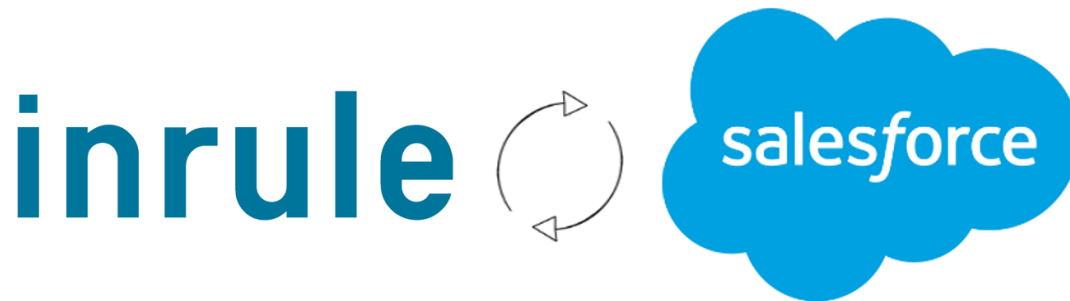


InRule® for Salesforce Deployment Guide



Document Updated against InRule v5.6.0

Document Updated against InRule for Salesforce v2.8.0

InRule does not upgrade this document after each Platform 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.

CONFIDENTIAL Any use, copying or disclosure by or to any other person than has downloaded a trial version of InRule or signed an NDA is strictly prohibited. If you have received this document by any other means than a download or an email from an InRule employee, please destroy it retaining no electronic or printed copies.

© Copyright 2020 InRule Technology, Inc.

Microsoft® is a registered trademark of Microsoft Corporation

Salesforce® is a registered trademark of Salesforce.com, Inc.®

All rights reserved. No parts of this work may be reproduced in any form or by any means – graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems – without the written permission from InRule Technologies, Inc.

InRule, InRule Technology, irAuthor, irVerify, irServer, irCatalog, irSDK and irX are registered trademarks of InRule Technology, Inc. All other trademarks and trade names mentioned herein may be the trademarks of their respective owners and are hereby acknowledged.

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document. The publisher and author reserve the right to make corrections, updates, revisions, or changes to the information contained herein. InRule Technology, Inc. does not warrant the material described herein to be free from patent infringement.

Table of Contents

Table of Contents	2
1 Introducing InRule® for the Salesforce Platform	3
2 Understanding your options	4
2.1 Salesforce – PaaS with Microsoft Azure	4
2.2 Salesforce - with IIS On-Premises Rule Execution Service	4
3 Performing the Installation	5
3.1 An overview of the Components needed	5
3.2 Gathering prerequisites	6
3.3 Deploying and Configuring Components	11
3.3.1 Catalog App Service	11
3.3.2 Rule Execution App Service for Salesforce	14
3.3.3 Upload License file	20
3.3.4 InRule for Salesforce App	21
Appendix A: Additional Resources	32
InRule's Support Website	32
InRule's Support Team	39
InRule's ROAD Team	39
InRule Training Services	39
Appendix B: Anatomy of a Request for Execution of Rules Diagram	40
Appendix C: irX General Integration Concepts	41
Appendix D: Accessing Salesforce Directly from Rule Helper	42
Appendix E: Methods for Executing Rules from Salesforce	50
1 Adding a Lightning Button	50
2 Adding a Classic UI Button	54
3 Executing Rules from Apex	57
4 Executing Rules from Triggers	59
5 Executing Rules from Lightning Flow	68
Appendix F: Azure App Service Plan Configuration	72
Appendix G: Azure Application Insights Configuration	75
Appendix H: Endpoint Override Configuration	77
Appendix I: Salesforce and Rule Execution Service Event Logging	79

1 Introducing InRule® for the Salesforce Platform

InRule provides *InRule® for the Salesforce Platform* to enable rule execution integration with Salesforce. This Salesforce package and solution framework serves as a runtime companion to the *irX® for the Salesforce Platform* rule authoring extension.

This guide focuses on the deployment of the Platform to your environment. While there are a number of options available when it comes to how you choose to integrate InRule with Salesforce, this guide details the primary deployment path for cloud-based integration with Microsoft Azure®.

Before beginning this guide, you may first want to familiarize yourself with the *irX for the Salesforce Platform* product by reading the [irX for the Salesforce Platform Help Documentation](#).

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.

If you are interested in getting rule execution integration working between InRule and Salesforce as quickly and easily as possible, this guide is the best place to start!

Additional material is available on the Downloads section of our support website. Please see the [Additional Resources](#) section of this document for support website detail.

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



Important: The InRule® for the Salesforce® Platform plugin is comprised of several Apex components. Apex is not supported by the Salesforce Professional Edition, thus InRule cannot function in a Professional Edition environment. The InRule for Salesforce App can only be deployed to Salesforce Unlimited and Developer Edition orgs, where Apex is supported.



Important: Not all Salesforce environments support API access. However, InRule® for the Salesforce® Platform 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 environment can also be used if required.

2.1 Salesforce – PaaS with 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.

2.2 Salesforce - with IIS On-Premises Rule Execution Service

If for any reason you can't use 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.

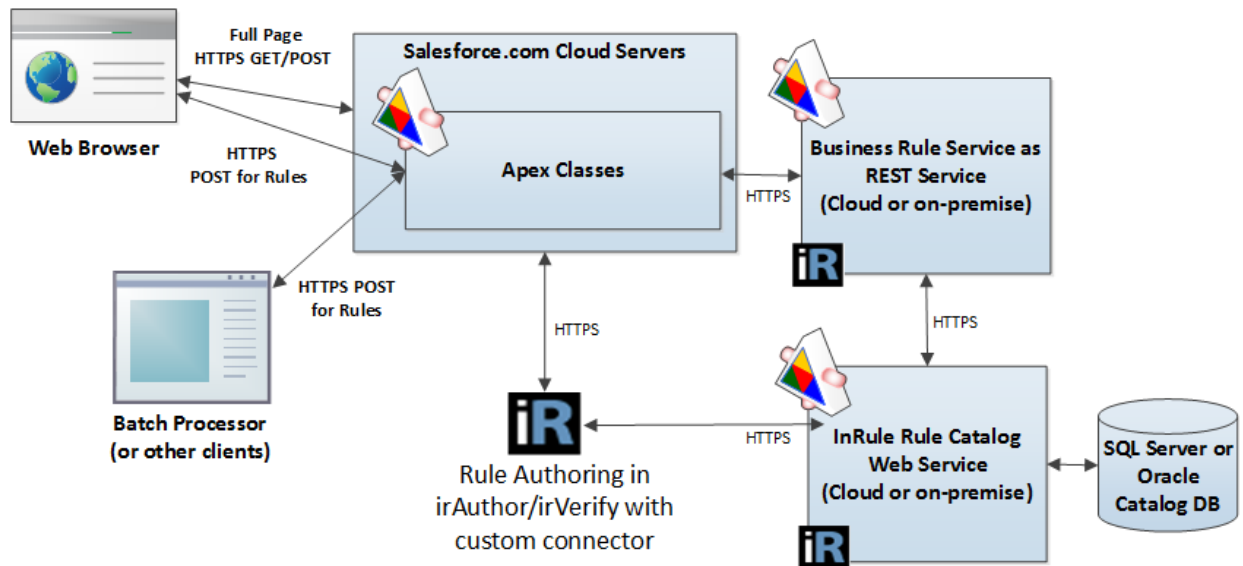
Detailed installation steps are not included with this guide, but the InRule.Salesforce.WebService.zip package in the RuleExecutionAzureService folder can also be deployed in an on-prem environment. This is a Web Deploy package, so it can be deployed via MSDeploy or by copying the contents to an IIS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

3 Performing the Installation

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

3.1 An overview of the Components needed

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

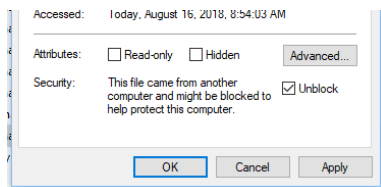
This section reviews what you will want to have prepared before you begin with the integration steps in the next section.

Required Files

The following file should be downloaded from [our support website's downloads section](#) before you begin:

- InRule for Salesforce.zip

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

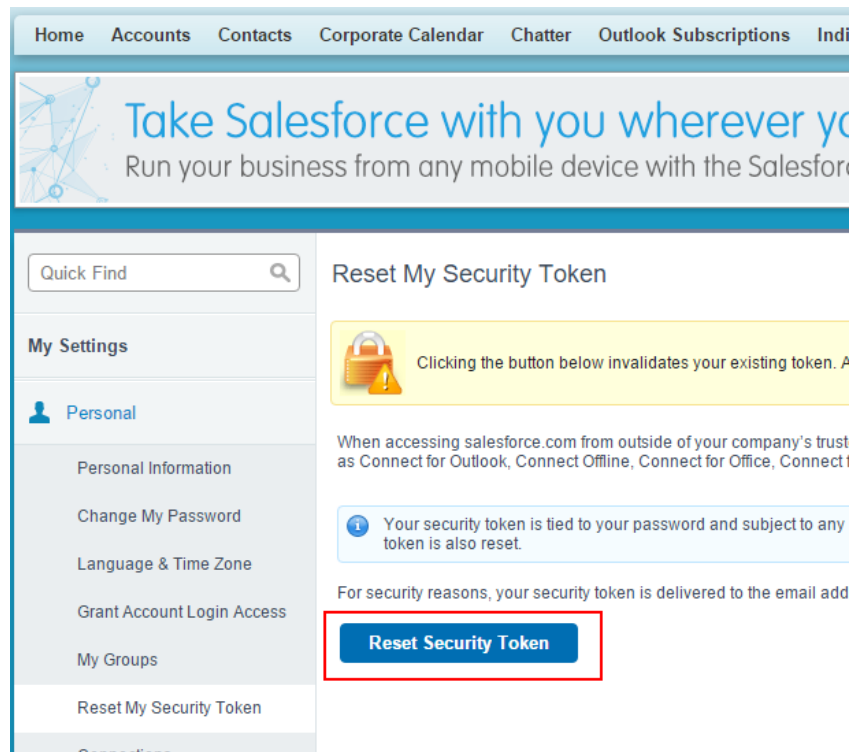
A rule authoring environment is used to upload a Rule Application to your Catalog Service. A rule authoring environment is a machine or virtual machine where irAuthor has been installed with the irX for the Salesforce Platform extension. If you followed the instructions outlined in [irX for the Salesforce Platform 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 the Salesforce Platform 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 choose 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 API Credentials: You will need to configure an API token and Connected App within your Salesforce instance for use in both irX and the Rule Execution Service. When authoring in irX you will typically use your personal account, but the execution service will typically use a separate service account. You will need to generate an API token for each account you want to use, but you can use the same Connected App if desired.

If the account does not already have an API token, then log into the Salesforce portal and navigate to User menu → My Settings → Personal → Reset My Security Token. You should receive an email within a few minutes that contains an API token.



Next, create a new Connected App in Salesforce that is configured for OAuth authentication. Navigate to Setup → Create → Apps → Connected Apps → New and create a new app. Fill out the Basic Information and configure OAuth authentication. The Rule Execution Service and irX use 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.

New Connected App

To publish an app, you need to be using a Developer Edition organization with a namespace prefix chosen.

Basic Information

Connected App Name

InRule Rule Execution

API Name

InRule_Rule_Execution

Contact Email

someemailaddress@inrule.com

Contact Phone

5555555555

Logo Image URL

Upload logo image or Choose one of our sample logos

Icon URL

Choose one of our sample logos

Info URL

<http://www.inrule.com/>

Description

InRule Service that executes rules against Salesforce data

API (Enable OAuth Settings)

Enable OAuth Settings

☒

Callback URL

<https://test.salesforce.com/services/oauth2/callback>

Use digital signatures

☐

Selected OAuth Scopes

Available OAuth Scopes

Access and manage your Chatter data (chatter_api)
Access and manage your Wave data (wave_api)
Access and manage your data (api)
Access custom permissions (custom_permissions)
Access your basic information (id, profile, email, address, phone)
Allow access to your unique identifier (openid)
Perform requests on your behalf at any time (refresh_token, offline_access)
Provide access to custom applications (visualforce)
Provide access to your data via the Web (web)

Add

Remove

Selected OAuth Scopes

Full access (full)

Web App Settings


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

Connected App Name
irX for the Salesforce Platform

[Help for this Page](#)

[Back to List: Custom Apps](#)

[Edit](#) [Delete](#) [Manage](#)



Version	1.0
API Name	irX_for_Salesforce
Created Date	5/10/2016 8:45 AM
By	[Redacted]
Contact Email	[Redacted]
Contact Phone	
Last Modified Date	9/13/2016 9:22 AM
By	[Redacted]
Description	
Info URL	

▼ API (Enable OAuth Settings)

Consumer Key	[Redacted]	Consumer Secret	Click to reveal
Selected OAuth Scopes	Full access (full)	Callback URL	https://ap1.salesforce.com/services/oauth2/token

Once you've 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.

Username and Password for the Rule Service: The rule execution service is protected using basic authentication. You will need to choose a username and password when setting up the service in Azure, and use those same credentials 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 Password to use for Catalog Service: You should decide what username and password you want to use for administrative privileges within irCatalog. 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.*

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}{OrganizationAbbreviation}{EnvironmentAbbreviation}

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 <i>must be lower case</i>	myappinruledevsqlserver
Catalog Database Name	MyAppInRuleDevCatalogDb
Catalog App Service Name	MyAppInRuleDevCatalogService
Rule Execution App Service Plan Name	MyAppInRuleDevRuleExecutionAppServicePlan
Rule Execution App Service Name	MyAppInRuleDevRuleExecutionAppService

3.3 Deploying and Configuring Components

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#ircatalog>

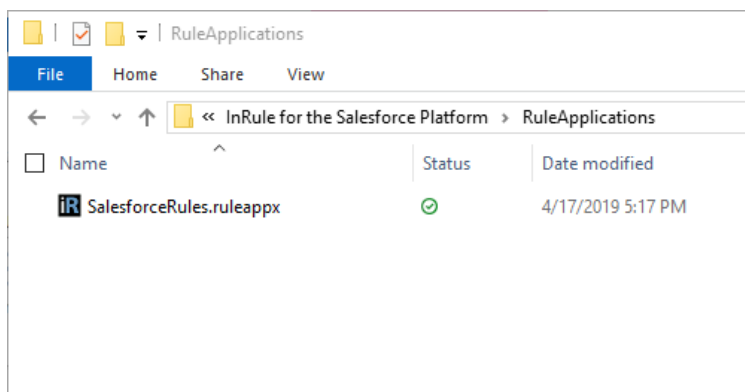
Please ensure you are installing the irCatalog service, not the irServer Rule Execution Service, which is found on the same page and is not compatible with Salesforce.

