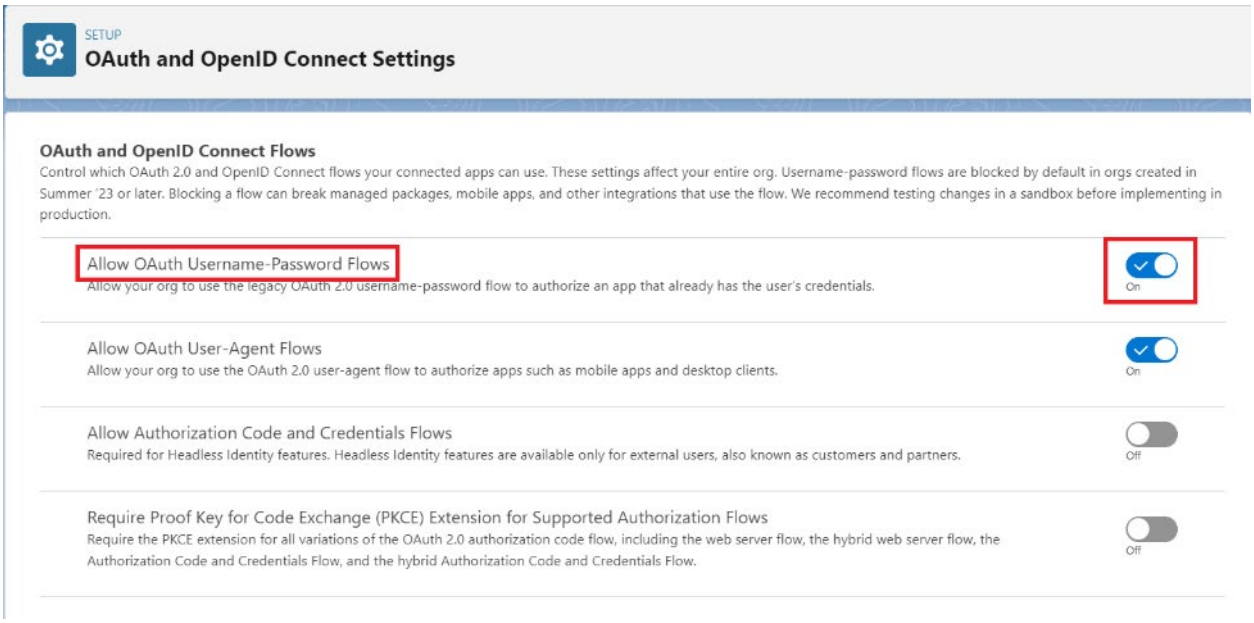
Fill out the Basic Information and configure OAuth authentication. The Rule Execution Service uses the Username-Password authentication flow, so there is no callback URL, but because Salesforce requires this field to be populated, enter any properly formatted URL; it will not be used. Also, be sure to add “Full access” to the Selected OAuth Scopes. Any other values can be left as defaults.



After saving, select the newly created Connected App and inspect the OAuth settings. Record the Consumer Key and the Consumer Secret.



If your Salesforce instance was created **after** the Summer ’23 update, verify that you have enabled the ‘Allow OAuth Username-Password Flows’ toggle on the OAuth and OpenId Connect Settings page.



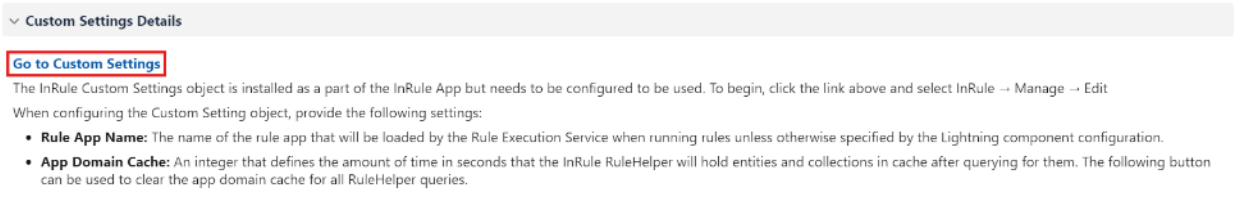
Once you have completed these steps, you should have the token, consumer key and consumer secret, all of which are required for authentication with Salesforce in addition to the username and password.

Named Credential

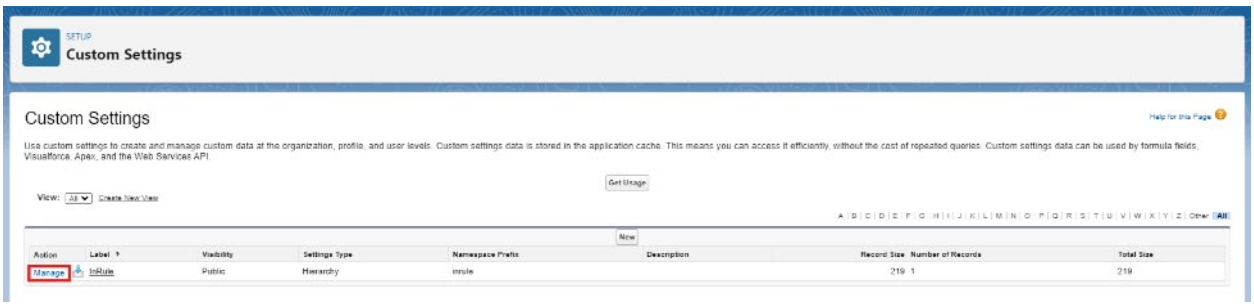
Next, we need to configure the Named Credential in Salesforce. This will be used by the Apex code to make secure HTTP requests to the Rule Execution Service. See Appendix H: Named Credential Configuration for instructions on how to set up your Named Credential.

Custom Setting

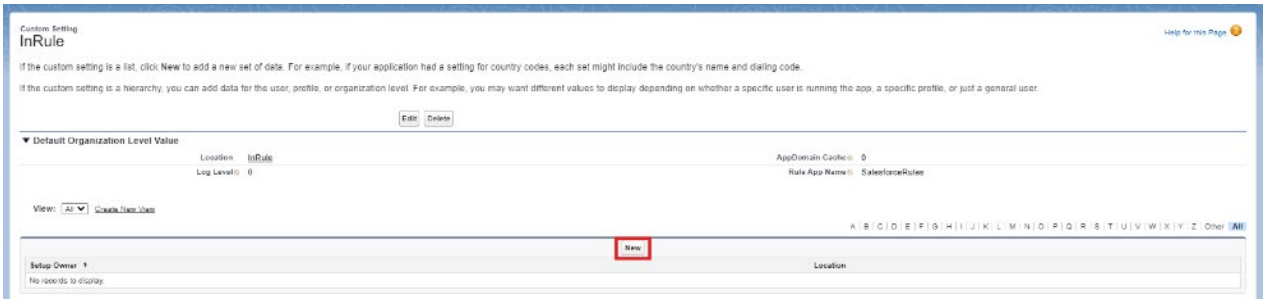
Once you have set up the Named Credential, navigate back to the InRule App and return to the configuration tab. Under the “Custom Settings” header, click the “Custom Settings” shortcut link to configure your default rule app name and logging level.

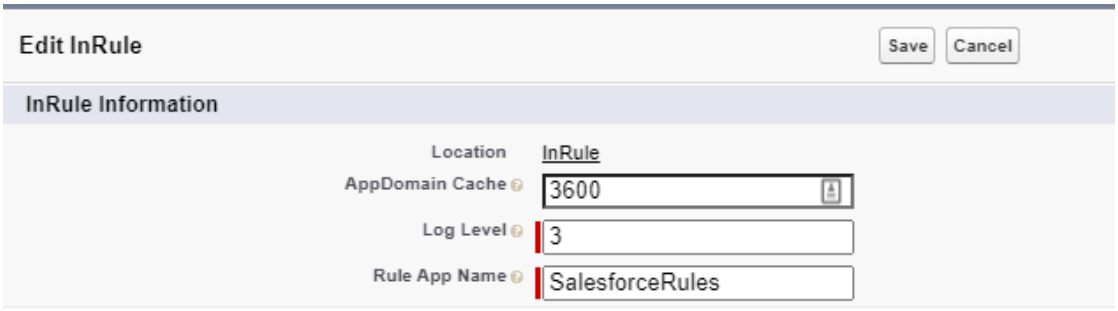


Click “Manage” next to the **InRule** label



Click “New”. Be sure to select the button above “Default Organization Level Value”, not the one below it. The “New” button in the grid below will create a setting that is only applicable to a subset of users, but the org level value will apply to all users. Scoped values can be used if desired, but be sure to at least define an org level value.



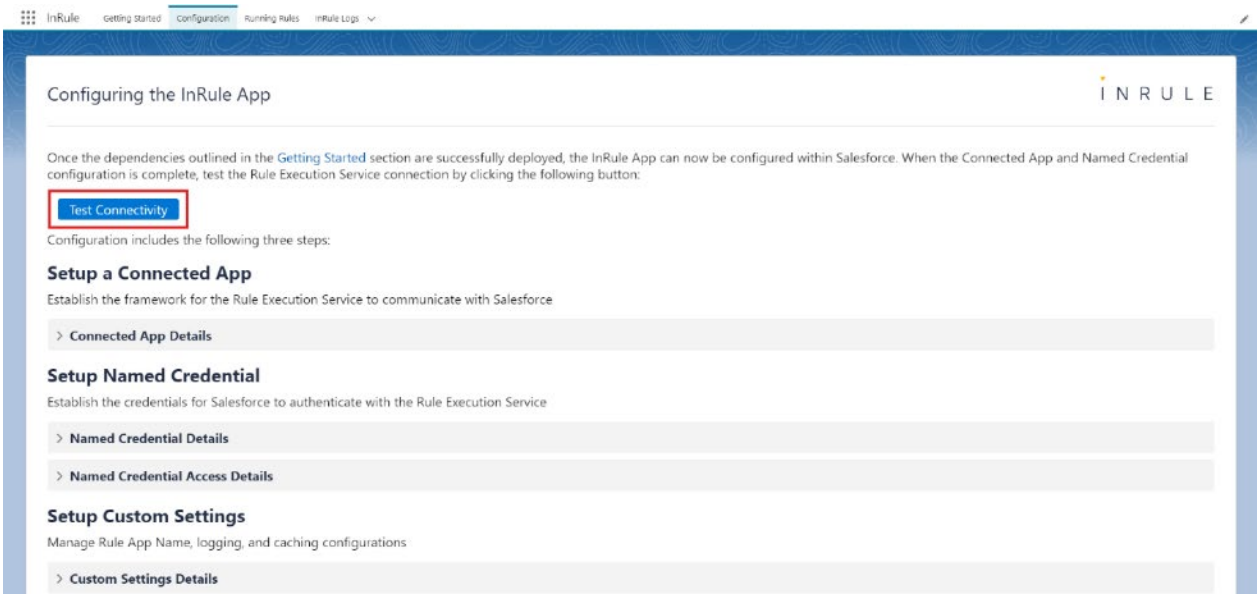


Configuration Form Fields	
Rule App Name	The name of the rule app that will be loaded by the Rule Execution Service when running rules
App Domain Cache	An integer that defines the amount of time in seconds that the InRule RuleHelper will hold entities and collections in cache after querying for them. You can use the “Clear Cache” button on the InRule Configuration page to clear the cache at any time. Additionally, re-setting the configuration value to 0 will clear the cache.
Logging Level	An integer that denotes the amount of information to log after rule execution completes. This is a default value that will be used by all invocations of the DecisionClient, but can be overridden in the calling script. Results are written to the InRule_Log__c object. Values can be 0, 1, 2, or 3. *These log levels are different than the log levels in the execution service which must be configured independently*: • 0 – disable all logging • 1 – Logs errors and a minimal amount of information on successful requests • 2 – Logs errors, rule engine notifications, and rule engine validations • 3 – Logs the same information as 2, but also includes JSON from the HTTP request and response payloads, and additional trace messages

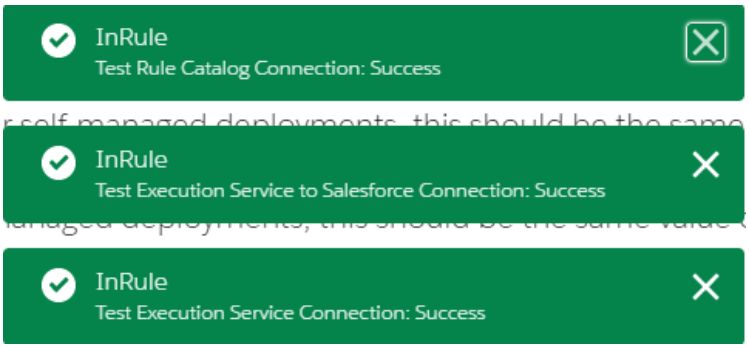
Once configured, press Save.

4: Verify a successful deployment

Now that all of our resources are properly deployed and configured, we can test to ensure they're all working as they should be. Navigate back to the InRule App and then the Configuration tab once again. Find the "Test Connectivity" button and press it. This button will make a mock request out to your Rule Execution Service to verify that the InRule App and all related Azure resources are deployed and configured properly.



If successful, you should get 3 green notifications like below:



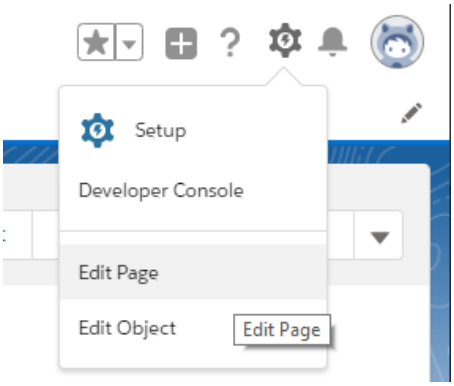
5: Add the Run Rules button

As a part of the deployment of the InRule for Salesforce App, a new "Run Rules" Lightning component has been installed. To add and configure this component to an entity or entities, you can follow the steps below. This example uses the Account entity and the rule app you uploaded in the Testing the Catalog App step, but this process should work for any entity, default or custom.

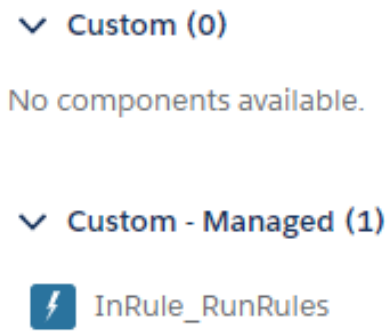
Alternatively, a sample Run Rules JavaScript button has now been installed for the account and contact entities in the Classic (non-Lightning) UI. For guidance on how to add a Run Rules button in the Classic UI, refer to Appendix E: Methods for Executing Rules from Salesforce.

To begin, navigate to the entity you wish to add the button to. Note that you will have to add the button to each entity individually.

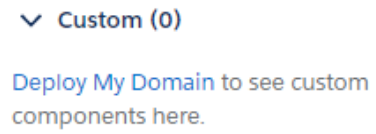
Once on the desired entity page, find the settings icon in the top right corner and select Edit Page.



On the left-hand side of the page in the Lightning Components section scroll to the bottom of the page until you find the InRule_RunRules in the Custom-Managed Section

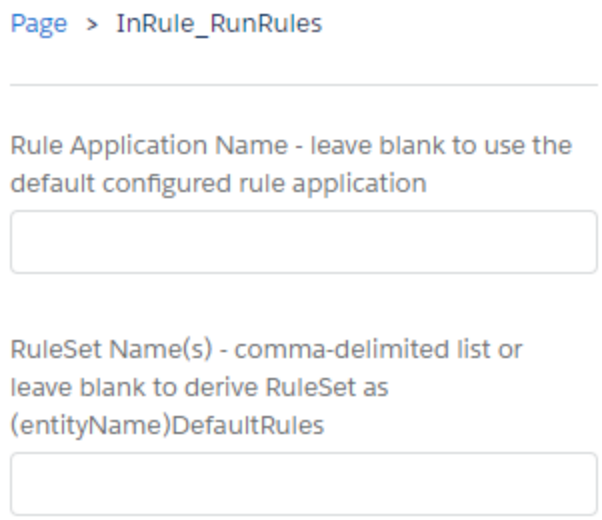


Note: *If you have not setup your Salesforce domain, you will see this message in the Custom-Managed Section.*



Simply follow the link to setup your domain. Once it has been set and you have logged in with the new domain come back to edit page and you should now see the InRule_RunRules Lightning Component.

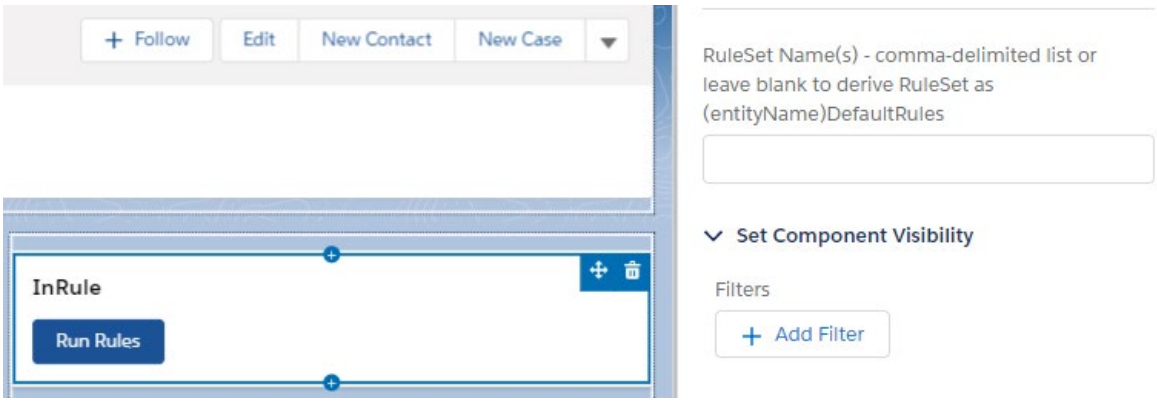
Drag the InRule_RunRules Lightning component to the desired location on the page view. Once you have placed the component, click on it and a menu bar on the right will appear with two available configuration values.



The first configuration field, Rule Application Name, allows you to define a Rule Application to use for this specific lightning component. Placing a value in this field will override the Rule Application you have defined in your Custom Setting as the default. Leave this field blank to use the default configured Rule Application.

The “Rule Set Name(s)” field accepts a comma delimited list of Rule Set names. Adding multiple rule sets here will create multiple Run Rules buttons for each defined Rule Set on the page. Leaving this field blank will leave only the singular “Run Rules” button, and this button will execute the Default Rule Set for the given entity. The Default Rule Set is defined as the entity’s name + “DefaultRules.” For example, if you are adding the Lightning component to the Account entity, the Rule Set name it will default to if this field is left blank is “AccountDefaultRules.”

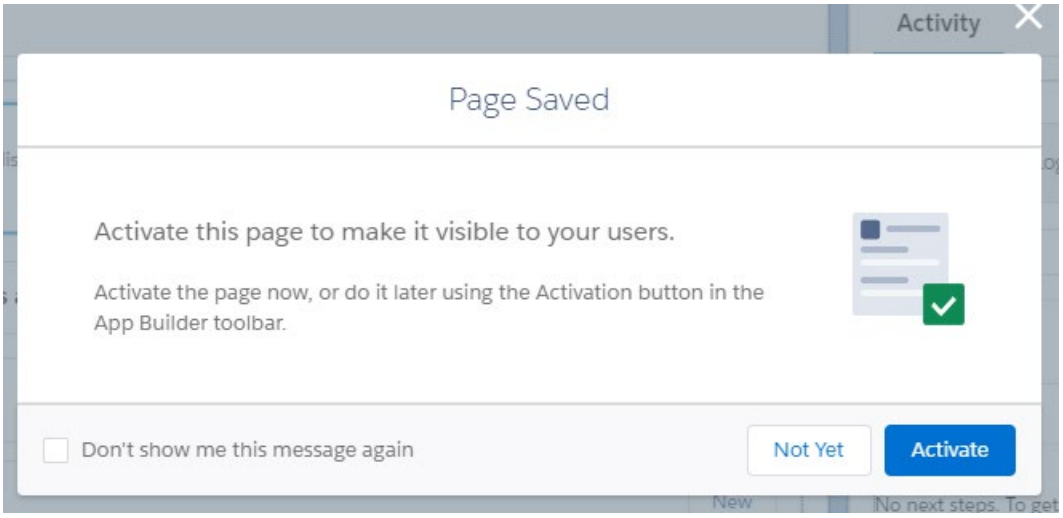
If no Rule Sets are defined, it will create 1 button that uses the default Rule Set for the entity. This button will be generically titled “Run Rules”:



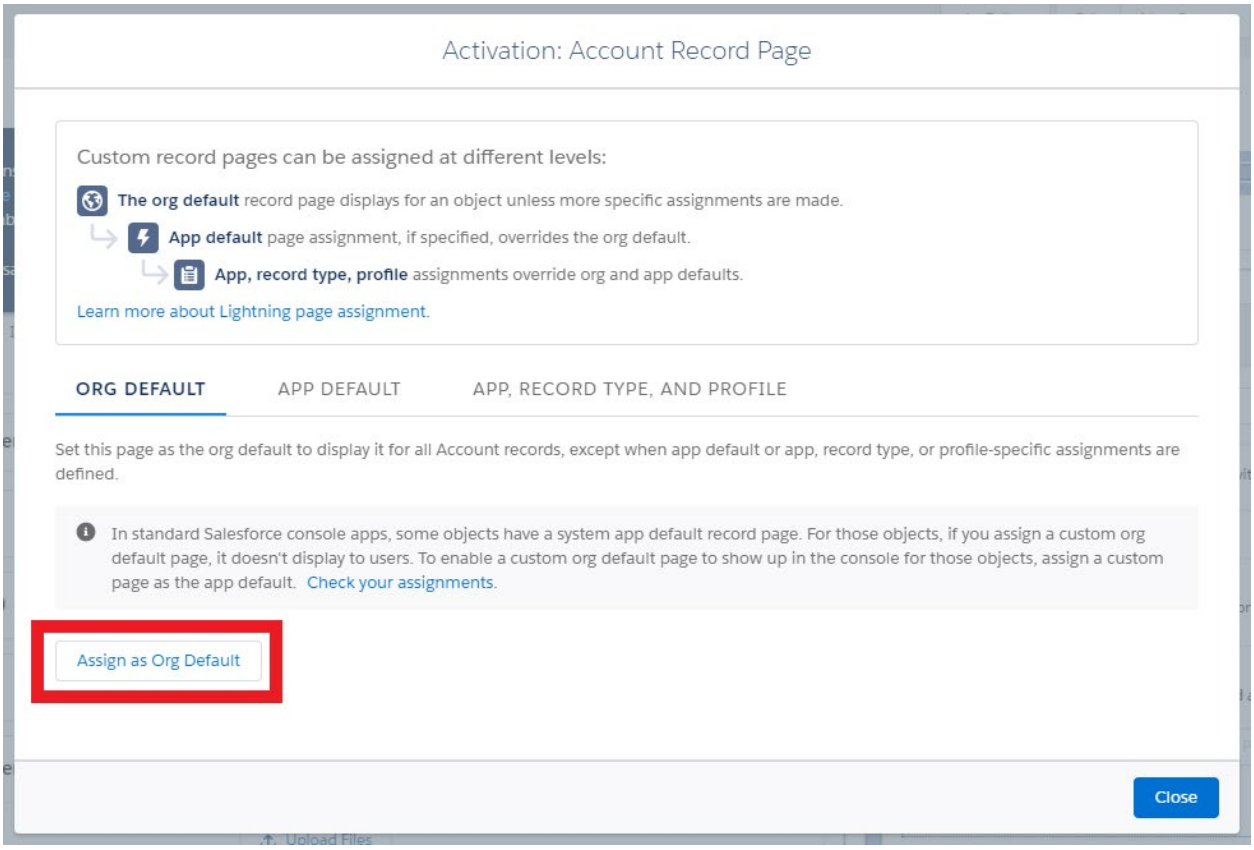
If one Rule Set is defined, will create 1 button that will use that defined Rule Set and will be titled with that Rule Set’s name:

If 2+ Rule Sets are defined, will create a button per Rule Set that will use the respective Rule Sets, each button titled with the rule set they map to:

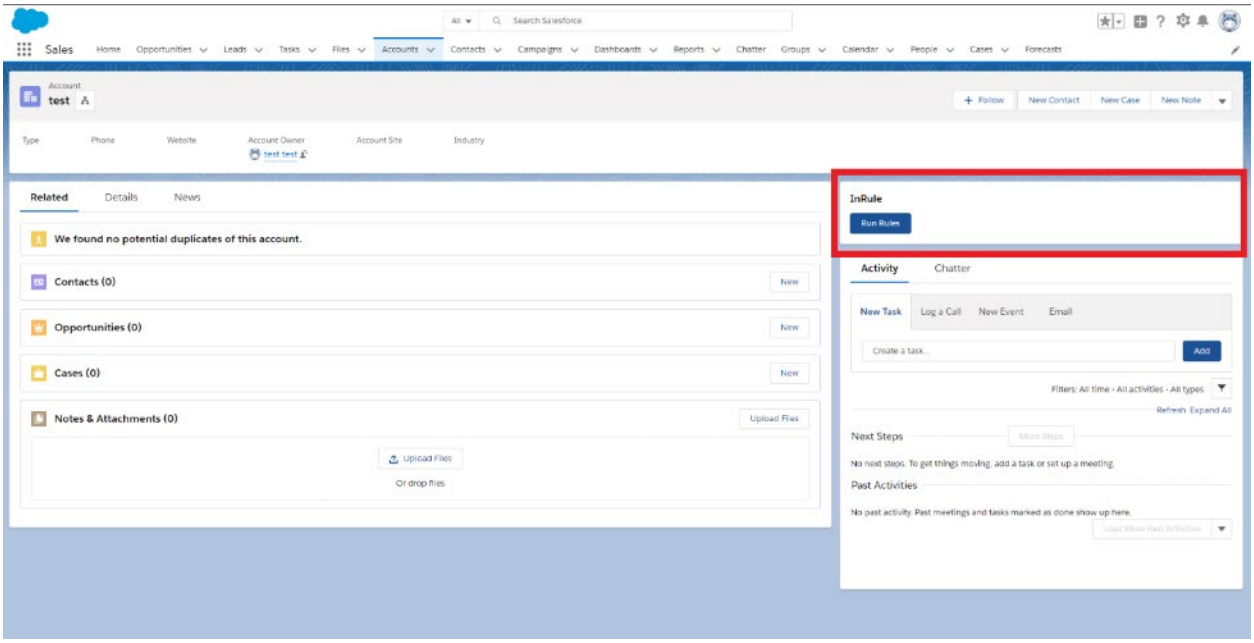
Once you have configured your Rule Sets, click save in the top right corner. It will prompt you to activate this page to make it visible to your users, click activate.



Salesforce will now ask for the scope to activate the record page. Select the desired scope and click Assign and then save on the subsequent prompt.



Navigate back to the main entity page and the component should appear.



Appendix A: Additional Resources

Having trouble? Relax! InRule offers many additional resources to help you get InRule correctly integrated with Salesforce.

InRule's Support Website

InRule's support website can be found at http://support.inrule.com. If you do not already have a login for our support site, the client administrator at your company has the ability to create an account for you. If you are unsure of who your client administrator is, please email support@inrule.com.

InRule's Support Team

The support team at InRule is available to help with any product support needs, troubleshooting suspected product bugs, resolving any licensing issues, and free tele-hugs.

The best way to reach Support is through a detailed email sent to support@inrule.com.

You can also reach our support team by calling +1 (312) 648-1800.

InRule's ROAD Team

ROAD Services agreements can be used to engage with ROAD, InRule's professional services team.

ROAD can provide your organization with specialized consulting and tailored Architecture and Authoring Guidance.

ROAD can assist with less common installation requirements, such as deployment to third party cloud providers or integration with custom software.

ROAD can be contacted by emailing ROADServices@InRule.com

InRule Training Services

InRule offers the following interactive training services:

- Onsite and Remote attendance courses
- Hands-On multi-day courses with interactive labs
- Virtual Express training courses delivered online for rapid knowledge transfer

If you are interested in scheduling training services, please contact us at Training@InRule.com.

Appendix B: Anatomy of a Request for Execution of Rules

The steps below help to give a top-level understanding of how InRule is integrated with Salesforce. Please note these steps are a simplification that does not cover topics like caching, iterations, and multiple environments. It serves to show how the request moves through different Azure resources.

1. The DecisionClient in Salesforce is called by a button or event. This generates a call to the Rule Execution App Service.
2. The Rule Execution App Service makes a request to the Catalog Service, asking for a copy of the requested RuleApp.
3. The Catalog Service Queries its SQL Server based database for a copy of the requested RuleApp.
4. The SQL Server responds with the RuleApp.
5. The catalog service responds to the Rule Execution App Service with the RuleApp.
6. The RuleApp executes inside the Rule Execution App Service.
7. Optionally, the RuleApp has an opportunity to query Salesforce for additional data needed to execute rules.
8. The RuleApp completed execution
9. The Rule Execution App Service relays the response to Salesforce, where the InRule for Salesforce App synchronizes changes

Appendix C: irX General Integration Concepts

Runtime Mapping across Nested Relationships

Much like Salesforce, the InRule rule engine offers strong support for hierarchical and relational data. Within a given Rule Application, data can be considered across parent-child relationships within a single rule request. These relationships can take the form of Collections (1 - * relationships) or 1 – 1 relationships.

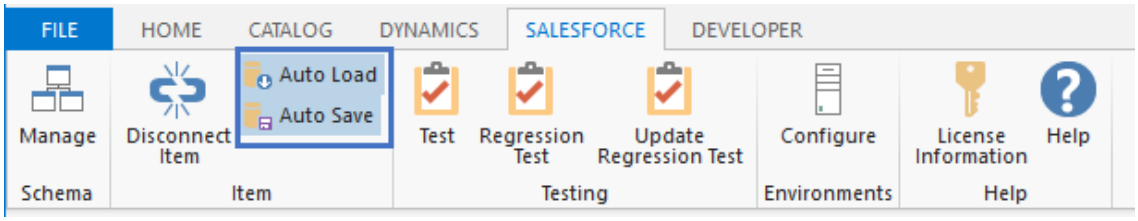
Note: When N:N relationships are imported into InRule, they behave as 1:N Collection relationships within the Rule Application.

In addition to the abilities of both products to handle relational data, both products also offer the ability to declaratively configure “Entities” and “Fields”. Both products also allow for different strongly-typed Entities and Fields to be accessed with loosely-typed SDK interfaces. Because of these inherent similarities and flexible interfaces, it is possible to build a reusable mapping component that can convert any given graph of loosely-typed Salesforce Entities to InRule Entities, and vice versa.

