IBM Automation Decision Services 23.0.2

Tables of Contents

Automation Decision Services

Benefit from a comprehensive set of decision automation features to capture and automate repeatable decisions.

Release notes

The release notes contain information that is important for the successful use of Automation Decision Services.

- **What's new**
  Discover the new features and other improvements in Automation Decision Services 23.0.2.
- **Privacy and compliance**
  Regularly check compliance with security configuration goals to help you identify configuration issues.
- **Known limitations**
  This topic describes known limitations in the current version and provides workarounds if available.

What's new

Discover the new features and other improvements in Automation Decision Services 23.0.2.

Develop

Extended support for Predictive Model Markup Language (PMML) files
: Connect to the newly available embedded machine learning provider and import any type of PMML file to implement your predictions. Learn more...

Execute

Emit execution information to an external Kafka instance
: You can now configure the decision runtime to emit events that contain technical and business data to an external Kafka instance. You can use this event emission for various purposes, such as real-time monitoring and data analytics. Learn more...

Execution Java API must be used with Java 17
: The execution Java API must now be used with an environment with Java 17. Learn more...

Privacy and compliance

Regularly check compliance with security configuration goals to help you identify configuration issues.

- **Guidelines for GDPR readiness**
  Protecting your data can help you comply with the General Data Protection Regulation (GDPR). GDPR establishes a regulatory framework for processing personal data.
- **FIPS compliance**
  IBM Automation Decision Services cannot claim to be FIPS-compliant.

Guidelines for GDPR readiness

Protecting your data can help you comply with the General Data Protection Regulation (GDPR). GDPR establishes a regulatory framework for processing personal data.

Notice:

Use the following sections to help you prepare for GDPR readiness. Each section provides information about the features of Automation Decision Services that you can configure, and aspects that you can consider to help your organization with GDPR readiness. The information is not an exhaustive list, due to the many ways that features can be configured, and the large variety of ways that the products and containers can be used.

When you install and use Automation Decision Services, you are responsible for ensuring that it is compliant with various laws and regulations, including the GDPR adopted by the European Union in May 2018. Obtain the advice of competent legal counsel as to the identification and interpretation of any relevant laws and regulations that might affect your business. Take the resulting actions to comply with such laws and regulations.

IBM® does not provide legal, accounting, or auditing advice, or represent or warrant that its services and products ensure compliance with any law or regulation. The responsibility to ensure that an IBM product is compliant with laws and regulations is on the company that purchases the product. Always consider how protected your data is and whether it is ready for GDPR.

Table of Contents

GDPR

General Data Protection Regulation (GDPR) has been adopted by the European Union ("EU") and applies from May 25, 2018.

Why is GDPR important?

GDPR establishes a stronger data protection regulatory framework for processing of personal data of individuals. GDPR brings:

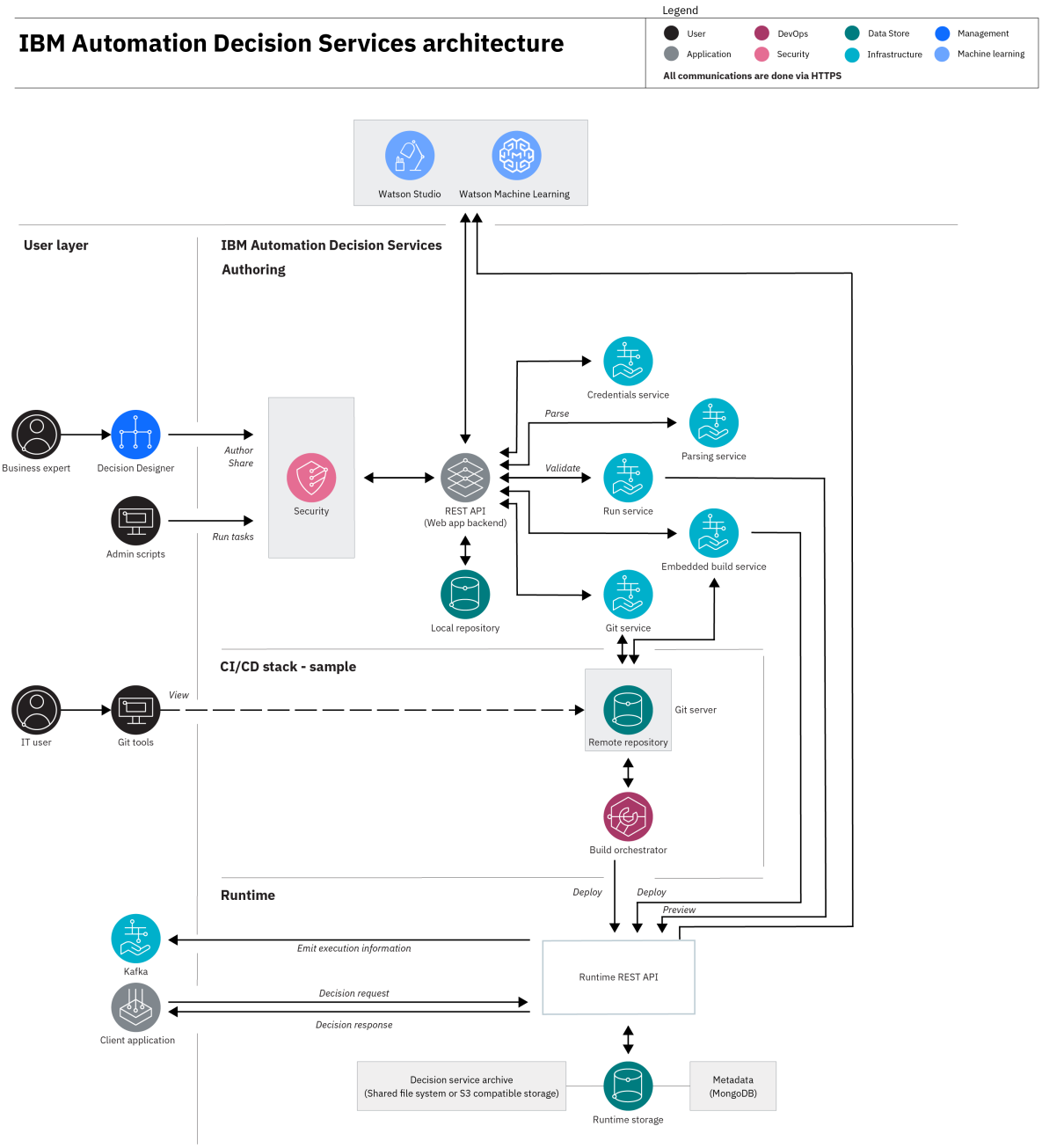
- New and enhanced rights for individuals
- Widened definition of personal data
- New obligations for companies and organizations handling personal data
- Potential for significant financial penalties for non-compliance
- Compulsory data breach notification

Read more about GDPR

- ibm.com/GDPR website

Security architecture

Automation Decision Services provides a comprehensive environment for authoring, managing, and running decision services.
The following diagram shows the architecture and communications of Automation Decision Services.



The sample CI/CD stack is provided with Automation Decision Services. If you decide to use the sample CI/CD stack, you need to install, configure, and manage it, and make sure to harden the security around the CI/CD stack as it is provided only as an example.

Data collection

In general, data that is used for authentication must be in a directory service or LDAP. Databases are provisioned during installation and store decision services. The databases evolve with the development and deployment of the decision services. Make sure to maintain them throughout product lifecycle:

- Make sure to maintain the databases throughout the lifecycle of the Automation Decision Services product use.
- Regularly back up data, according to your business needs and to the risk level.
- When data is no longer used, delete the databases or archive them for future use.
- As a data controller, provide means to satisfy data access requests for personal information or other compliance requests.