Testing the Catalog App Service by uploading the starter Rule App

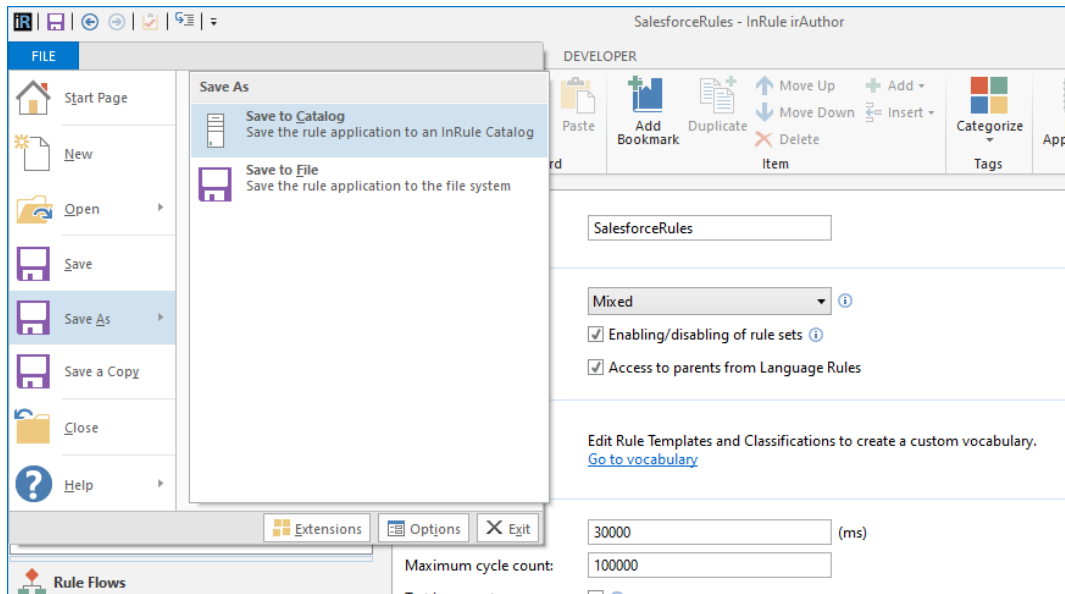
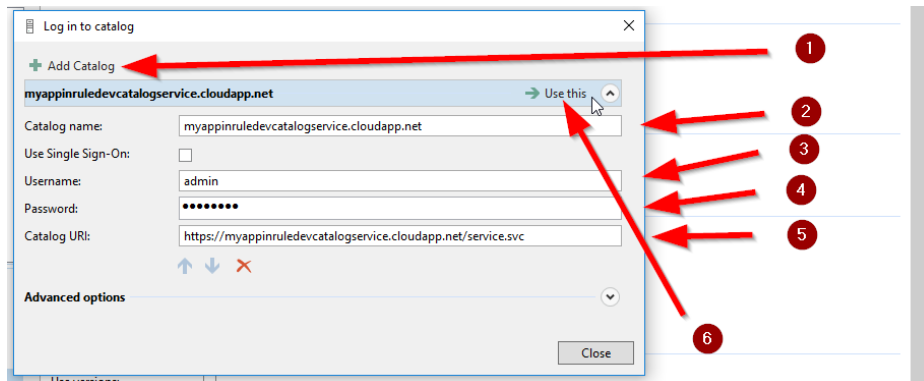
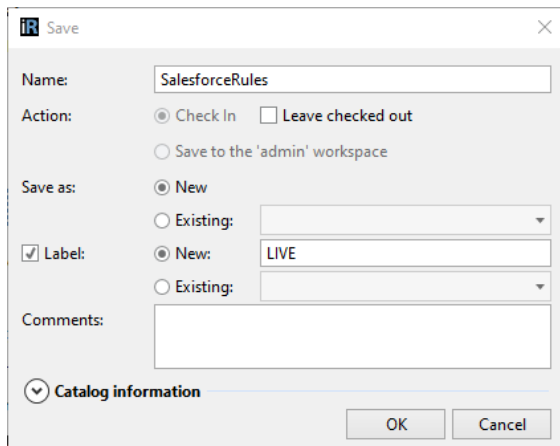
At the conclusion of the installation process outlined above you should have a Catalog URI, Username, and Password to use to connect to the catalog service.

Next, we will utilize your rule authoring environment to upload the Rule Application that you extracted into your working directory at:

`\InRule for Salesforce\RuleApplications\SalesforceRules.ruleappx`



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

2: Save the Rule Application by choosing File → Save As → Save to Catalog**3: Choose Add Catalog, enter connection information for the Catalog Server that you deployed, and then select Use This.****4: Save with the name SalesforceRules and the Label LIVE**

Save the Rule Application to the Catalog using the name of *SalesforceRules*. We must also be sure to label this Rule Application with the text '*LIVE*', as configured in the app service configuration. Be sure not to forget this label, it is a small but important step!

At this point, if you can click OK without an error, we 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.

3.3.2 Rule Execution App Service for Salesforce

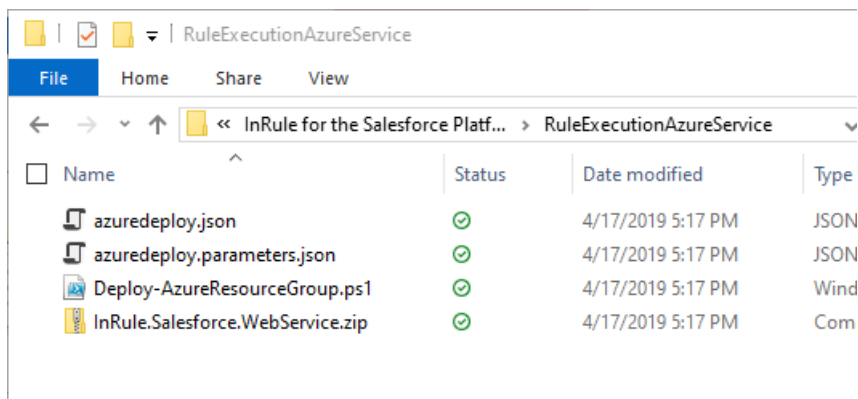
Next, we'll deploy the InRule Rule Execution service as an Azure App Service, along with all its Azure resource dependencies. To make this process easier, we'll be using an Azure Resource Manager (ARM) template, which allows us to deploy and configure all the Azure resources the Rule Execution Service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

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 this, reference Microsoft's documentation: <https://docs.microsoft.com/en-us/azure/azure-resource-manager/templates/deploy-portal>

1: Locate `azuredeploy.parameters.json`

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

Locate the `azuredeploy.parameters.json` file in the `RuleExecutionAzureService` folder within the *InRule for Salesforce.zip* file downloaded in [Section 3.2: Required Files](#).



2: Update parameters

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

```

"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"
  },
  "executionServiceBasicUsername": {
    "value": ""
  },
  "executionServiceBasicPassword": {
    "value": ""
  },
  "appServicePlanName": {
    "value": ""
  },
  "inRuleVersion": {
    "value": "5.5.1"
  },
  "appInsightsResourceName": {
    "value": ""
  },
  "appInsightsInstrumentationKey": {
    "value": ""
  }
}

```

1 appServiceName	<p>Provide a name for the Azure App Service that the Rule Execution Service will run on.</p> <p>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 Configuration</p>
2 catalogUri	<p>The URI for the Catalog Service that will be used</p> <p>Example: https://myappinruledevcatalogservice.cloudapp.net/service.svc </p>
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 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
6 sfUsername	Salesforce username for the service account used to connect from the execution service to the Salesforce API

7 sfPassword	Password for the Salesforce service account
8 sfSecurityToken	API token for the Salesforce service account
9 sfConsumerKey	OAuth consumer key for the connected app
10 sfConsumerSecret	OAuth consumer secret for the connected app
11 executionServiceUsername	Username used to secure the rule service. This same username will need to be provided to Salesforce during configuration
12 executionServicePassword	Password used to secure the rule service. This same password 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 Configuration
15. appInsightsInstrumentationKey	If you want to use Application Insights as a log sink in addition to the app service logging already enabled, provide an instrumentation key from an existing Application Insights resource here. When providing a value for the 'appInsightsResourceName' parameter, leave this value blank, as the template will automatically fill this value in for you based on the newly created resource Rule Execution Service Event Log .
16. appInsightsResourceName	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.

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

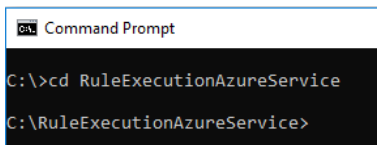
3: Option 1: Deploy ARM Template with Azure CLI

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

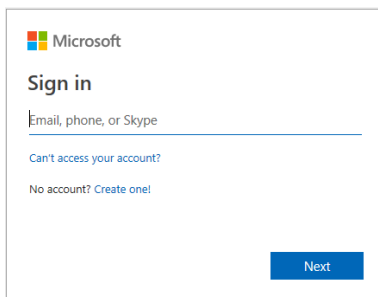


3.3 Enter "az login" to login into Azure


```
Command Prompt

C:\>cd RuleExecutionAzureService
C:\RuleExecutionAzureService>az login
```

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.parameters.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

 Windows PowerShell

1.2 Navigate to the RuleExecutionAzureService folder

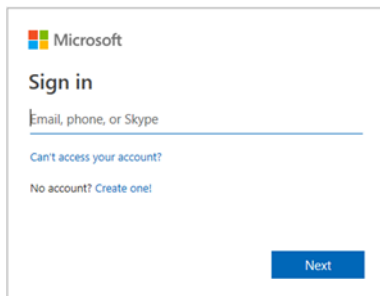
 Windows PowerShell

```
PS C:\> cd .\RuleExecutionAzureService\  
PS C:\RuleExecutionAzureService>
```

1.3 Enter “Connect-AzureRmAccount” to login into Azure

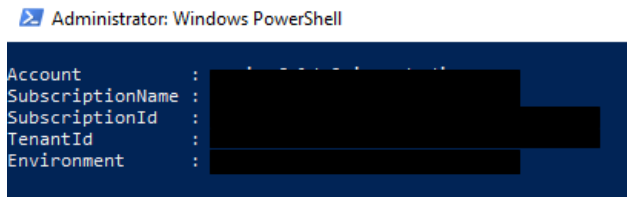
```
PS C:\RuleExecutionAzureService> Connect-AzureRmAccount
```

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:

Administrator: Windows PowerShell

```
Account      : [REDACTED]  
SubscriptionName : [REDACTED]  
SubscriptionId  : [REDACTED]  
TenantId      : [REDACTED]  
Environment    : [REDACTED]
```

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:

 Windows PowerShell

```
PS C:\RuleExecutionAzureService> Select-AzureRmSubscription -SubscriptionName SubscriptionNameHere
```

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

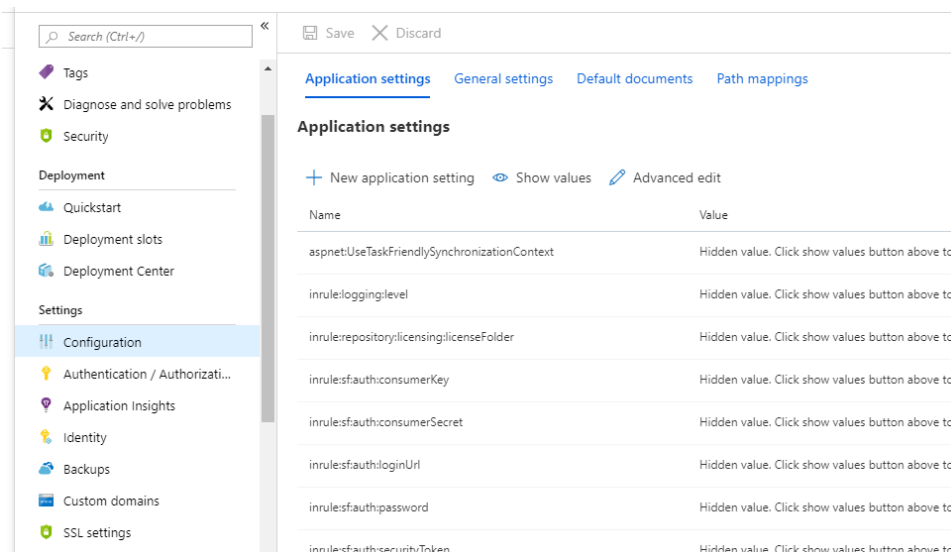
5: Verify Setup

Navigate to the Azure portal and locate the deployed App Service

1 of 2 items selected ☐ Show hidden types

NAME	TYPE
 armTestAppService	App Service

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



Search (Ctrl+/)

Save Discard

Application settings General settings Default documents Path mappings

Application settings

+ New application setting Show values Advanced edit

Name	Value
aspnet:UseTaskFriendlySynchronizationContext	Hidden value. Click show values button above to
inrule:logging:level	Hidden value. Click show values button above to
inrule:repository:licensing:licenseFolder	Hidden value. Click show values button above to
inrule:auth:consumerKey	Hidden value. Click show values button above to
inrule:auth:consumerSecret	Hidden value. Click show values button above to
inrule:auth:loginUrl	Hidden value. Click show values button above to
inrule:auth:password	Hidden value. Click show values button above to
inrule:auth:securityToken	Hidden value. Click show values button above to