Controlling irVerify Behavior with Load, Save and Inactive Record Settings

When working with an Entity Schema composed of many related Salesforce Entities, it is often useful to have explicit control over which relationships are either automatically loaded or automatically considered in change detection for persistence. If a relationship is skipped during the initial load routines, then it is available to be conditionally populated later using rules.

In the irX rule authoring ribbon, there are two buttons that give the rule author the control to denote if a relationship should be automatically loaded or saved excluded.



Note: Automatic loading and saving is enabled by default for all relationships that are imported from Salesforce. The rule author can opt-out of these automatic behaviors by unselecting “Auto Load” or “Auto Save”. When these buttons are selected, metadata attributes are written into the Rule Application for the given relationship. These metadata attributes are used by the irVerify data loader when recursively loading data or detecting changes for persistence.

Important: If loading or saving is disabled for a given relationship, then it is also disabled for all Entities that are children of that relationship.

Appendix D: Accessing Salesforce Directly from Rule Helper

In the default Rule Execution setup, all relationships between Entity types must be established before these entities can be used by the rule app. This behavior is intuitive, but it is not ideal for all business problems. InRule for Salesforce provides a 'Rule Helper' assembly that can be used directly in a rule app and allows rules to load, compare, and assign data that is not related in Salesforce before rules are executed.

When to use the Query from Rules Approach

The query from rules approach adds value for the following business problems:

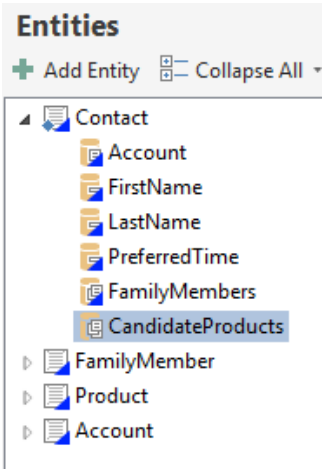
- The rules need to reference "lookup" information that may be in a list or set of Entities that are not specifically related to the current Entity hierarchy
- The purpose of rules is to create new relationships between Entity instances that already exist in Salesforce
- The rules need to compare many combinations of unrelated Salesforce Entities and produce results about best possible matches or scores
- A custom filter is required when loading data for 1:N relationships

Working with Disconnected Fields when Loading and Saving Data

One of the most important integration concepts when loading Salesforce data from rules is the notion of "Disconnected" Fields and Fields that have "Auto Load" and "Auto Save" disabled.

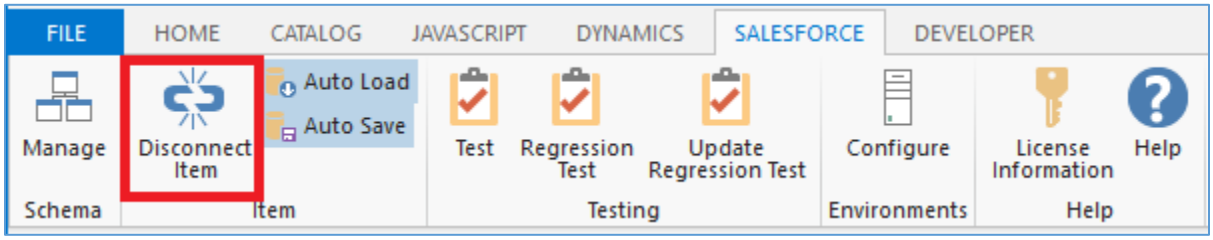
irX allows the rule author to explicitly control the "Auto Load" and "Auto Save" behaviors of Fields that are connected to Salesforce.

The example below shows a Collection named CandidateProducts. Since the Collection is not marked with a blue triangle, it is not considered to be attached to Salesforce.

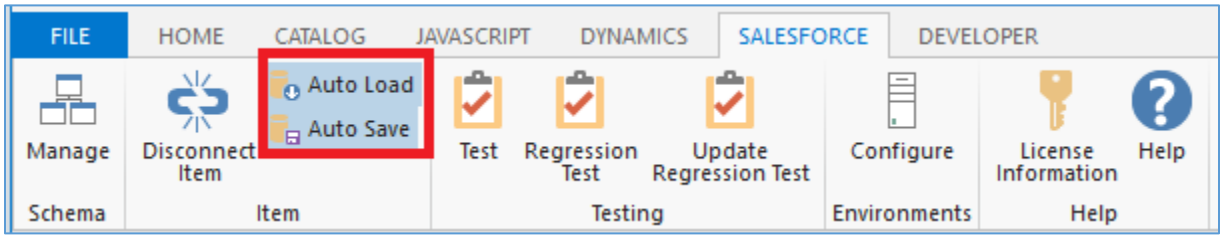


Important: Although Entity Fields and Collections may be "Disconnected" from Salesforce, the types contained by the Collections can be set to types that were imported from Salesforce.

Note: If a Field is added to the schema using irAuthor, then it will be disconnected from Salesforce. If a Field has been imported from Salesforce using irX, then it can be disconnected from Salesforce by clicking the “Disconnect Item” button in the irX ribbon.



Important: Two additional settings appear in the irX ribbon that offer additional control over automatic loading and saving behaviors for Fields that remain connected to Salesforce. In the example below, both buttons are “lit up”, which denotes that the settings are enabled. By default, automatic loading and saving is enabled for all Fields that are connected to Salesforce.

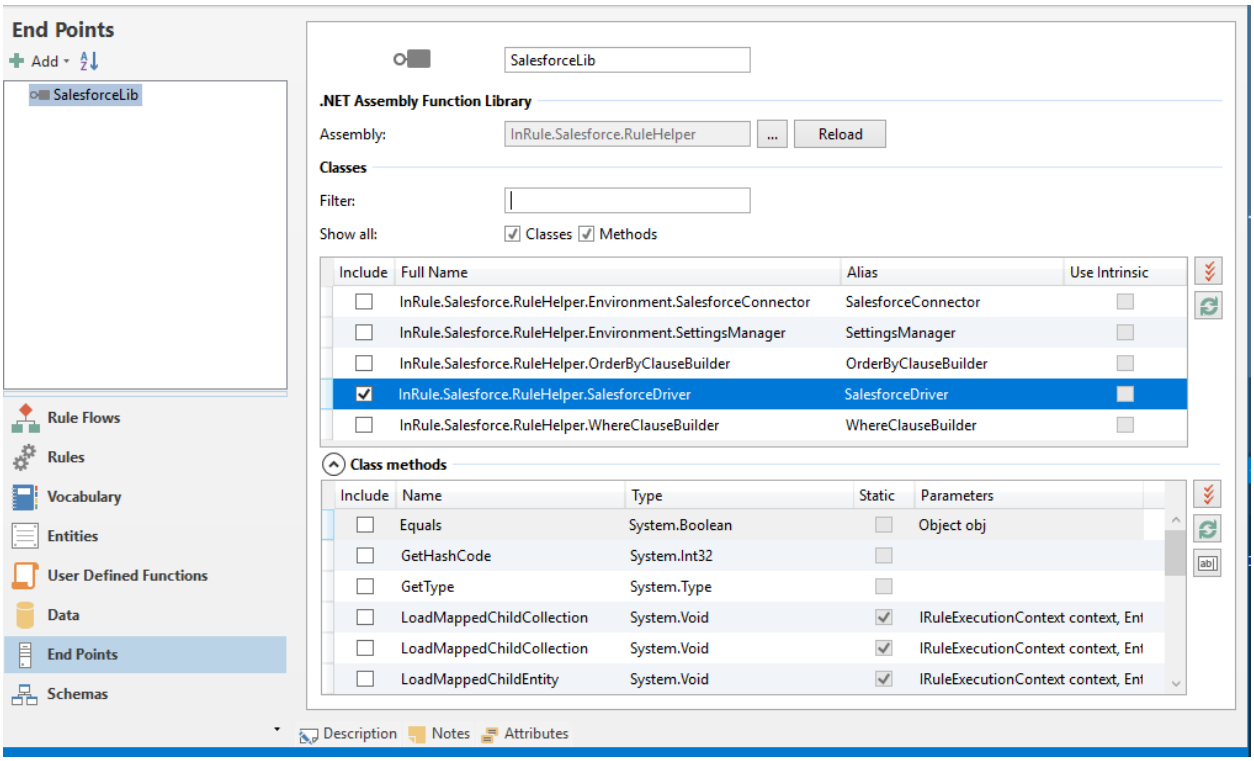


Integrating the Rule Helper Component

InRule provides a sample rule application (SalesforceRules) that is already configured for RuleHelper usage. You can simply edit this rule app, or, if you wish to integrate the Rule Helper into an existing rule app, you can copy both the UDF Library “RuleHelper” and End Point ‘SalesforceHelper” from the SalesforceRules rule to another rule application.

If you wish to manually create the UDF Library and End Point in irAuthor, follow the steps below.

1. Create a new rule app using the irX add-in for irAuthor.
2. Create a new “.NET Assembly Function Library” end point and bind the end point to the InRule.Salesforce.RuleHelper.dll assembly. Select the SalesforceDriver class and then select the methods that should be callable from rules. Edit the name of the end point to “SalesforceLib” or similar. Select the methods from the SalesforceDriver that are needed for the Rule Application. You do not need to select all the methods—only import the methods that will actually be used by rules. Additional methods can always be imported later be revisiting the endpoint screen and reloading the assembly.



3. Add a User Defined Function library and set the name to “SalesforceHelpers” or similar. This library will contain functions that the rules will call to query Salesforce.

4. Add a User Defined Function to the new library. The example below shows a UDF that will be used to execute the QueryCollection method on the SalesforceDriver. Fill out the UDF with script that will call a method on the SalesforceDriver.

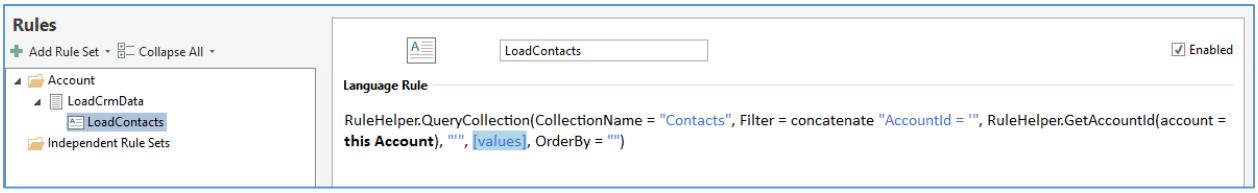
   Note: The methods on the SalesforceDriver are designed to be reused for more than one Entity type, Field, or set of Fields. The name of the Target Field or Collection should be supplied as a string. When querying a Collection of results, an optional “where” clause can be provided that will be forwarded to calls against the Salesforce SDK. In addition, an “order by” clause can be provided to return sorted results.

   Important: This integration pattern relies on the “Context” object that is available from irScript. The Context object returns information based on the context under which a given UDF is executed. For example, when executing an Entity Rule Set, the Context.Entity returns a reference to the Entity against which the current Rule Set is executing. The Context and its child properties are passed to the SalesforceDriver so it has enough information to form calls to Salesforce and map responses back to the InRule Rule Session.

   Note: The Context.FunctionLibraries property can be used to create calls to the .NET assembly library methods, such as the methods imported in Step 5 above. The following script example demonstrates how to use the Context object in irScript to form a call to a static .NET method:

   ```
   Context.FunctionLibraries.SalesforceDriver.QueryCollection(Context,
   Context.Entity, collectionName, filter, orderBy, connectionString);
   ```

5. Rules can now be authored to execute methods on the SalesforceDriver. These methods can be used to load Collections, single Entities, or single Fields from Salesforce based on conditional logic within rules.



Important: The Target Collection in the sample rule is called “FamilyMembers”. This is a Field that either does not exist in Salesforce (only for use in rules), or has been imported and then “disconnected” from Salesforce using the “Disconnect Field” button, or has “Auto Load” disabled.

Note: Please see the following sections for more details on the creating the “filter” clauses similar to the one used in this example.

Filtering Queries using the Where Clause Builder

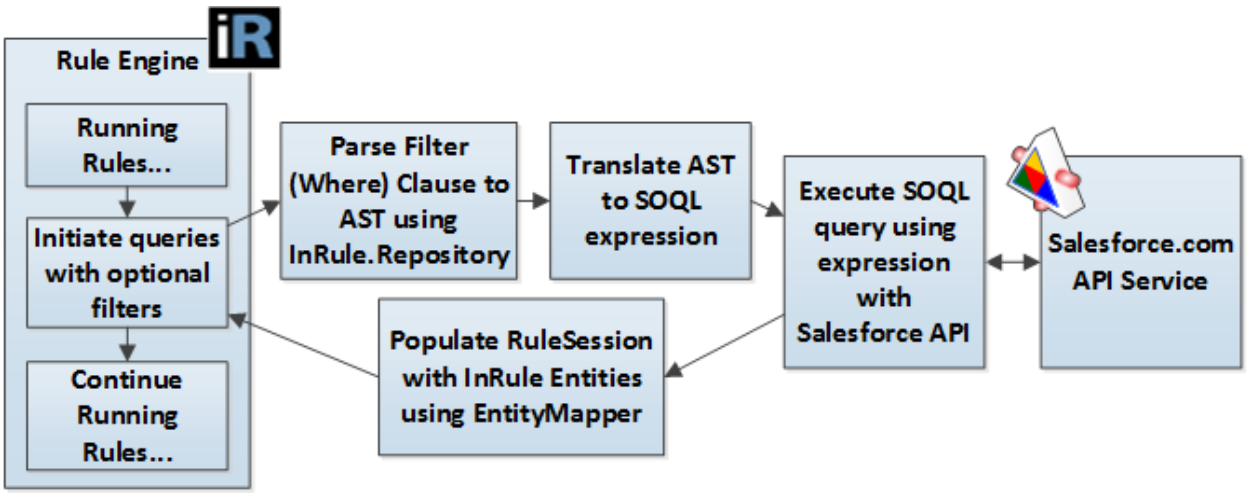
When loading data from Salesforce during rule execution, it is critical that the rule author is able to author logic to specify which Entity data to load. Using the RuleHelper, this is accomplished by allowing the rule author to pass in a “filter” or “where” clause into the calls against the SalesforceDriver class.

During execution of the SalesforceDriver, the filter clause is parsed into an Abstract Syntax Tree (AST) and then translated into SOQL (Salesforce Object Query Language) so it can be executed against the Salesforce data. The filter clause is based on the InRule function syntax format.

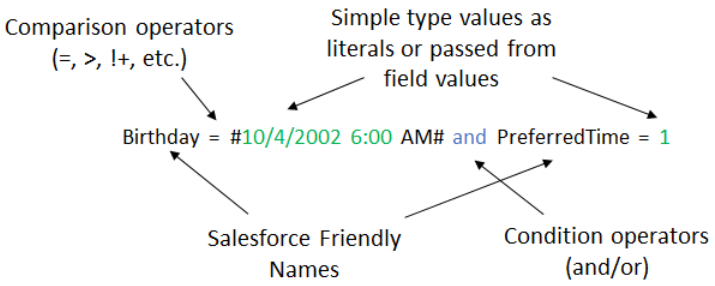
Note: The InRule function syntax format is used for the following reasons:

- The syntax rule format is consistent with the rule authoring experience used throughout irAuthor
- This format can make good use of the InRule AST parser that is included as part of irSDK

The diagram below depicts the logical flow of steps used by the SalesforceDriver and WhereClause builder classes to query data from rules.

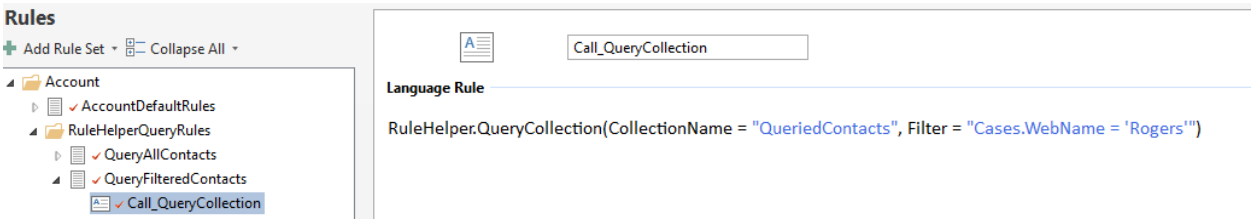


The diagram and notes below contain some additional information about forming the filter clause in a rule:

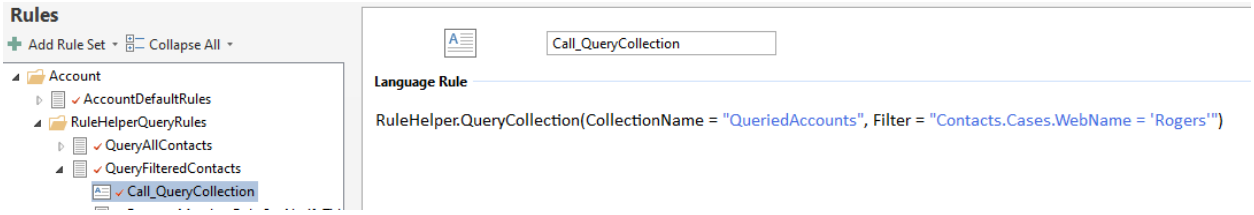


- All the Field names that are used are the Salesforce friendly names (Salesforce Field Label) that are used in the Rule Application. These names are mapped back to the Salesforce Field names in the parser.
- String literals should be wrapped in single quotes, date literals should be wrapped in pound signs.
- Simple operators are supported to compare values, such as =, !=, >, < (ex. Age > 21, Name != 'Ralph').
- Multiple conditions can be chained together using 'and' and 'or' keywords.
- The following keywords and operators are supported by the InRule AST parser and expression tree translation code: **=, <>, !=, +, -, *, /, or, and, xor, >, >=, <, <=, ^,**

The filter expression also supports querying against related entities, simply by appending the related entity name in front of the relevant query field. Querying against related entities requires that all **entities and fields** queried in the relationship chain be imported into irAuthor.

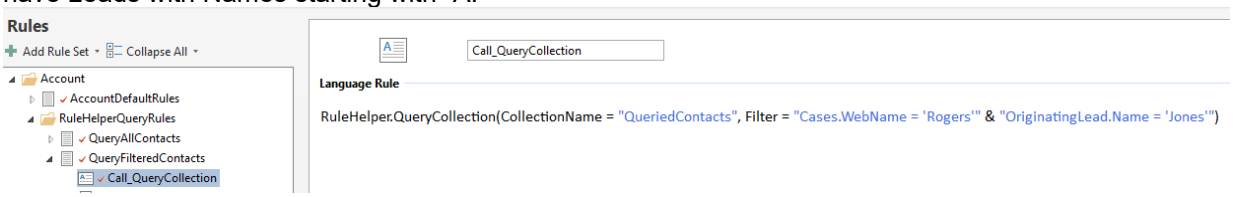


In this example, we are populating a collection by querying the Contact entity, which is the "parent" entity here. We are then applying a filter statement to return only contacts with related Cases that have a Web Name of "Rogers." Cases in this example is the "child" entity. Notice how the hierarchy of the related entity down to field is denoted. If you wanted to drill down another layer to a "grandchild" entity (in this example, an entity related to Case), it would be accomplished by simply continuing the chain from entity to field. Below is an example of a "grandchild" case:



This example would return Accounts that have related Contacts with Cases that have a Web Name of "Rogers." The filter expression can support querying in this manner up to 5 "layers" deep, not including the initially queried entity. Put another way, you can have up to 6 total different related entities in a single filter expression.

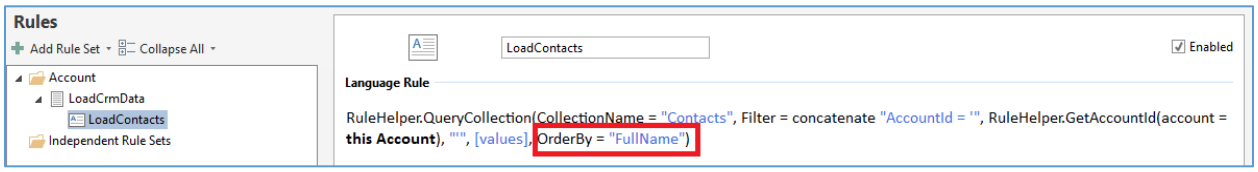
The filter expression supports querying against multiple properties from different related entities. In the below example, we are querying for Contacts with Cases that have Descriptions starting with “A” and also have Leads with Names starting with “A.”



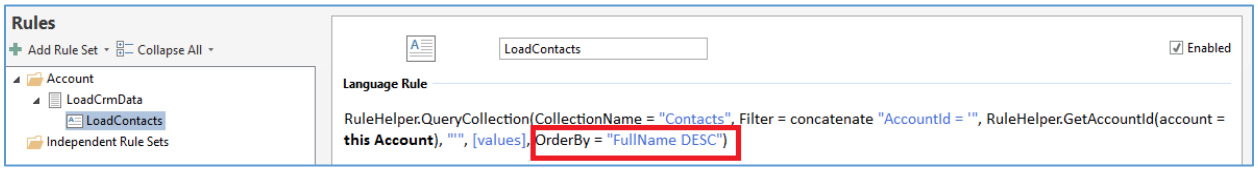
Ordering Query Results with the OrderByClauseBuilder

The SalesforceDriver class also supports the ability to control the order of the results returned from Salesforce by passing in an optional “order by” clause. The order by clause can accept only a single Field name, which should be the name of the Field in the Rule Application. The results are always sorted in ascending order unless the Field name is followed by the “desc” syntax. Please see the examples below:

To sort ascending, pass the Field name to use in the sort:



To sort descending, pass the Field name to use in the sort followed by the “desc” keyword:



Note: The “order by” clauses generally contain much simpler expressions than “where” clauses. However, InRule syntax rules format is used for the order by clause to be consistent with the where clause approach.

Methods Available in the Rule Helper

The following table lists the public, static methods that are available in the SalesforceDriver

Method Name	Description
`LoadMappedChildCollection`	Populates a child Entity Collection based on an existing 1:N relationship in Salesforce. The Collection is populated based on existing parent-child relationship data in Salesforce.
`LoadMappedChildEntity`	Populates a child Entity Field based on an existing 1:1 relationship in Salesforce. The Field is populated based on existing parent-child relationship data in Salesforce.
`QueryCollection`	Populates an Entity Collection with a set of a given Entity type. An optional filter clause (where clause) can be used to define selection criteria for the Entity set. The Collection does not need to correspond to a 1:N relationship in Salesforce.
`QueryEntity`	Populates an Entity Field or variable based on a query to Salesforce. An optional filter clause (where clause) can be used to define

Method Name	Description
	selection criteria for the Entity. The Field does not need to correspond to a 1:1 relationship in Salesforce. If more than one Entity is returned from the query to Salesforce, then the first Entity in the set is used.
`QueryField`	Populates a primitive Field or variable based on a query to Salesforce. An optional filter clause (where clause) can be used to define selection criteria for the Entity. If more than one Entity is returned from the query to Salesforce, then the Field value from the first Entity in the matching set is used.

Additional Flags Available to Control Loading and Caching Behaviors in the Rule Helper

During a given query operation, there may be advanced use cases that require specific control over loading or reloading data from Salesforce. The optional overloads of the QueryEntity and QueryCollection methods expose a set of optional Boolean flags that help control caching and depth of loading behaviors. The table below list these parameters:

Parameter Name	Description
`loadChildren`	Denotes if the execution service should recurse the Entity graph and load all children. If false, no children are loaded below the Collection Members that are loaded. The default value is true.
`useCaching`	Denotes if previously loaded Salesforce Entities should be reused from the InRule entity cache, or if new entity instances should be created. If false, the original entity data will be requested from Salesforce, and a copy of the Entity is created. The Instance ID is not set to the ID of the Salesforce Entity, which will also prevent changes to this entity from being written back to Salesforce. This functionality can, for example, be used to load the original values for an entity persisted in Salesforce when rules are run on update and compare the original and updated values. The default value is true.
`overwriteIfLoaded`	Denotes if a previously loaded Salesforce Entity should be repopulated with the latest values in Salesforce. This behavior will overwrite Field values stored in the cache. The default value is false.
`cacheInAppDomain`	Denotes if the result of the query should be saved in the persistent AppDomain cache. The difference between this parameter and the 'useCaching' parameter above is that enabling this parameter will save the query result in a cache that will persist across multiple different rule executions, where the above parameter only enables caching within the scope of a single rule execution.

Note: The default values should always be used for the cache settings unless there is a specific use case that requires different behaviors.

Using the Rule Helper with the Native REST Execution Service

While using the Salesforce rule execution service documented in this guide is the suggested way to interact with Salesforce via rules, you can also use the rule helper from the native REST execution service, which does not connect to Salesforce out of the box.

To do this, you will first need to copy the Salesforce rule helper assemblies to the bin directory of your execution service. These assemblies can be found in the `RuleHelperDeployment` folder of the framework zip you download from the support site. The method for copying these assemblies will differ

based on your exact hosting setup, but if your execution service is hosted in Azure you can copy these files over via the App Service Editor or FTP.

Once you have copied the assemblies to the bin directory, you will need to add the required app settings for authenticating to Salesforce. These are the same parameters provided in the Update parameters step of the template deployment, but you will need to provide the exact app setting names instead of the template parameter names. These are the settings you will need to provide, along with their mapping to the template parameters from the section linked above:

App Setting Name	Template Parameter Name
Inrule:sf:api:loginUrl	sfLoginUrl
inrule:sf:api:username	sfUsername
inrule:sf:api:password	sfPassword
inrule:sf:api:securityToken	sfSecurityToken
inrule:sf:api:consumerKey	sfConsumerKey
inrule:sf:api:consumerSecret	sfConsumerSecret

Once you have set these app settings, you should be able to use the rule helper to manually load and save data to Salesforce from rules, just like you would from the Salesforce rule execution service.

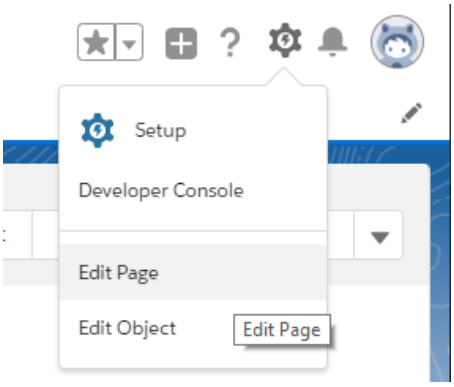
Appendix E: Methods for Executing Rules with Salesforce

Once the Salesforce components are installed, end-to-end connectivity can be tested by adding a call into a Page Layout. Many possible approaches can be applied to call the DecisionClient Apex class—six approaches are documented here, adding a button to the Lightning interface, adding a JavaScript button to the Classic UI, executing rules from Apex, executing rules from Apex triggers, executing rules from Salesforce Flow, and executing rules from the Salesforce REST Endpoint

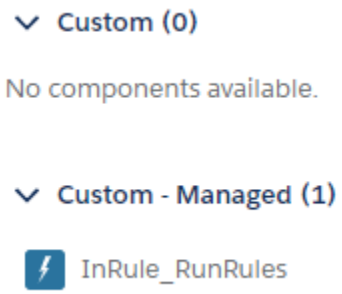
1 Adding a Lightning Button

Navigate to the entity you wish to add the button to, note that you will have to add the button to each entity individually.

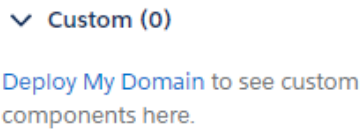
Once on the desired entity page find the settings icon in the top right corner and select Edit Page.



On the left-hand side of the page in the Lightning Components section scroll to the bottom of the page until you find the InRule_RunRules in the Custom-Managed Section

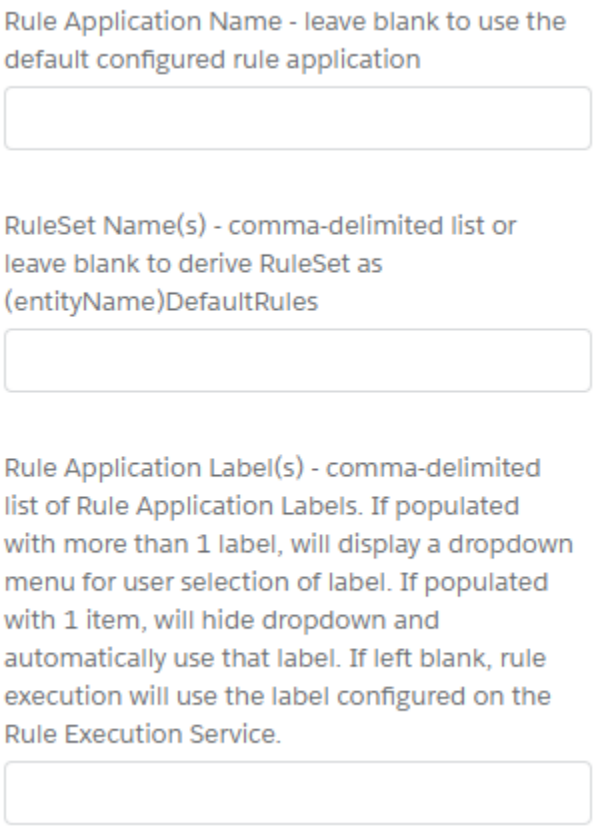


If you have not setup your Salesforce domain, you will see this message in the Custom-Managed Section.



Simply follow the link to setup your domain. Once it has been set and you have logged in with the new domain come back to edit page and you should now see the InRule_RunRules Lightning Component.

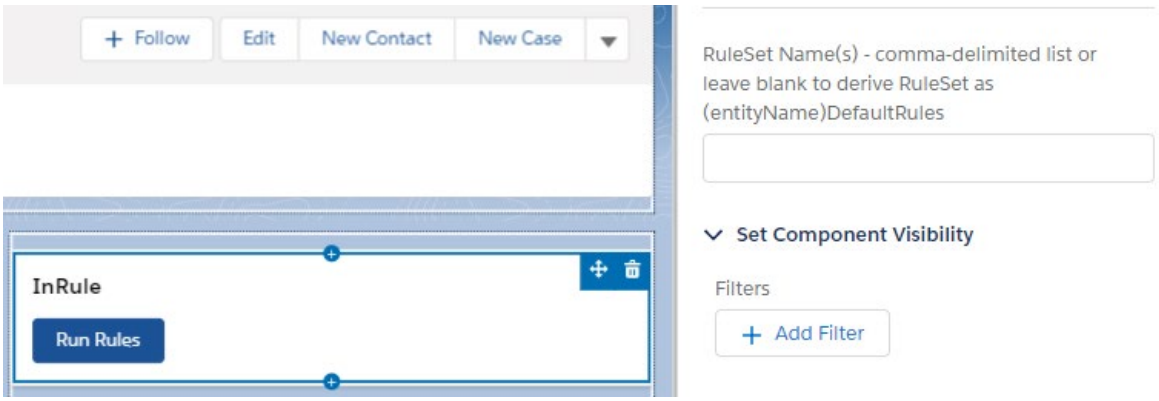
Drag the window to the desired location on the page view. Once you have placed the component, click on it and a menu bar on the right will appear three two available configuration values.