User IDs are used across Automation Decision Services to uniquely identify users, their decision automations and activities, including in the logs. Care must be taken if the user IDs are personally identified information. In addition, individual email addresses are used for Git operations (commits) in the local repository, and are sent to the remote repository.

Secrets

IT administrators are responsible for the configuration and the security of the cloud cluster. In particular, an administrator is the person to provide and manage the secrets that are used for all of the deployments. In Automation Decision Services, the container images do not include any secrets. Before deployment, the administrator must review the content of the custom resource file and provide all of the Kubernetes secrets that are needed for the deployment.

Typically, secrets are needed for user credentials, passwords, certificates, encryption keys, API keys, tokens, and other code. After the secrets are created, the containers can be installed. The administrator must take note of these secrets, store them in a safe place like a password manager or encrypted files, and then manage them with the proper attention.

Data storage

The databases and LDAP provisioned by the customer should be protected by using appropriate security controls.

Data access

Implement protective measures concerning data access:

- Make sure that control of access to databases is in place and effective.
- Use HTTPS or equivalent secure communication protocols for all the connections.
- Automation Decision Services provides APIs to access project data and decision runtime services. Consider implementing certain protections, including:
  - Use of basic authentication or other authentication methods.
  - Proper authorization, so that only authorized roles can use the corresponding API.

Data processing

A new endpoint becomes available for each decision service that is deployed. Decision services are invoked by passing payloads.

Consider implementing the following security guidelines when invoking the REST APIs with Automation Decision Services:

- HTTPS with secure ciphers should be used.
- The input payload should be properly sanitized.
- The output from Automation Decision Services should be manipulated carefully, although it has been sanitized by Automation Decision Services.
- The security infrastructure should protect against DOS attacks.

Data monitoring

Make sure to regularly test, assess, and evaluate the effectiveness of your technical and organizational measures to comply with GDPR. These measures should include ongoing privacy assessments, threat modeling, centralized security logging, and monitoring among others.

Data deletion

Article 17 of the GDPR states that data subjects have the right to request that their personal data be removed from the systems of controllers and processors, without undue delay. Implement appropriate controls and tools to satisfy this right.

Automation Decision Services does not require any special method for data deletion, providing that it is secure.

Data that reflects personally identifiable information (PII) can be in all stages of the data processing pipeline. Data deletion must include all these stages.

Responding to requests from individuals

The personal data that is stored and processed by Automation Decision Services falls into one or more of the following categories:

- Basic personal data, such as names, usernames, and passwords.
- Technically identifiable personal information, such as IP addresses and hostnames to which user activity might potentially be linked.
- Personal data that could be stored in the text of the rules. Rule authors should control the rules they write.

This data is key for efficient operation of an automated system. Consider and implement secure and lawful responses to the following requests:

- Delete data.
- Correct data.
- Modify data.
- Extract specific data for export to another system.
- Restrict the use of the data within the overall system securely and responsibly.

FIPS compliance

IBM Automation Decision Services cannot claim to be FIPS-compliant.
Federal Information Processing Standards (FIPS) includes compelling security and interoperability standards when no acceptable industry standards or solutions exist. The National Institute of Standards and Technology (NIST) defines FIPS.

The status of the FIPS 140-2 module is Historical. For more information, see CMVP Historical Validation List. FIPS 140-3 is the relatively new US and Canadian federal standard, and Automation Decision Services is working on its support of this module.

Important: Agencies and institutions who use these standards must assess the risk of where and how Automation Decision Services is used before they decide to install it.

Known limitations

This topic describes known limitations in the current version and provides workarounds if available.
For the most up-to-date information, see the support page IBM Automation Decision Services Known Limitations, which is regularly updated.