3.3.3 Upload License file

Regardless of how you choose to deploy the ARM template, you'll need to upload a license file to the web app service in order for both the catalog service and the rule execution app service to properly function. The simplest way to upload the license file is via 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 + "/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 + "/InRuleLicense.xml");$client.UploadFile($uri, "LICENSE_FILE_ABSOLUTE_PATH")
```

3.3.4 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.

1: Install from AppExchange:

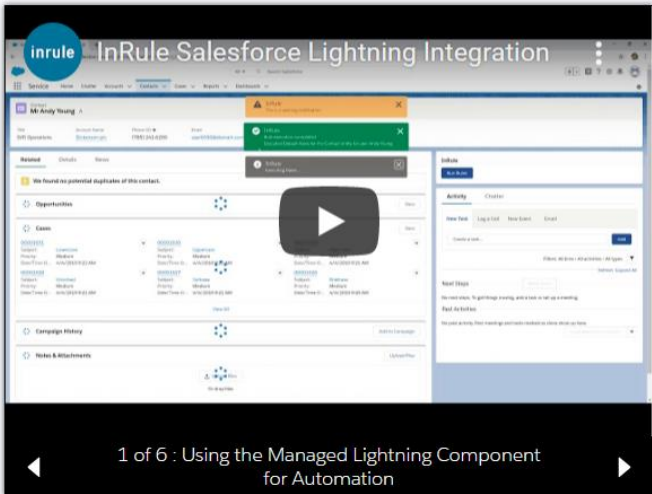
InRule for Salesforce can be installed from the Salesforce AppExchange marketplace. You can either search for 'InRule for Salesforce' or go directly to the listing here:

<https://appexchange.salesforce.com/appxListingDetail?listingId=a0N3A00000FMd5cUAD> 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.

< SEARCH RESULTS | ALL APPS

InRule® for Salesforce

By InRule



Get It Now

RATING
★★★★★ (0)

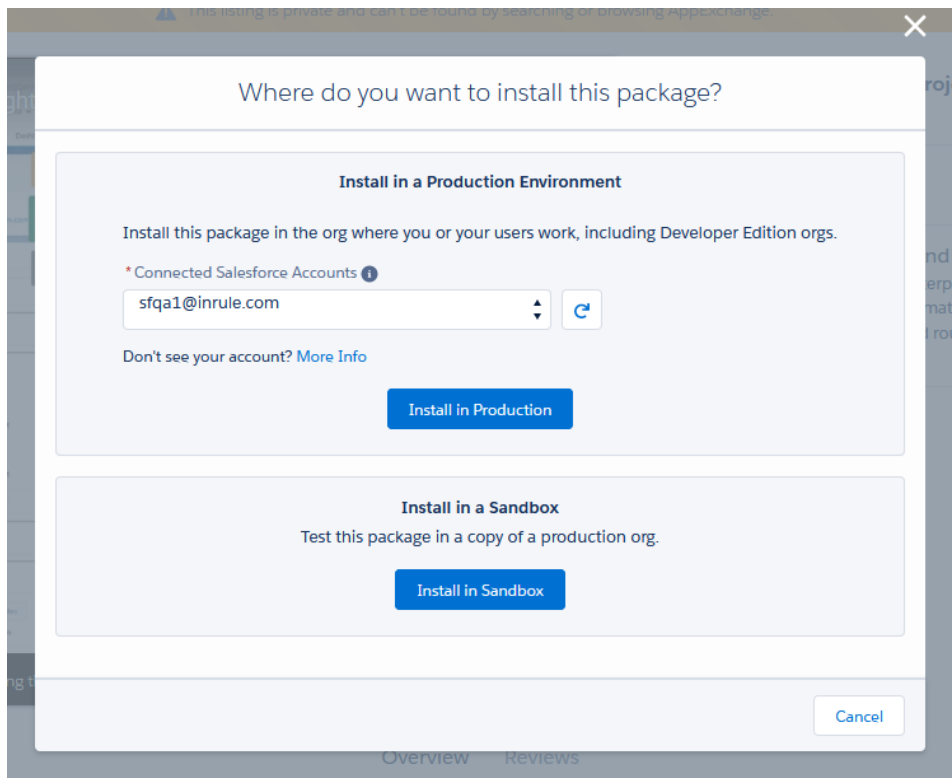
LISTED ON
9/25/2020

LATEST RELEASE
9/10/2020

Watch Demo

Automate Business Decisions and Rules Without Code

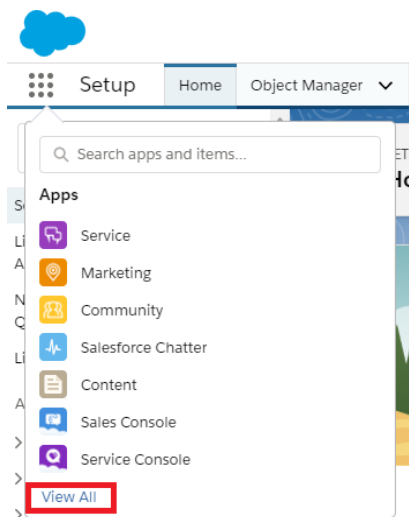
InRule for Salesforce is the leading enterprise-grade decision platform that provides the ability to create and automate complex business decisions in Salesforce. InRule can be used for lead routing, claims adjudication, loan eligibility, and much more.



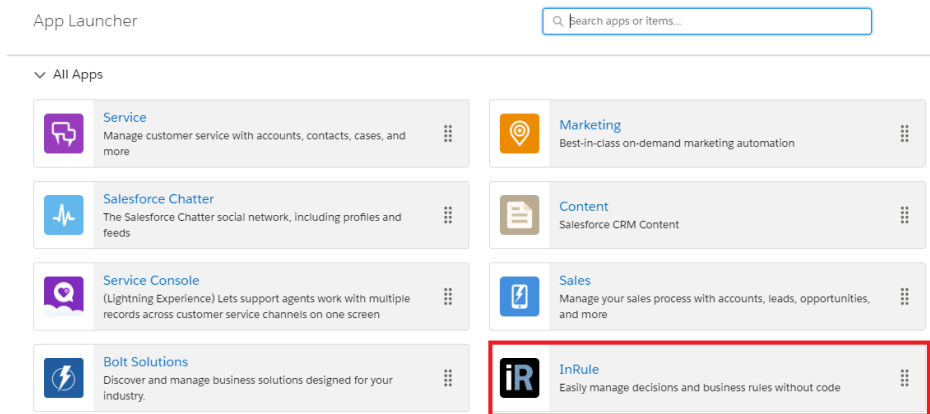
Once you've selected an org and logged in, you'll see a screen prompting you to install the package. Select 'Install for all users' and hit ok. Installing this package installs all the components required for integrating with InRule, including a configuration app, Apex classes, and custom settings and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

Named Credential

Next, we need to create a Named Credential in Salesforce. This will be used by the Apex code to make secure HTTP request to the Rule Execution Service.

In the InRule App, select the “Configuration” tab



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

Named Credentials

First create a Named Credential in Salesforce. This will be used by the DecisionClient deployed as part of the Salesforce application to make secure HTTP requests to the Rule Execution Service. Navigate to Setup → Security Controls → Named Credentials → New Named Credential. You can click here as a shortcut to get there: [Edit Named Credentials](#)

Press the “New Named Credential” button

Named Credentials

A named credential specifies a callout endpoint and its required authentication parameters. When setting up callouts, avoid setting authentication parameters for each callout by referencing named credentials.

View: All [Create New View](#)


A | B | C | D | E


New Named Credential


Named Credential Edit: InRule Rule Execution Service

Specify the callout endpoint's URL and the authentication settings that are required for Salesforce to make callouts


Save Cancel


Label  InRule Rule Execution Service


Name  InRule_Rule_Execution_Ser


URL  https://sampleservice.azurewebsites.net


Authentication

Certificate 


Identity Type  Named Principal


Authentication Protocol  Password Authentication


Username  BasicUserName

Password 

Callout Options

Generate Authorization Header  ☒

Allow Merge Fields in HTTP Header  ☐

Allow Merge Fields in HTTP Body  ☐

Save Cancel

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 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 service. This should be the same value chosen earlier for the 'ruleServiceUsername' parameter when setting up the service in Azure
Password	Password for accessing the rule service. This should be the same value chosen earlier for the 'ruleServicePassword' parameter when setting up the service in Azure
Callout Options	Make sure 'Generate Authorization Header' is checked

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.

Custom Settings

Once you've set up the Named Credential, navigate to the Custom Setting installed by the package at Setup → Develop → Custom Settings → InRule → Manage → Edit. You can click below as a shortcut to that location: [Custom Settings](#)

Click on the InRule label

Custom Settings

Use custom settings to create and manage custom data at the organization, profile, and user levels. Custom settings data is stored in the application cache. This means you can access it efficiently, without the cost of repeated queries. Custom settings data can be used by formula fields, Visualforce, Apex, and the Web Services API.

View: All Create New View

Get Usage

Action	Label ↑	Visibility	Settings Type	Namespace Prefix	Description	Record Size	Number of Records	Total Size
Manage	InRule	Public	Hierarchy	inrule		201	0	0

Click "Manage"

Custom Setting Definition

InRule

Create the fields for your custom setting. The data in these fields are cached with the application.

Custom Setting Definition Detail

Label	InRule	Object Name	InRule_Settings
API Name	inrule__InRule_Settings__c	Setting Type	Hierarchy
Visibility	Public	Description	
Namespace Prefix	inrule	Created Date	5/4/2020, 10:26 AM
Last Modified Date	5/4/2020, 10:26 AM	Record Size	201

Click "New"

Custom Setting

InRule

If the custom setting is a list, click **New** to add a new set of data. For example, if your application had a setting for country codes, each set might include the country's name and dialing code.

If the custom setting is a hierarchy, you can add data for the user, profile, or organization level. For example, you may want different values to display depending on whether a specific user is running the app, a specific profile, or just a general user.

New

InRule Edit

Provide values for the fields you created. This data is cached with the application.

Edit InRule Save Cancel

InRule Information

Location

Log Level ⓘ

Rule App Name ⓘ

Configuration Form Fields	
Rule App Name	The name of the rule app that will be loaded by the Rule Execution Service when running rules
Logging Level	<div>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:</div> <ul style="list-style-type: none">• 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

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. Under the Named Credential Header, 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.

Named Credentials

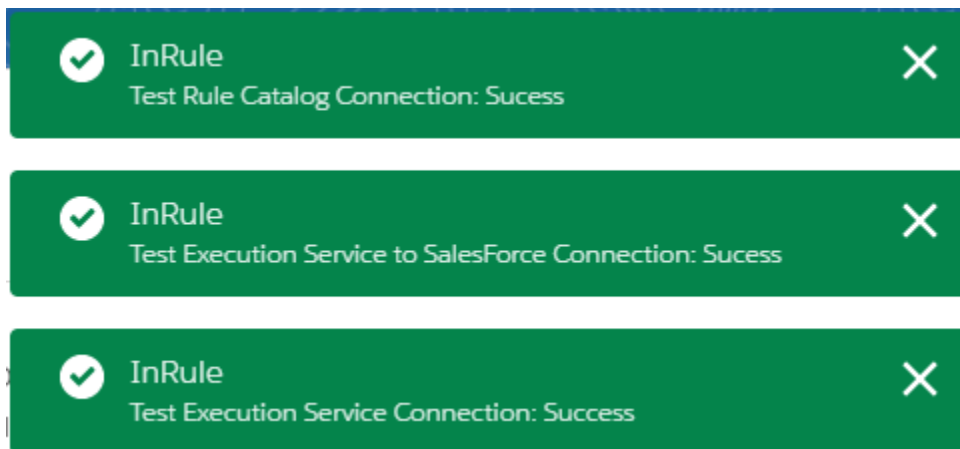
First create a Named Credential in Salesforce. This will be used by the Decision Navigator to Setup → Security Controls → Named Credentials → New Named Credential. The newly created Named Credential should be configured with the following:

- **Label:** This should be set to "InRule Rule Execution Service". This value is with the correct one
- **Name:** This value must be set to "InRule_Rule_Execution_Service". The r
- ***URL:** This setting contains the base URL for the rule service in Azure. For created earlier. For example, <https://sampleservice.azurewebsites.net>
- **Identity Type:** Named Principal
- **Authentication Protocol:** Password Authentication
- ***Username:** The username used for accessing the rule service. For self-managing the service in Azure
- ***Password:** The password for accessing the rule service. For self-managing the service in Azure
- **Callout Options:** Make sure 'Generate Authorization Header' is checked

* **Note:** Items marked with an asterisk will have their values provided by InRule. Once you've configured the Named Credential, click below to test the connection.

Test Connectivity

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



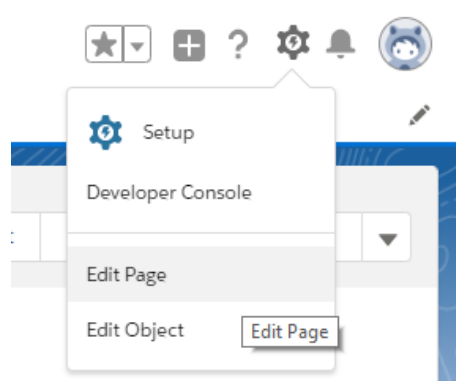
5: Add the Run Rules button

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, 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

▼ Custom (0)

No components available.

▼ Custom - Managed (1)

 InRule_RunRules

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

▼ Custom (0)

[Deploy My Domain](#) to see custom components here.

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.

Page > InRule_RunRules

Rule Application Name - leave blank to use the default configured rule application

RuleSet Name(s) - comma-delimited list or leave blank to derive RuleSet as (entityName)DefaultRules

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

If no RuleSets are defined, it will create 1 button that uses the default RuleSet for the entity:

The screenshot shows the InRule configuration interface. At the top, there is a navigation bar with buttons: "+ Follow", "Edit", "New Contact", and "New Case" with a dropdown arrow. Below this, the main configuration area is divided into two sections. The left section, titled "InRule", contains a "Run Rules" button. The right section, titled "RuleSet Name(s)", contains a text input field with the placeholder text "RuleSet Name(s) - comma-delimited list or leave blank to derive RuleSet as (entityName)DefaultRules". Below this, there is a section titled "Set Component Visibility" with a "Filters" section containing a "+ Add Filter" button.

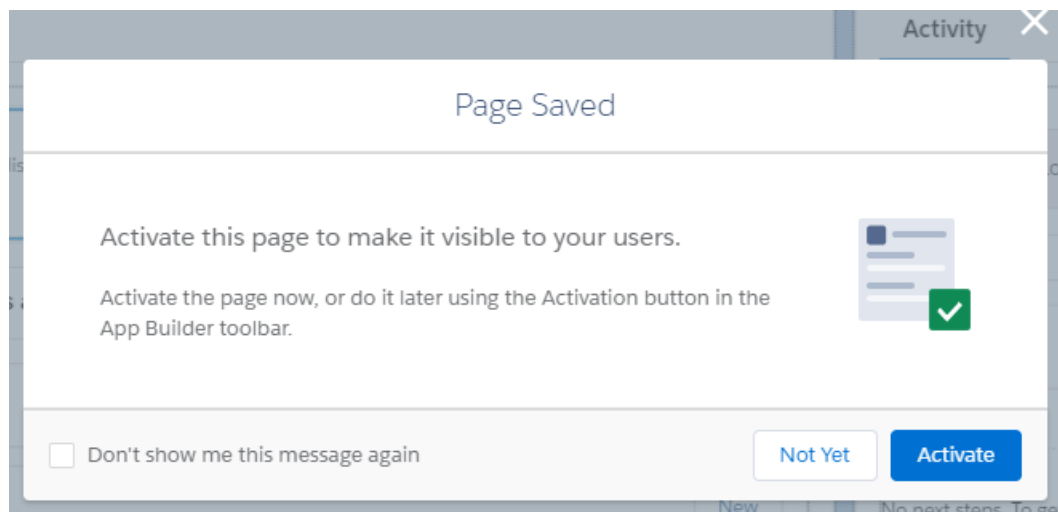
If one RuleSet is defined, will create 1 button that will use that defined RuleSet:

This screenshot is similar to the previous one, but the "RuleSet Name(s)" text input field now contains the text "Rule1". The rest of the interface, including the navigation bar, "InRule" section, and "Set Component Visibility" section, remains the same.

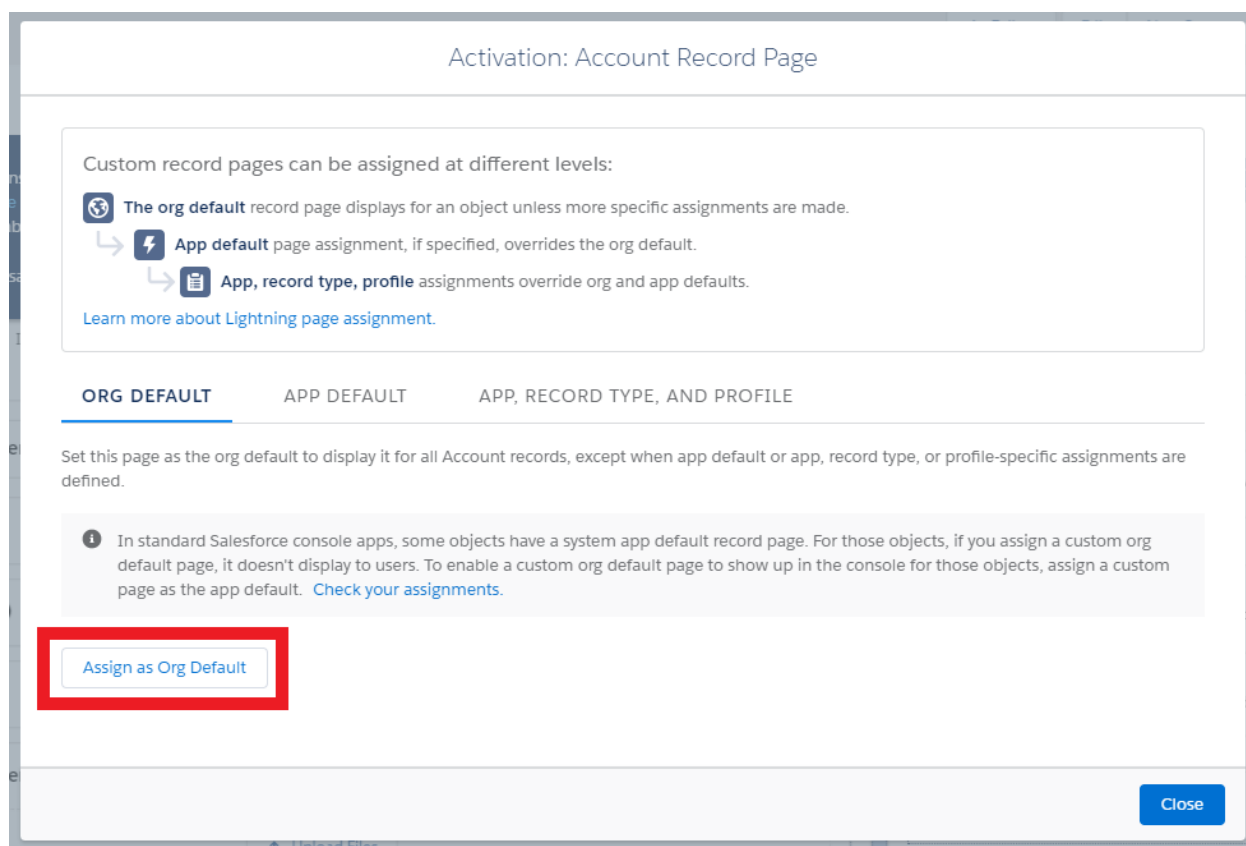
If 2+ RuleSets are defined, will create a button per RuleSet that will use the defined RuleSets.

This screenshot shows the InRule configuration interface with multiple RuleSets. The "RuleSet Name(s)" text input field now contains the text "Rule1, Rule2". The "InRule" section on the left now contains two buttons, "Rule1" and "Rule2", instead of just the "Run Rules" button. The rest of the interface, including the navigation bar and the "Set Component Visibility" section, remains the same.

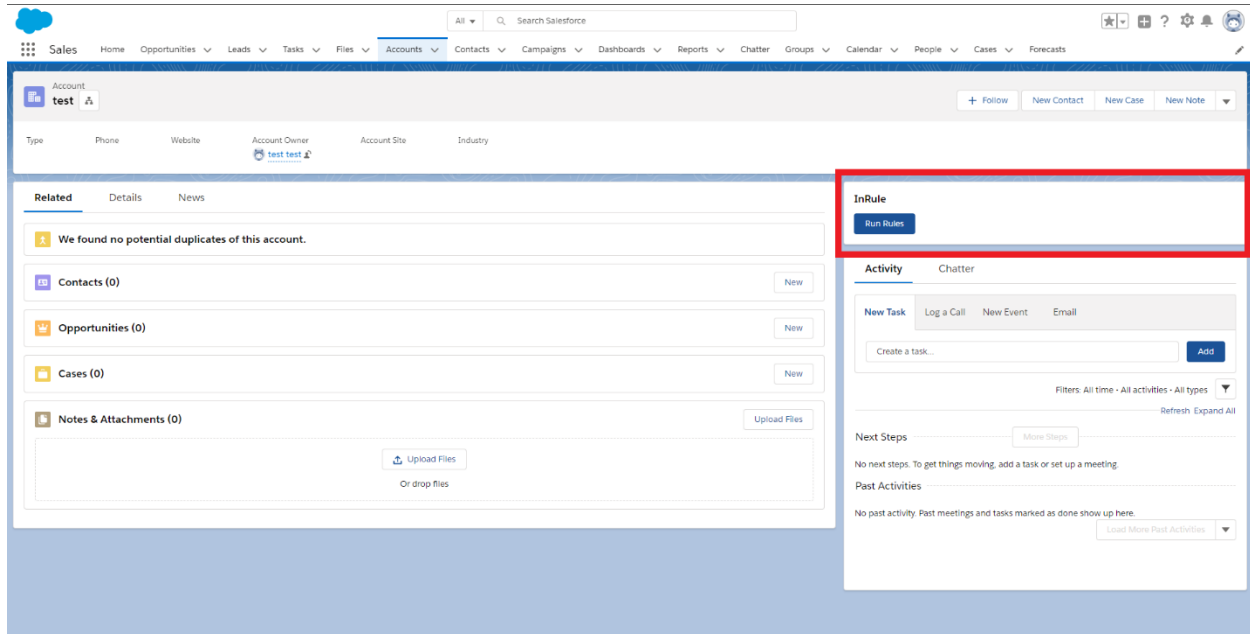
Once you have configured your RuleSets, click save in the top right corner. It will prompt you to activate this page to make it visible to your users, click activate.



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

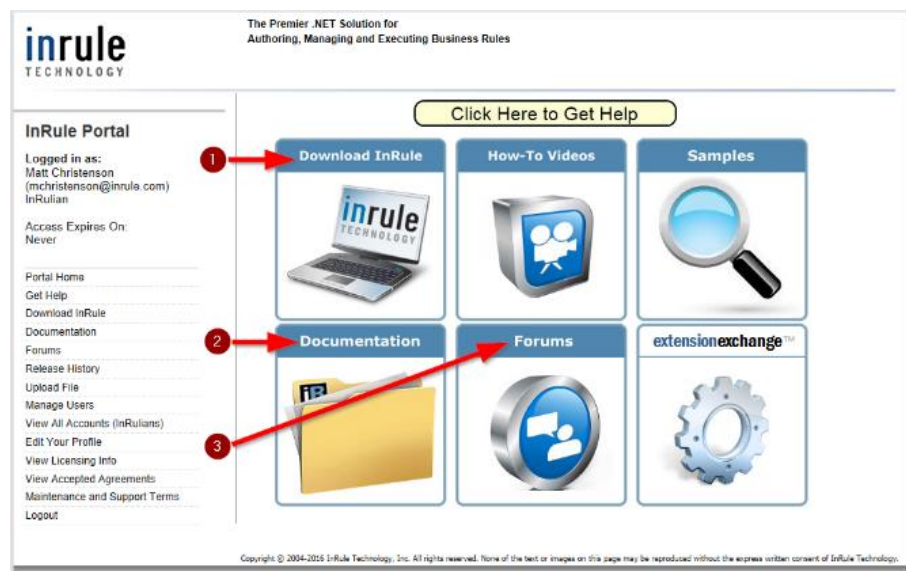
Having trouble? Relax! InRule offers many additional resources to help you get InRule correctly integrated with Salesforce.

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.

The InRule Support Website has three different areas that readers should be aware of:

1. Downloads
2. Documentation
3. Forums



The Downloads section

The download section of the Support Website contains numerous resources that will likely be of interest to you:

InRule Installer*

Filed under the 'InRule' section of the downloads page, the InRule installer package will launch a graphical user interface that will walk the user through installing many of InRule's offerings. These offerings include both irAuthor which is used to edit Rule Applications, and irX for the Salesforce Platform, which is an extension to irAuthor that allows synchronizing schema between Rule Applications and Salesforce.

*It is recommended that the reader of this document has established a Rule Authoring Environment by installing both [irAuthor](#) and [irX for the Salesforce Platform](#) before attempting to deploy [InRule for Salesforce](#).