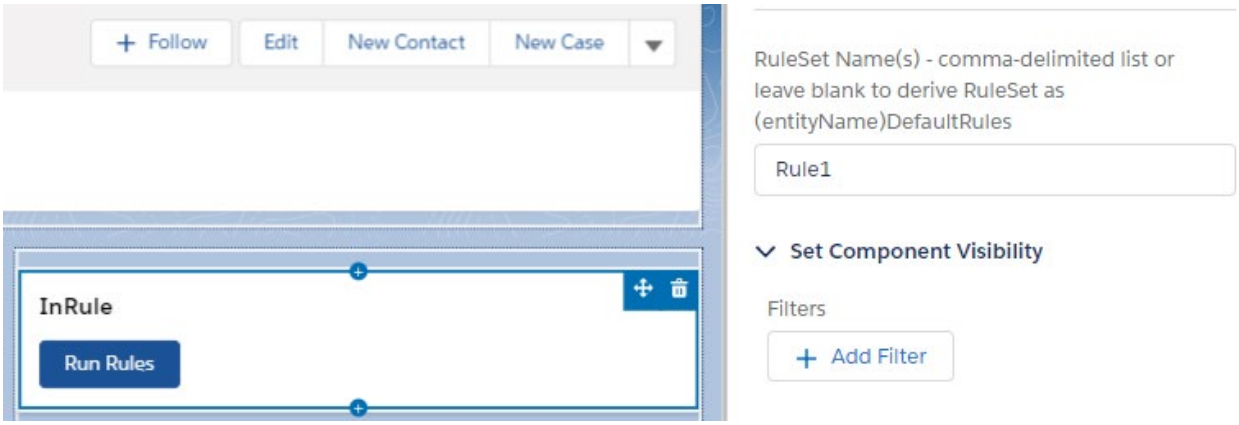
The first configuration field, Rule Application Name, allows you to define a Rule Application to use for this specific lightning component. Placing a value in this field will override the Rule Application you have defined in your Custom Setting as the default. Leave this field blank to use the default configured Rule Application.

The “Rule Set Name(s)” field accepts a comma delimited list of Rule Set names. Adding multiple rule sets here will create multiple Run Rules buttons for each defined Rule Set on the page. Leaving this field blank will leave only the singular “Run Rules” button, and this button will execute the Default Rule Set for the given entity. The Default Rule Set is defined as the entity’s name + “DefaultRules.” For example, if you are adding the Lightning component to the Account entity, the Rule Set name it will default to if this field is left blank is “AccountDefaultRules.”

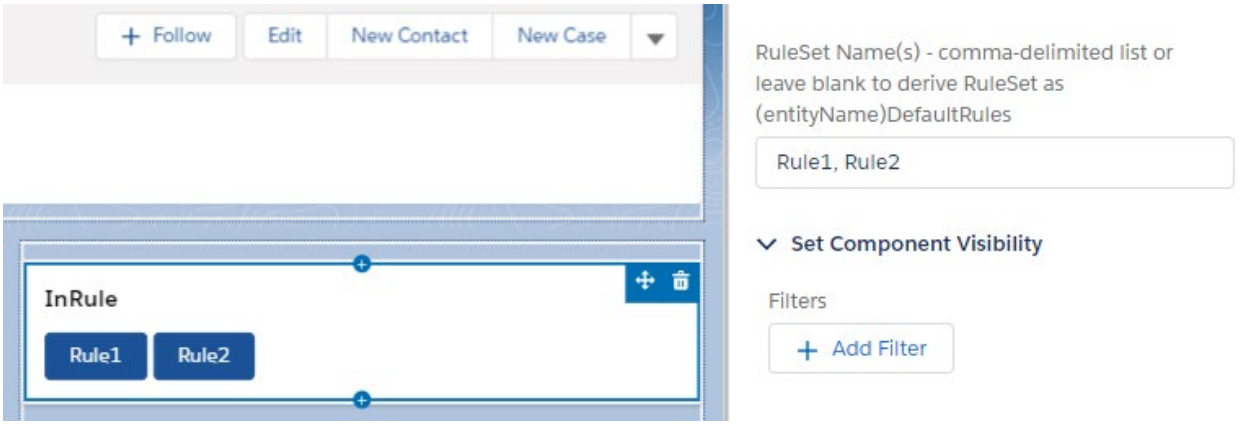
If no Rule Sets are defined, 1 button will be displayed that uses the default Rule Set for the entity:



If one Rule Set is defined, 1 button will be displayed that will use that defined Rule Set:

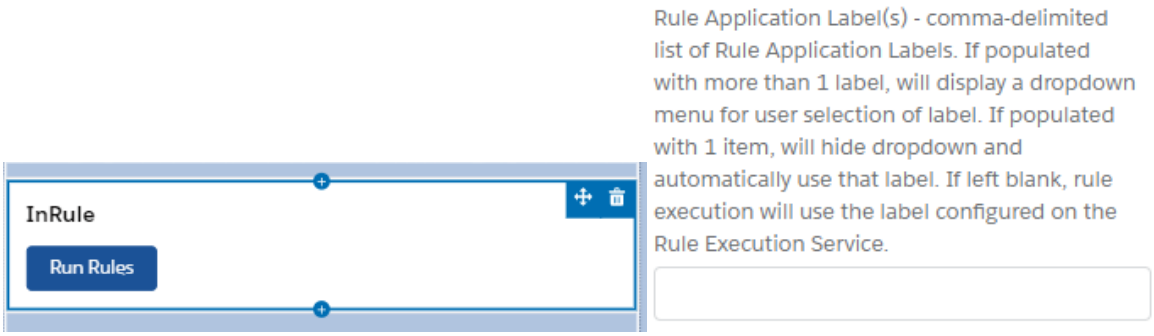


If 2+ Rule Sets are defined, a button per Rule Set will be displayed that will use the defined Rule Sets.

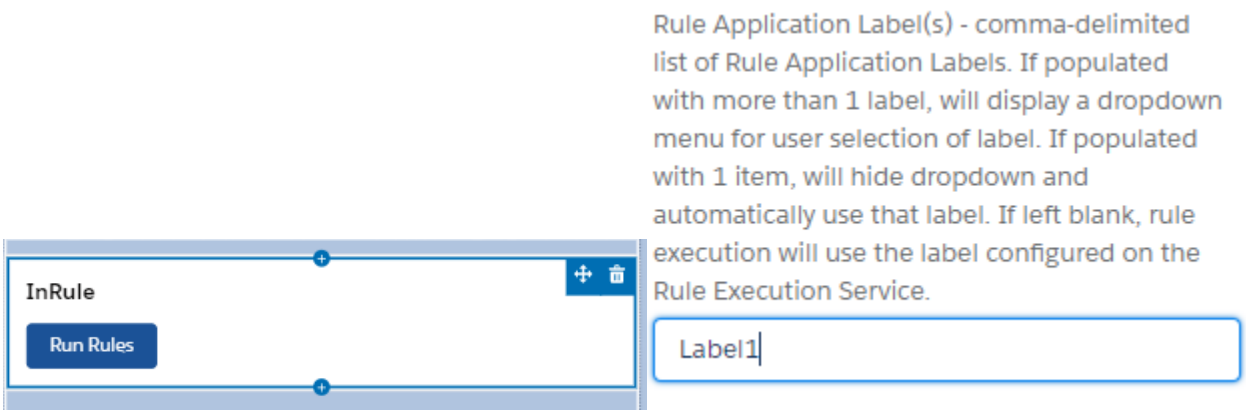


The last configuration field accepts a comma-delimited list of Rule Application Labels. This allows for overriding the Rule Application Label defined on the Rule Execution Service and providing users the ability to change what label to run rules against from the defined list of options.

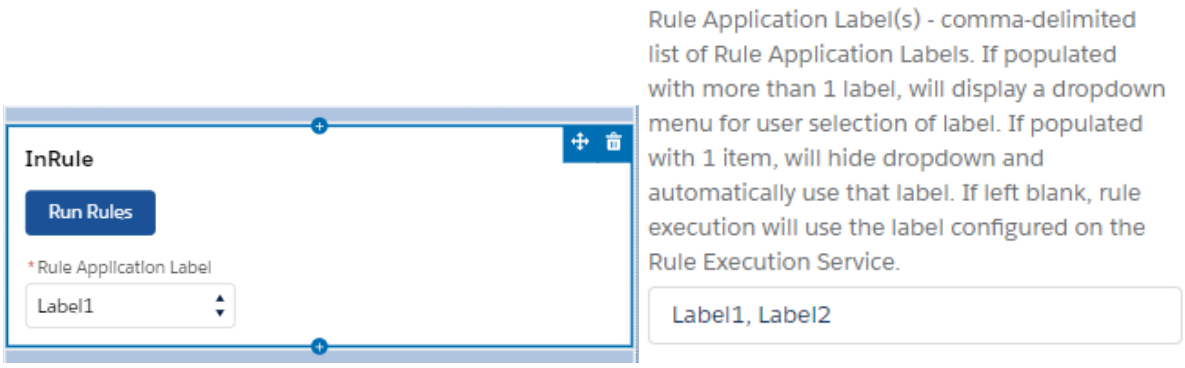
When no Rule Application Labels are defined, the Run Rules button appears as normal and rules will be executed against the label configured on the Rule Execution Service:



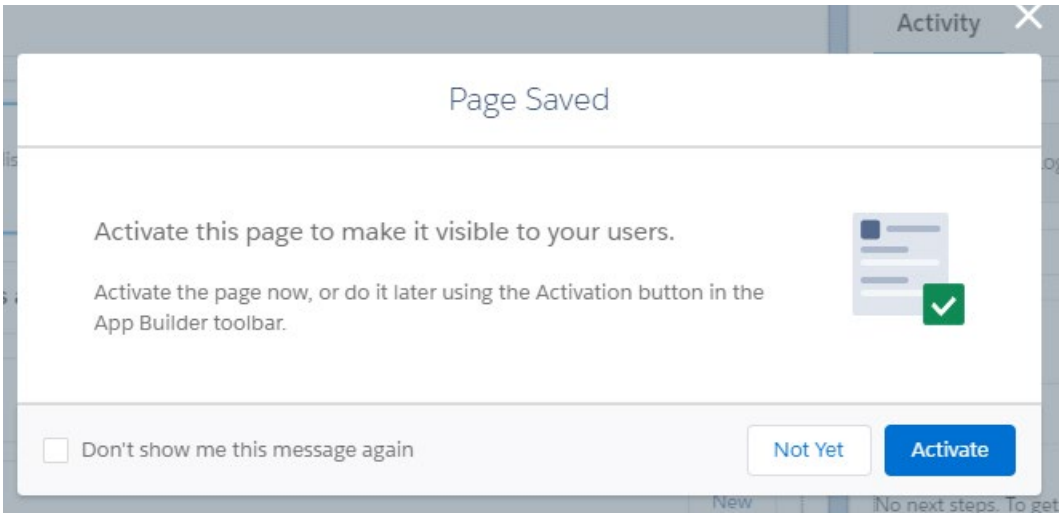
If one Rule Application Label is defined, the Run Rules button appears as normal and rules will be executed against that defined label:



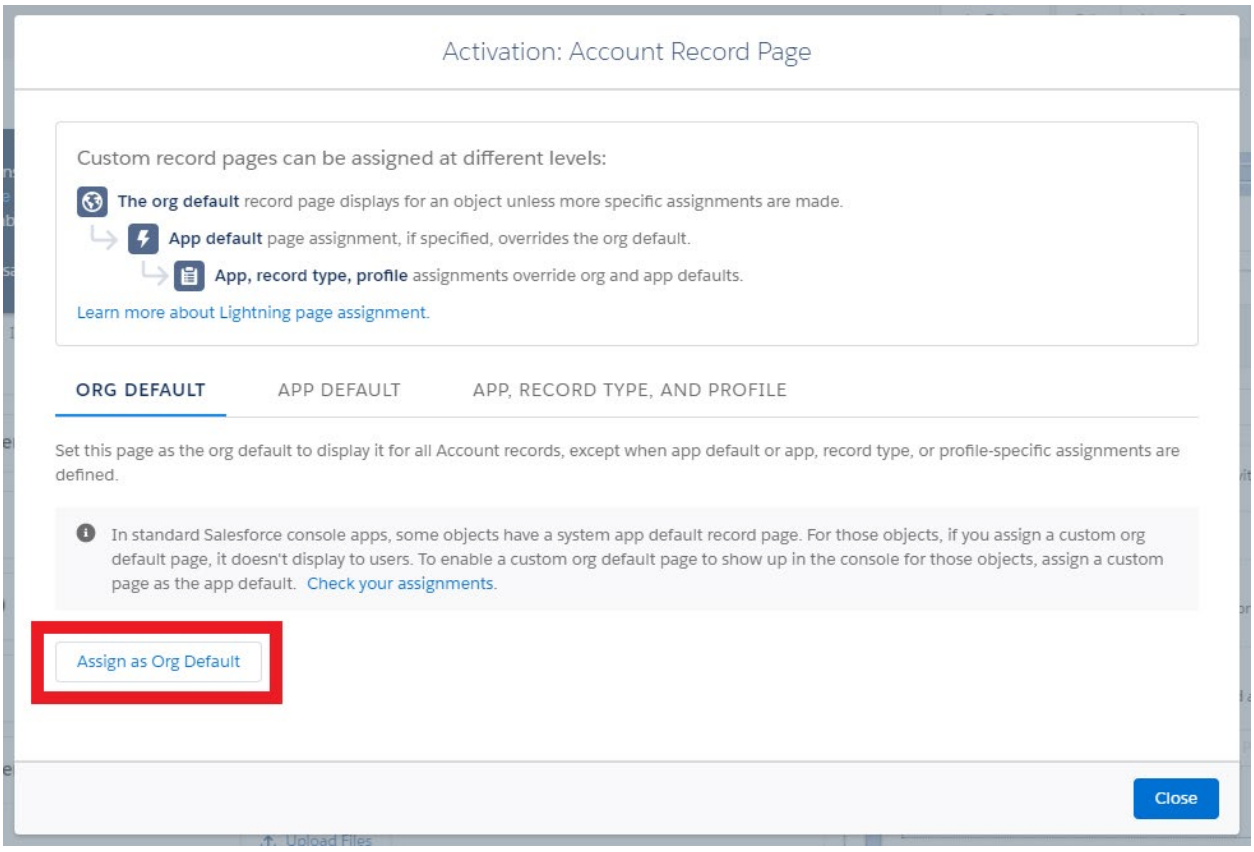
If more than one label is defined, a dropdown box will appear underneath the Run Rules button that allows users to select what label to run rules against. All defined Rule Sets and their associated buttons will execute against the selected label:



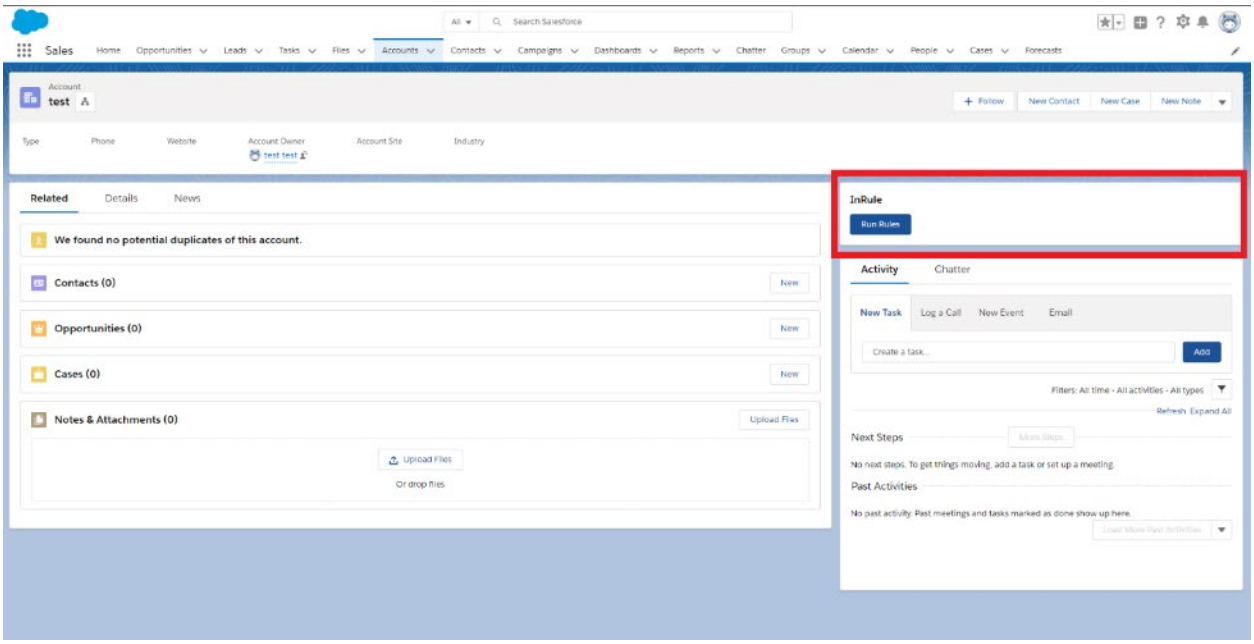
Once you have configured these fields, click save in the top right corner. It will prompt you to activate this page to make it visible to your users, click activate.



Salesforce will now ask for the scope to activate the record page. Select the desired scope and click Assign and then save on the subsequent prompt.



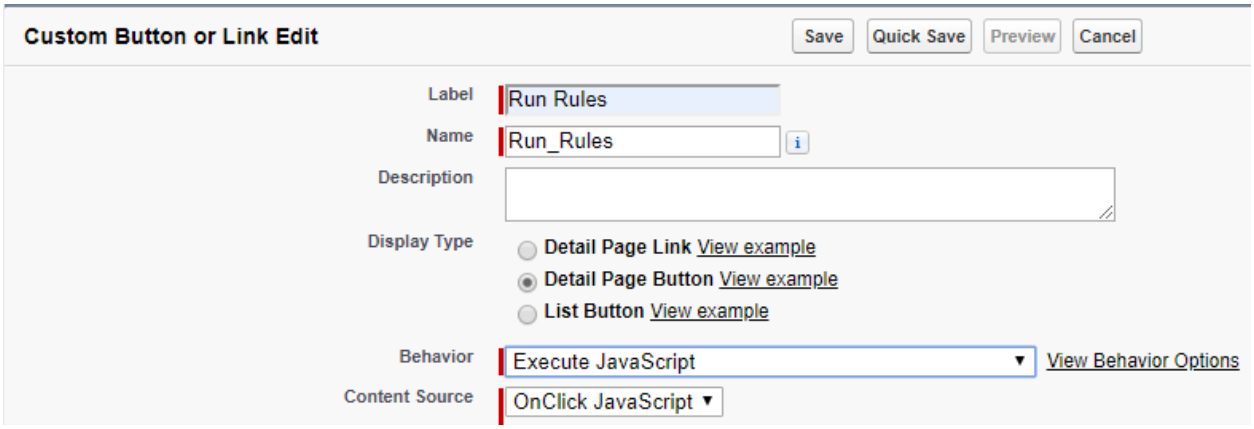
Navigate back to the main entity page and the component should appear.



2 Adding a Classic UI Button

Classic UI buttons are not included directly in the InRule for Salesforce app, but we do provide a helper library and sample code here to make it easier to add a button to a classic form for running rules. This button will run rules, show notifications, and refresh field changes when clicked.

To add a Classic UI button, navigate to Setup → Customize or Create, add a new Button. Set the Behavior to Execute JavaScript and then the Content Source to OnClick JavaScript.



Within the script window, copy and paste over the sample JavaScript found below:

```
{!REQUIRESCRIPT("/soap/ajax/33.0/connection.js")}
{!REQUIRESCRIPT("/soap/ajax/33.0/apex.js")}
{!REQUIRESCRIPT('/resource/' & LEFT(SUBSTITUTE(SUBSTITUTE(SUBSTITUTE(TEXT(NOW()),':',''),'-
','')),' ',''),10) & '000/inrule__DecisionClientHelper')}

var config = {
entityId : '{!Account.Id}',
```

```
entityType : "Account",
ruleSetName : "AccountDefaultRules"
};

var response = executeRules(config);

displayNotifications(response);

window.location.reload();
```

Your script window should now look similar to this:

```
{!REQUIRESCRIPT("/soap/ajax/33.0/connection.js")}
{!REQUIRESCRIPT("/soap/ajax/33.0/apex.js")}
{!REQUIRESCRIPT('/resource/' &
LEFT(SUBSTITUTE(SUBSTITUTE(SUBSTITUTE(TEXT(NOW()),':',''),'-',''),' ',''),10) &
'000/inrule__DecisionClientHelper')}

var config = {
entityId : '{!Account.Id}',
entityType : "Account",
ruleSetName : "AccountDefaultRules"
};

var response = executeRules(config);

displayNotifications(response);

window.location.reload();
```

To customize this button for the specific entity type you intend to use this button for, you can alter the "config" object in the JavaScript by editing the values of the properties to match the particulars you want.

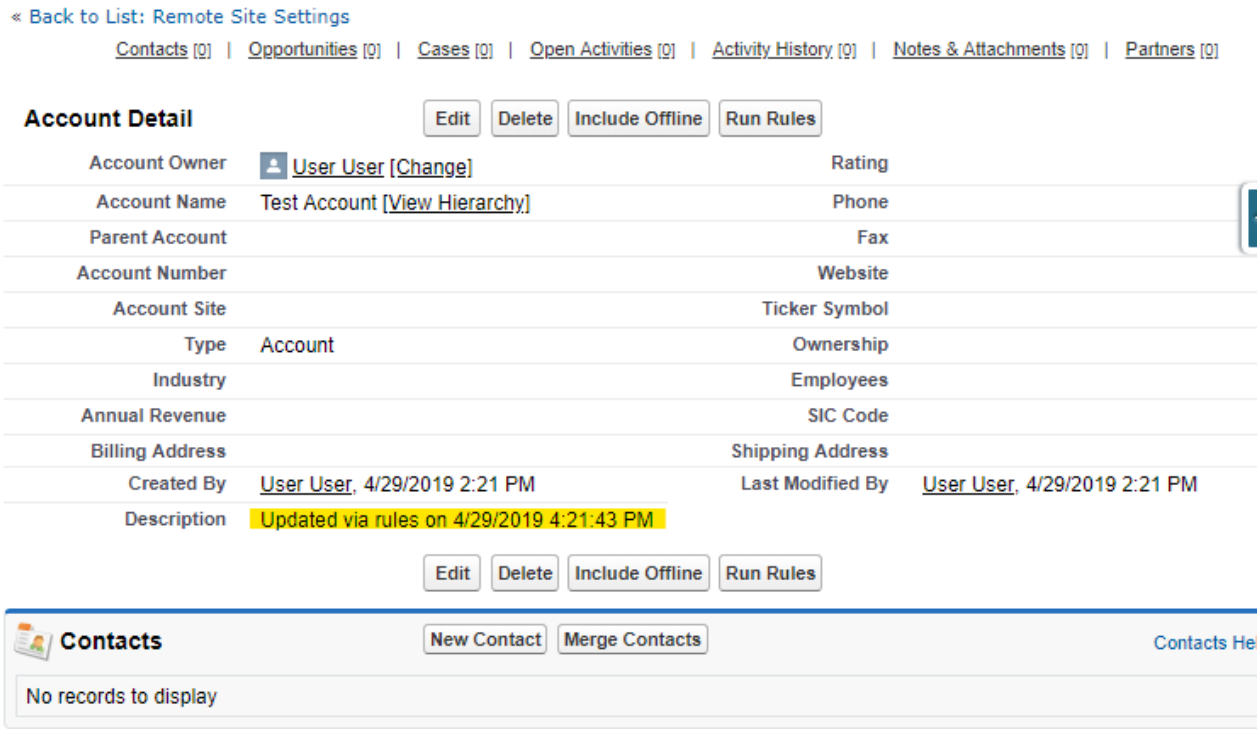
The table below includes a list of the properties that can be configured on the "config" object:

Parameter Name	Description
entityId	The unique Salesforce identifier for the root object in the request. This follows the naming convention of: "entityName.Id"
objectType	The Salesforce object type of the root object in the request
ruleSetName (optional)	Optional. The name of an Explicit Rule Set to call as the entry point for rule execution. If no value is supplied, then the Rule Sets configured as 'Auto' for that entity will be executed.
loggingLevel (optional, must be manually added to config object)	Optional. An integer that denotes the amount of information to log after rule execution completes. Results are written to the InRule_Log__c object. Values can be 0, 1, 2, or 3: • 0 – disable all logging • 1 – Logs errors and a minimal amount of information on successful requests • 2 – Logs errors, request information, rule engine notifications, and rule engine validations • 3 – Logs the same information as 2, but also includes JSON from the HTTP request and response payloads

	Defining a logging level here will override the logging level universally defined for the InRule for Salesforce App in the custom object settings for a specific button.
useEntityPrefix (optional, must be manually added to config object)	Optional. If set to true, the DecisionClient will append the entity label to the supplied rule set name.
ruleAppName (optional, must be manually added to config object)	Optional. Defining and passing a RuleAppName here allows you to override the default Rule App Name defined in your Custom Setting created during initial configuration for this specific button.

Once your config object has been appropriately configured, simply save your button and add it to the entity page layout for use. The button will leverage the DecisionClientHelper static resource to handle the communication between the JavaScript button and the DecisionClient, as well as handling any notifications returned from the rules.

To verify everything is setup correctly, open up a record for the configured object and click the 'Run Rules' button you added. What will happen next is dependent on your configuring rule set, but if you are running the included 'AccountDefaultRules', you should see the description of the account updated with the current date and time.



3 Executing Rules from Apex or Lightning Web Components

Rule Execution is handled on the Salesforce side by an Apex class installed with the InRule for Salesforce App called the DecisionClient. The DecisionClient is what accepts the run rules requests from the Run Rules button and trigger requests to handle sending it along to the Rule Execution Service, but it also accepts calls from other Apex classes or Lightning Web Components (LWCs). This section will document how to run rules from your own Apex classes or LWCs by calling the DecisionClient, as well as detailing what manner of response it returns.

To run rules from another Apex class, simply invoke the executeRules method in the DecisionClient with the following call:

inrule.DecisionClient.executeRules(String eventType, String id, String objectType, String ruleSetName, Boolean useEntityPrefix, String ruleAppName, String ruleAppLabel, string entityImage, Boolean persistChanges)

Alternatively, setting up a call to the DecisionClient from a LWC requires setting up an action. First, you must reference the DecisionClient as your LWC's controller in your .cmp file:

<aura:component controller="inrule.DecisionClient">

To create the run rules action to hit the DecisionClient in your component controller, new up your action with by setting it as below:

var action = component.get('c.executeRules');

From there, the action can be setup and enqueued as with any other action, with the following parameters to set:

action.setParams({
eventType: string
id: string
objectType: string
ruleSetName: string**,**
useEntityPrefix: boolean**,**
ruleAppName: string**,**
ruleAppLabel: string,
entityImage: string,
persistChanges: boolean
});

Regardless of whether you are calling from Apex or a LWC, the arguments will need to be defined with the following values:

Parameter Name	Description
Event Type (String)	The event type for invoking rules. This will always need to be either “insert,” “update” or “delete,” depending on what your rule is doing
entityId (String)	The unique Salesforce identifier for the root object in the request. This is a property available on all Salesforce objects and be accessed with: “entityName.Id”
objectType (String)	The Salesforce object type of the root object in the request. For example, if running rules against an Account entity, this will need to be set to a string value of “Account”.
ruleSetName (String)	The name of an explicit Rule Set to call as the entry point for rule execution.
useEntityPrefix (Boolean)	If set to true, the DecisionClient will append the entity label to the supplied rule set name. For example, if you pass in a ruleSetName of “DefaultRules” and set useEntityPrefix to true, the effective ruleSetName name would be “AccountDefaultRules”
ruleAppName (String) (Optional)	Optional. Defining and passing a RuleAppName here allows you to override the default Rule App Name defined in your Custom Setting created during initial configuration for this specific button. If you do not wish to override your Custom Setting, pass null here.
ruleAppLabel (String) (Optional)	Optional. Defining and passing a rule app label here allows you to override the rule app label configured on the execution service. If you do not wish to override your execution service’s configuration, pass null here.
entityImage (String) (Optional)	Optional. This allows you to pass in the serialized JSON string of an entity image as it exists at the time of calling the DecisionClient, which will have rules execute against the entity image passed in, not the entity image as it exists in Salesforce when the execution process reaches the execution service. If you do not wish to pass this in, pass null instead.
persistChanges (Boolean) (Optional)	Optional. This allows you to define whether to persist data changes made during rule execution to Salesforce. Pass in null to default this value to true.

The DecisionClient will always return a JSON string; a serialized version of the DecisionClientResponse object. Once you have received it back as a string, you will need to deserialize it to access its properties.

The available properties on the DecisionClientResponse are:

Property Name	Description
IsSuccess (Boolean)	Denotes whether or not rules successfully ran with no errors.
Notifications (List<NotificationMessage>)	Provides a list of all notification messages returned by rule execution. Each NotifactionMessage has 2 properties on it: • **Type (Integer):** The notification type is an integer that maps to a Salesforce notification type. 0 maps to Informational, 1 to Success, 2 to Warning, and 3 to Error. • **Message (String):** The notification text
Errors (List<ErrorMessage>)	Provides a list of all errors encountered during rule execution. These differ from errors thrown by the rule application itself, which are instead added to the Notifications list as NotificationMessages of type Error. Errors added into the Errors list are strictly errors encountered during rule execution runtime.

	Each ErrorMessage has 2 properties: • **Source (String):** At what point during runtime the error was thrown • **Message (String):** The error text

4 Executing Rules from Apex Triggers

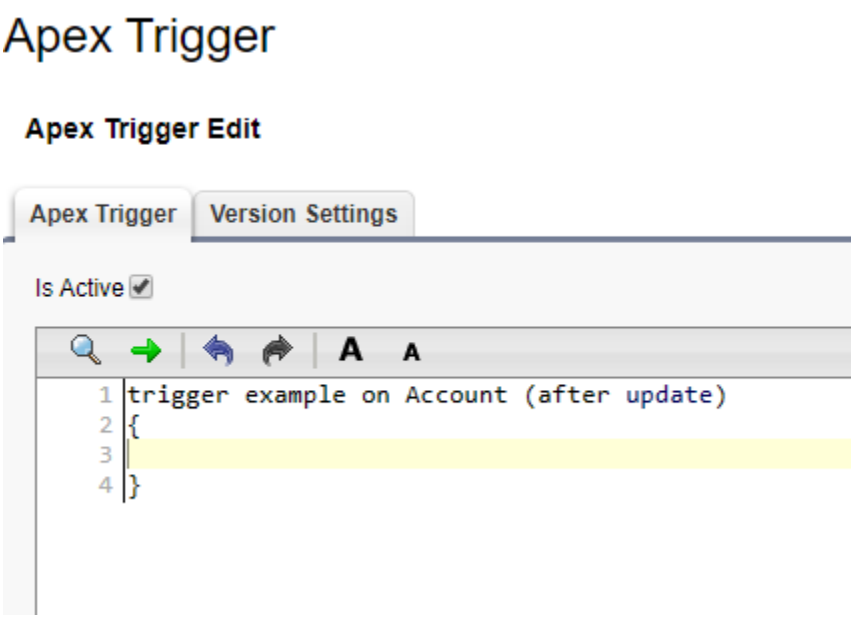
As of v5.5.1, the InRule for Salesforce App supports the execution of rules from **Update** triggers with some limitations. For a full list of these limitations, references the Known Issues and Limitations of Executing Rules from Triggers section below.