Setting up your Automation Decision Services environment

Feature	Limitation	Comment or workaround
Installing Automation Decision Services	The embedded MongoDB database is not highly available (HA).	The IT user can provide a remote MongoDB database, and set the MongoDB connection string URI to the remote MongoDB. For more information about configuring the MongoDB secret, see Configuring MongoDB storage.
	The embedded MongoDB persistent volume requires a fully POSIX-compliant storage.	Make sure to use a storage class that is fully POSIX-compliant. An azure-file storage, for example, does not meet this requirement.
Using the sample CI/CD stack	Some container images run as root user.	None

Managing decisions

Feature	Limitation	Comment or workaround
Importing decision services	Decision service archives that were compressed using the default compression tool on your computer cannot be opened in Decision Designer.	Import decision service archives created with the Export feature. For more information, see Managing decision services.
	When you import a decision automation that was initially created and exported from the same Decision Designer instance you are working in, the predictive models it contains will run even if they are not connected to a machine learning provider.	The provider details are found in the original decision automation settings at runtime.
Sharing changes	When you share a large decision service, an error might occur (HTTP response code 500).	The error is thrown when the operation has not completed on time. Check the Share changes tab after 2 minutes. If there are still pending changes, contact your administrator. To avoid running into this error, try to share your modifications often when you build a large decision service.
	You cannot upload a file for a decision service that is larger than 64 MB to a Git server.	Configure the maximum file size upload by setting the spec.decision_designer.git_service.max_upload_file_size parameter in the git-service.
Upgrading decision services	Upgrading a decision service fails when both of the following conditions are met: • The decision service uses external libraries. • The external libraries are missing when the decision service is upgraded.	Include the missing external libraries in the decision automation before the upgrade. For more information about using external libraries, see Importing an external library into.

Modeling decisions

Feature	Limitation	Comment or workaround
Building an external library	When you build an external library, no warning is displayed if: • Methods that cannot be used in rule conditions in decision models (that is, they are not pure functions) are verbalized. Note: These methods can still be used in rule actions and in task models. • Instances of a type cannot be copied. This might happen in rules in decision models.	If the methods that you want to use are a pure function, you can declare them as such by adding a Java or external annotation. For more information, see Pure functions.
Importing an external library into Decision Designer	If your decision service contains an external library with a different locale than the decision service, an error occurs in the data model and no data type is available.	1. Remove the dependency to the external library. For more information, see Declaring and managing dependencies. 2. Import the external library again. For more information, see Importing an external library into.
Creating a decision model	If the decision is multi-valued or if you selected the first rule applies policy, you cannot add a decision node with the same name as an attribute that is used in its decision logic.	Choose a different name for the decision node, or set a different output variable name.
Adding the decision logic	When you use the `copy of ... where ...` statement in a rule, an error occurs if one of the specified values must be converted (for example, from list to array).	None
	Static attributes from external libraries can be modified without restrictions, even though they should not.	None
Composing and reusing models	When you update a decision model that is used as a function in another decision model, the changes are not reflected in the decision model where it is integrated.	Reset the type of the function node.
	When you update a task model that is used as a function in another task model, the changes are not reflected in the task model where it is integrated.	Remove the function task node and add it again.

Integrating machine learning

Feature	Limitation	Comment or workaround
Managing machine learning providers	Watson™ Machine Learning functions cannot be used in Automation Decision Services.	None
	You cannot connect to Watson Machine Learning that is hosted on IBM Cloud Pak for Data.	None

Building and deploying decision services

Feature	Limitation	Comment or workaround
Building and deploying in a CI/CD stack	You cannot run unit tests in a CI/CD stack on predictive models or other models that call predictive models because credentials cannot be passed.	None
Building and deploying from Decision Designer	Custom POM files are ignored in the embedded build service.	None

Executing decision services

Feature	Limitation	Comment or workaround
Calling decision services	Users with the `decision service manager` and `decision runtime monitor` roles are also granted the `decision service user` role and can execute decision services.	None

Overview

IBM Automation Decision Services is an automation solution that helps business users capture and automate repeatable intelligent business decisions. It enables flexible and scalable deployment of automated decisions running on premises or in the cloud.
Organizations make numerous operational decisions every day as part of their business operations. The scope of these decisions can include areas such as product and service pricing, and credit and loan approval, as well as risk management and fraud detection.