irX for the Salesforce Platform Help Documentation

Filed under the 'InRule for Salesforce' section of the downloads page. This document discusses how to utilize the [irX for the Salesforce Platform](#) [irAuthor](#) extension to connect to Salesforce in a rule-authoring environment to synchronize rule application schema objects against Salesforce entities and to test Rule Applications directly against data coming from the Salesforce environment.

InRule for Salesforce Deployment Guide

This is the guide that you are reading now. The intention of this guide is to give a complete reference for those wanting to get the Platform up and going as quickly as possible without deviating from what we consider to be the most common deployment scenario.

This is a fantastic place to start if you already have some familiarity with [irX for the Salesforce Platform](#) and are ready to deploy [InRule for Salesforce](#) as a runtime companion.

InRule for Salesforce

This downloadable archive contains all of the following located off of the root:

- **Rule Applications \ SalesforceRules.ruleapp:** This Rule App is intended to be used both to test a basic deployment of the Platform, but also to serve as a starting point for authoring further rules against your Salesforce environment.
- **Rule Execution Azure Service:** This folder contains the app service package and configuration files that will be needed to deploy the Rule Execution App Service to Azure as discussed elsewhere in this document.
- **RuleHelper Deployment:** This folder contains the .NET assemblies necessary to utilize the Rule Helper library to dynamically query Salesforce for information during the execution of rules. RuleHelper is included with the Rule Execution Service, but also made available here for deploying to irAuthor's EndPointAssemblies folder. For more information about using the RuleHelper, please refer to [Appendix D: Accessing Salesforce Directly from Rule Helper](#)
- **ReadMe.txt:** This file contains some information about this Platform, including historic release notes, product version information, and license information.

Catalog App Service – Azure Package and Config File

Found in the 'InRule for Microsoft Azure' section of the downloads page, this archive contains both the app service package and app service configuration files that you will need to perform an Azure based installation of irCatalog.

Catalog App Service – Install Documentation

This document will discuss how to install InRule's irCatalog in Microsoft Azure using a PaaS model.

The Documents section

The documents section of the Support Website contains additional documentation about how the various components of InRule operate.

Although the documents section does not contain documentation that discusses integration with Salesforce (that documentation can be found in the downloads section), it does contain the following three areas of interest:

1. Online Authoring Help: This document discusses anything you would need to know about how to author Rule Applications. The majority of this document applies to utilization of irAuthor, InRule's custom graphical authoring product.
2. Online SDK Help: InRule exposes an extensive .NET based API. This document discusses the use of the full SDK for use by software developers that would like to integrate with InRule.
3. Miscellaneous documentation section: Here you will find a variety of special interest documents, such as an installation guide, an implementation guide, and a guide that focusses on InRule's UDF language, irScript.

The screenshot shows the InRule Portal's Documentation section. On the left is a sidebar with navigation links. The main content area has a search bar and sorting options. Below these are links for Online Authoring and SDK Help for versions 4.1, 4.5, 4.6, and 5.0. A table lists various documents, including an Evaluation Guide, Implementation Guide, JavaScript Documentation, Case Management Patterns, and WCF Setup. Red arrows and numbers 1, 2, and 3 highlight specific elements: 1 points to the SDK 4.1 link, 2 points to the SDK 4.5 link, and 3 points to the Implementation Guide document in the table.

InRule Portal
 Logged in as mchristenson@inrule.com
 Portal Home
 Get Help
 Download InRule
 Documentation
 Forums
 Release History
 Upload File
 Edit Your Profile
 View Licensing Info
 View Accepted Agreements
 Maintenance and Support Terms
 Logout

Browse Media
 General
 InRule Version 4 and 5
Documentation
 Authorized Licensee Docs
 Video Tutorials
 Samples for Rule Authors
 Samples for Developers
 extensionexchange
 Utilities
 InRule Version 3

The Premier .NET Solution for
 Authoring, Managing and Executing Business Rules

Media » InRule Version 4 and 5 » Documentation

Documentation

Sort by: Name | **Most Recent** | Most Downloads | Most Popular | Most Comments
 Show as: Thumbnails | **List**

For Online Authoring 4.1 Help ([click here](#)), **For Online SDK 4.1 Help** ([click here](#))
For Online Authoring 4.5 Help ([click here](#)), **For Online SDK 4.5 Help** ([click here](#))
For Online Authoring 4.6 Help ([click here](#)), **For Online SDK 4.6 Help** ([click here](#))
For Online Authoring 5.0 Help ([click here](#)), **For Online SDK 5.0 Help** ([click here](#))

Name	Date	Downloads	Comments
InRule Technology Evaluation Guide by Mark Malen	Tue, Aug 8 2017	6	0
InRule Implementation Guide by Alan Holley This guide is intended to provide high-level architectural and design guidance regarding the implementation...	Thu, Dec 10 2015	484	0
InRule for Javascript Documentation by Mark Malen This is the documentation for our new offering "InRule for JavaScript" where you can execute...	Wed, Nov 18 2015	202	0
Case Management Patterns and Practice Using InRule®... by Mark Malen Case management fundamentally changes the orientation of business applications from flow to data...	Tue, Jan 27 2015	287	0
Windows Communication Foundation (WCF) Setup by Mark Malen This document provides instructions on how to install IIS, ASP.NET, and WCF on various operating...	Tue, May 6 2014	299	0

Online User Forums

Numerous internal resources at InRule monitor InRule Forums. This is the first place to go if you want to benefit directly from **the helpful and knowledgeable people**

The Premier .NET Solution for
Authoring, Managing and Executing Business Rules

inrule
TECHNOLOGY

Forums > InRule Forums > InRule For Dynamics CRM

InRule Portal

Logged in as
mchristenson@inrule.com

Portal Home
Get Help
Download InRule
Documentation
Forums
Release History
Upload File
Edit Your Profile
View Licensing Info
View Accepted Agreements
Maintenance and Support Terms
Logout

Shortcuts

[View all users](#)
[Posts you have not read](#)
[Forum Subscriptions](#)

InRule For Dynamics CRM

Search this site

Sorting and Filtering Write a New Post | Mark all read

Topics	Replies	Views
<input type="checkbox"/> InRule CRM Integration Error returned from rule service: Label has... Latest post by Dustin Oxford, Mon, Jul 31 2017 11:14 AM	3	19
<input type="checkbox"/> InRule Custom Action does not return the Notification or Error Latest post by Josh Elster, Fri, Jul 28 2017 9:31 AM	1	7
<input type="checkbox"/> Attribute a numeric value to an option set list of metadata in order... Latest post by Matt Christenson, Tue, Mar 21 2017 6:16 PM	1	29
<input type="checkbox"/> Exception in OnPremisePlugin.cs Latest post by Haribalachandar..., Wed, Dec 21 2016 3:16 AM	4	30
<input type="checkbox"/> IRx and Quote Details Latest post by nick.welburn@bupa..., Fri, Oct 28 2016 11:09 AM	2	25
<input type="checkbox"/> InRule for Dynamics CRM Latest post by Josh Elster, Thu, Nov 5 2015 4:16 PM	1	41
<input type="checkbox"/> 400 - Bad Request Latest post by Duchan, Thu, Nov 27 2014 8:28 AM	0	37
<input type="checkbox"/> Accessing a Parent's Parent and a childs childs child from an entity... Latest post by Chris Miller, Wed, Dec 18 2013 12:34 PM	5	123

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 Diagram


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.

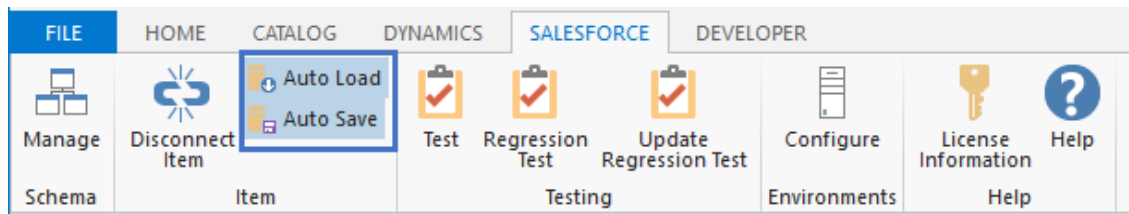
 **Note:** When N:N relationships are imported into InRule, they behave as 1:N Collection relationships within the Rule Application.


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:

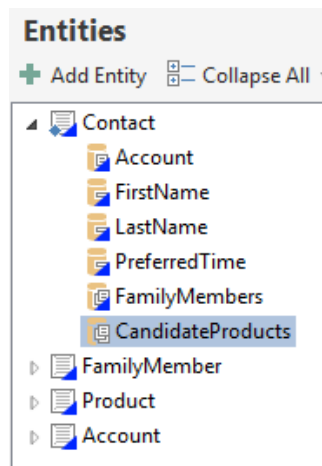
- The rules need to reference “lookup” information that may be in a list or set of Entities that are not specifically related to the current Entity hierarchy
- The purpose of rules is to create new relationships between Entity instances that already exist in Salesforce
- The rules need to compare many 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 or N: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.

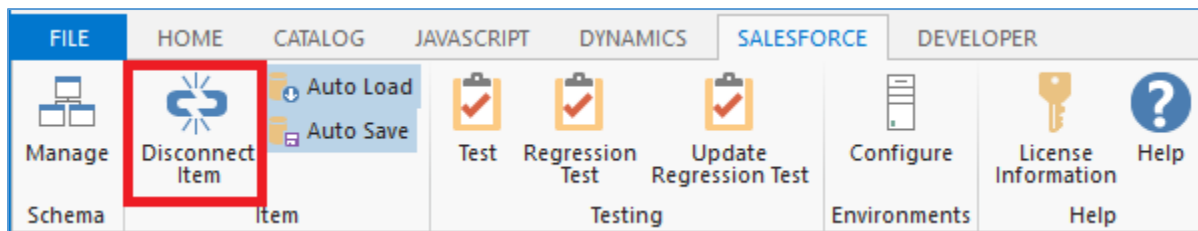
irX allows the rule author to explicitly control the “Auto Load” and “Auto Save” behaviors of Fields that are connected to Salesforce.

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.

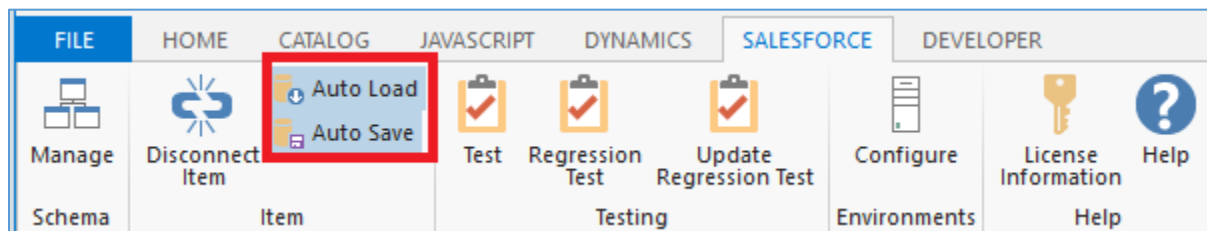


Important: Although Entity Fields and Collections may be “Disconnected” from Salesforce, the types contained by the Collections can be set to types that were imported from Salesforce.

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 get used by rules. Additional methods can always be imported later by revisiting the endpoint screen and reloading the assembly.

End Points

+ Add

o SalesforceLib

Rule Flows

Rules

Vocabulary

Entities

User Defined Functions

Data

End Points

Schemas

Assembly: InRule.Salesforce.RuleHelper Reload

.NET Assembly Function Library

Classes

Filter:

Show all: ☒ Classes ☒ Methods

Include	Full Name	Alias	Use Intrinsic
<input type="checkbox"/>	InRule.Salesforce.RuleHelper.Environment.SalesforceConnector	SalesforceConnector	<input type="checkbox"/>
<input type="checkbox"/>	InRule.Salesforce.RuleHelper.Environment.SettingsManager	SettingsManager	<input type="checkbox"/>
<input type="checkbox"/>	InRule.Salesforce.RuleHelper.OrderByClauseBuilder	OrderByClauseBuilder	<input type="checkbox"/>
<input checked="" type="checkbox"/>	InRule.Salesforce.RuleHelper.SalesforceDriver	SalesforceDriver	<input checked="" type="checkbox"/>
<input type="checkbox"/>	InRule.Salesforce.RuleHelper.WhereClauseBuilder	WhereClauseBuilder	<input type="checkbox"/>

Class methods

Include	Name	Type	Static	Parameters
<input type="checkbox"/>	Equals	System.Boolean	<input type="checkbox"/>	Object obj
<input type="checkbox"/>	GetHashCode	System.Int32	<input type="checkbox"/>	
<input type="checkbox"/>	GetType	System.Type	<input type="checkbox"/>	
<input type="checkbox"/>	LoadMappedChildCollection	System.Void	<input checked="" type="checkbox"/>	IRuleExecutionContext context, Ent
<input type="checkbox"/>	LoadMappedChildCollection	System.Void	<input checked="" type="checkbox"/>	IRuleExecutionContext context, Ent
<input type="checkbox"/>	LoadMappedChildEntity	System.Void	<input checked="" type="checkbox"/>	IRuleExecutionContext context, Ent

Description Notes Attributes

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.



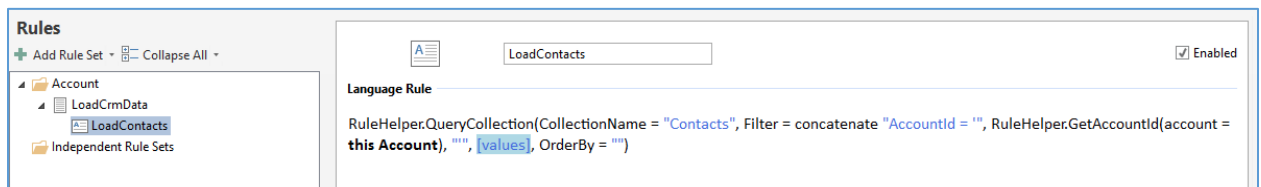
Note: The methods on the SalesforceDriver are designed to be reused for more than one Entity type, Field, or set of Fields. The name of the Target Field or Collection should be supplied as a string. When querying a Collection of results, an optional “where” clause can be provided that will be forwarded 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);
```