To emphasize, only **After Update** and **After Insert** triggers are supported. Due to Salesforce's enforcement of all external service methods running asynchronously, trigger execution and persistence to the Salesforce database will always complete before rule execution has had time to finish, thus preventing the ability to execute rules on the "in-progress" entity image before it has been persisted. As an extension of this limitation, no entity image rollbacks can occur as a result of any validation done during or after the rule execution process.

As an alternative to Apex triggers, Salesforce Flows are a no-code alternative that can also be used to call InRule. For an overview of the differences between these two options, refer to the following Salesforce documentation: Record-Triggered Automation

Adding a Rule Execution Trigger

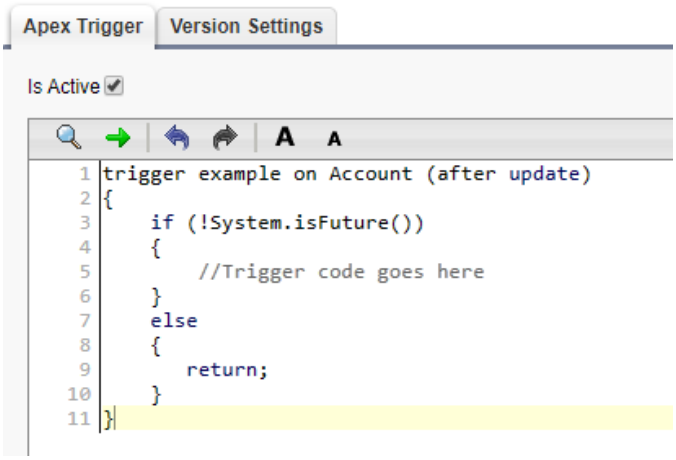
To define a rule-executing trigger, create a new trigger on the entity of choice. The following example will use Account:



Next, all rule executing triggers must contain the boilerplate wrapper around their code defined below. This is required to prevent infinite loops that may occur if the invoked rule edits the same entity.

Apex Trigger

Apex Trigger Edit

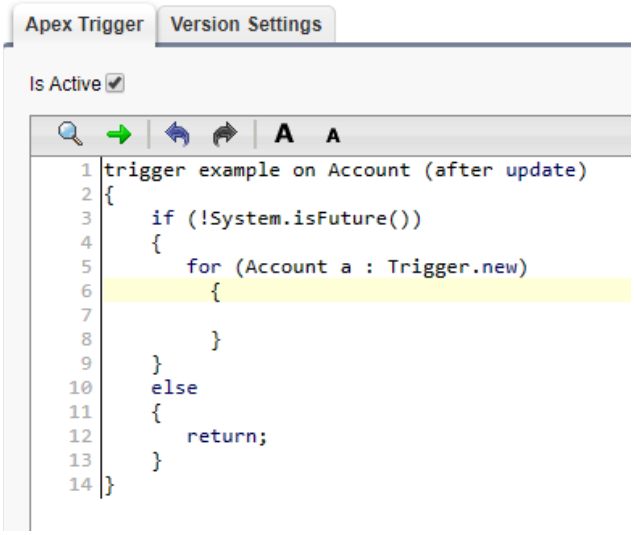


All operations, including calls to the rule execution service, will be contained within the **If (!System.isFuture())** block.

Based on whether you're using an insert or update trigger and whether you want to pass the original or new entity values to the rules, use either Trigger.new or Trigger.old to access the entity image:

Apex Trigger

Apex Trigger Edit



At this point, the only remaining required component is to call the rule execution service. To call the rule execution service from a trigger, invoke the inrule.DecisionClient.executeRulesFromTrigger method and pass the appropriate parameters:

Is Active

```
trigger example on Account (after update)
{
    if (!System.isFuture())
    {
       for (Account a : Trigger.new)
         {
            inrule.DecisionClient.executeRulesFromTrigger('update', a.id, 'Account', 'AccountDefaultRules', false, null);
         }
    }
    else
    {
       return;
    }
}
```

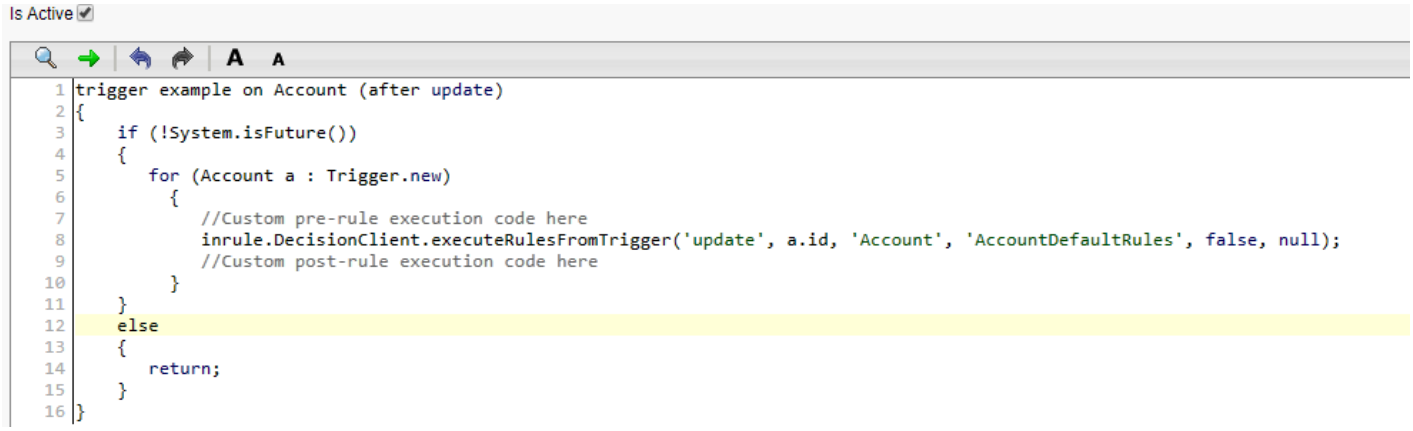
The parameters for calling executeRulesFromTrigger, in the order they're passed into the signature:

Parameter Name	Description
eventType	The event type for invoking rules. This will always be 'update'.
entityId	The unique Salesforce identifier for the root object in the request. This follows the naming convention of: "entityName.Id"
objectType	The Salesforce object type of the root object in the request
ruleSetName	The name of an explicit Rule Set to call as the entry point for rule execution.
useEntityPrefix	If set to true, the DecisionClient will append the entity label to the supplied rule set name. For example, if you pass in a ruleSetName of "DefaultRules" and set useEntityPrefix to true, the effective ruleSetName name would be "AccountDefaultRules"
ruleAppName (optional)	Optional. Defining and passing a RuleAppName here allows you to override the default Rule App Name defined in your Custom Setting created during initial configuration for this specific button. If you do not wish to override your Custom Setting, pass "null" here, as displayed in the example above.
entityImage (optional)	Optional. Defining and passing an entity image allows you to capture the entity image at time of trigger execution to ensure its state for rule execution, whereas leaving it out will require the execution to query Salesforce to get the current entity state. This state may differ at that point from its state when trigger execution began due to the asynchronous nature of trigger execution. Passing an entity image requires you first to serialize it into a JSON string. An example of this can be found in the screenshot below.

An example of serializing and passing an entity image:

```
trigger example on Account (after update)
{
    if (!System.isFuture())
    {
        for (Account a: Trigger.new)
            {
                String entityImage = JSON.serialize(a)
                inrule.DecisionClient.executeRulesFromTrigger('update', a.id, 'Account', 'AccountDefaultRules', false, null, entityImage)
            }
    }
    else
    {
        return;
    }
}
```

At this point, all required components are set. Any operations you wish to include before/after the invocation of the rule execution service can be added at the points denoted below:



It's important to note that any custom code written after the rule execution service has been called **must not** have any dependencies on outcomes of rule execution, as rule execution will be operating asynchronously, and trigger execution will continue and likely finish well before rule execution does.

Known Issues and Limitations of Executing Rules from Triggers

Currently, there are a number of known issues and limitations with executing rules from triggers:

- Currently, only **After Update** and **After Insert** triggers are supported. Due to Salesforce's enforcement of all external service methods running asynchronously, trigger execution and persistence to the Salesforce database will always complete before rule execution has had time to finish, thus preventing the ability to execute rules on the "in-progress" entity image before it has been persisted. As an extension of this limitation, no entity image rollbacks can occur as a result of any validation done during or after the rule execution process.

- Due to the asynchronous nature of trigger execution, the trigger will not wait on a response from the rule execution service before continuing with any additional code contained within it. Therefore, you cannot rely on having a rule response at any point during trigger execution, even if your code relying on it comes after your call to the execution service. It is not recommended to have any code in your trigger that relies on any data contained in the rule response.

- As a result of triggers' asynchronous nature, automatic data refresh on a record page as a result of rule execution is not supported in Classic view, since the page has no way of knowing when rule execution has completed. Notifications will still display once rule execution completes, but the page will have to be manually refreshed by a user for them to be able to see any updates to the record itself.

- Automatic data refresh on record pages **is** supported in Lightning, however, it requires giving all users that will be initiating triggers that execute rules Read permissions to the InRule_Event platform event. This can be done in two ways, both of which are explained in the below section Adding Permissions to InRule Platform Event

- Due to the fact that triggers execute asynchronously and Formula fields on entities are calculated and persisted asynchronously as well, any rules you attempt to execute from a trigger that make use of a Formula field will be caught in a race condition between the Formula field being calculated and persisted to the database and the rule execution service querying Salesforce to fetch the currently persisted value of that field. If persistence for the Formula field has not yet completed by the time the rule execution service reaches that point, it will pull down an out of date value for that field and your rules will execute using the wrong value. Therefore, it is strongly recommended that you **do not** execute rules that act on Formula fields from triggers. While there is a possibility for it to work correctly, there is a high chance for unexpected behavior to occur and there is no means of controlling whether it will work properly or not on any given execution of that Rule Set from a trigger.

Adding Permissions to InRule Platform Event

Displaying notifications and refreshing the entity form after rules are run from a trigger relies on a Salesforce Platform Event that is installed as part of the InRule package. However, by default, most Salesforce default user profiles do not have read access to Platform Events. **In order for notifications to be displayed and data to be properly refreshed on a record page after rule execution via trigger, you must give your users read access to the InRule_Event platform event.**

There are 2 methods of doing this, both with their own advantages and disadvantages. Which approach is best is ultimately dependent on the circumstances of your organization.

Add Platform Event Permissions via Editing Profiles

The first way of giving the needed permissions to your users is by editing their Profiles. In Salesforce, all users have an assigned Profile that defines their permissions in the environment. Granting the needed permissions to your users through this method can be as simple as editing the profile permissions of the user types that will be setting off your trigger(s). This approach has some pros and cons.

Pros:

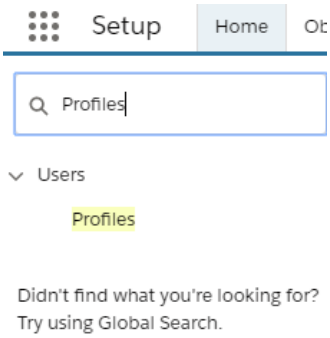
- Makes changes to entire profiles rather than specific users, meaning once it is done, all that will need to be done for future users is to assign them to a profile with these permissions already granted and they'll be set
- If you are already using predominately custom/cloned Salesforce profiles in your environment, making this change is very quick, easy, and doesn't require any future overhead.

Cons:

- Cannot be accomplished with default profiles, which mandates cloning all of your profiles if your users are using predominately default profiles.
- Migrating all users from default profiles to new profile types can be highly time-intensive depending on the size of your org.

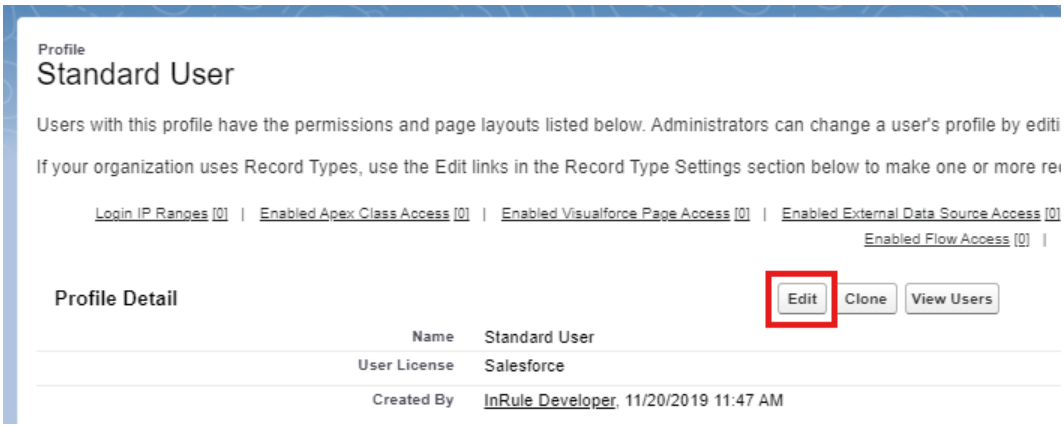
To do this approach, a user with system admin permissions must login to Salesforce and go to Setup.

From there, search for Profiles:



Once you've navigated to the Profiles page, select the profile you wish to edit. For this example, we'll be granting the necessary permission to Standard Users

Once on that profile's page, select Edit:



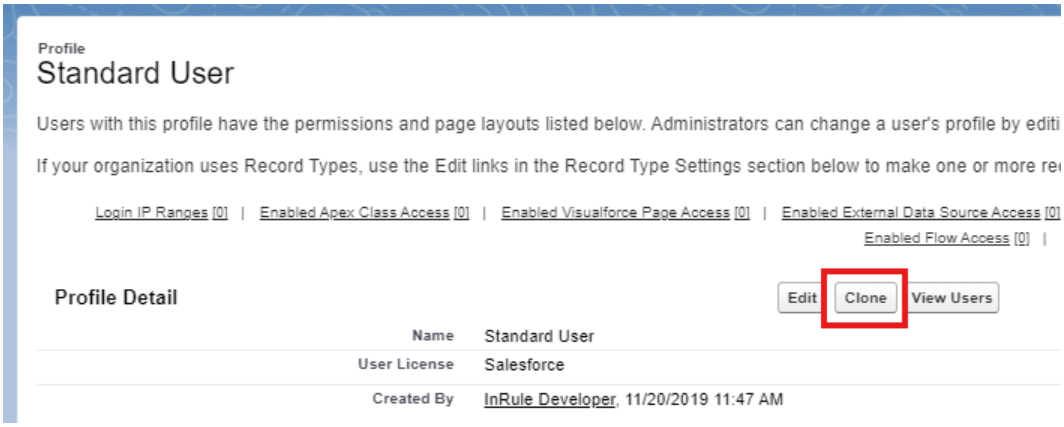
Scroll down until you see the "Platform Event Permissions" header:

Check the "Read" checkbox, then scroll to the bottom of the page and press Save.

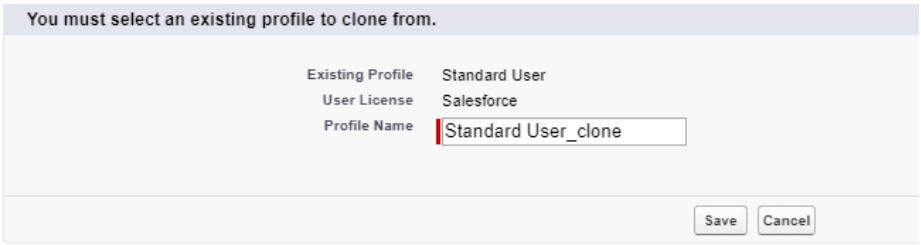
Repeat this process as needed for all profile types that need to be able to initiate rule executing triggers.

If the Read checkbox is greyed out for you and you can't select it, this means you're trying to edit a default Salesforce profile. Salesforce has several "default" profile types that are commonly used; the Standard User profile used in the example above is one such default profile. Unfortunately, Salesforce default profiles **cannot** be edited. This means that if you have users that will be setting off your trigger(s) that use default profile types, you will have to clone that profile, move your users from the default Salesforce profile over to the clone of it, and add the permission on the new profile clone.

To clone a profile, navigate to that profile's detail page as above, but instead of clicking "Edit," click "Clone."



Name your new profile appropriately:



Hit save. Your new profile clone has been created. You should now be able to follow the steps above to add the appropriate Platform Event permission.

Once that's done, you need to move your users from the original profile over to the clone.

You will need to repeat this process for every profile type in your organization that needs UI updates when triggers run

Add Platform Event Permissions via InRule User Permission Set

The alternative approach to granting the necessary platform event to your users is to apply the InRule User Permission Set that is installed as a part of the InRule Salesforce package. This approach has its own pros and cons:

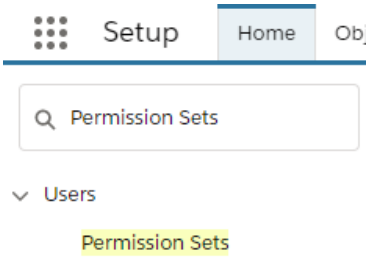
Pros:

- The InRule User Permission Set comes pre-installed and pre-configured with the InRule Salesforce package, making the only required step being to assign the permission set to the appropriate users

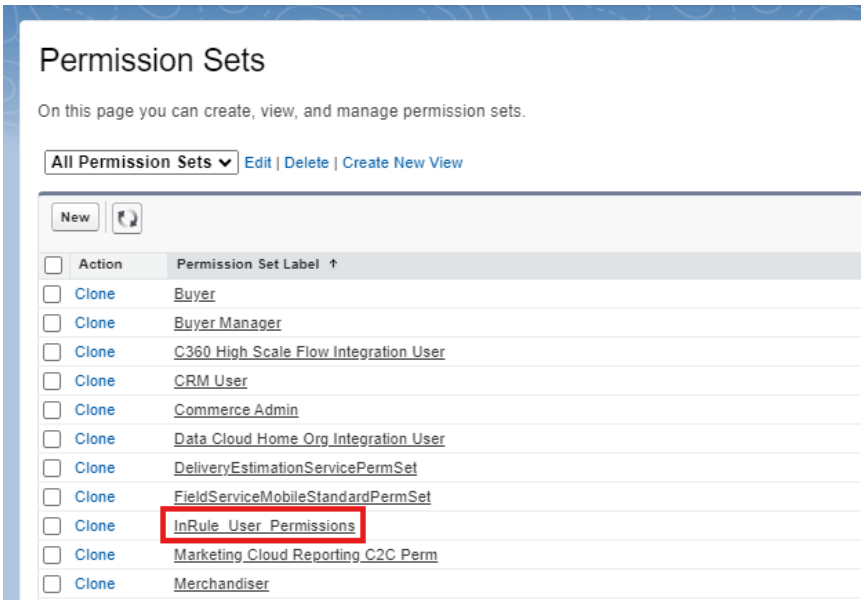
Cons:

- Permission Sets must be assigned to specific users, rather than profiles. This means that anytime a new user is created, the permission set must be independently added to that user, creating user management overhead. Managing this may not be viable within a large organization.

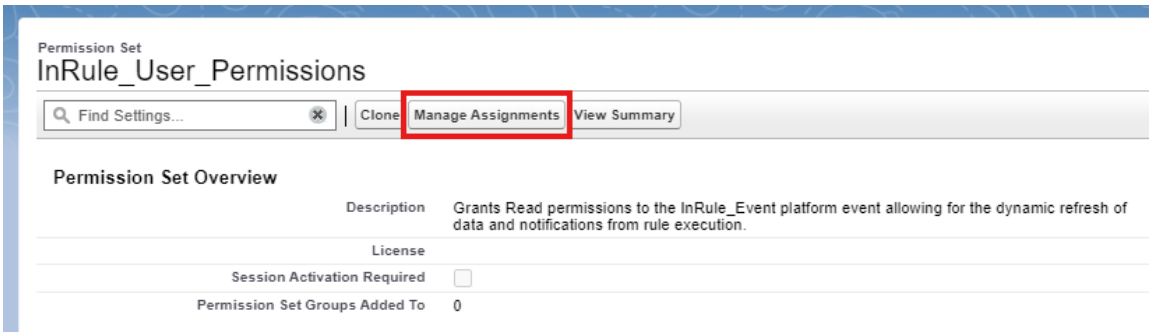
To add users to the InRule User Permission Set, go to Setup and search for Permission Set:



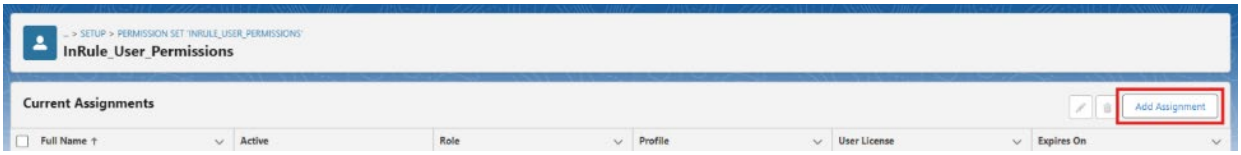
Find and select InRule_User_Permissions:



Select “Manage Assignments”



Select “Add Assignments”



You’ll be presented with a list of all users in your environment. You can select all users that will need the ability to initiate rule executing triggers. Once you’ve selected all the relevant users, you can select “Assign,” and the permission set will be assigned to all of those users.

These users will now be able to see UI updates when rules are run from triggers.

5 Executing Rules from Salesforce Flow

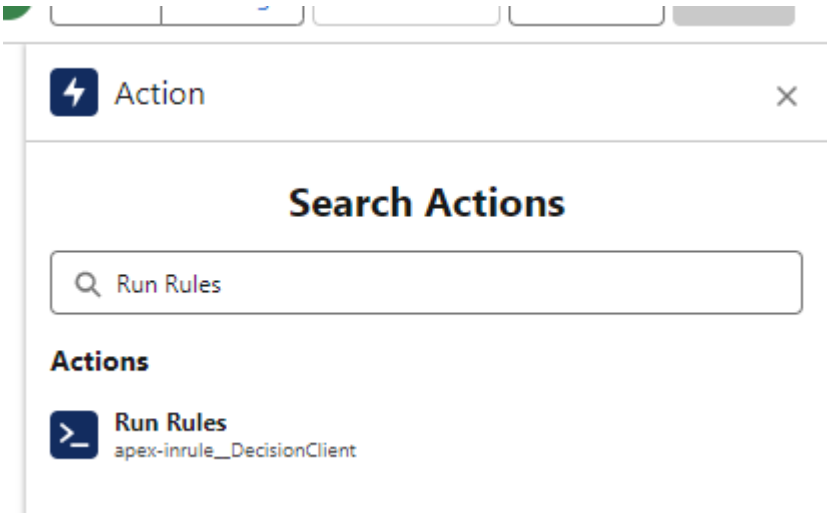
If you need to integrate rules with more complex process automation, rules can also be run from Salesforce Flow. The InRule integration with Flow lets you run rules against a particular entity record and receive back the status and notifications from rule execution.

There are currently 5 types of Flows in Salesforce:

- **Screen Flow** - With Screen Flow you can create a custom UI (user interface) and guide users through a business process that can be launched from Lightning Pages, Experience Cloud (previously known as Community Cloud), quick actions and more.
- **Record-Trigger Flow** - Record-Triggered Flow launches when a record is created, updated, or deleted. So far, we have used Apex triggers for this automation, some of which can now be done using Flows.
- **Scheduled-Triggered Flow** - This flow launches at the specified time and frequency for each record in a batch. Traditionally, we have met this kind of requirement by using Apex batch jobs.
- **Platform Event Flow** - Platform event flow Launches when a platform event message is received. For example, you can pump the data from an external system in Platform Events and then use Flows to split and save the records in different objects.
- **Autolaunched Flow** - Auto Launches flow launch when invoked by Apex, Process Builder or even REST API.

For more information about these Flows, you can visit the Introduction to Salesforce Flows.

To run rules from a flow, you will need to add an 'Action' element to your flow. You can find the InRule action by searching for 'Run Rules' in the search box. The display name will be 'Run Rules' and the ID will be apex-inrule__DecisionClient



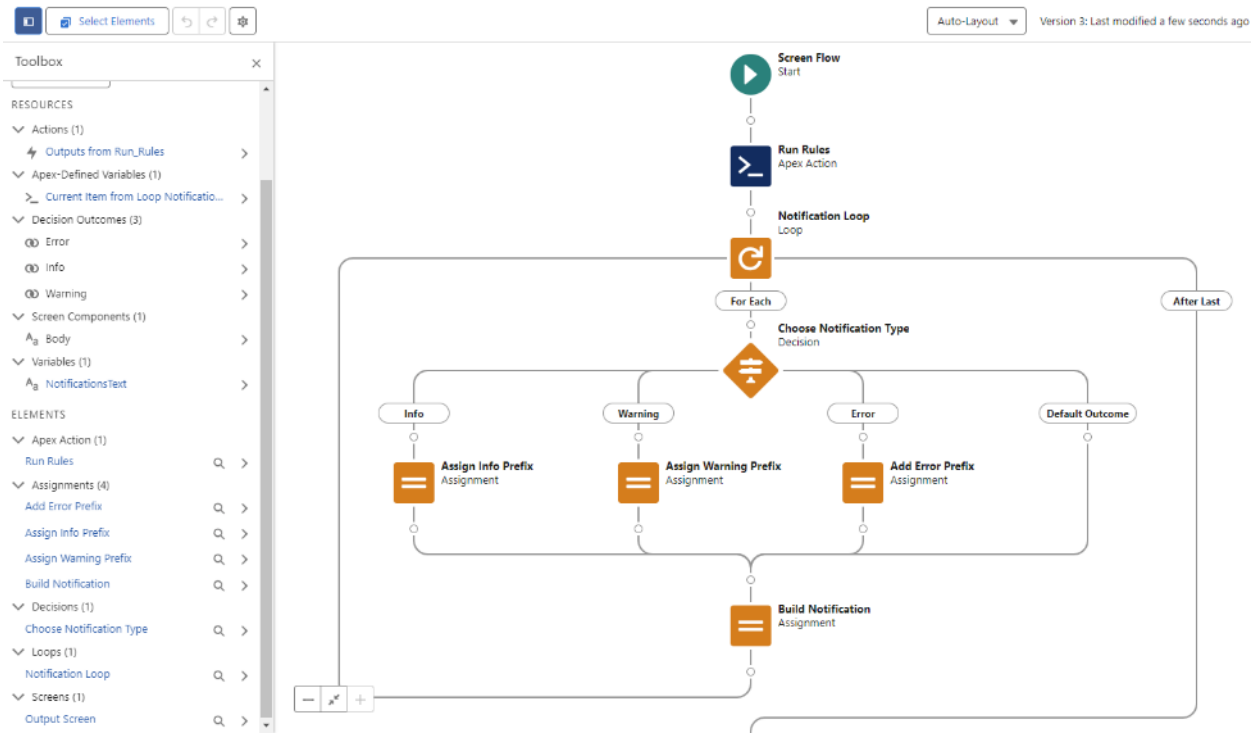
When adding the action, you will need to fill in the following required parameters

Parameter Name	Description
Entity ID (String)	The unique Salesforce identifier for the root object in the request. If running a flow from an object page, you can configure this value to be passed into an input variable
Object Type (String)	The Salesforce object type of the root object in the request. For example, if running rules against an Account entity, this will need to be set to a string value of "Account".
Persist Changes (Boolean)	Enable or disable saving changes made to entities during rule execution
Rule App Label (String)	Set to use a specific catalog version label when executing rules, otherwise leave blank to always use the latest version of the Rule App
Rule App Name (String)	The name of the InRule Rule App that contains the rule set to run
Rule Set Name (String)	The name of an explicit Rule Set to call as the entry point for rule execution.
Use Entity Prefix (Boolean)	Typically set to false. If set to true, the DecisionClient will append the entity label to the supplied rule set name. For example, if you pass in a ruleSetName of "DefaultRules" and set useEntityPrefix to true, the effective ruleSetName name would be "AccountDefaultRules"