Operational decisions apply business policies, which are often influenced by numerous factors that can be both internal and external to an organization. Today's ever-changing business climate requires business users to directly control their decision-making processes, and the data driving these decisions. They must also make their processes accessible immediately to their enterprise applications.

Automation Decision Services provides a fully-fledged decision modeling environment where business users can take control of the decision-making process and initiate and build enterprise-scale decision automations.

Automation Decision Services comprises the following two components:

Decision Designer
: A web-based, low-code application where business users can model, test, and deploy decisions. Decision Designer provides all the tools business users need to model complex business decisions, such as decision models and ruleflows, and define the data that governs these decisions.
In Decision Designer, business users can also infuse their decisions with predictions delivered by machine learning models. Combining business decisions and predictions allows business users to make better, more informed decisions bases on data already available in their organization.

Decision runtime
: A container-based execution server where decisions can be deployed and executed at scale. The decision runtime offers APIs for executing decisions and managing decision service resources: run specific versions of decisions, manage deployment spaces and decision metadata, troubleshoot executions, and much more.

Read the following topics for more details about the Automation Decision Services features.

- **Modeling and authoring business decisions**
  The intuitive and graphical user interface of Decision Designer brings together all the tools business users need to model complex business decisions.
- **Collaborating on decision services**
  Automation Decision Services provides enhanced collaboration workflows and greater control over decision files.
- **Infusing business decisions with machine learning**
  The data of your organization contains highly valuable information that you can use to help you make better decisions. You can integrate predictive models with decision models to gain insight into your data and make business decisions with greater accuracy.
- **Deploying and executing decision services**
  When a decision service is ready for deployment, it can be securely deployed through a continuous integration and delivery (CI/CD) stack before being rolled out to production. Alternatively, you also can deploy decision services directly from Decision Designer.

Modeling and authoring business decisions

The intuitive and graphical user interface of Decision Designer brings together all the tools business users need to model complex business decisions.
Business users develop and validate decision services by using Decision Designer, a graphical user interface that provides powerful capabilities along with intuitive menus and wizards. In Decision Designer, business users can use three decision artifacts to develop their decision services: data models, decision models, and task models.

Data model

Business decisions apply to real-life data such as loans, cars, or customers. These real-life data are generally referred to as business concepts. In Automation Decision Services, you describe the data that governs your business activity in a data model. Creating a data model vocabulary allows you to populate decisions in your own language and puts you in control of the business logic.

The data model vocabulary can be easily created by using text fields and drop-down menus to list the specific characteristics and attributes of your business objects. For example, you describe a car that has a model, a category, and a registration year. These attributes have values in the real world, and decisions depend on these values.

The data model vocabulary can be used across a decision service, in the different models it contains.

Decision model

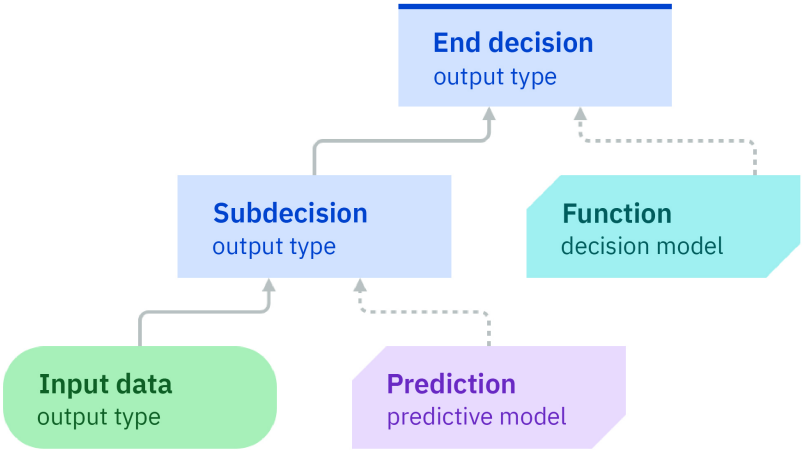
Decision modeling is a straightforward and low-code approach to express and refine operational decisions through a structured, visual representation of a decision. Using this approach, business users can model operational decisions by specifying:

- The information that is required to make the decision.