- 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 this example.

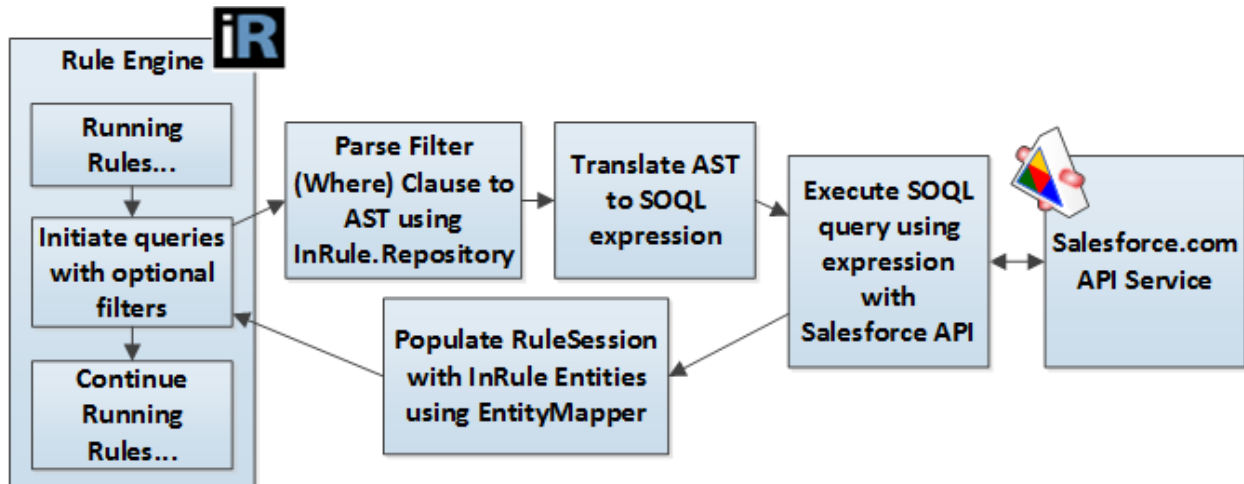
Filtering Queries using the Where Clause Builder

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.

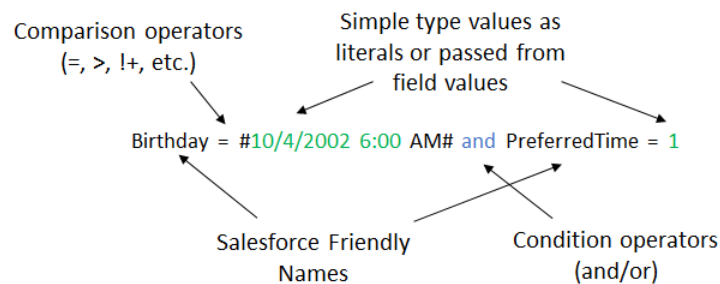
During execution of the SalesforceDriver, the filter clause is parsed into an Abstract Syntax Tree (AST) and then translated into SOQL (Salesforce Object Query Language) so it can be executed against the Salesforce data. The filter clause is based on the InRule function syntax format.

- 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:

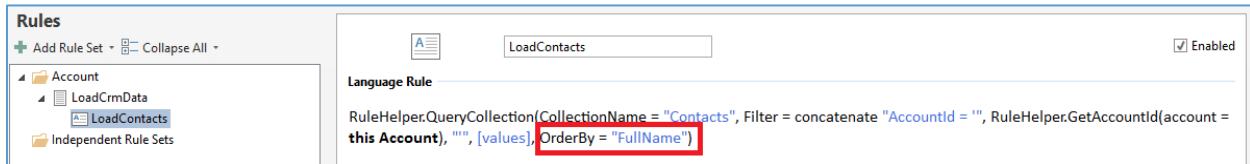


- All the Field names that are used are the Salesforce friendly names (Salesforce Field Label) that are used in the Rule Application. These names are mapped back to the Salesforce Field names in the parser.
- String literals should be wrapped in single quotes, date literals should be wrapped in pound 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`, `>`, `>=`, `<`, `<=`, `^`,

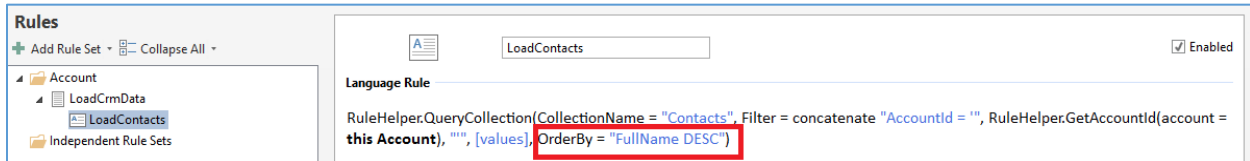
Ordering Query Results with the OrderByClauseBuilder

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 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 Dynamics 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 Dynamics, and a copy of the Entity is created. The Instance ID is not set to the GUID of the Dynamics Entity, which will also prevent changes to this entity from being written back to Dynamics. This functionality can, for example, be used to load the original values for an entity persisted in Dynamics 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.



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

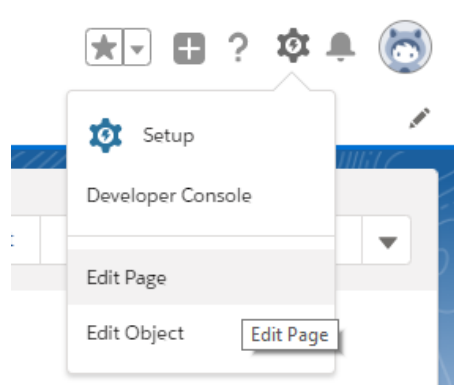
Appendix E: Methods for Executing Rules from Salesforce

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—five 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 triggers, and executing rules from Lightning Flow.

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

▼ Custom (0)

No components available.

▼ Custom - Managed (1)

 InRule_RunRules

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

▼ Custom (0)

[Deploy My Domain](#) to see custom components here.

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 with two available configuration values.

Page > InRule_RunRules

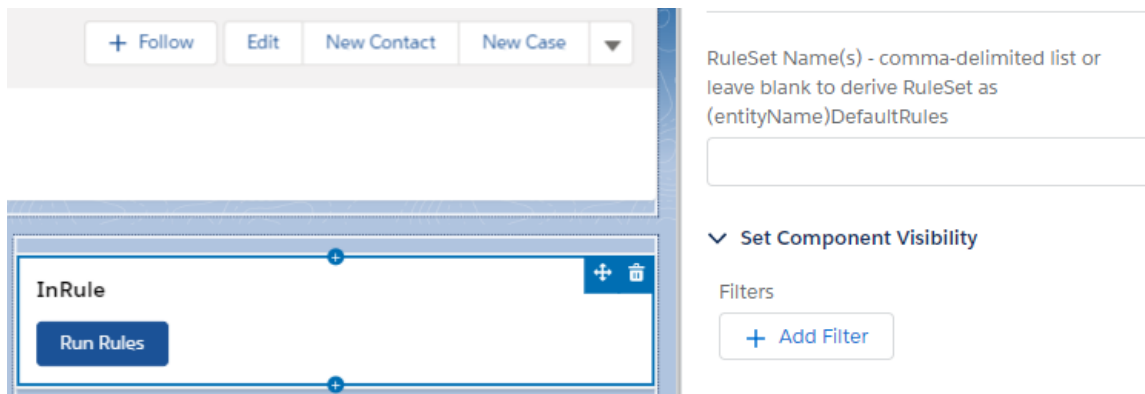
Rule Application Name - leave blank to use the default configured rule application

RuleSet Name(s) - comma-delimited list or leave blank to derive RuleSet as (entityName)DefaultRules

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

If no RuleSets are defined, will create 1 button that uses the default RuleSet for the entity:



If one RuleSet is defined, will create 1 button that will use that defined RuleSet:

The screenshot shows the InRule configuration interface. At the top, there is a toolbar with buttons: '+ Follow', 'Edit', 'New Contact', 'New Case', and a dropdown arrow. Below the toolbar, there is a large blue-bordered box labeled 'InRule' containing a 'Run Rules' button. To the right of the 'InRule' box, there is a text input field for 'RuleSet Name(s)' with the value 'Rule1'. Below this, there is a section titled 'Set Component Visibility' with a 'Filters' section containing a '+ Add Filter' button.