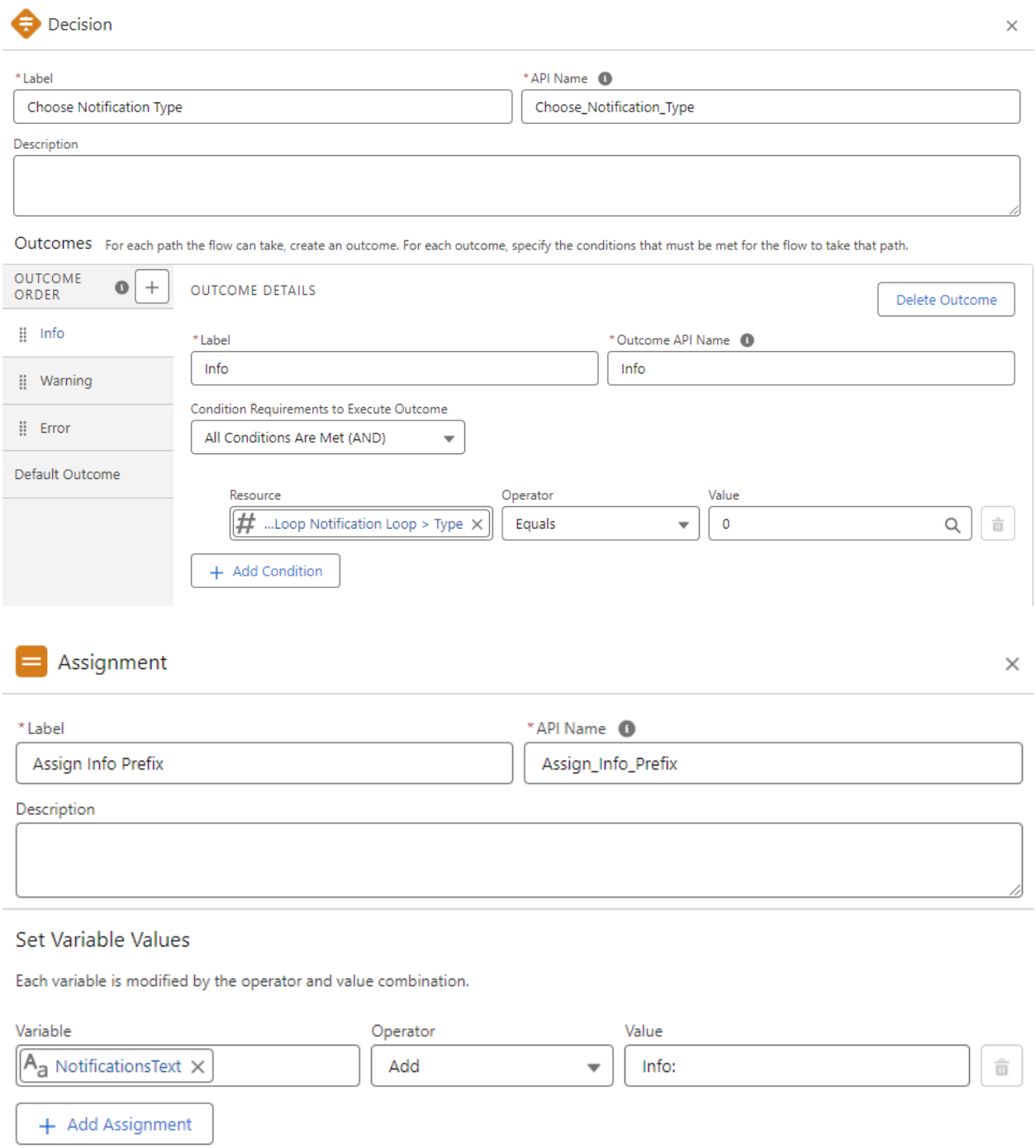
This action has three outputs

Output Name	Description
Notifications (Apex-Defined collection)	Includes notifications fired from rules during execution. This output can be assigned to a resource variable of the same type for use in subsequent steps. The Apex class for this collection is 'inrule__NotificationMessage'. This class has the following properties: • Type: integer value representing notification type. 0 for Info, 1 for Warning, and 2 for Error • Message: string value containing the notification text
Errors (Apex-Defined collection)	Includes any errors that prevent rule execution from completing. This output can be assigned to a resource variable of the same type for use in subsequent steps. The Apex class for this collection is 'inrule__NotificationMessage'. This class has the following properties: • Message: string value containing text of the error • Source: string value containing the error source
Is Success (Boolean)	If rules are not able to be executed for any reason, this value will be set to true. If true, additional error information will be available in the 'Errors' output collection

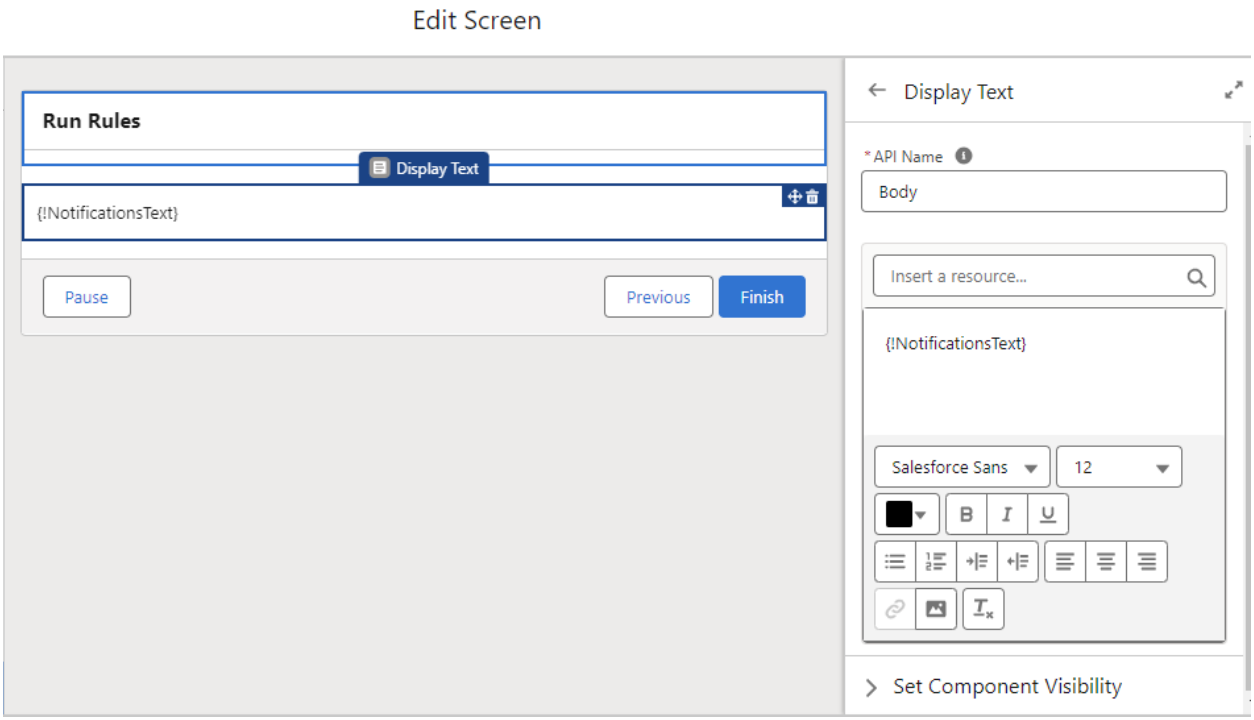
The following is an example of what a Flow using rules might look like. You can do other things with the output from the Apex action, but this demonstrates a common use case, displaying rule notifications to the user. The flow starts with Apex action, assigns the output collections to variables, loops over the notifications to build a single string containing all notifications, and then outputs the value to the screen



When looping through the notifications, a decision checks the notification type integer to determine what text to prefix the message with.



The combined notification text value is then displayed on the screen



6 Executing Rules from REST Endpoint

As of v5.7.3 rules may also be executed by interacting with the Decision Client over REST. This easily enables external workflows, such as Power Automate, to execute rules from the Decision Client. A POST endpoint is available on the Decision Client that will accept a request to execute rules and return a response containing information on the entity changes or information about errors encountered.

Authentication with Salesforce

Before sending your request to execute rules, an access token needs to be retrieved. This will be used in the rule execution request to authenticate with the Salesforce REST API. Below is an example request body that might be used in a POST request to your login URL to receive an access token. More information about REST API authorization can be found in the Salesforce REST API Developer Guide.

```
{
    "grant_type": "password",
    "client_id": "{{clientId}}",
    "client_secret": "{{clientSecret}}",
    "username": "{{username}}",
    "password": "{{password}}{{secretToken}}"
}
```

Rule Execution Request

The URI for Decision Client rule execution REST endpoint will use the subdomain for your Salesforce org.

```
https://{{DomainName}}.my.salesforce.com/services/apexrest/inrule/executerules
```

The access token from the authorization request will then need to be provided as a bearer token in your authorization headers.

Finally, you will need to supply the body for your salesforce request. Below is information on the request parameters that can be sent as part of your Rule Execution Request, as well as an example request body.

Parameter Name	Description
Id (String)	The unique Salesforce identifier for the root object in the request. This is a property available on all Salesforce objects and be accessed with: “entityName.Id”
ObjectType (String)	The Salesforce object type of the root object in the request. For example, if running rules against an Account entity, this will need to be set to a string value of “Account”.
RuleSetName (String)	The name of an explicit Rule Set to call as the entry point for rule execution.
RuleAppName (String) (Optional)	Optional. Defining and passing a RuleAppName here allows you to override the default Rule App Name defined in your Custom Setting created during initial configuration for this specific button. If you do not wish to override your Custom Setting, pass null here.
RuleAppLabel (String) (Optional)	Optional. Defining and passing a rule app label here allows you to override the rule app label configured on the execution service. If you do not wish to override your execution service’s configuration, pass null here.
EntityImage (String) (Optional)	Optional. This allows you to pass in the serialized JSON string of an entity image as it exists at the time of calling the DecisionClient, which will have rules execute against the entity image passed in, not the entity image as it exists in Salesforce when the execution process reaches the execution service. If you do not wish to pass this in, pass null instead.
PersistChanges (Boolean) (Optional)	Optional. This allows you to define whether to persist data changes made during rule execution to Salesforce. Pass in null to default this value to true.

```
{
    "Id": "0016g000015JnOnAAK",
    "ObjectType": "Account",
    "RuleSetName": "AccountDefaultRules",
    "RuleAppName": "SalesforceRules",
    "PersistChanges": false
}
```

Appendix F: Azure App Service Plan & Application Insights Configuration

Azure App Service Plan Overview

The Salesforce Rule Execution Azure App Service runs on an Azure App Service Plan. The ARM Template deployment process outlined in Section 3.3.2: Rule Execution App Service for Salesforce will, by default, automatically deploy an App Service Plan for you.

This App Service Plan will be deployed to the subscription and resource group provided during the deployment process. Additionally, the plan will be configured with a “F1” (Free) pricing tier, but this can be increased based on your scaling and configuration needs.

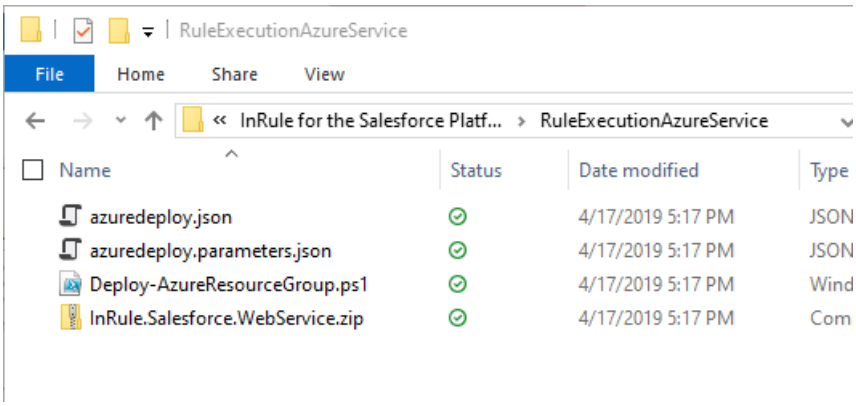
Important: If you leave the ARM template configured to re-deploy the App Service Plan, updating an existing one, the pricing tier of that App Service Plan **will be set to the default pricing tier, regardless of what it may currently be set to.** If you do not wish the pricing tier to be reverted to default, it is recommended you follow the steps below.

Should you wish to use a pre-existing Azure App Service Plan rather than have a new one created for you, a few configuration steps within the ARM template parameters file are necessary.

Configuring the ARM Template to Use an Existing Azure App Service Plan

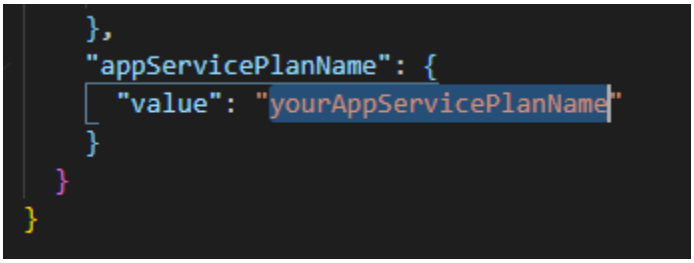
1: Locate azuredeploy.parameters.json

The ARM template parameters file is located in the *RuleExecutionAzureService* folder as, defined in Section 3.3.2: Rule Execution App Service for Salesforce



2: Populate “appServicePlanName” parameter

Open the file in your text editor of choice. First, populate the **“appServicePlanName”** parameter. Set the value equal to **the name of your app service plan**.

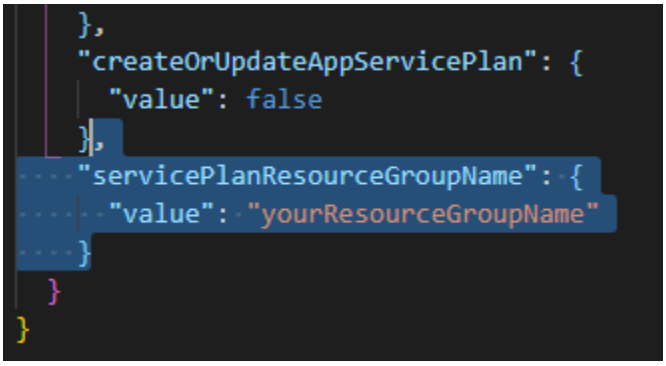


3: Set “createOrUpdateAppServicePlan” parameter

Next, just below the “appServicePlanName” parameter be sure to set the parameter called “**createOrUpdateAppServicePlan**” to **false**

4: [Optional] Create "servicePlanResourceGroupName" parameter

In the event the App Service Plan you intend to use is located in a different resource group than the one you are deploying the ARM template against, you need to add a parameter to inform the ARM template what resource group your App Service Plan is in. Create the "servicePlanResourceGroupName" parameter as shown below and define the value as **the name of the resource group your App Service Plan exists in.**



5: Save azuredeploy.parameters.json and continue deployment

Save and close the file. You can now proceed with the deployment process outlined in Section 3.3.2: Rule Execution App Service for Salesforce as normal; your rule execution app service will now deploy to the App Service Plan you defined in the steps above

Azure App Insights Overview

For improved logging capabilities, the Salesforce Rule Execution Service is configured to use an Azure App Insights resource as a logging sink in addition to the logging the App Service itself already has. ARM Template deployment process outlined in Section 3.3.2: Rule Execution App Service for Salesforce will, by default, automatically deploy an App Insights resource for you.

The app insights resource will aggregate all the logs generated from the execution service. These logs can be tremendously useful for debugging any issues encountered with the Rule Execution Service. Depending on the level of event logging configured for the Rule Execution Service Event Log, these logs can add insight into rule executions, entity loading, overall execution timing, and any errors encountered during rule execution.

App Insights in non-Standard Azure Instances (Government Cloud)

If this is your first time deploying the arm template and you would like for the template to create the app insights resource for you, then no other configuration is required.

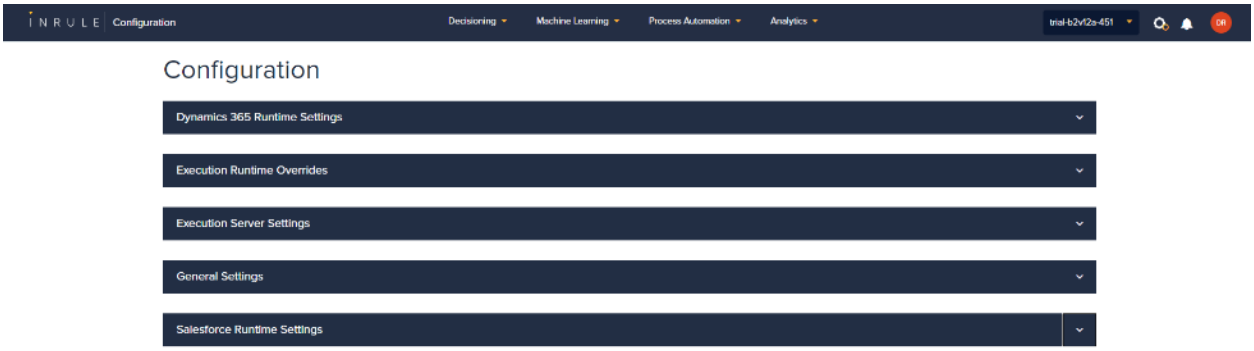
However, if you would like to use a pre-existing app insights resource then you need to set the **appInsightsInstrumentationKey** and the **appInsightsConnectionString** for that resource in the azuredeploy.parameters.json. Then proceed with the rest of the deployment steps outlines in Section 3.3.2: Rule Execution App Service for Salesforce

Appendix G: InRule SaaS Portal Configuration

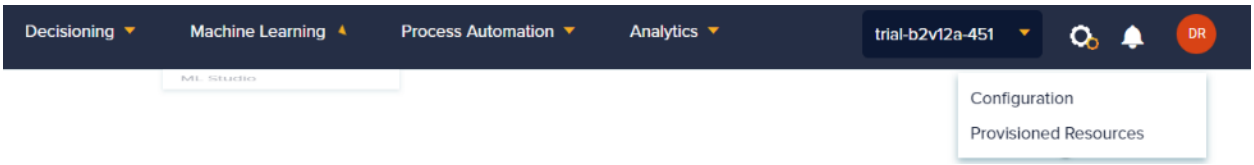
InRule SaaS offers customers the most streamlined deployment process, eliminating the need to install and manage the InRule App Services in Azure, as InRule will manage the deployment of the Rule Execution Service for Salesforce for you. SaaS customers are also able to use their SaaS portal to access the information needed for configuring the InRule App for Salesforce and provide Service Account credentials for the SaaS-hosted Rule Execution Service to connect to a Salesforce environment.

Access Solution Configuration Information

SaaS Customers can find the information needed for configuring the InRule App for Salesforce through the Configuration page of their SaaS portal. This page will have a section for each Rule Execution Service instance hosted for the customer by InRule. Please note that only the Salesforce Runtime Settings are currently used by the Salesforce Execution Service. In a future release, settings from the Execution Server Settings section will also contribute to the Salesforce Execution Service. In the interim, if you need any of those settings changed, please reach out to InRule Support.



Additional configuration information can be located on the Provision Resources page available from the cogwheel icon in the top right corner of the SaaS portal.



The Provision Resources screen is where you can find read-only configuration information such as the Rule Execution Service URL and the Rule Execution Service API Key.

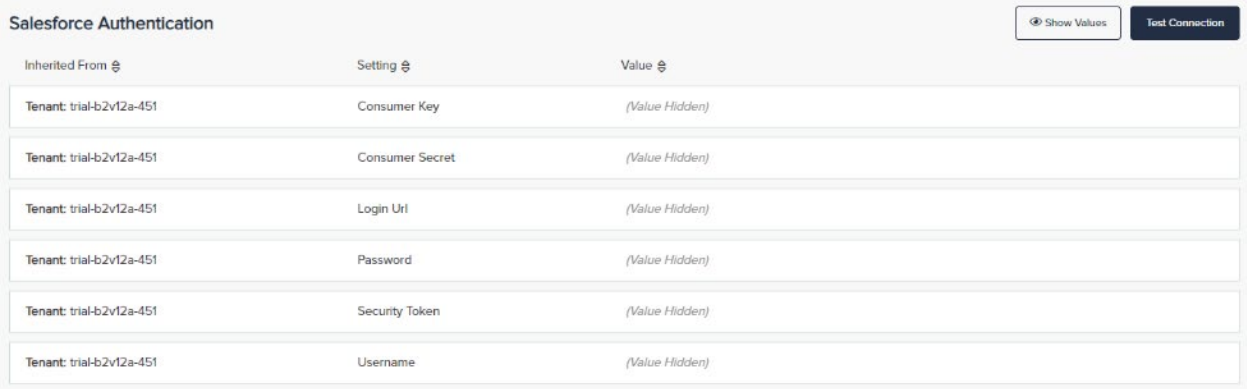
Providing Connection Information

Clicking the drop-down arrow for a Salesforce section will allow you to provide connection information that will allow the SaaS-hosted Rule Execution Service to interact with your Salesforce environment. You must configure authentication information for both Salesforce and for your Rule Execution Service

Salesforce Authentication

In order to authenticate to Salesforce, you must configure the appropriate login URL for your Salesforce environment, the service account's username, password, and security token, and the consumer key and

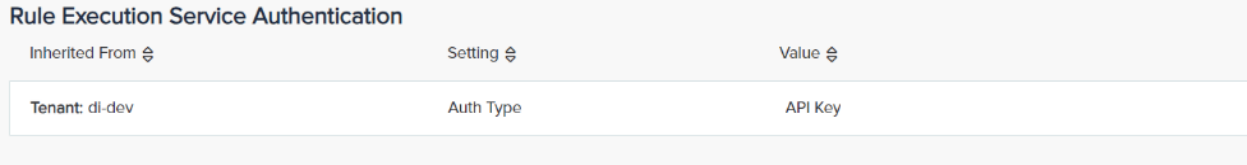
consumer secret associated with the Connected App you created as a part of Section 3.3.3: Configure the InRule App.



Rule Execution Service Authentication

This configuration section defines the authentication type that will be used by Salesforce to authenticate to the Rule Execution Service. Currently, the supported authentication types are API Key and a legacy username/password basic authorization. Support for the basic auth configuration will be discontinued at an unspecified date in the future. If using the basic auth configuration, you will additionally need to supply a value for the username and password settings.

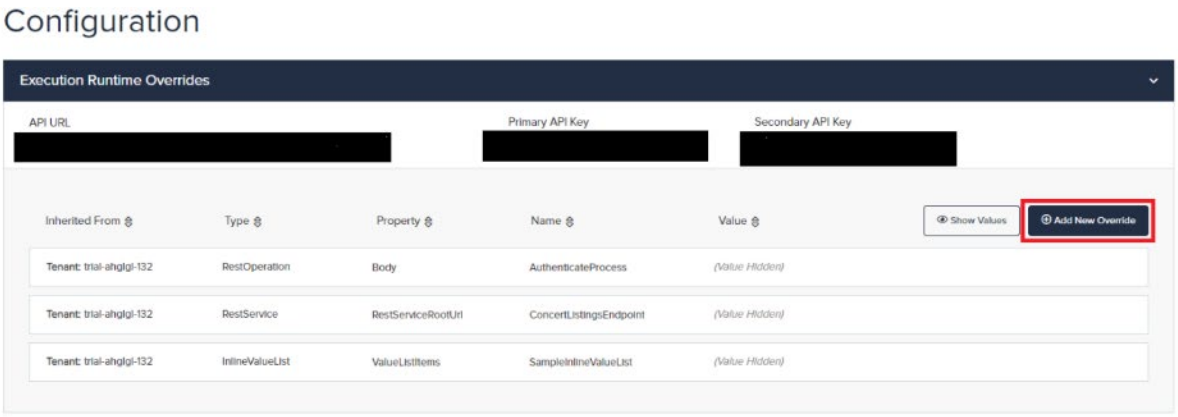
On the Salesforce side, the same credentials will need to be configured on the Named Credential described in Section 3.3.3: Configure the InRule App.



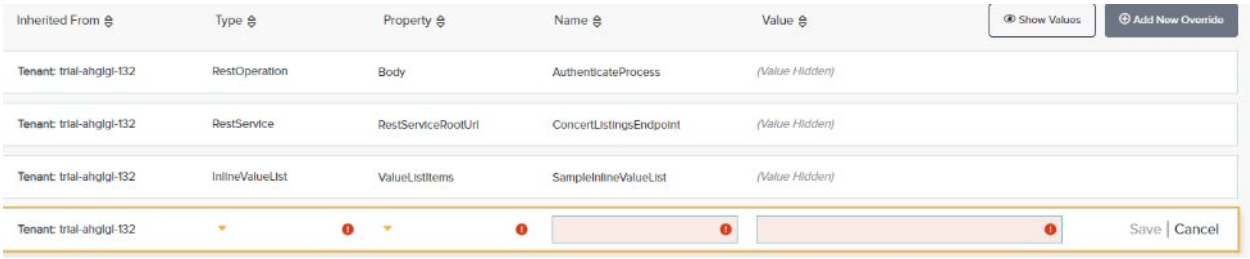
Endpoint Override Configuration (Optional)

As of version 5.5, InRule for Salesforce now supports Overriding Endpoint Configuration via Azure App Service App Settings. This allows for the overriding of various endpoint settings configured on a rule app, such as REST API URLs or Database Connection Strings, by setting App Settings on your execution service App Service.

To set an endpoint override on your app service, simply navigate to your rule execution app service, go to the Configuration view, and click the drop-down arrow for Execution Runtime Overrides.



Click the Add New Override button:

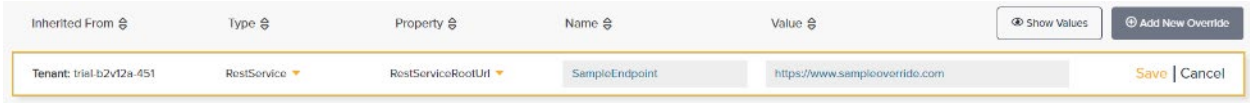


Click the drop-down arrow under the Type column and select the corresponding type for your override. The Property column should auto-populate for you. Note that there is a drop-down arrow on the Property column as well – some Types have multiple properties for you to choose from.

Your endpoint name should match the name of the endpoint in the rule app you wish to override.

The value of the override would then be set to whatever value you wish to override with.

Below is what a properly configured end-result would look like, using an InlineValueList override as an example:



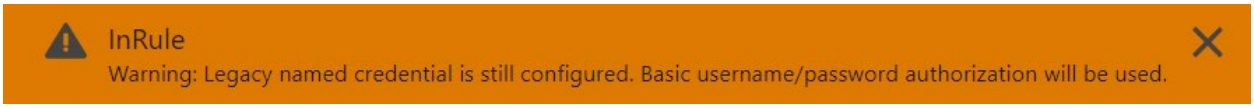
Once your override is set, simply save the changes to your app service. Upon the next execution of rules, the specified endpoint type will be overridden with the supplied value.

Appendix H: Named Credential Configuration

Named Credentials are used by the Apex code to make secure HTTP requests to the Rule Execution Service. Currently, there are two types of Named Credentials available:

- **Named/External Credentials (new)** – These are created with the newly named credential process introduced when Salesforce renewed named credentials in 2023. This new process provides additional authentication options, more granular access control, and will be supported for the foreseeable future.
- **Legacy Named Credentials (old)** – These are named credentials that were created prior to Salesforce's renewal of the named credential process. While it is still possible to create legacy named credentials today, it is not recommended – Salesforce has stated they will be discontinuing them at an unspecified date in the future. For additional information regarding setting up a Legacy Named Credential, you can find those instructions here.

Important: If you have a legacy named credential set up from prior to the InRule 5.8.1 release, you will need to delete your legacy named credential before you will be able to take advantage of the new Named/External credentials. A warning message will display to make you aware of this when testing your connectivity on the InRule application's configuration page.

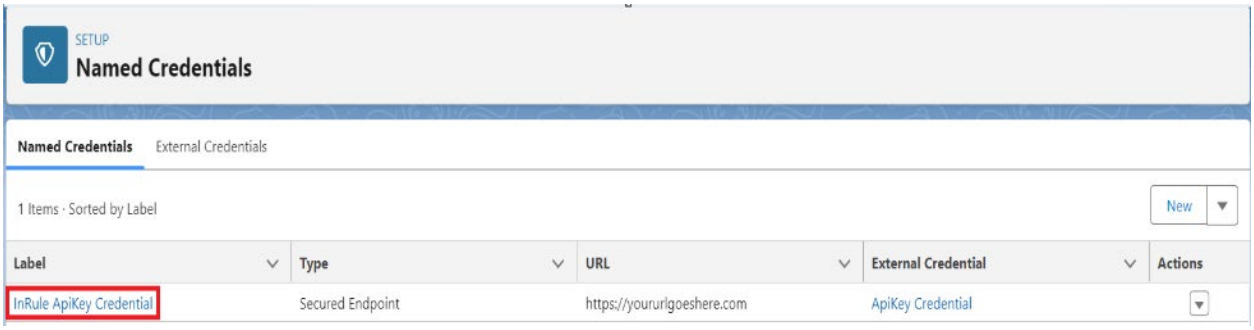


The instructions below will walk you through the process for setting up a Named/External Credential.

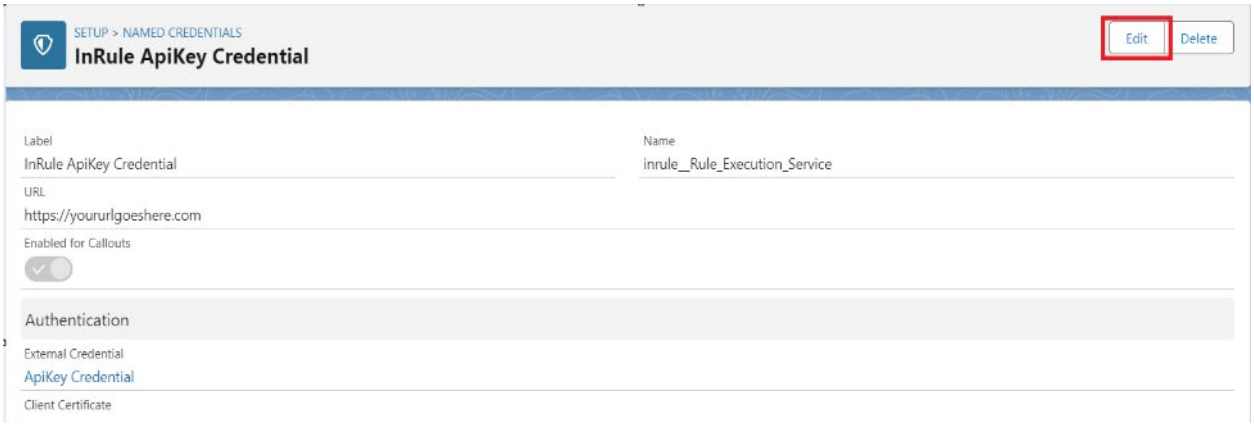
Setting up the Named/External Credentials

The InRule managed package includes pre-configured named and external credentials for your convenience. Before you can make use of them, you will need to follow the steps below to finish the setup using your user-specific information.

To begin, navigate to the Named Credentials settings page and locate the named credential called InRule ApiKey Credential.