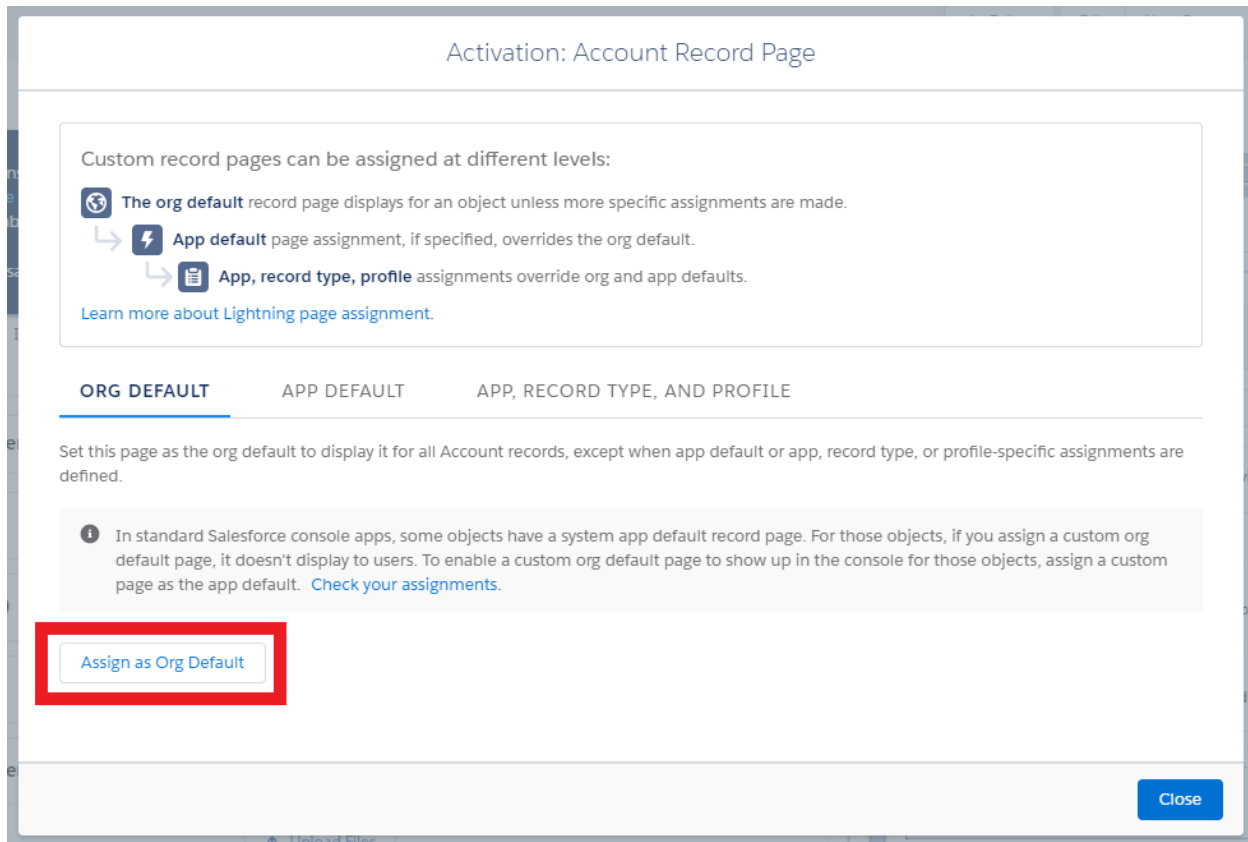
If 2+ RuleSets are defined, will create a button per RuleSet that will use the defined RuleSets.

The screenshot shows the InRule configuration interface with two RuleSets defined. The 'RuleSet Name(s)' field now contains 'Rule1, Rule2'. The 'InRule' box now contains two buttons, 'Rule1' and 'Rule2', instead of the 'Run Rules' button. The 'Set Component Visibility' section remains the same.

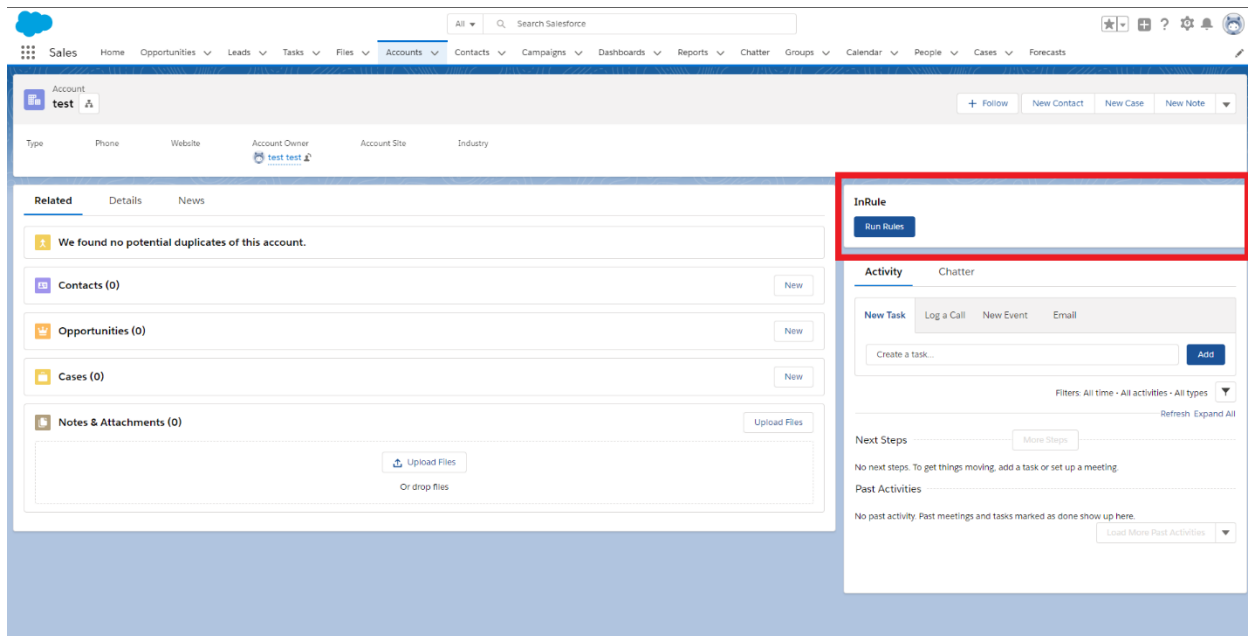
Once you have configured your RuleSets, click save in the top right corner. It will prompt you to activate this page to make it visible to your users, click activate.

The screenshot shows a 'Page Saved' dialog box. The title is 'Page Saved'. The main text says 'Activate this page to make it visible to your users.' Below this, it says 'Activate the page now, or do it later using the Activation button in the App Builder toolbar.' There is a green checkmark icon. At the bottom, there is a checkbox labeled 'Don't show me this message again' and two buttons: 'Not Yet' and '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.

Custom Button or Link Edit [Save] [Quick Save] [Preview] [Cancel]

Label:

Name: ⓘ

Description:

Display Type:
☐ Detail Page Link [View example](#)
☒ Detail Page Button [View example](#)
☐ List Button [View example](#)

Behavior: ▼ [View Behavior Options](#)

Content Source: ▼

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:

```

{!REQUIRESSCRIPT("/soap/ajax/33.0/connection.js")}
{!REQUIRESSCRIPT("/soap/ajax/33.0/apex.js")}
{!REQUIRESSCRIPT('/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 RuleSets configured as ‘Auto’ for that entity will be executed.
loggingLevel (optional, must be manually added to config object)	<p>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:</p> <ul style="list-style-type: none"> 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 <p>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.</p>
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.

<< [Back to List: Remote Site Settings](#)

[Contacts](#) [0] | [Opportunities](#) [0] | [Cases](#) [0] | [Open Activities](#) [0] | [Activity History](#) [0] | [Notes & Attachments](#) [0] | [Partners](#) [0]

Account Detail

[Edit](#) [Delete](#) [Include Offline](#) [Run Rules](#)

Account Owner	 User User [Change]	Rating
Account Name	Test Account [View Hierarchy]	Phone
Parent Account		Fax
Account Number		Website
Account Site		Ticker Symbol
Type	Account	Ownership
Industry		Employees
Annual Revenue		SIC Code
Billing Address		Shipping Address
Created By	User User , 4/29/2019 2:21 PM	Last Modified By User User , 4/29/2019 2:21 PM
Description	Updated via rules on 4/29/2019 4:21:43 PM	

[Edit](#) [Delete](#) [Include Offline](#) [Run Rules](#)



Contacts

[New Contact](#) [Merge Contacts](#)

[Contacts He](#)

No records to display

3 Executing Rules from Apex

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. This section will document how to run rules from your own Apex classes 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)

The method's 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 be will 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, as displayed in the example above.

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>)	<p>Provides a list of all notification message returned by rule execution. Each NotificationMessage has 2 properties on it:</p> <ul style="list-style-type: none">• 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>)	<p>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.</p> <p>Each ErrorMessage has 2 properties:</p> <ul style="list-style-type: none">• Source (String): At what point during runtime the error was thrown• Message (String): The error text

4 Executing Rules from Triggers

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

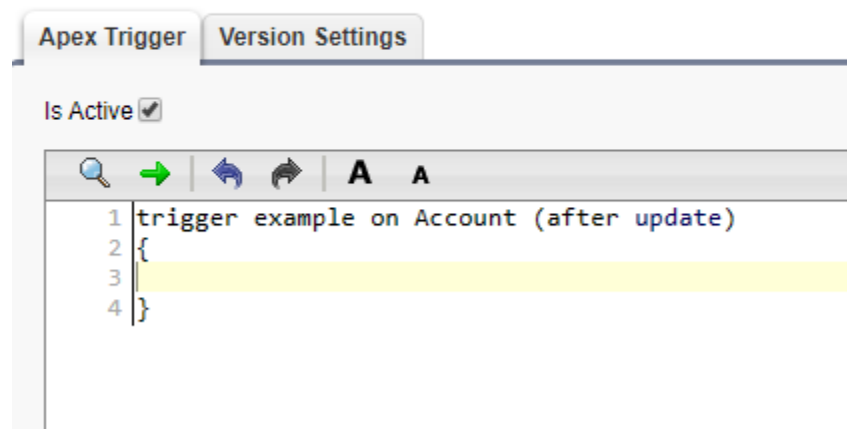
To emphasize, only **After Update** and **After Insert** triggers are supported. Due to Salesforce's enforcement of all external service methods running asynchronously, trigger execution and persistence to the Salesforce database will always complete before rule execution has had time to finish, thus 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.

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:

Apex Trigger

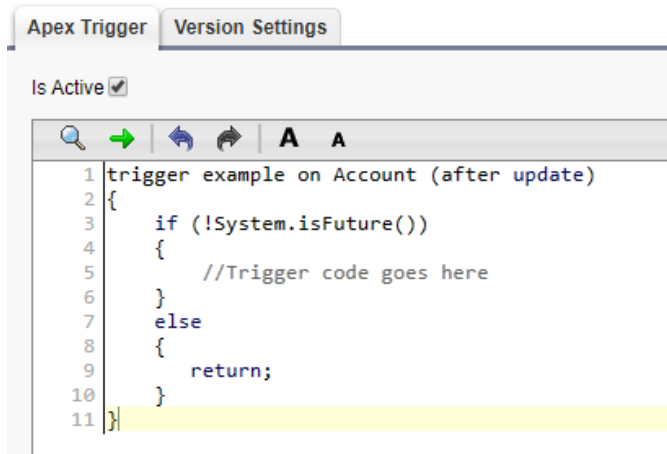
Apex Trigger Edit



Next, all rule executing triggers must contain the boilerplate wrapper around their code defined below. This is required to prevent endless recursions of triggers calling rules which then update fields on the entity image, which would then re-invoke the Update trigger.

Apex Trigger

Apex Trigger Edit

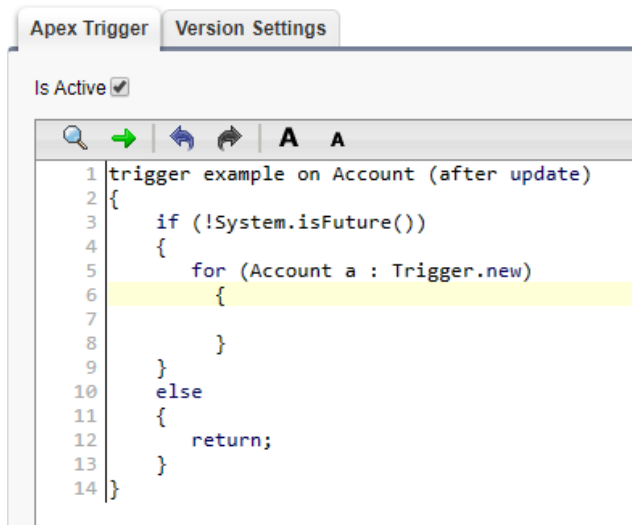


All operations, including calls to the rule execution service, will be contained within the **If (!System.isFuture())** block.

All triggers require defining whether you will be operating on the pre or post entity image (old/new, respectively). As we can only operate on the post image, we'll be using `Trigger.new`:

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 ☒


```

1 trigger example on Account (after update)
2 {
3     if (!System.isFuture())
4     {
5         for (Account a : Trigger.new)
6         {
7             inrule.DecisionClient.executeRulesFromTrigger('update', a.id, 'Account', 'AccountDefaultRules', false, null);
8         }
9     }
10    else
11    {
12        return;
13    }
14 }

```

The parameters for calling `executeRulesFromTrigger`, in order they fit into the signature:

Parameter Name	Description
Event Type	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)	<p>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, which may differ at that point from its state when trigger execution began due to the asynchronous nature of trigger execution.</p> <p>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.</p> <p>If you do not wish to pass the entity image, you do not need to define it as null like with the ruleAppName. You can simply leave it out of the execution method signature altogether.</p>

An example of serializing and passing an entity image:

Is Active ☒

```
1 trigger example on Account (after update)
2 {
3     if (!System.isFuture())
4     {
5         for (Account a : Trigger.new)
6         {
7             String entityImage = JSON.serialize(a)
8             DecisionClient.executeRulesFromTrigger('update', a.Id, 'Account', 'AccountDefaultRules', false, null);
9         }
10    }
11    else
12    {
13        return;
14    }
15 }
```

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:

Is Active ☒

```
1 trigger example on Account (after update)
2 {
3     if (!System.isFuture())
4     {
5         for (Account a : Trigger.new)
6         {
7             //Custom pre-rule execution code here
8             inrule.DecisionClient.executeRulesFromTrigger('update', a.id, 'Account', 'AccountDefaultRules', false, null);
9             //Custom post-rule execution code here
10        }
11    }
12    else
13    {
14        return;
15    }
16 }
```

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 ruleset 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:

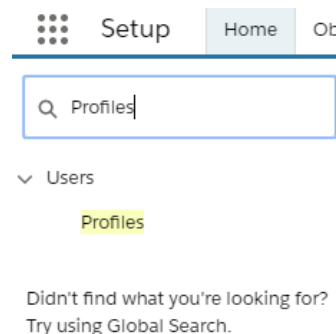
- Makes changes to entire profiles rather than specific users, meaning once it is done, all that will need to be done for future users is to assign them to a profile with these permissions already granted and they'll be set
- If you are already using predominately custom/cloned Salesforce profiles in 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

All Profiles ▾ Edit | Delete | Create New View

New Profile

Action	Profile Name ↑	User License
Edit Clone	Silver Partner User	Silver Partner
Edit Clone	Solution Manager	Salesforce
Edit Clone	Standard Platform User	Salesforce Platform
Edit Clone	Standard User	Salesforce
Edit Clone	System Administrator	Salesforce

Once on that profile's page, select Edit:

Profile
Standard User

Users with this profile have the permissions and page layouts listed below. Administrators can change a user's profile by editing that user. If your organization uses Record Types, use the Edit links in the Record Type Settings section below to make one or more record types:

[Login IP Ranges \(0\)](#) | [Enabled Apex Class Access \(2\)](#) | [Enabled Visualforce Page Access \(0\)](#) | [Enabled External Data Source Access \(0\)](#) | [Enabled Service Presence \(0\)](#)

Profile Detail

Name	Standard User
User License	Salesforce
Created By	InRule Developer 11/20/2019, 11:45 AM

[Edit](#) [Clone](#) [View Users](#)

Scroll down until you see the “Platform Event Permissions” header:

Platform Event Permissions		
	Basic Access	
	Read	Create
InRule_Events	<input type="checkbox"/>	<input type="checkbox"/>

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.”

Profile
Standard User

Users with this profile have the permissions and page layouts listed below. Administrators can change a user's profile by editing that user.

If your organization uses Record Types, use the Edit links in the Record Type Settings section below to make one or more record types available.

[Login IP Ranges \(0\)](#) | [Enabled Apex Class Access \(2\)](#) | [Enabled Visualforce Page Access \(0\)](#) | [Enabled External Data Source Access \(0\)](#) | [Enabled Service Presence Settings \(0\)](#)

Profile Detail

[Edit](#) [Clone](#) [View Users](#)

Name	Standard User
User License	Salesforce
Created By	InRule Developer 11/20/2019, 11:45 AM

Name your new profile appropriately:

You must select an existing profile to clone from.

Existing Profile	Standard User
User License	Salesforce
Profile Name	<input type="text" value="Standard User_clone"/>

[Save](#) [Cancel](#)

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:

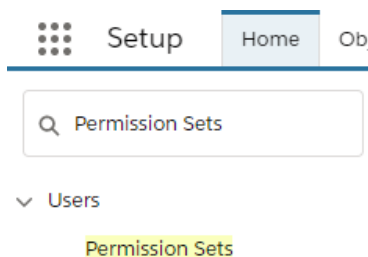
Pros:

- 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

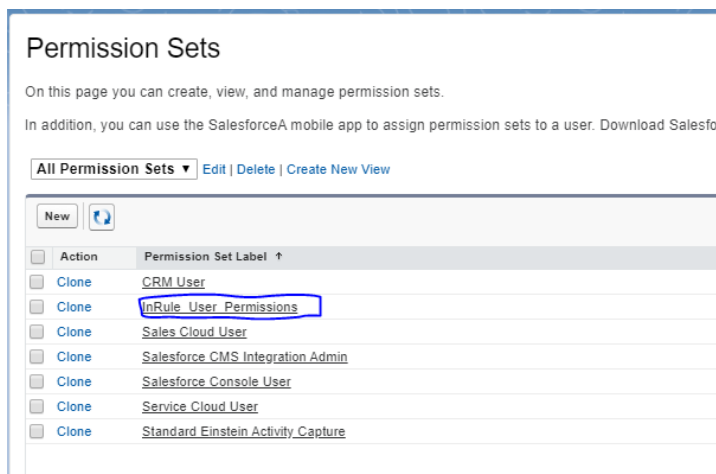
Cons:

- 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.

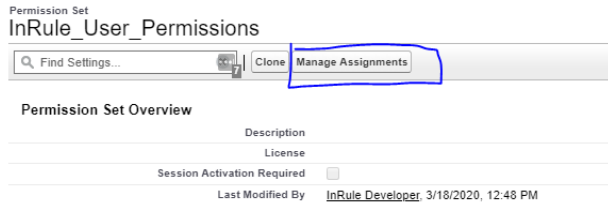
To add users to the InRule User Permission Set, go to Setup and search for Permission Set:



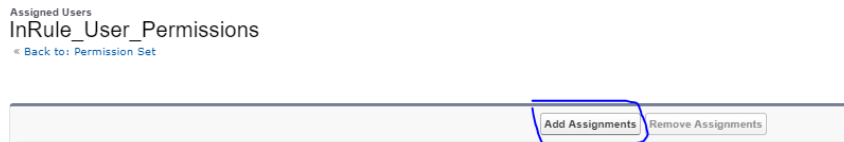
Find and select InRule_User_Permissions:



Select “Manage Assignments”



Select “Add Assignments”



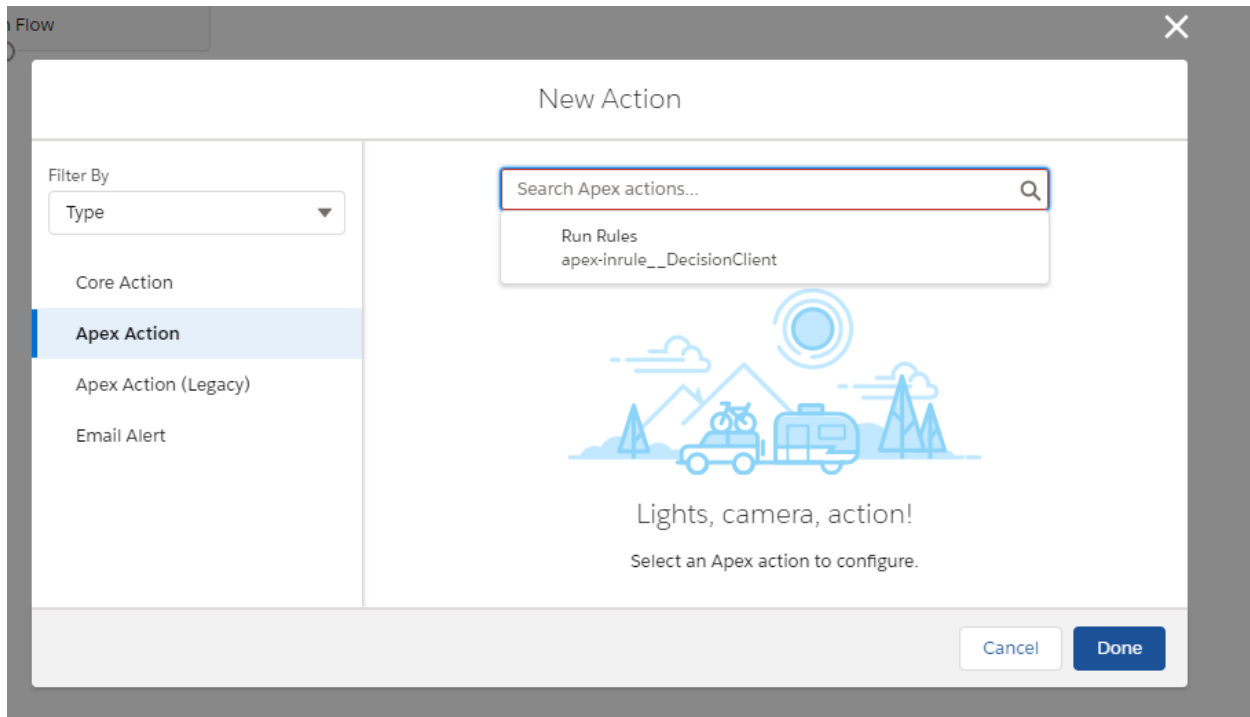
You'll be presented with a list of all users in your environment. You can select all users that will need the ability to initiate rule executing triggers. Once you've selected all the relevant users, you can select “Assign,” and the permission set will be assigned to all of those users.

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

5 Executing Rules from Lightning Flow

If you need to integrate rules with more complex process automation, rules can also be run from Lightning 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.

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



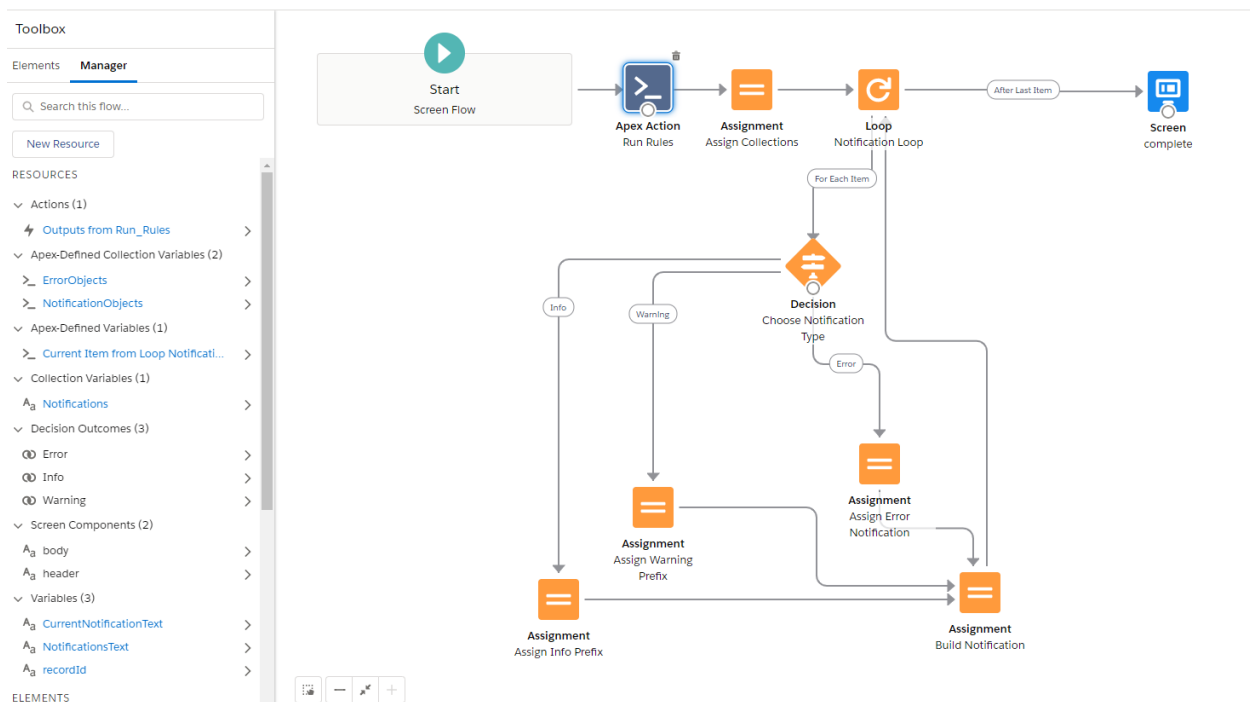
When adding the action, you will need to fill in the following required parameters

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 in to 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".
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"

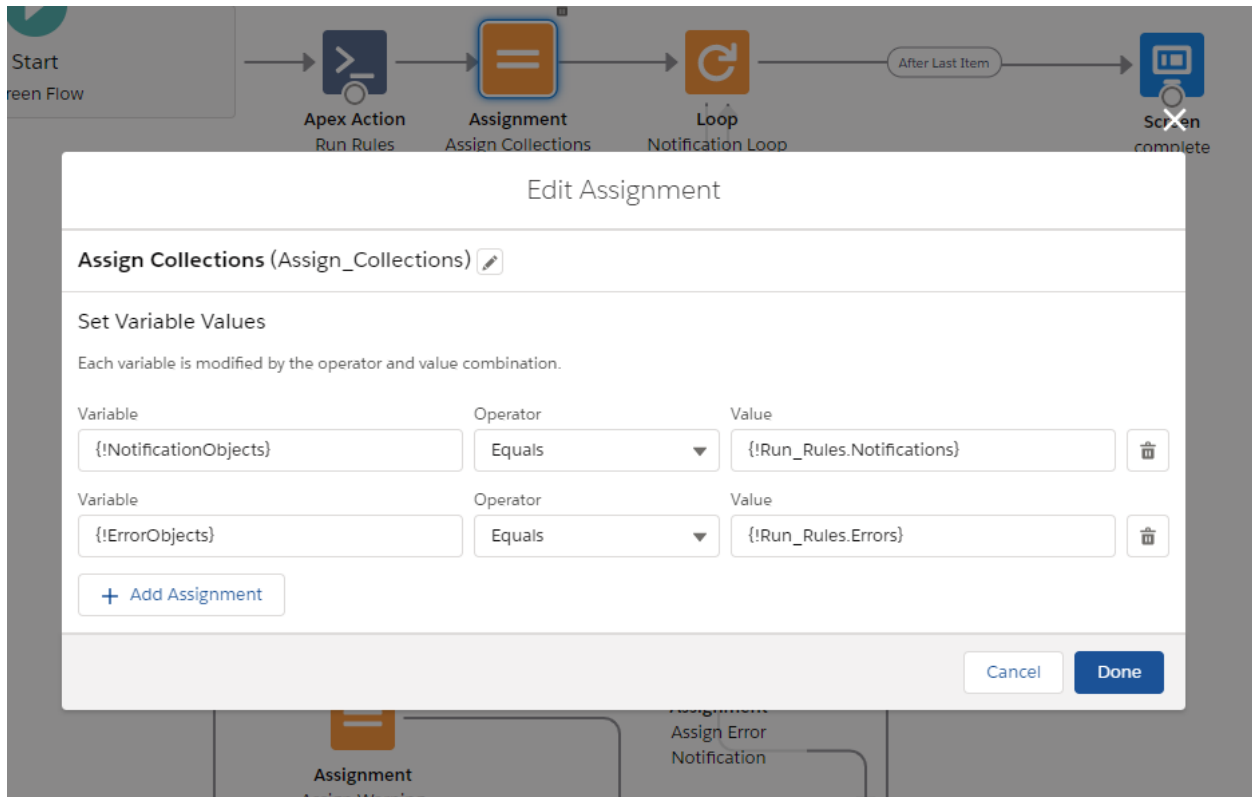
This action has three outputs

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: <ul style="list-style-type: none"> 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: <ul style="list-style-type: none"> 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

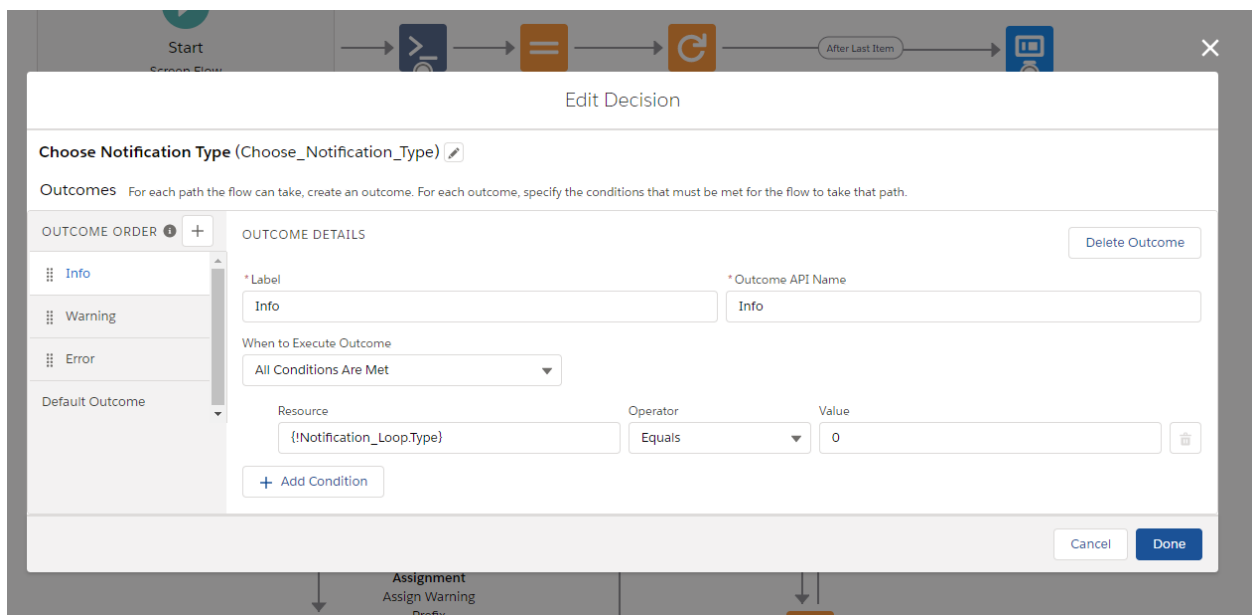
The following is an example of what a Flow using rules might look like. You can do other things with the output from the Apex action, but this demonstrates a common use case, displaying rule notifications to the user. The flow starts with Apex action, assigns the output collections to variables, loops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



Here the Notifications and Errors output are assigned to dedicated resource variables, so that they can be looped over later



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



Edit Assignment

Assign Info Prefix (Assign_Info_Prefix)

Set Variable Values

Each variable is modified by the operator and value combination.

Variable	Operator	Value
{!NotificationsText}	Add	Info:

+ Add Assignment

Cancel Done

The combined notification text value is then displayed on the screen

Edit Screen

Notifications

{!NotificationsText}

Pause Previous Finish

Display Text

* API Name

body

Insert a resource...

{!NotificationsText}

Salesforce Sans

12

Appendix F: Azure App Service Plan Configuration

Azure App Service Plan Overview

The Salesforce Rule Execution Azure App Service 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.

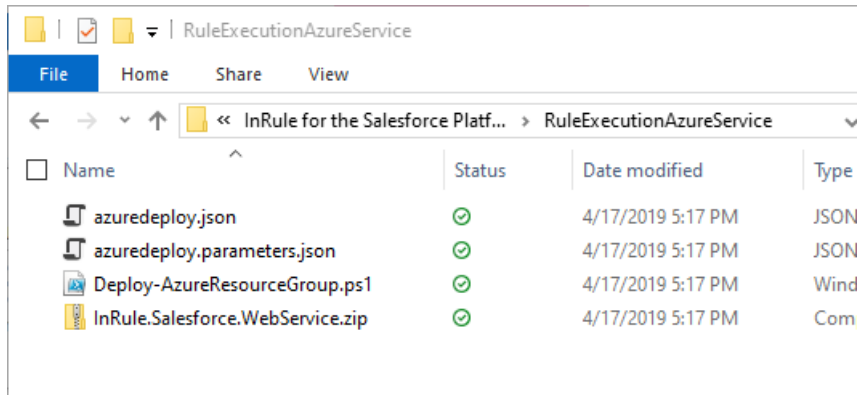
This App Service Plan will be deployed to the subscription and resource group provided during the deployment process. Additionally, the plan will be configured with a “F1” (Free) pricing tier, but this can be increased based on your scaling and configuration needs.

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 itself are necessary.

Configuring the ARM Template to Use an Existing Azure App Service

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**.