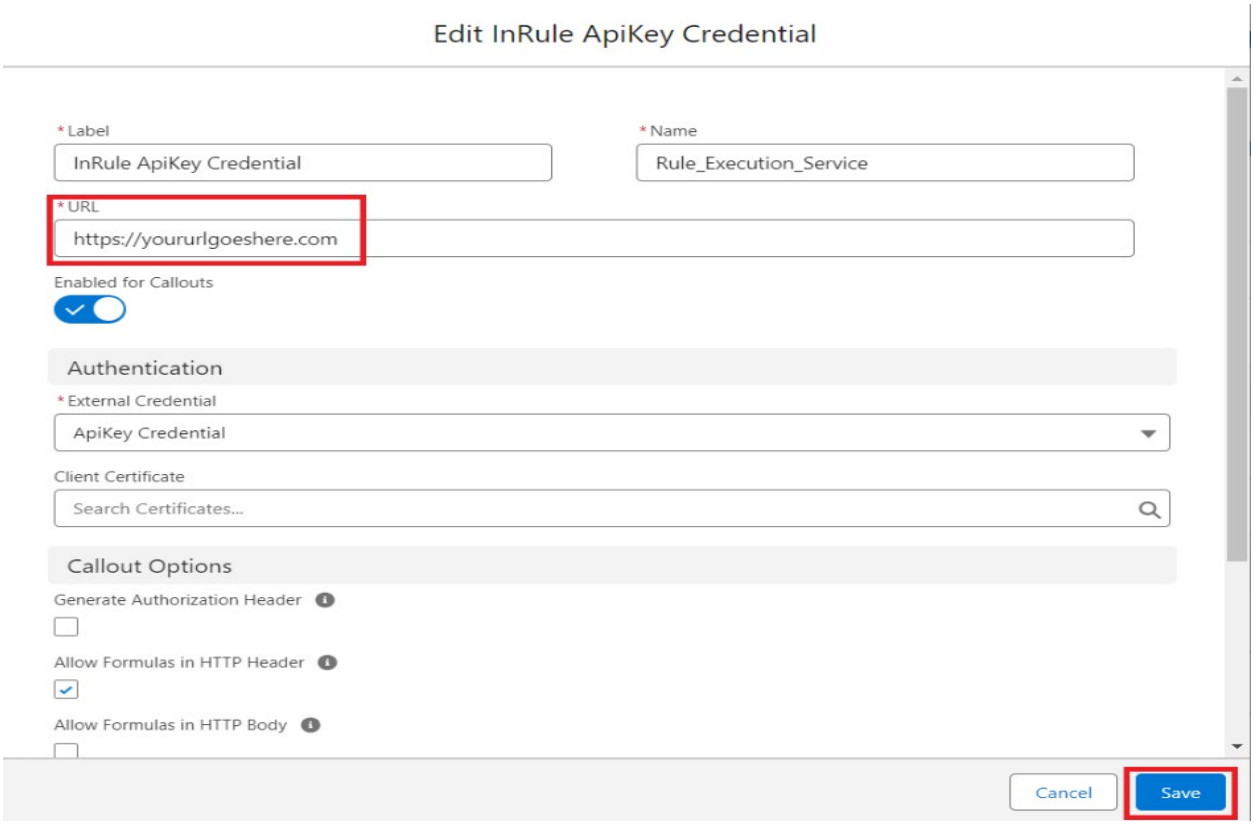


Click on the label to be taken to the settings page for this named credential. Click on the edit button in the top right corner.

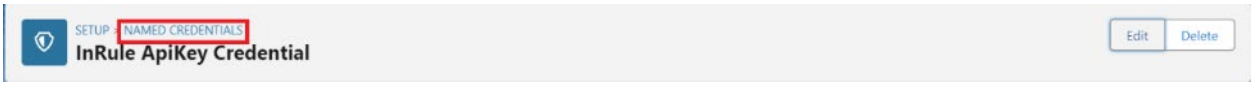


URL	URL for the rule execution service. • For self-hosted tenants, this can be found in the Azure portal in the overview section for the app service created earlier. For example, https://sampleservice.azurewebsites.net • For SaaS tenants, this can be found in the Provisioned Resources page available from the cogwheel icon in the top right corner of the InRule SaaS Portal.

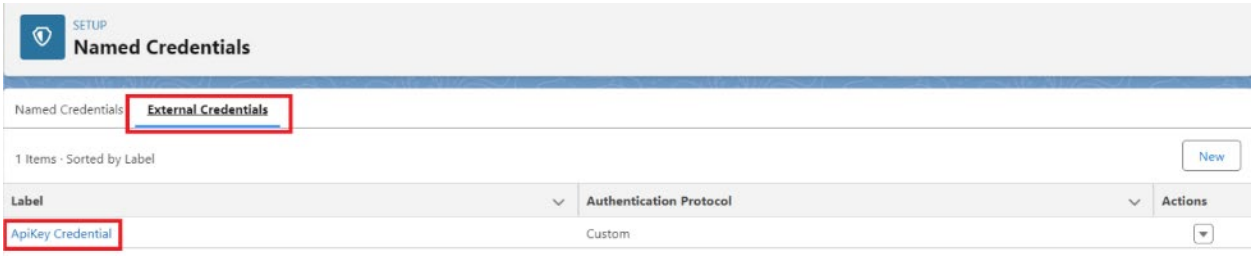
Replace the value in the **URL** field with the URL for the rule execution service. Save your changes when done.



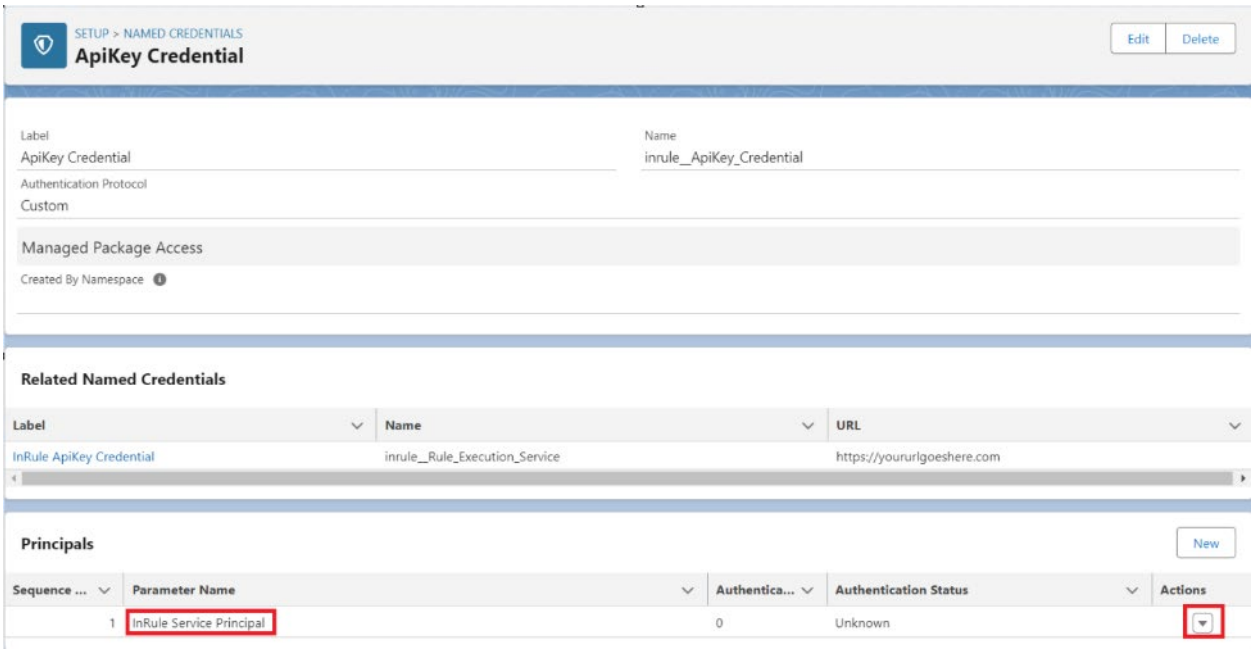
Navigate back to the named credential settings page by clicking the Named Credentials link at the top of the screen.



Once there, click on the External Credential tab and find the external credential named **ApiKey Credential**. Click on its' label to be taken to its' overview page.



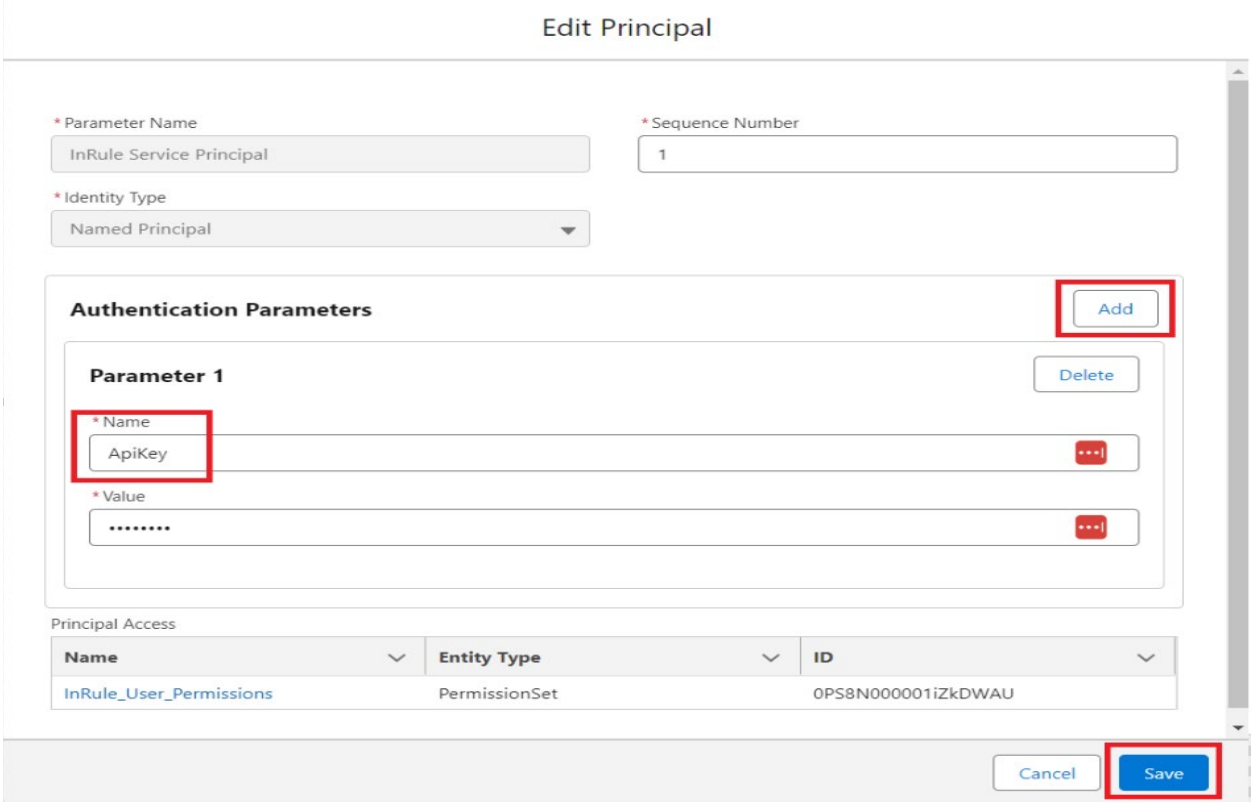
Locate the Principals section and find the principal named **InRule Service Principal**. Click the drop-down arrow on the right side of the screen and click the edit button.



Click the **Add** button next to the **Authentication Parameters** section to add a new authorization parameter. The **Name** field **must be set to ApiKey**.

API key	A unique key that allows users to verify their identity when interacting with the rule execution service. Your key can be located in one of two locations: • For self-hosted tenants, if you have not done so already, you will need to update your parameters as shown here – the api key configured will be the same one you use in Salesforce. • For SaaS tenants, your key will be located on the Provision Resources page available through the cogwheel icon in the top right corner of the InRule SaaS Portal.

The **Value** field will need to be set to your API key. Save your changes when done.

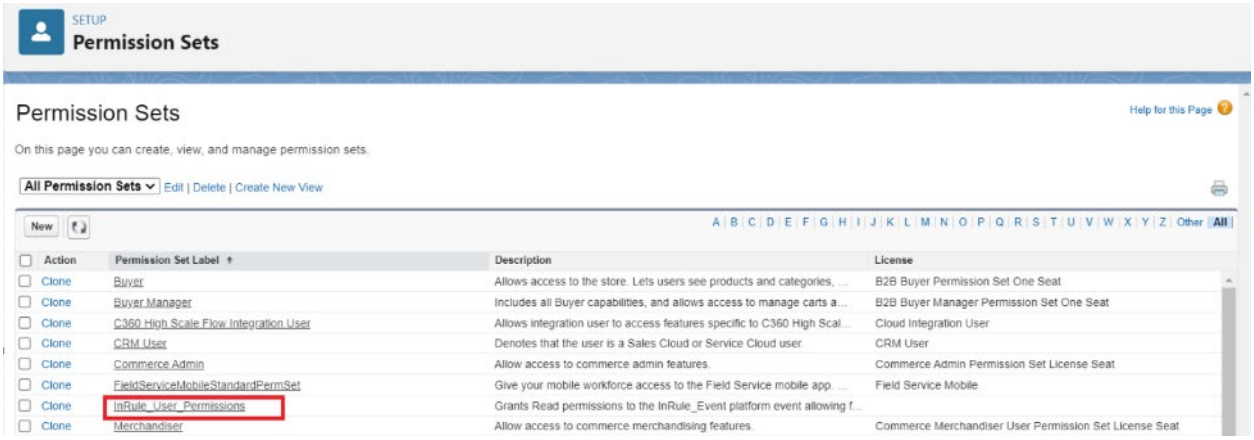


The named credential should now be configured correctly. Before your users will be able to make use of the named/external credentials, you will have to give them access. This can be accomplished either by assigning a permission set, or by granting the user's profile access.

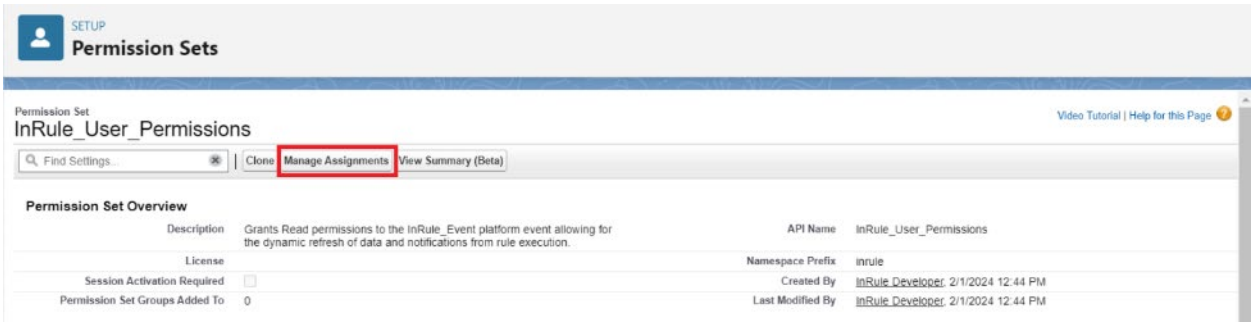
Providing Named Credential Access via Permission Set

In order to grant access via a permission set, you will need to assign the **InRule_User_Permissions** permission set to any user(s) you want to enable the named credential for.

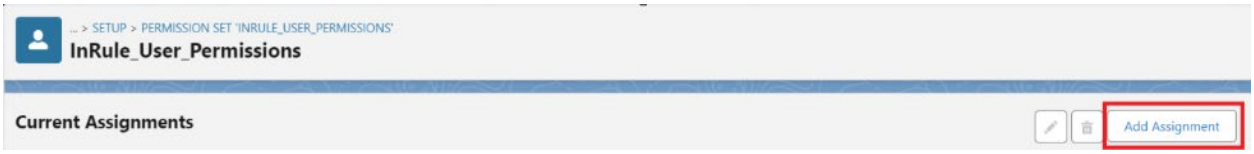
To begin, navigate to the Permission Sets setup page. Once there, locate the **InRule_User_Permissions** permission set in the list and click on its' label.



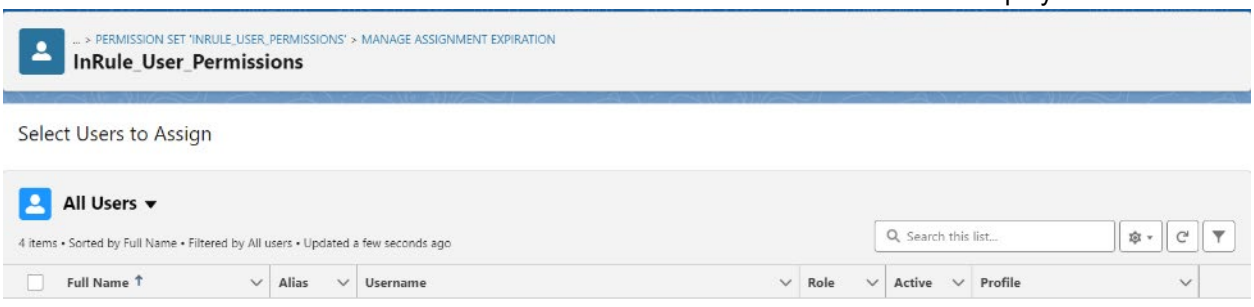
Once on the permission set's overview page, click on the Manage Assignments button above the permission set's description.



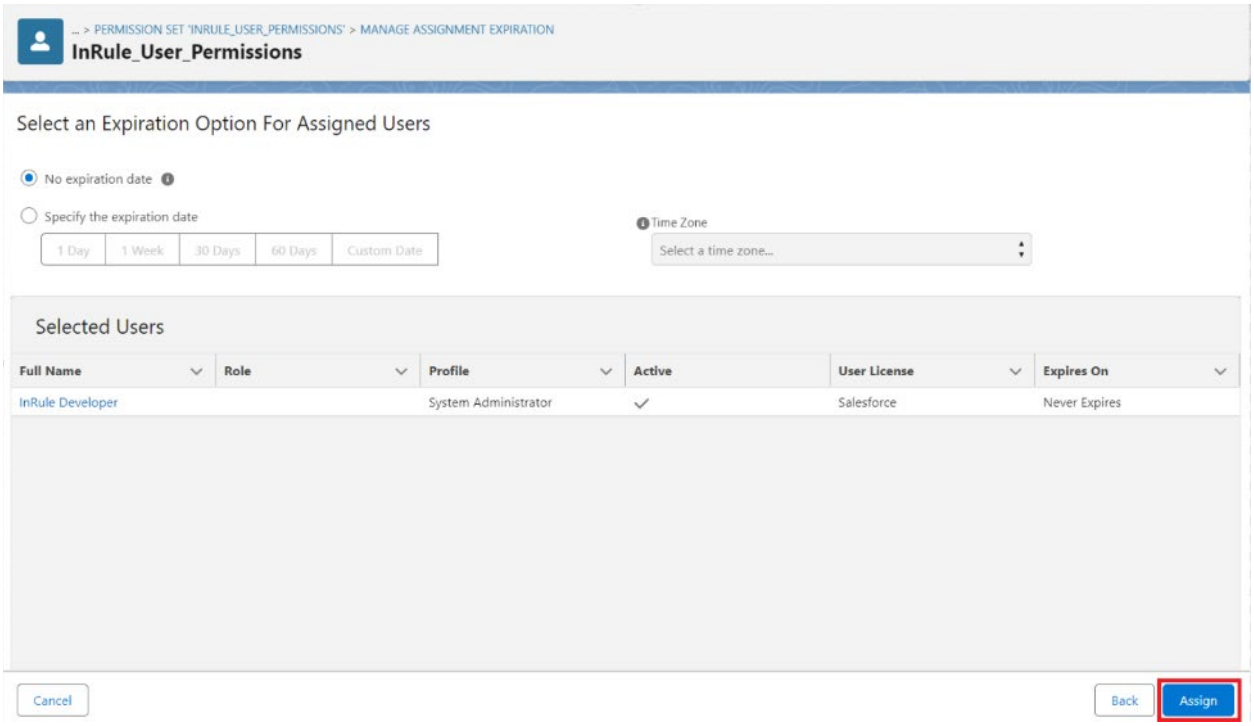
On the following screen, click the Add Assignment button on the right side.



From here, select any user(s) you wish to grant access to the named/external credential to.
Note: You can click the checkbox next to the Full Name column to select all users displayed.



Once done, select the **Next** button in the bottom right corner of the screen. You will be taken to the assignment expiration management screen. From here, you can optionally provide an expiration date for the permission set assignment. Once done, click the **Assignment button** in the bottom right corner.

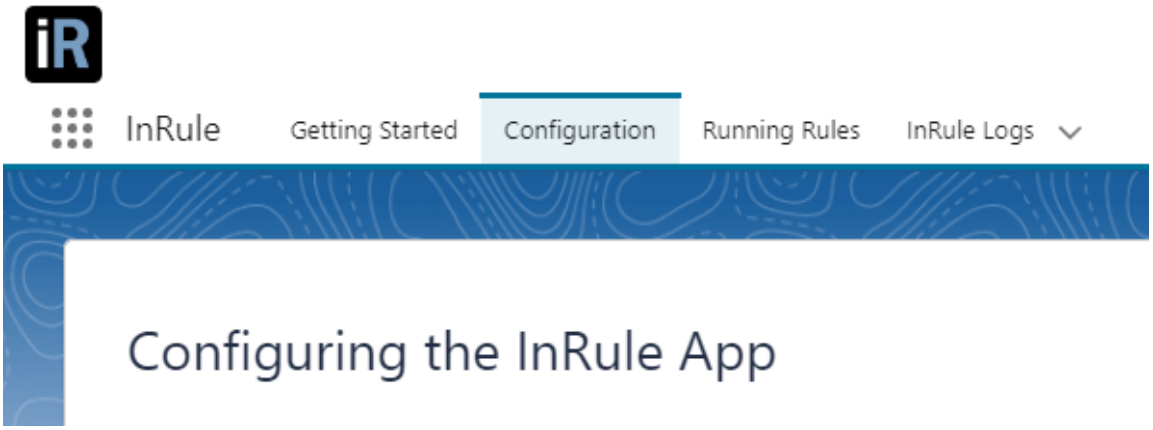


Your user(s) should now have access to the named/external credentials and should now have access to make authenticated callouts.

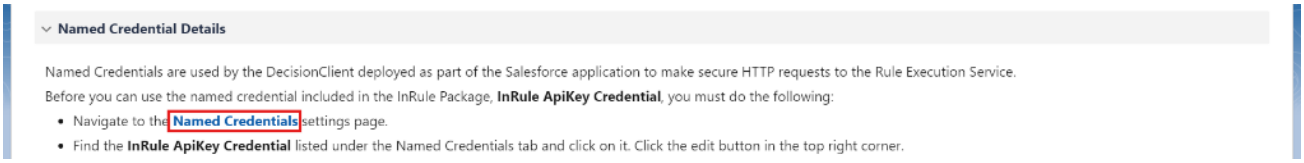
Legacy Named Credential Configuration

Legacy named credentials offer an older, less robust way of setting up authenticated callouts. They are used by the Apex code to make secure HTTP requests to the Rule Execution Service.

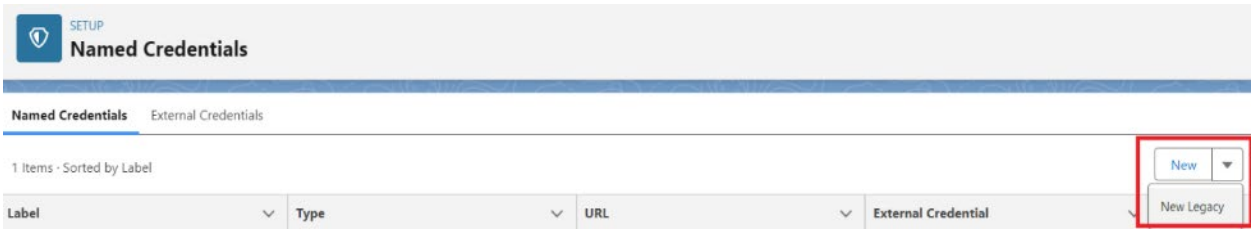
In the InRule App, select the “Configuration” tab

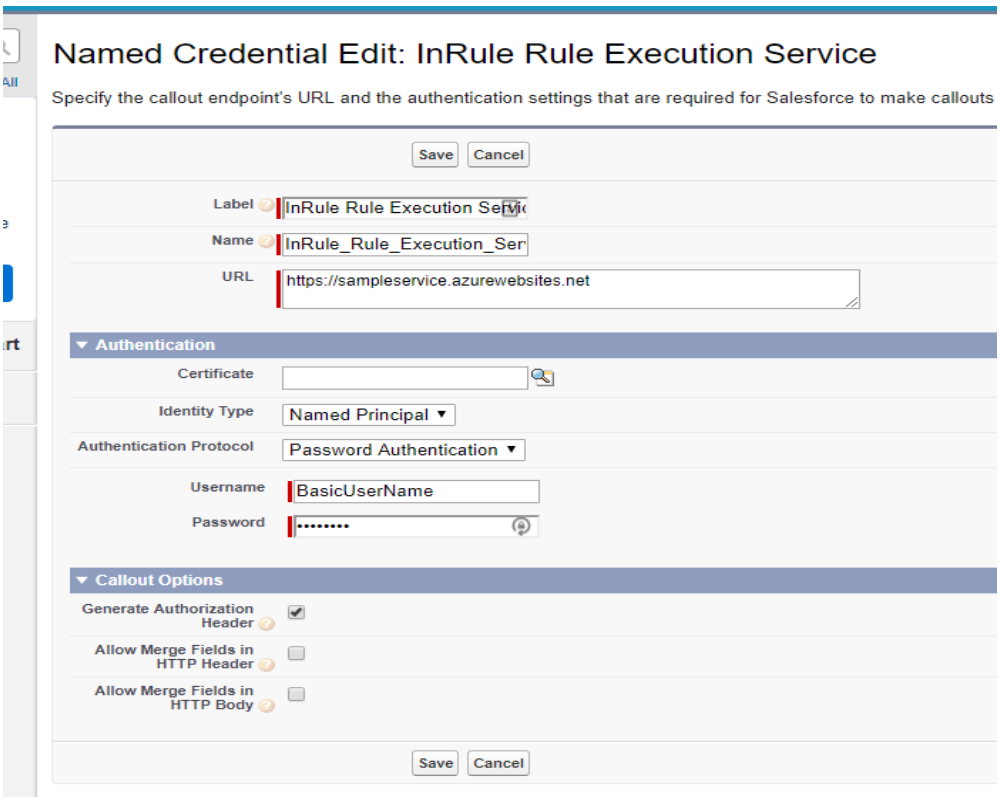


Under the “Named Credentials” header, click the “Named Credentials” link



Press the drop-down arrow on the “New” button and select “New Legacy”.





Configuration Form Fields	
Label	This should be set to InRule Rule Execution Service. This value is not required to match exactly, but if you choose something else, make sure to replace the autogenerated Name with the correct one
Name	This value must be set to **InRule_Rule_Execution_Service**. The name field is how the Named Credential is retrieved in the Apex code, so this needs to match exactly
URL	URL for the rule execution service in Azure. This can be found in the Azure portal in the overview section for the app service created earlier. For example, https://sampleservice.azurewebsites.net
Identity Type	Named Principal

Authentication Protocol	Password Authentication
Username	Username for accessing the rule execution service. This should be the same value chosen earlier for the 'ruleServiceUsername' parameter when setting up the service in Azure or established via the InRule SaaS Portal
Password	Password for accessing the rule execution service. This should be the same value chosen earlier for the 'ruleServicePassword' parameter when setting up the service in Azure or established via the InRule SaaS Portal
Callout Options	Make sure 'Generate Authorization Header' is checked

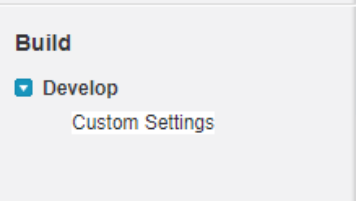
Appendix I: Salesforce and Rule Execution Service Event Logging

InRule Salesforce Logging

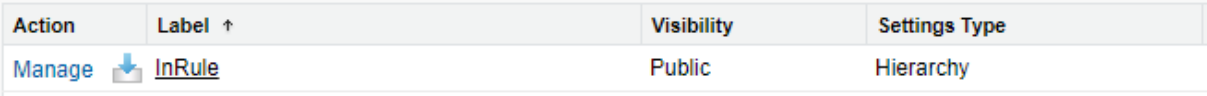
This section will highlight how to configure and view event logging for the InRule for Salesforce App.

Enable InRule for Salesforce App Logging

To enable logging from the InRule for Salesforce App, first navigate to **Setup** > **Develop > Custom Settings**.



Once you're in the Custom Settings page, locate the InRule custom settings object and select "Manage"



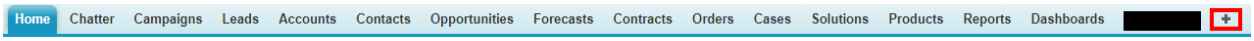
Select Edit. Here, you can configure the LoggingLevel custom setting that will define whether or not the InRule for Salesforce App logs to Salesforce and how verbose of logs it will create. The field can be configured with values from 0-3, with each signifying the following:

- 0 – Disables all logging
- 1 – Logs errors and a minimal amount of information on successful requests
- 2 – Logs errors, request information, rule engine notifications, and rule engine validations
- 3 -- Logs the same information as 2, but also includes JSON from the HTTP request and response payloads

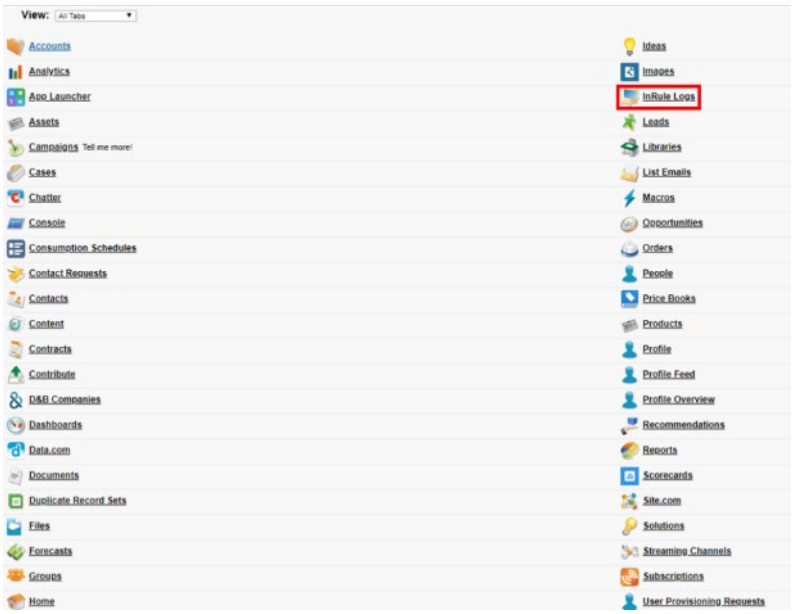
Once you have configured the LoggingLevel, select "Save."

Viewing InRule for Salesforce App Log

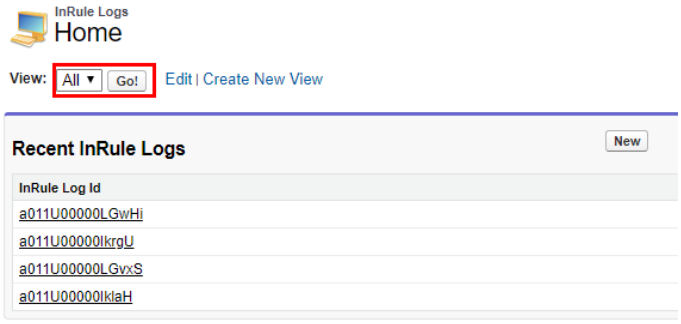
After installing the InRule for Salesforce App, a new custom tab to access the InRule for Salesforce App logs will have been added. To add it to your navigation bar, click the "+" at the end of the navigation bar.