```
},
"appServicePlanName": {
  "value": "yourAppServicePlanName"
}
}
```

3: Create “createOrUpdateAppServicePlan” parameter

Next, just below the “appServicePlanName” parameter you will add an additional parameter called “createOrUpdateAppServicePlan” Detailed below (Note, be sure to add the pictured comma immediately following the first bracket after the “appServicePlanName” preceding parameter):

```
    },  
    "appServicePlanName": {  
      "value": "yourAppServicePlanName"  
    },  
    "createOrUpdateAppServicePlan": {  
      "value": false  
    }  
  }  
}
```

4: Create “servicePlanResourceGroupName” parameter

Lastly, we 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**.

```
    },  
    "createOrUpdateAppServicePlan": {  
      "value": false  
    },  
    "servicePlanResourceGroupName": {  
      "value": "yourResourceGroupName"  
    }  
  }  
}
```

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

Appendix G: Azure Application Insights Configuration

Azure App Service Plan Overview

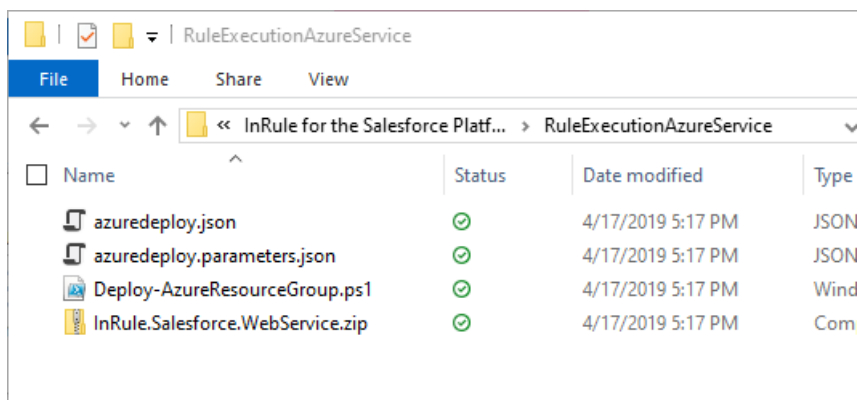
For improved logging capabilities, the Salesforce Rule Execution Service is configured to use an Azure App Insights resource as a logging sink in addition to the logging the App Service itself already has. ARM Template deployment process outlined in [Section 3.3.2: Rule Execution App Service for Salesforce](#)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

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

Configuring the ARM Template to Use an Existing App Insights resource

1: Locate InRule.Dynamics.Service.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 “appInsightsInstrumentationKey” parameter

Open the file in your text editor of choice. First, populate the “**appInsightsInstrumentationKey**” parameter at the bottom of the parameters file. Set the value equal to the instrumentation key of your App Insights resource.

```
"appInsightsInstrumentationKey": {  
  "value": "yourAppInsightsInstrumentationKey"  
},
```

3: Create “createOrUpdateAppInsightsResource” parameter

Next, just below the “**appInsightsInstrumentationKey**” parameter you will add an additional parameter called “**createOrUpdateAppInsightsResource**” detailed below and set its value to false:

```
"createOrUpdateAppInsightsResource": {  
  "value": false  
},
```

4: 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 associate the created app service with your existing app insights resource without creating a new one.

Appendix H: Endpoint Override Configuration

As of version 5.5, InRule for Dynamics 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 and go to the Configuration view:

+ New application setting Show values Advanced edit Filter

Name	Value	Source	Deployment slot setting	Delete
inrule:crm:catalog:label	Hidden value. Click show values button :	App Config		
inrule:crm:catalog:password	Hidden value. Click show values button :	App Config		
inrule:crm:catalog:ruleAppDirectory	Hidden value. Click show values button :	App Config		
inrule:crm:catalog:sso	Hidden value. Click show values button :	App Config		
inrule:crm:catalog:uri	Hidden value. Click show values button :	App Config		
inrule:crm:catalog:useInRuleCatalog	Hidden value. Click show values button :	App Config		
inrule:crm:catalog:user	Hidden value. Click show values button :	App Config		
inrule:crms2s:azureAppId	Hidden value. Click show values button :	App Config		

Select “New Application Setting”

Add/Edit application setting

Name

Value

The name of the override uses the following convention:

inrule:overrides:<YourEndpointNameHere>:<EndpointType>:<EndpointSetting>


Your endpoint name should match the name of the endpoint in the rule app you wish to override. The available endpoint types are:

- DatabaseConnection
- MailServerConnection
- WebServerAddress
- WebServiceWsdlUri
- WebServiceMaxReceivedMessageSize
- XmlDocumentPath
- XmlSchema

- XmlSchemaValidation
- InlineTable
- InlineXmlDocument
- InlineValueList
- SqlQuery
- RestServiceRootUrl
- RestServiceAuthenticationType
- RestServiceX509CertificatePath
- RestServiceAllowUntrustedCertificates

The endpoint setting is the name of the setting to override for that Endpoint Type. Some Endpoint Types have several Endpoint Settings; the available settings for each Endpoint Type can be read about in the [InRule support site documentation](#).


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

 **Important:** If an override type contains multiple settings, like RestServiceX509CertificatePath or RestServiceAuthenticationType, be sure to include an App Setting for each of the settings listed in the documentation.

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

Add/Edit application setting

Name	<input type="text" value="inrule:overrides:EndpointName:RestServiceAuthenticationType:RestServiceUserName"/>
Value	<input type="text" value="username"/>

 **Important:** A notable exception to this format type is for the RestServiceRootUrl endpoint type. If you wish to override a REST root URL, you will need to follow the following format instead:

inrule:overrides:<YourEndpointNameHere>:<EndpointType>

Below is what a properly configured RestServiceUrl override would look like:

Add/Edit application setting

Name	<input type="text" value="inrule:overrides:TestRestService:RestServiceRootUrl"/>
Value	<input type="text" value="https://8a388bfd.ngrok.io"/>

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.

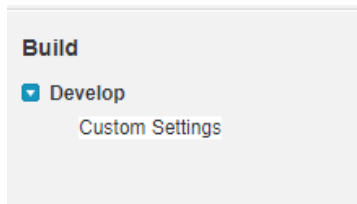
Appendix I: Salesforce and Rule Execution Service Event Logging

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**.

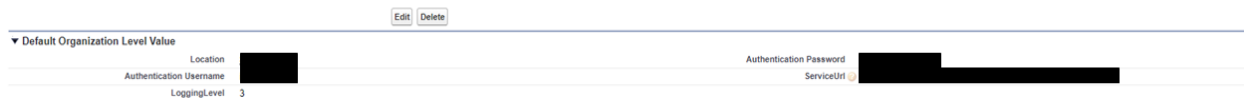


Once you're in the Custom Settings page, locate the InRule custom settings object and select "Manage"

Action	Label ↑	Visibility	Settings Type
Manage	InRule	Public	Hierarchy

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:

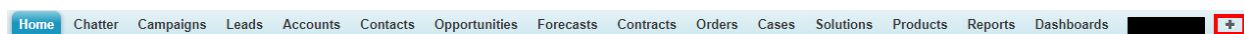
- 0 – Disables all logging
- 1 – Logs errors and a minimal amount of information on successful requests
- 2 – Logs errors, request information, rule engine notifications, and rule engine validations
- 3 -- Logs the same information as 2, but also includes JSON from the HTTP request and response payloads



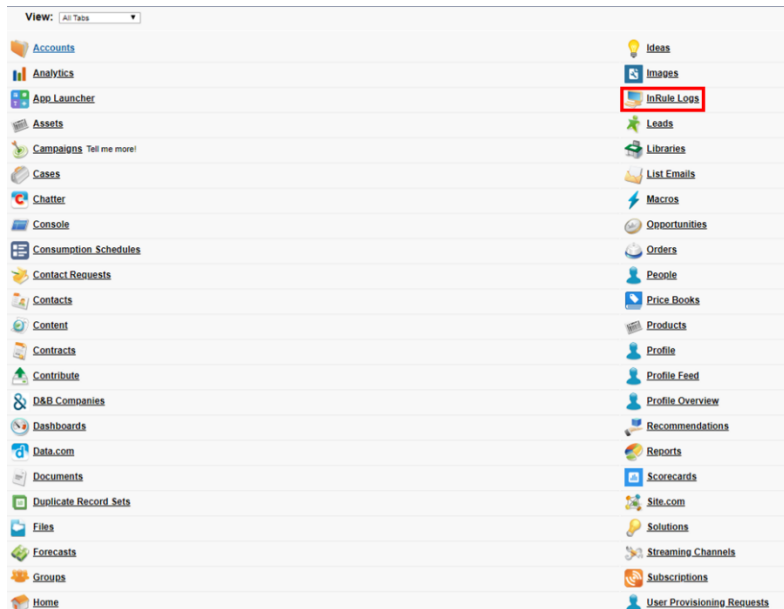
Once you have configured the LoggingLevel, select "Save."

Viewing InRule for Salesforce App Log

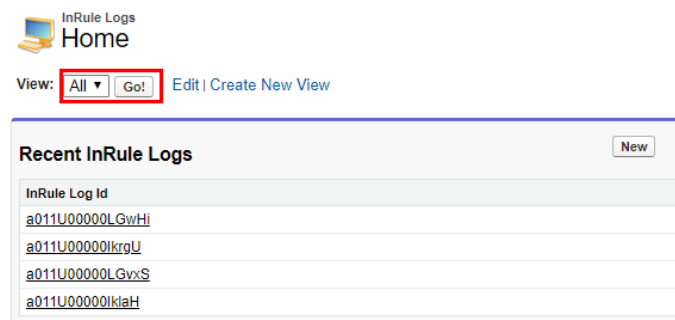
After installing the InRule for Salesforce App, a new custom tab to access the InRule for Salesforce App logs will have been added. To add it to your navigation bar, click the "+" at the end of the navigation bar.



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

App Service Diagnostics

Use App Service Diagnostics to investigate how your app is performing, diagnose issues, and discover how to improve your application. Select the problem category that best matches the information or tool that you're interested in:

Availability and Performance
Is your app experiencing downtime or slowness? Click here to run a health checkup to discover issues that may affect your app's high availability, by either platform or app issues.
Keywords: Health Check, Downtime, Six Errors, 4xx Errors, CPU, Memory

Configuration and Management
Are you having issues with something that you configured specifically for your app? Find out if you misconfigured App Service features, such as backups, deployment slots, and scaling.
Keywords: Scaling, Swaps, Failed Backups, IPs, Migration

SSL and Domains
Having trouble with certificates and custom domains? Discover any issues related to SSL certificates, authentication, and domain management.
Keywords: 4xx Errors, SSL, Domains, Permissions, Auth, Cert

Best Practices
Are you running your application in production? Review best practice recommendations to best ensure that you running your production application with the optimal configurations and suggestions.
Keywords: AutoScale, Traffic Manager, AlwaysOn, ARR Affinity

Diagnostic Tools
Sometimes deeper investigation is necessary. With Diagnostic Tools, you can run language-specific tools to profile your app, collect network traces, memory dumps, and more.
Keywords: Profiler, Memory Dump, DaaS, AutoHeal, Metrics, Security

Select Application Events

App Service Diagnostics Tools and Resources

ASP.NET | ASP.NET Core | Java | PHP

Sometimes deeper investigation is necessary. With Diagnostic Tools, you can run language-specific tools to profile your app, collect network traces, memory dumps, and more.

🔍 The stack used by your Web App was automatically detected to be **Static Only**. If incorrect, please select the correct option on the right.

- Diagnostic Tools



- Support Tools



- Premium Tools



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.

Level	Date Time ▲	Event Id	Source	Computer	
Level	Date Tim	Event I	Source	Computer	
Info	01/21/2019 20:12:13	2000	HttpPlatformHandler	RD00155D70EBC2	
Info	01/21/2019 20:12:13	2000	HttpPlatformHandler	RD00155D70EBC2	
Info	01/21/2019 20:12:12	2000	HttpPlatformHandler	RD00155D70EBC2	

Recycling application MACHINE/WEBROOT/APPHOST/~

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 provide 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:

The screenshot displays the Azure Portal interface for an App Service. On the left, the 'Settings' menu is expanded, and 'Application settings' is highlighted with a red box. The main content area shows the 'Overview' tab for the App Service. Key details include: Resource group (IntegrationsTeam), Status (Running), Location (North Central US), Subscription (Engineering-DevTest), and Subscription ID (cf3242e8-7c98-4b60-a714-3be7b91c5df1). The 'Tags' section shows 'Billable: UpdateMe', 'CreatedBy: YourName', 'CreationDate: YYYY-MM-DD', 'Department: Engineering', and 'Project: UpdateMe'. Below the overview, there are three tiles: 'Diagnose and solve problems', 'Application Insights', and 'App Service Advisor'.

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:stap:consumerKey	 Hidden value. Click show values button above to view
inrule:stap:consumerSecret	 Hidden value. Click show values button above to view
inrule:stap:loginUrl	 Hidden value. Click show values button above to view
inrule:stap:password	 Hidden value. Click show values button above to view
inrule:stap:securityToken	 Hidden value. Click show values button above to view
inrule:stap:username	 Hidden value. Click show values button above to view
inrule:stap:catalog:label	 Hidden value. Click show values button above to view
inrule:stap:catalog:password	 Hidden value. Click show values button above to view
inrule:stap:catalog:ruleAppDirectory	 Hidden value. Click show values button above to view
inrule:stap:catalog:iso	 Hidden value. Click show values button above to view
inrule:stap:catalog:uri	 Hidden value. Click show values button above to view
inrule:stap:catalog:useInRuleCatalog	 Hidden value. Click show values button above to view
inrule:stap:catalog:user	 Hidden value. Click show values button above to view
inrule:stap:ruleService:basicPwd	 Hidden value. Click show values button above to view
inrule:stap:ruleService:basicRealm	 Hidden value. Click show values button above to view
inrule:stap: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

For more detailed, technical explanations of what is included with each logging level, please reference the [Runtime Event Log Details documentation on the support site.](#)

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



Click here to upgrade to a higher SKU and enable additional features.

New application setting Show values Advanced edit Filter

Name

Value

inrule:logging:level

Hidden value. Click show values button above to view