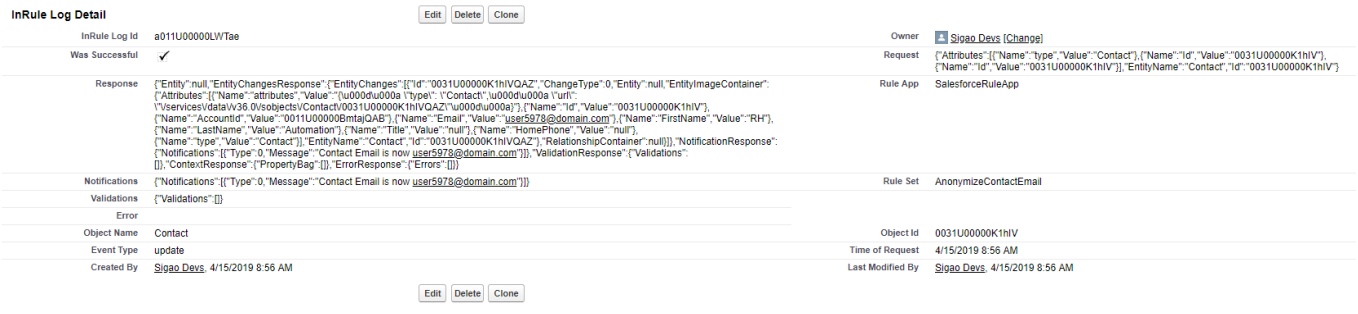
A list of objects will appear. Select "InRule Logs"



You will be navigated to a new page displaying a list of any recent logs that may have been created. To view a list of all logs, select “All” from the dropdown list at the top of the page and select “Go!”



You can now view all logs created by the InRule for Salesforce App. To view more information about an individual log, you can click on the log ID to view the log details



What fields are and aren’t populated is determined by what logging level you configured.

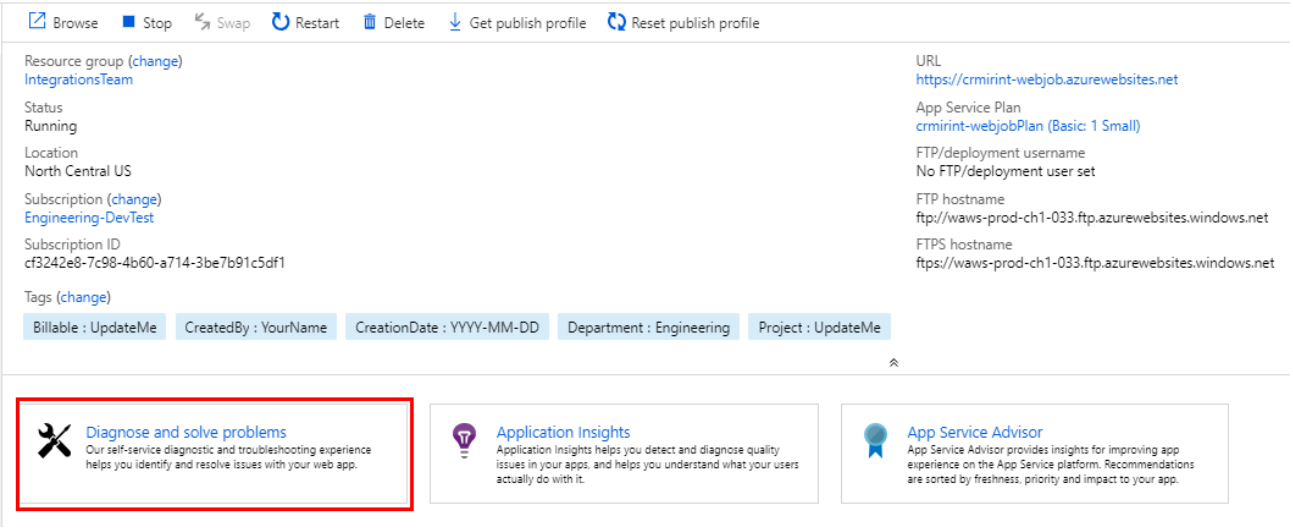
Rule Execution Service Event Log

Event logging can be enabled in the Rule Execution App Service to monitor application events. These logs can be tremendously useful for debugging any issues encountered with the Rule Execution Service.

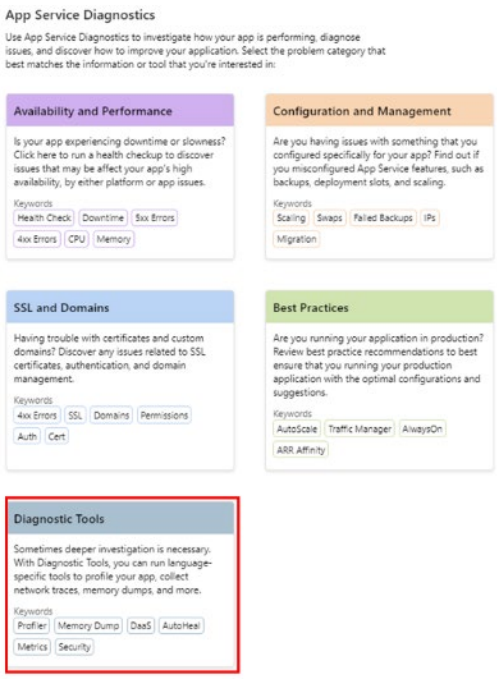
Viewing Application Event Logs

To enable event logging, login to Azure and navigate to your App Service as created as a part of the Azure deployment process detailed in Section 3.3.2: Rule Execution App Service for the Salesforce.

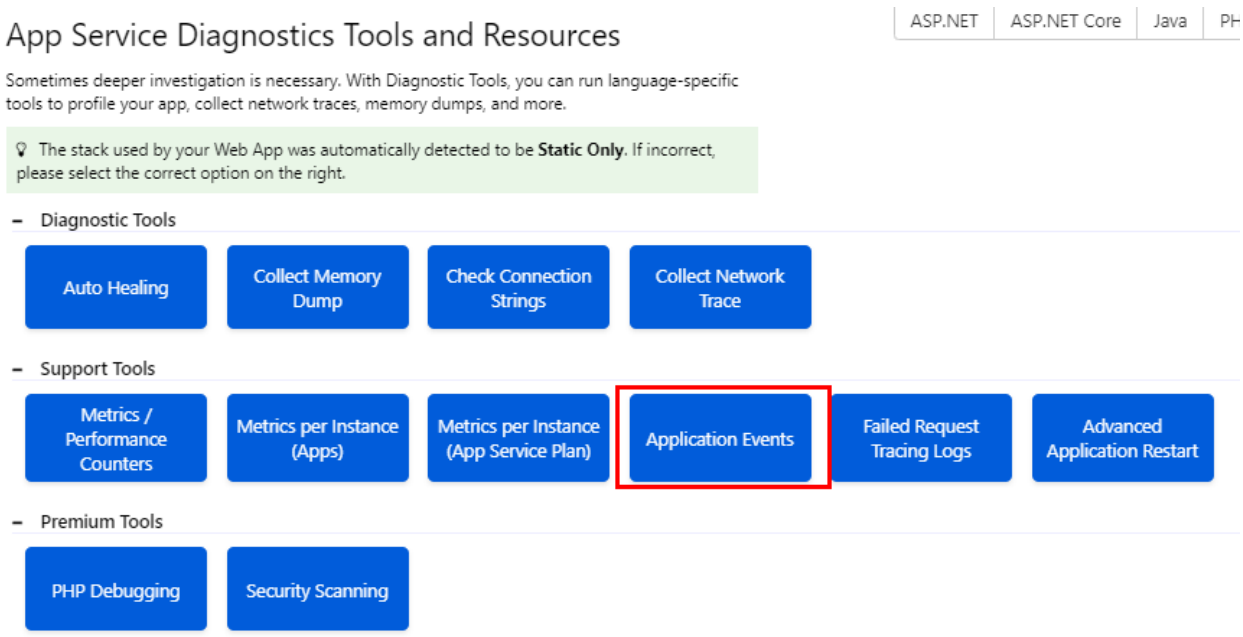
Once you're looking at the overview of your app service, select **Diagnose and solve problems**



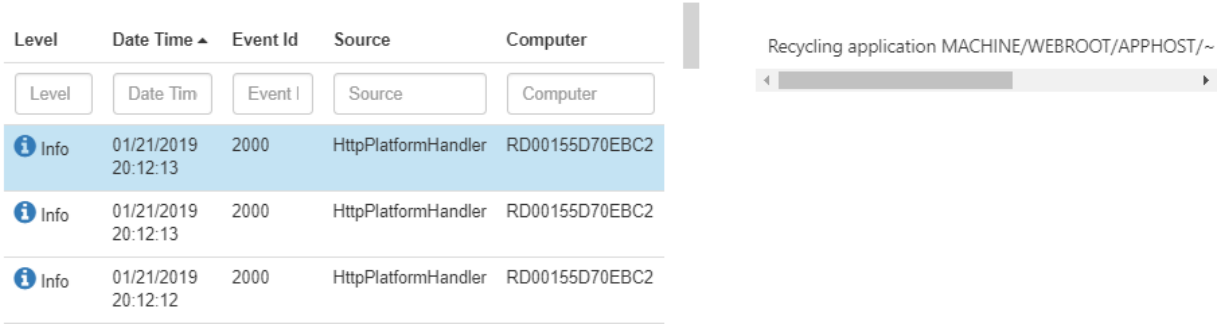
Select **Diagnostic Tools**



Select **Application Events**



You should see a list of all application events logged by the rule execution service, denoted with the notification level, timestamp, event ID, source and web server. Selected an event log will cause the log details to appear in a separate column on the right-hand side of screen.



In the event of rule service issues, error events in this log stream can be useful for debugging purposes. For example, below is an example of an error event log in an instance where the rule service contacting the catalog looking for a rule application that didn't exist:

```
Time: 10/30/2018 7:19:54.872 PM, Source: WcfCatalogProxy, Message:
Rule application 'DynamicsRules2' does not exist in the catalog.
Installer Version: 5.2.0.166
irSDK Version: 5.2.0.166
HostAppDomainHeapMemoryMB: 18
, Detail:

Operating System: Microsoft Windows NT 10.0.14393.0
Processor Count: 1
CLR Version: 4.0.30319.42000
CLR Mode: 32-bit
Thread Culture: 1033
ThreadID: 15
ProcessID: 0
Installer Version: 5.2.0.166
irSDK Version: 5.2.0.166

Error Information:
InRule.Repository.Service.InRuleCatalogException: Rule application 'DynamicsRules2' does not exist in
the catalog.
```

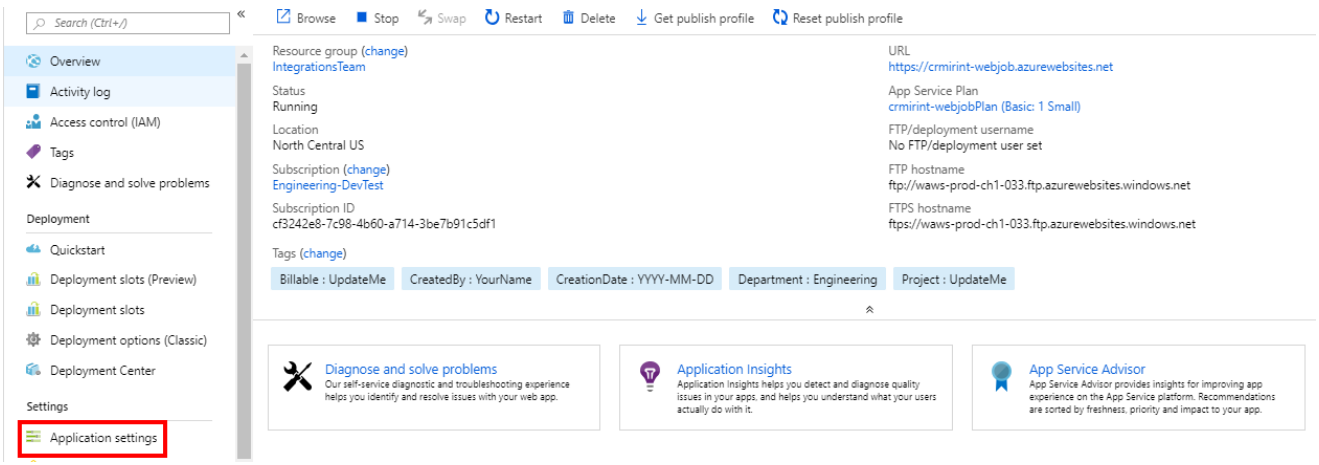
Typically, the response message at the beginning of the log and the Error Information section provides the most pertinent debugging information.

Adjusting Logging Levels

The Application Event Log can quickly become bogged down with too many logs, making finding specific logs that you may be interested in more difficult. To cut down on excessive informational logs, the Rule Execution Service, by default, will be deployed with a logging level of “Warn,” meaning only Warnings and Errors will be logged. However, this can be adjusted as needed for whatever your needs may be.

To adjust your Rule Execution Service’s logging level, navigate to your App Service as created as a part of the Azure deployment process detailed in Section 3.3.3: Rule Execution App Service for the Salesforce.

Once you’re looking at the overview of your app service, select **Application Settings** in the settings menu:



Scroll down until you see the **Application settings** section:

Name	Value
inrule:logging:level	Hidden value. Click show values button above to view
inrule:repository:licensing:licenseFolder	Hidden value. Click show values button above to view
inrule:sf:api:consumerKey	Hidden value. Click show values button above to view
inrule:sf:api:consumerSecret	Hidden value. Click show values button above to view
inrule:sf:api:loginUrl	Hidden value. Click show values button above to view
inrule:sf:api:password	Hidden value. Click show values button above to view
inrule:sf:api:securityToken	Hidden value. Click show values button above to view
inrule:sf:api:username	Hidden value. Click show values button above to view
inrule:sf:catalog:label	Hidden value. Click show values button above to view
inrule:sf:catalog:password	Hidden value. Click show values button above to view
inrule:sf:catalog:ruleAppDirectory	Hidden value. Click show values button above to view
inrule:sf:catalog:sso	Hidden value. Click show values button above to view
inrule:sf:catalog:uri	Hidden value. Click show values button above to view
inrule:sf:catalog:useInRuleCatalog	Hidden value. Click show values button above to view
inrule:sf:catalog:user	Hidden value. Click show values button above to view
inrule:sf:ruleService:basicPwd	Hidden value. Click show values button above to view
inrule:sf:ruleService:basicRealm	Hidden value. Click show values button above to view
inrule:sf:ruleService:basicUserName	Hidden value. Click show values button above to view

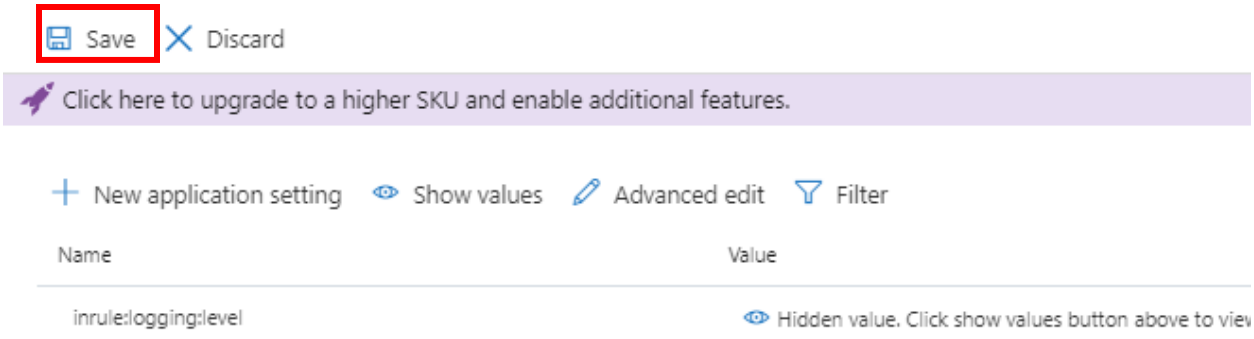
Locate the **inrule:logging:level** setting. Note that its value is currently set to “Warn.”

inrule:logging:level	Warn

Simply change the value to the desired logging level. You may select from one of the following levels that the Rule Execution Service leverages:

Logging Level	**Details**
Info	Logs all application events, including Informational events that track the general flow of the application
Warn	Logs Warning and Error events. Warning events highlight abnormal or unexpected events in the application flow, but don’t otherwise cause application execution to stop
Error	Logs only Error events. Error events result in the halted execution of the application’s current activity due to a failure

Once you have configured the setting to the desired to level, press Save at the top of the page:



Entity Loading Log

Enabling Info level logging on the Execution Service can provide more detailed performance data. The plugin trace log will include some summary data about the entire rule execution run, but the execution service can log specifics about entity data loading and rule execution from the service. To enable the info level logging set the “inrule:logging:level” app setting to “Info”. For all installations this can be done by editing the web.config, but Azure-hosted services can also edit this setting via the Azure portal.

Setting this to “Info” will enable two sets of logs, the standard InRule SDK logs that appear for all InRule products, and the data loading logs that measure entity loading specific to the Salesforce integration. You can find all the details about the Event Logs on the InRule support site here. There are a variety of different events, but the ‘ExecuteRuleSet’ entry will contain some of the most important measurements, such as compile, execution and external call times.

However, these measurements only capture actual rule execution, and before rule execution happens the entity tree is loaded from Salesforce. The data loading log captures this information. When executing rules from Salesforce, only the root entity is sent from the plugin to the execution service, and the execution service will then use the rule app metadata to determine which relationships and entities need to be loaded. The execution service batches these requests for greater efficiency, but very large entity trees or network latency can still lead to long load times. The entity loading log breaks down the total loading time into each ‘batch’ of loading requests made, and lists the total load time and all entities loaded in the batch. This information can be used to identify where large numbers of entities are being loaded and how long particular batches take to retrieve from Salesforce.

Taken together, these logs can be used with the plugin trace log to get an end-to-end picture of performance, and help identify which steps consume the most time. An extract of a sample of the data loading log is shown below.

```
Entity Loading:
Total Loading Time: 911.5114ms
Total Org Service Time: 701ms
Root Entity: inr_projectprofile

Related Entities: inr_projectprofile -> account lookup <inr_account_inr_projectprofile> (1),
inr_projectprofile -> inr_agreement lookup
<inr_inr_agreement_inr_projectprofile_PrimaryAgreement> (1), inr_projectprofile ->
inr_projectprofilegroup lookup <inr_inr_projectprofilegroup_inr_profile> (1),
inr_projectprofile -> inr_agreement collection <inr_inr_projectprofile_inr_agreement> (2),
inr_projectprofile -> annotation collection <inr_projectprofile_Annotations> (33),
inr_projectprofile -> inr_businessapplicationtype collection
<inr_inr_businessapplicationtype_inr_projectpr> (1), inr_projectprofile ->
inr_integrationdetail collection <inr_inr_integrationdetail_inr_projectprofile> (5),
```

```
inr_projectprofile -> inr_runtimeapplicationtype collection
<inr_inr_runtimeapplicationtype_inr_projectpro> (4), Time: 180.1637ms
Related Entities: account -> incident collection <incident_customer_accounts> (43),
inr_agreement -> serviceappointment collection <inr_agreement_ServiceAppointments> (69), Time:
128.366ms
Related Entities: incident -> subject lookup <subject_incidents> (8), serviceappointment ->
service lookup <service_service_appointments> (5), Time: 109.0635ms
```

In addition to always being logged on the execution service when Info logging is enabled, this log can also be returned in the response and logged to the plugin trace log for easier access.

Appendix J: New Releases and Upgrading Versions

In most cases, updating InRule for Salesforce is a relatively straight forward process. You will effectively follow the same order of operations of the initial installation steps outlined in this project.

The components that will need to be upgraded are:

- irAuthor
- Azure Rule Execution Service
- InRule for Salesforce App

All 3 of these components must be upgraded in all upgrade scenarios, and all three must be upgraded to the same version.

You can review the InRule Version Release Notes here. If you're upgrading a version that's not the latest, keep in mind that you'll need to use a versioned package link for the InRule for Salesforce App – App Exchange only offers the latest version. This distinction is discussed in more detail in Section 3.3.3: InRule for Salesforce App. The link provided in this section is for the version matching the version of this document. If you need an older version than this document's version, **do not** use the link located in Section 3.3.3. You'll need the Deployment Guide for that version.

This appendix discusses some special cases and considerations to be aware of when upgrading.

Release History

InRule for Salesforce Version Matrix

Version	Status	Release date	Description
v5.8.1	UNRELEASED	May 31, 2024	New auth type allowing API key-based named credentials, irAuthor Salesforce login fixes, RuleHelper fixes
v5.8.0	RELEASED	Nov 7, 2022	Maintenance Release
v5.7.3	RELEASED	Apr 7, 2022	Azure Marketplace listing, DecisionClient callable from REST, RuleHelper query and caching enhancements
v5.7.2	RELEASED	Jun 30, 2021	InRule for Salesforce configuration in SaaS Portal. Salesforce API limit fixes
v5.7.1	RELEASED		Maintenance Release
v5.7.0	RELEASED	Feb 17, 2021	Expanded execution options for Decisions and Version Labels, data loading performance improvements, Call InRule from Lightning Flow, advanced irX OAuth login flow
v5.6.0	RELEASED	Jun 23, 2020	InRule App on the AppExchange, InRule App content and configuration pages, 'Test Connectivity' button, DecisionClient enhancements
v5.5.1	RELEASED	Feb 18, 2020	Maintenance Release
v5.5.0	RELEASED	Dec 10, 2019	AppInsights logging, Execution App Service endpoint overrides
v5.4.1	RELEASED	Jun 18, 2019	Maintenance Release
v5.4.0	RELEASED	May 27, 2019	ARM Template, hosted InRule App package deployment
v5.3.0	RELEASED	Feb 26, 2019	Maintenance Release
v5.2.0	RELEASED	Oct 25, 2018	Rule Execution Azure App Service for Salesforce

Version Compatibility Considerations

The below list of items for upgrade consideration includes specific fixes or changes that may require migration or configuration action. Issue number and title are included here, but you can refer to the

release notes section of the support site for further information: Decision Platform Release Notes – InRule Technology. This page includes release notes for all InRule products, but Salesforce release notes will be prefixed with “SF”

Important: Again, when upgrading to a new version, irAuthor, InRule for Salesforce App, and the Azure Rule Execution Service should always be upgraded together

Issue Number	Version	Description
SFC-279	5.8.1	A new configuration setting, Auth Type, has been added. For existing users upgrading to 5.8.1, this setting will be set to basic username/password authorization. Authorization credentials configured in the Named Credential must now match the Auth Type configured for the Rule Execution Service.
SFC-278	5.8.1	Added support for the new Named/External Credentials introduced by recent Salesforce updates. Named Credentials that existed prior to these updates are now referred to as Legacy Named Credentials and should be deleted once the new credentials have been configured.
SFC-272	5.8.1	Added support for Webview2 to the irAuthor Salesforce login, resolving an issue with lightning-based login URLs such as https://your-username.lightning.force.com. Webview2 will now be used to render the login page if the runtime is present. If the runtime is not present, the legacy implementation will be used. A new warning log was added when the runtime is not detected.
SFC-261	5.7.3	Added ruleAppLabel parameter to execution service arm template. By default this label is empty. No ruleAppLabel will result in the most recent version of a rule app being used. Customers relying on a label to denote the production ready version of a rule app, such as the LIVE, will want to add the production label to the arm template prior to upgrade deployment.
SFC-48	5.7.0	Added improvements to the InRule Log Entity. Views will need to be manually updated to take advantage of these updates.
SFC-126	5.7.0	Added support for interactive login to irX, allowing login without requiring the Salesforce API token and the Consumer Secret/Key. This also enables support for MFA authentication.

SaaS Upgrades

For SaaS customers, the upgrade process will be communicated to you by InRule via email. Keep an eye out for these communications, as they will be the primary source of guidance around what to do in upgrade scenarios. Relevant information from these communications includes when an update will be occurring, the new features coming with the update, the maintenance times required to apply the update, and mandatory steps you’ll need to take immediately following the update. It is recommended to update via the Power Platform Admin.

Execution Service App Settings

When running an ARM template deployment to upgrade an existing app service, the new deployment will remove all app settings that currently exist and replace them with the settings during the new deployment. Because of this, it is important to save your parameters file when you do a deployment. If you do not have your previous parameters file you can find it in the deployments section of the Azure Portal. This also means that any settings that were changed or added manually, such as logging level or endpoint

overrides, will be removed.

Alternatively, current app setting values can be manually set in the ARM template parameter file prior to deployment to have the template deploy using those values. Should you go this route, be thorough during the transfer process, it is common for values to have been manually changed over time. For a thorough breakdown of the relevant parameters, reference Section 3.3.2: Rule Execution App Service for Salesforce

Lastly, redeploying or upgrading via the ARM template will remove your InRule license from the resulting app service. The license file will need to be manually added back; for a walkthrough on how to do this, reference Appendix K: License Management.

irX Salesforce Login Updates

As of v5.7, the irX Salesforce plugin now uses web login to Salesforce, rather than the former form-based authentication configuration. This allows for simply signing into Salesforce through Salesforce's native login experience, rather than needing to configure security tokens and consumer secrets/keys within irX. Additionally, this allows for login from accounts with Multi-Factor Authentication (MFA) required, whereas MFA-required accounts were not compatible before (Note: MFA support only applies to irX. Executive service accounts with MFA required are still not supported).

If upgrading from an older version, old usernames and environments will be preserved, but users will need to re-login.

Default Rule App Label Change

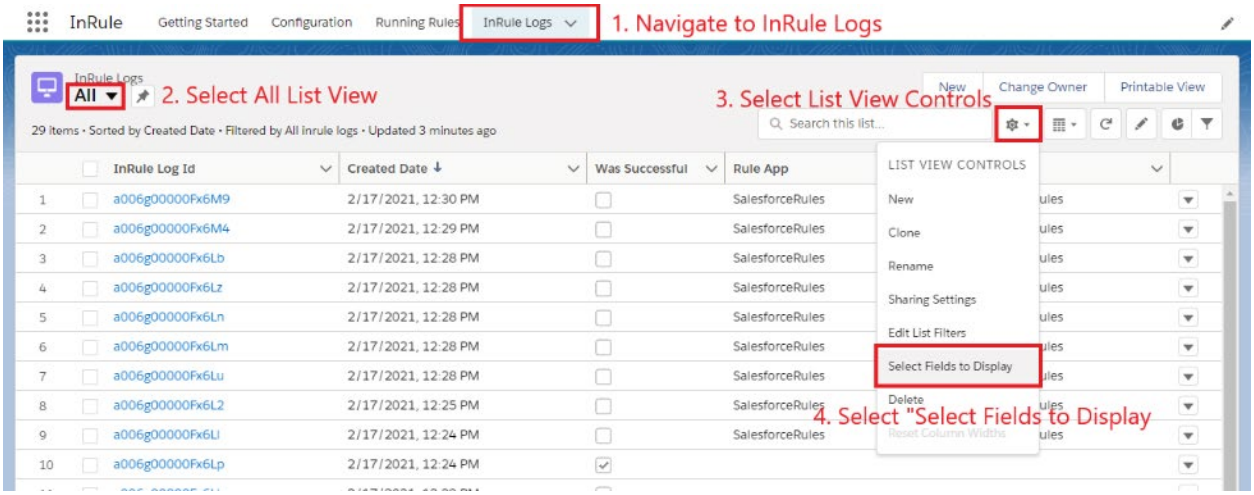
As of v5.7.3, the default Rule App Label configured on the execution service has been changed from "LIVE" to not being set with a value. If a default label is not configured on the execution service, it will now default to using the most recent version of the rule app being run. This has the potential to create an issue during upgrade scenarios where users are relying on the LIVE label to denote their production-ready rule. If using the most recent version of the given rule app is not the desired outcome, the default label on the execution service will need to be set back to LIVE (or whatever the appropriate production label for your existing rule apps may be). This can be done in the ARM template prior to upgrade deployment.

InRule Log Entity

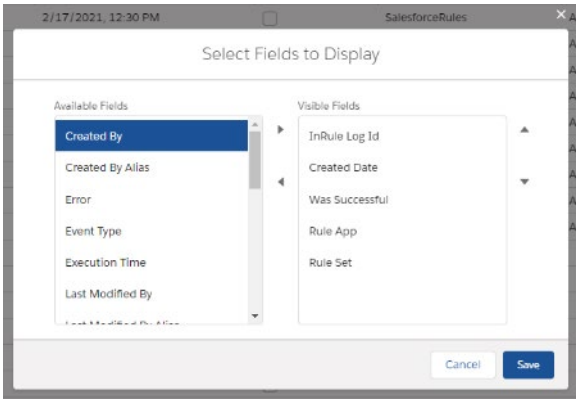
As of v5.7, updates have been made to improve the information displayed in InRule Logs. This includes updates made to the list views. Salesforce does not allow list views to be updated with a package update, so list view updates will need to be performed manually.

To update the "All" list view:

1. Navigate to the InRule Logs tab
2. Select the "All" list view
3. Select the List View Controls
4. Select the "Select Fields to Display" option



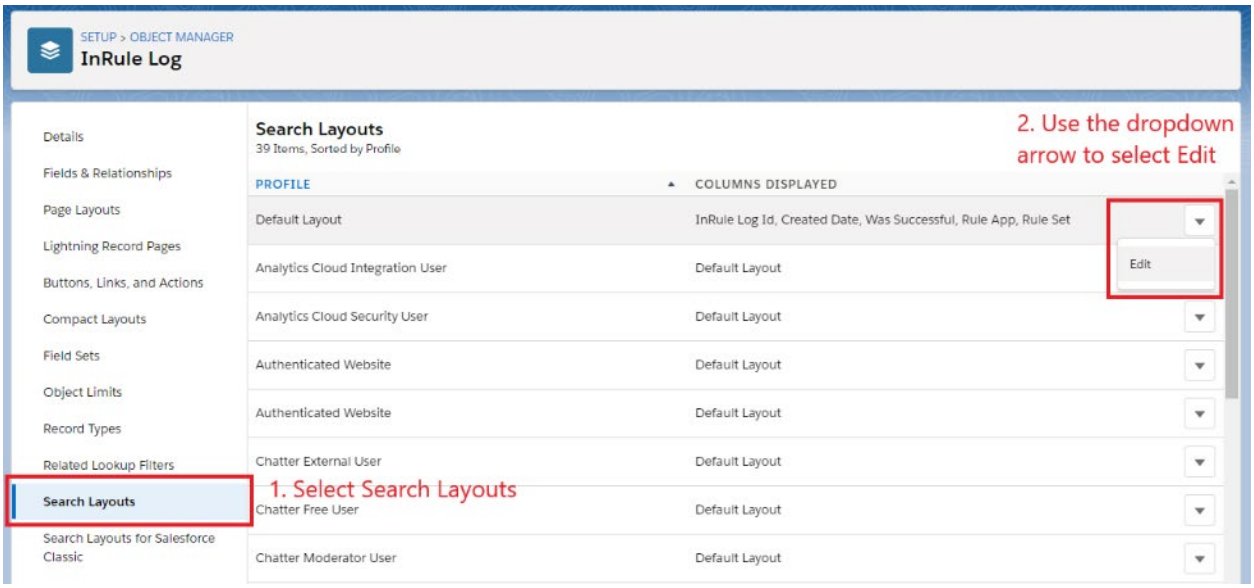
5. Use the “Select Fields to Display” window to add and order fields for the All view. Below is the recommended configuration for the “All” list view.



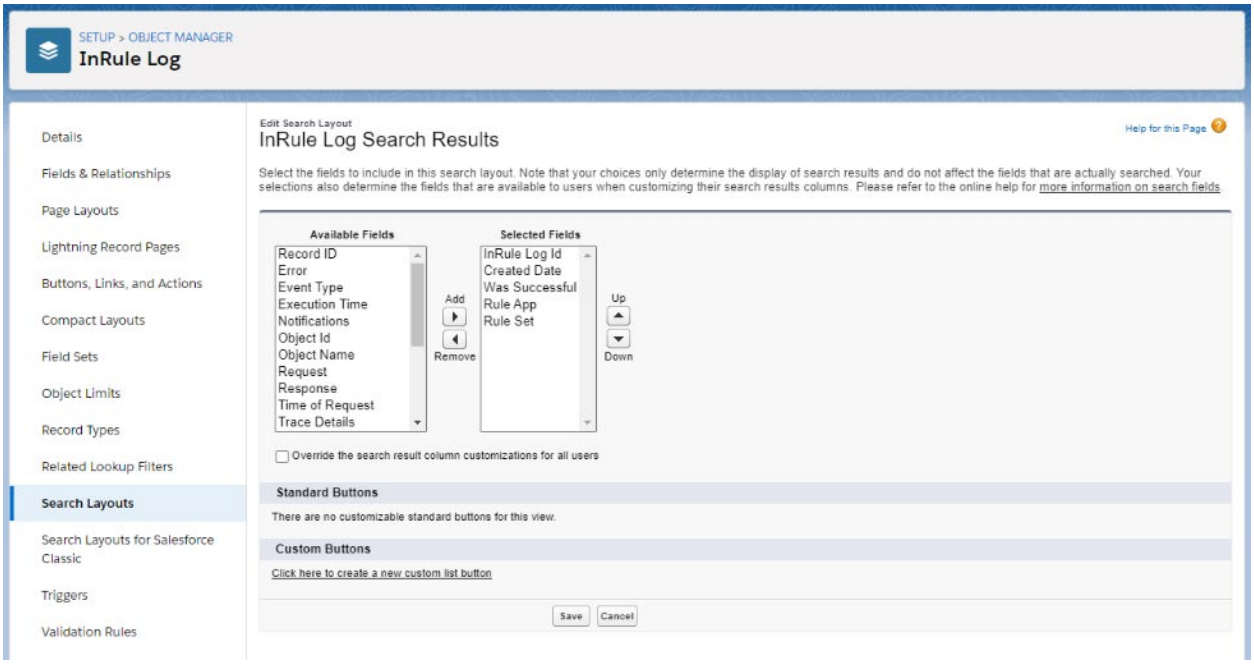
The “Recently Viewed” list view is controlled by the Search Layout. As such updating the “Recently Viewed” list view is done by updating the Search Layout for the InRule Log object in the Object Manager.

To update the “Recently Viewed” list view:

1. Select “Search Layouts” from the InRule Log Object Manager Screen
2. Use the dropdown arrow for the Default Layout to select the option to Edit the Default Layout



3. Use the “Edit Search Layout” page to add and order fields for the InRule Log Search Layout. Once the changes to the Search Layout are saved, it will update the “Recently Viewed” list view. Below is the recommended configuration for the InRule Log Search Layout.



Named Credential Changes

Salesforce has revamped their Named Credential process – existing Named Credentials are now referred to as Legacy Named Credentials and will be discontinued at an unspecified date in the future. Starting with InRule version 5.8.1, we have added support for the new Named Credential process to the InRule managed package. We recommend following the configuration steps here and then deleting any InRule specific named credentials that existed prior to upgrading to version 5.8.1.

Important: Failure to remove the existing legacy credential will result in continued usage of that credential, regardless of whether you have configured the new credential included in the package upgrade.

Important: Failure to configure the new credential puts you at risk of a breaking change being introduced by Salesforce when they deprecate support for Legacy Named Credentials.

Deleting the InRule Package

In some situations, it may be necessary to delete the InRule package from your organization to resolve an issue. Before deleting the package, ensure you have backed up any config or log data you want saved. Then you will need to remove all references to the InRule package from any customizations you have made, including things like forms and flows. If you attempt to delete the package without removing these customizations, Salesforce will list any remaining dependencies you need to remove.

Appendix K: License Management

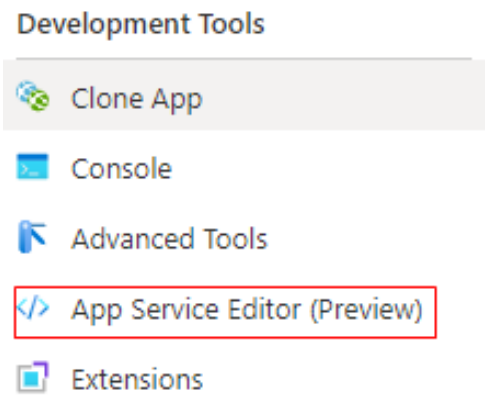
Licenses are required for both the Catalog Service and the Rule Execution Service. Whether you're deploying an Online or On-Prem service will determine the appropriate method for activating your InRule licenses. For Online installations, you will have an Azure license file provided to you by InRule which can uploaded either through Azure's App Service Editor tool or via FTP. Walkthroughs of both means are provided below.

For On-Prem services, you will need to leverage the InRule License Activation Utility. A walkthrough of this process can be found here.

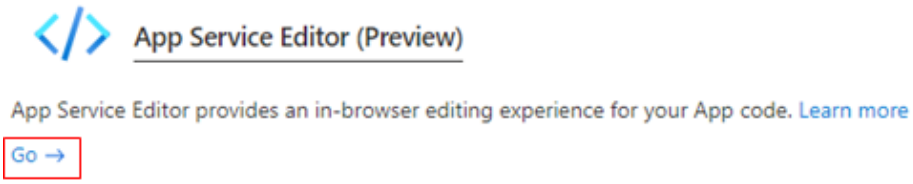
Uploading a License File

Azure App Service Editor

To upload the InRule license file to your execution service or your catalog service, navigate to the Azure portal and to the appropriate app service. On the left-hand nav-bar, scroll down until you find the App Service Editor option, under the Development Tools header:

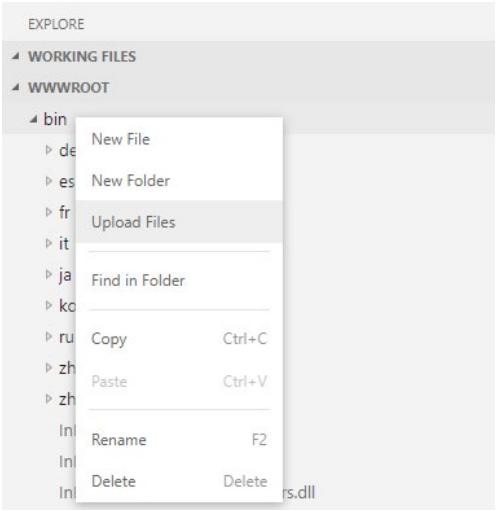
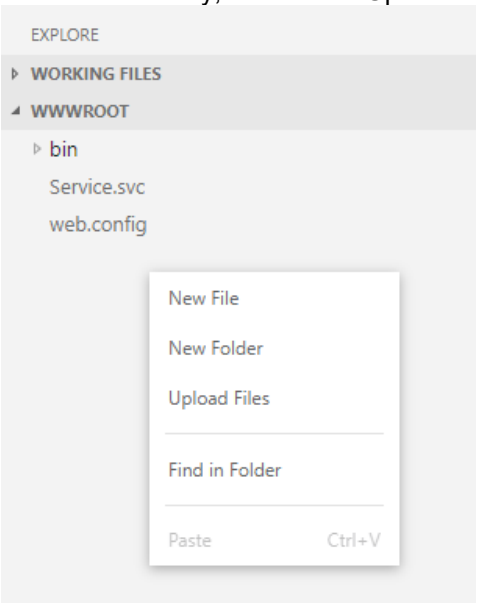


On the resulting page, press "Go"



From here, you'll need to navigate to the appropriate file location, which is different depending on if you're adding it for the Execution Service or the Catalog Service:

Execution Service	Catalog Service

Right-click on the bin folder on the left-hand side of the screen, then select Upload File:  Find your InRuleLicense.xml file wherever you have it stored and select Open to finish.	Right-click anywhere in the top level of the wwwroot directory, then select Upload File:  Find your InRuleLicense.xml file wherever you have it stored and select Open to finish.

With that, your license file should be uploaded and usable by your Rule Execution or Catalog Service.

FTP

This example leverages Azure CLI in addition to PowerShell commands. If you intend to use this method, please run the CLI from PowerShell.

Alternative approaches using PowerShell only should be possible if desired but are not detailed in this document.

First, retrieve the FTP deployment profile (url and credentials) with the az webapp deployment list-publishing-profiles command and put the values into a variable:

```
# Example: az webapp deployment list-publishing-profiles --name contoso-execution-prod-wa --resource-group inrule-prod-rg --query "[?contains(publishMethod, 'FTP')].{publishUrl:publishUrl,userName:userName,userPWD:userPWD}[0]" | ConvertFrom-Json -OutVariable creds | Out-Null

az webapp deployment list-publishing-profiles --name WEB_APP_NAME --resource-group RESOURCE_GROUP_NAME --query "[?contains(publishMethod, 'FTP')].{publishUrl:publishUrl,userName:userName,userPWD:userPWD}[0]" | ConvertFrom-Json -OutVariable creds | Out-Null
```

Then, upload the license file using those retrieved values:

```
# Example:
$client = New-Object System.Net.WebClient;$client.Credentials = New-Object System.Net.NetworkCredential($creds.userName,$creds.userPWD);$uri = New-Object System.Uri($creds.publishUrl + "/bin/InRuleLicense.xml");$client.UploadFile($uri,
```

```
"$pwd\InRuleLicense.xml");

$client = New-Object System.Net.WebClient;$client.Credentials = New-Object
System.Net.NetworkCredential($creds.userName,$creds.userPWD);$uri = New-Object
System.Uri($creds.publishUrl + "/bin/InRuleLicense.xml");$client.UploadFile($uri,
"LICENSE_FILE_ABSOLUTE_PATH")
```

Appendix L: Known Issues, Limitations and Troubleshooting

This portion of the document lists current known issues and limitations that may be encountered in usage of the Integration Framework. Most should only be encountered in limited edge cases.

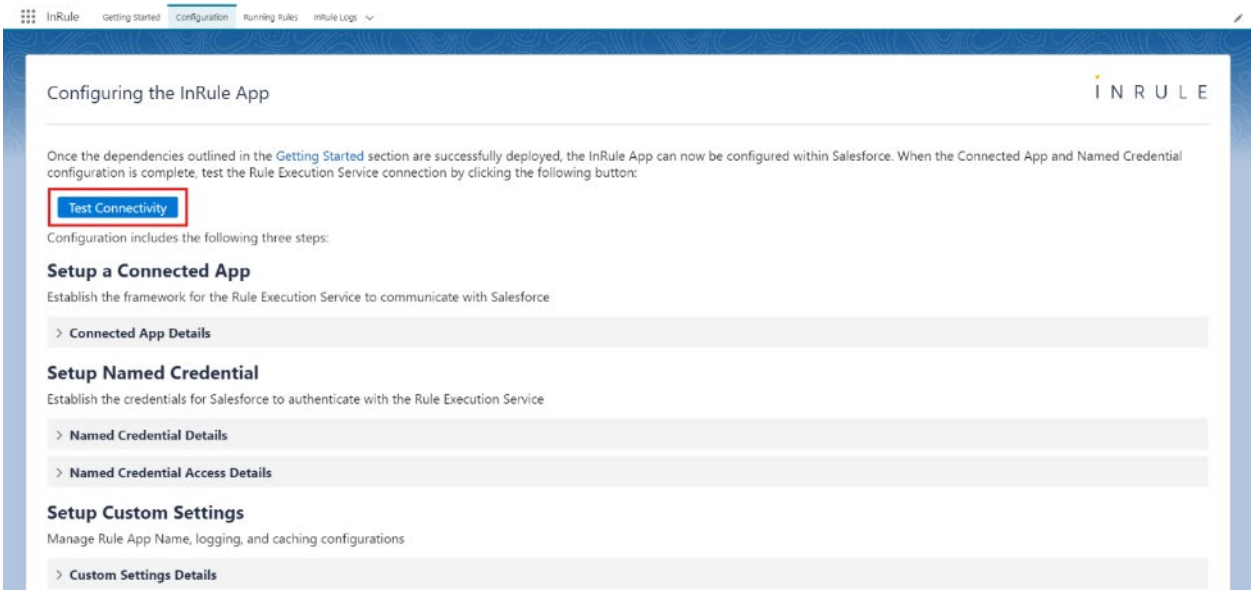
In general, when beginning to troubleshoot a given issue, it is critical that you verify that you are running matching versions of irAuthor, the Rule Execution Service, and the InRule for Salesforce App. Mismatched versioning can be the root of a plethora of unpredictable problems. For more information on this and upgrading InRule components, refer to Appendix J: New Releases and Upgrading Versions.

API Authentication to Salesforce

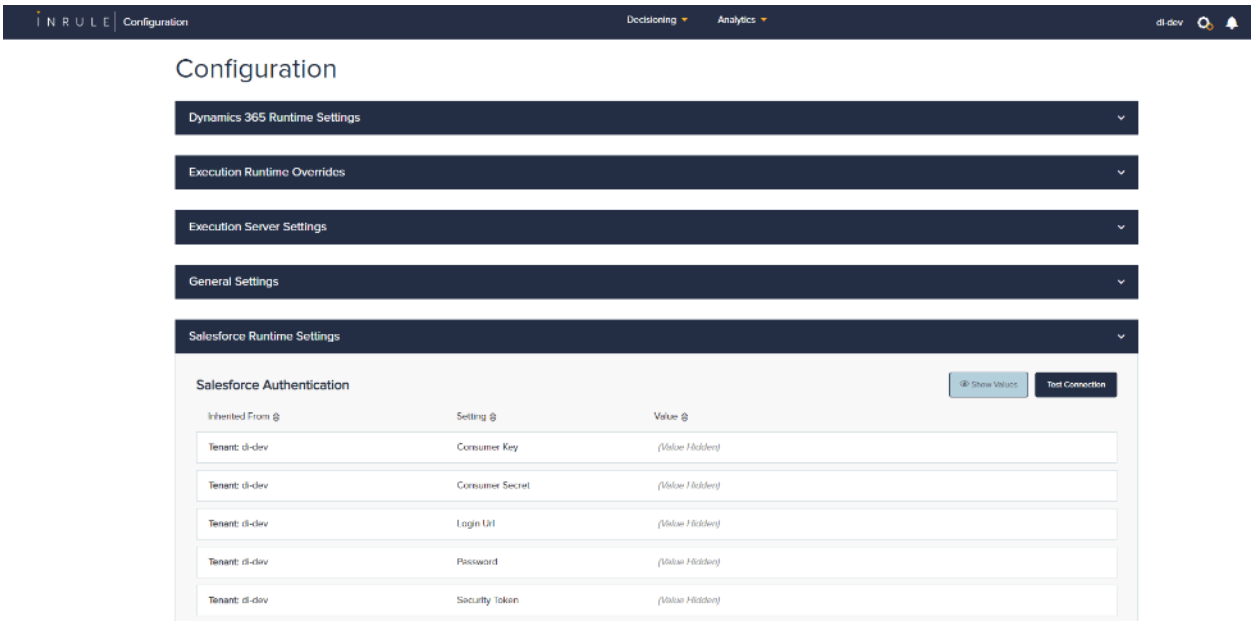
An authentication error response when attempting to login to Salesforce during rule execution is generally the result of a bad username, password, or security token. All three of these values must be properly configured to be able to authenticate.

You can test your ability to authenticate to Salesforce via API in a number of different ways, depending on your specific scenario:

Within Salesforce, you can navigate to the InRule Configuration page and leverage the Test Connectivity button. This button will verify your execution service's ability to authenticate to Salesforce and provide more specific error handling in the event of an issue.



For SaaS customers, a similar Test Connectivity button can be found in the SaaS portal's Salesforce configuration section



Alternatively, you can leverage Salesforce's available collections to test authentication via Postman. These collections can be found here.

Security Tokens

A common cause of authentication issues is confusion around how Salesforce handles API security tokens. **In most scenarios, security tokens are required.** Security tokens are only optional in scenarios when the account attempting to authenticate is connecting from a Trusted IP address. Trusted IPs can be set globally, meaning any user that connects from an IP in the trusted range will be regarded as trusted, or on the Profile level, meaning that only users assigned to the given Profile will be regarded as trusted when connecting from an IP that falls in the trusted range. If the account in question does not fall in a configured Trusted IP range, it **must** have a security token configured.

Multi-Factor Authentication

Currently, service accounts configured to require multi-factor authentication (MFA) are not supported by Salesforce's authentication endpoint. As a result, MFA cannot be set to "required" for your service accounts, or authentication will fail.

Apex Callout Timeout

When calling out to an external system from Apex via HTTP, the maximum total communication time limit set by Salesforce is 120 seconds. Executing rules involves a single HTTP callout, and the framework is configured to use all 120 seconds for the timeout. If the timeout is reached, you will receive a "Read Timeout Error" when running rules. This can occur because of latency between Salesforce and the rule execution service, or anything else that causes the rule execution service to take more than 2 minutes to respond.

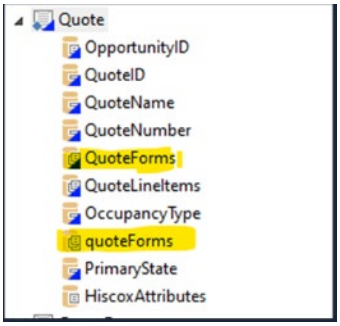
Salesforce API Limits

In order to maintain performance and availability, Salesforce has various limits on the number of requests that can be made to its API. Full documentation of these limitations can be found here, but as an example, in a Developer Edition instance, you are limited to 15,000 calls per 24-hour period. The rule

execution service makes requests via the API to Salesforce to load and save data, and these will count against any limits, but these requests are batched to be as efficient as possible.

Multiple Collections with the Same Name

There is a known issue that causes a null reference exception to be thrown when loading entities that have multiple collections of the same name on them, with the only difference being capitalization. An example of this scenario can be seen below:



This is currently not a supported scenario; all collections on a single given entity need to have wholly unique names; differences in capitalization do not adequately distinguish the two collections as unique from each other.

Application Insights Location Error

Application Insights resources are not available in every region, the list of supported regions can be found in Microsoft's Product Availability. By default, the ARM template will attempt to deploy the App Insights resource in the resource group specified for the template deployment. If this resource group is in one of the unsupported regions you will get the following error:

```
New-AzureRmResourceGroupDeployment : 5:26:53 PM - Resource Microsoft.Insights/components
'ConnorArmTestAppInsights' failed with message '{
  "error": {
    "code": "MissingRegistrationForLocation",
    "message": "The subscription is not registered for the resource type 'components' in the location
'northcentralus'. Please re-register for this provider in order to have access to this location."
  }
}'
```

To fix this error we will have to choose a specific region for the Application Insights resource in the ARM template parameters file.

1. **Locate InRule.Salesforce.Service.parameters.json**
   The ARM template parameters file is located in the *RuleExecutionAzureService* folder as, defined in Rule Execution App Service for Salesforce
2. **Create an "appInsightsLocation" parameter**
   Open the file in your text editor of choice. First, create the **appInsightsLocation"** parameter at the bottom of the parameters file. Set the value equal to a region where Application Insights resources are offered.

```
    "appInsightsResourceName": {
      "value": ""
    },
    "appInsightsLocation": {
      {
        "value": "SouthCentralUS"
      }
    }
}
```

3. **Save InRule.Salesforce.Service.parameters.json and continue deployment**
   Save and close the file. You can now proceed with the deployment process outlined in Rule Execution App Service for Salesforce as normal; your rule execution app service will now deploy to the App Service Plan you defined in the steps above

Performance

Azure Platform

The Rule Execution App Service for Salesforce can use either a 64-bit or 32-bit platform in Azure. As of v5.7.3, the configuring the platform will be managed in the Arm Template and will be based on the app service plan. Using the 64-bit platform requires a Basic or higher app service plan, so when a Basic or higher app service plan is used in the Arm Template, the Rule Execution App Service will be deployed with the 64-bit platform. Otherwise 32-bit is used.

Apex Trace Log

When troubleshooting performance issues, the InRule Log entity included with the package provides detailed metrics to help analyze the various components of rule execution performance. At a high level, the logs provide run times for the various execution steps. The 'Information' section of the log form includes a field for 'Execution Time', which is the total run time for the Apex rules method. In addition, the InRule log also provides a more detailed breakdown of steps in the trace text. These are broken into Execution Service and Save Time.

Execution Service time will typically be longer, as this includes communication latency over HTTP, loading of additional entity data, and execution of rules. Save time will not always show up in trace logs, but if any entity changes come back from rule execution the time needed to save them will be reflected here. The trace log will also include a message noting the number of entities returned from the execution service for saving. This saving should not take much time for a handful of entities but can take several seconds for large change sets.

For a further breakdown of the data loading and execution time from the rule execution service, the 'Info' level event logs for the App Service can be accessed through the Azure Portal.

Included below is a sample trace log that includes these features.

```
Starting rule execution
```

```
Sending request to rule execution service at
callout:InRule_Rule_Execution_Service/SfRuleExecution.svc/ExecuteRulesFromPlugin?eventType=upd
ate&id=0018A00000cPk1CQAS&entityName=Account&ruleAppName=SalesforceRules&ruleSetName=RuleSet1&
ruleAppLabel=&appDomainCache=0.0
Received response from rule execution service
Execution Service Time: 3,221ms

Saving 4 changes to Salesforce
Processing change -- Insert on entity ID 73c95b94-df07-44c6-8895-fcbc3c422a3c -- type Contact
Processing change -- Insert on entity ID 01737977-eb66-46c9-89d2-2d1c94756e7d -- type Account
Processing change -- Insert on entity ID 6ed24d2a-77dd-497b-ac7c-4b692f9e8f6c -- type Case
Processing change -- Insert on entity ID 3cb009a2-a3e8-46d6-a368-e3892511b39a -- type
WorkOrder
Save Time: 676ms

Exiting rule execution
```

Request and Response Message Size Limitations

Salesforce includes a variety of different governors to track and enforce limits on resources used by Apex. One of these is request and response size limitations to HTTP callouts. Salesforce sets this limit at 12 MB for both requests and responses. These limits should not commonly be exceeded, but if you get this error during rule execution, try to reduce the number of notifications, validations, or entity changes you are making in a rule set. These are all returned in the response for processing, and reducing these can reduce the response message size.

Enable Background Compilation

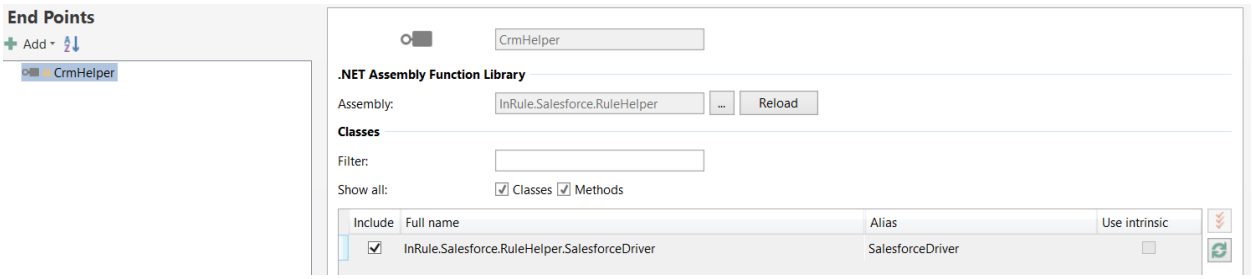
This setting controls whether the download, compilation, and caching of updates to rule applications from the catalog is handled on a background task. When enabled, rule execution will not be blocked while updating the rule app and will instead run immediately against the currently cached version. Self-hosted tenants should ensure this value is set to true for their deployment to avoid unnecessary catalog overhead. For additional information, please view the documentation available on the Configuration Overview page.

Disabling State Refresh

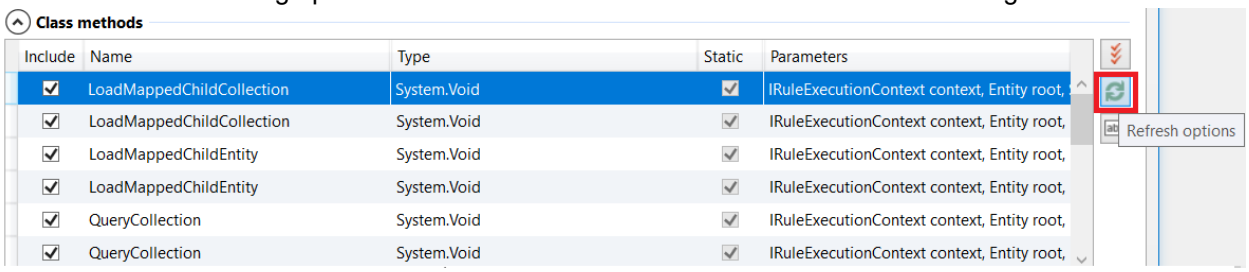
Disabling the state refresh option could improve performance for applications with many RuleHelper calls. State refresh is configured for both .Net Assembly Function Libraries, and User-Defined Functions.

.Net Function Assembly Libraries

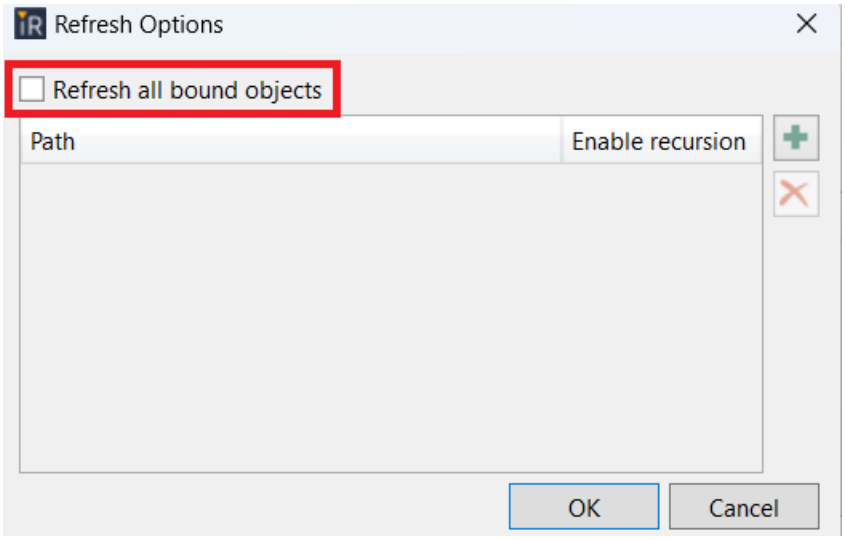
To disable state refresh for .Net Assembly Function Libraries, begin by navigating to the Endpoints tab in irAuthor. Once there, select the function library containing the function you wish to disable state refresh for.



Select the function being updated from the list and then click the refresh icon on the right.



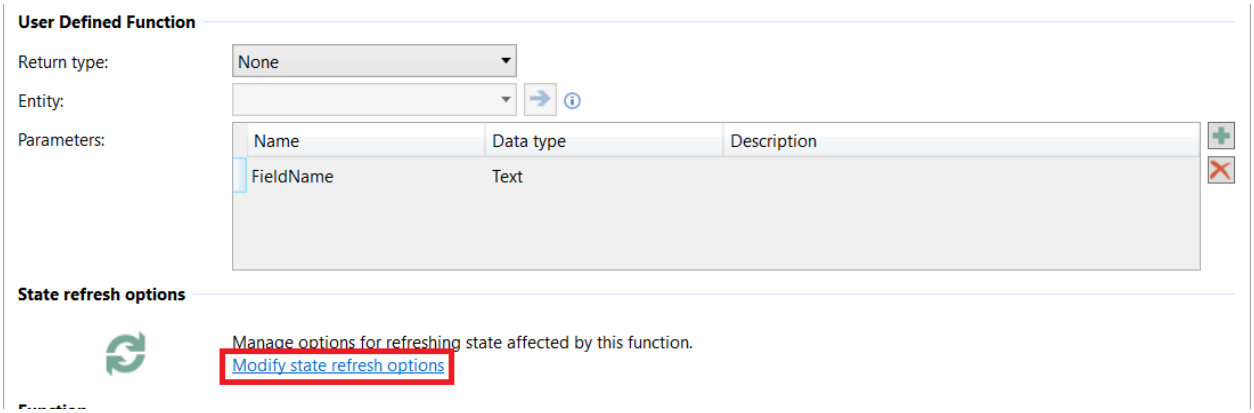
A Refresh Options pop-up should appear – locate the “Refresh all bound objects” checkbox and uncheck it. Click OK.



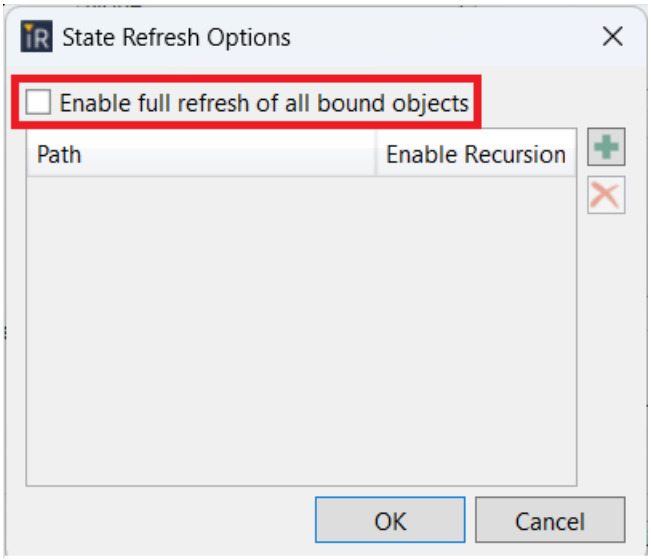
State refresh should now be disabled for this function.

User-Defined Functions

To disable state refresh for User-Defined Functions, begin by navigating to the User-Defined Functions tab of your app in irAuthor. Once there, select the function you wish to disable state refresh for and click the “Modify state refresh options” link under the State Refresh Options section.



A State Refresh Options pop-up should appear. Deactivate the “Enable full refresh of all bound objects” checkbox and lick Ok.



State refresh should now be disabled for this function.