- How to make the final decision with that information – the decision logic.

Each decision model contains a decision diagram. Diagrams provide an abstract, high-level representation of how decisions and the data that is required to make these decisions are structured and related to each other. Creating diagrams is an iterative process where you decompose the decision that you want to make into smaller decisions. Decision decomposition helps to make the decision model simpler and easier to manage.

Diagrams are composed of a set of nodes that are used as building blocks to represent decisions in a graphical way:



The drag-and-drop feature available in the modeling environment helps you create and arrange nodes in a logical flow and draw dependencies from node to node.

Each decision in the diagram is associated with a logic. The decision logic is implemented as a set of rules that generate a result based on one or several input data. In other words, the decision logic precisely defines how a particular decision is made:

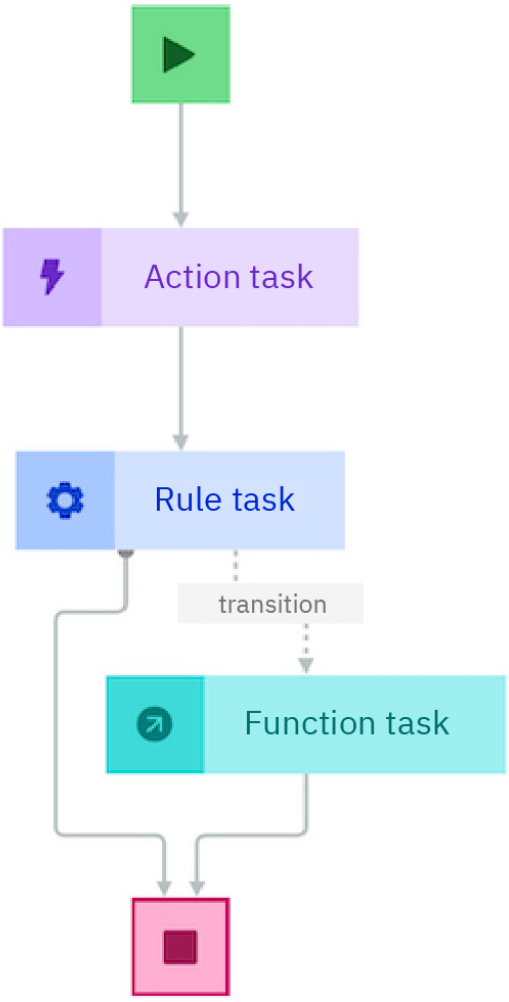
1. It evaluates the data available as input against a set of conditions that you define.
2. It takes action to produce the output of the decision, depending on whether these conditions are satisfied.

Task model

Task models offer business users a more advanced way to define decisions.

In a task model, the decision logic is defined at the root of the model, outside of a diagram. It is implemented as sets of business rules and decision tables that can be organized and grouped in folders. Each task model contains at least one ruleflow to control the execution of rules.

Whereas a decision diagram represents the dependencies between decisions and input data, a ruleflow represents a sequence. A ruleflow chain together tasks, and specifies how, when, and under what conditions they are run. It consists of several task nodes that are connected by logical links:



The task nodes of a ruleflow contain the instructions for what to execute and in what order. Depending on their type, task nodes can contain:

- Sets of business rules and decision tables to be executed at that point in the ruleflow, and in a specific order.
- Rule action statements to be executed.
- A reference to another ruleflow that is contained in the same task model.
- A reference to another model that is contained in the same decision service. It can be any type of model: decision model, predictive model, or task model.

The ruleflow editor palette helps you create and arrange ruleflow elements. The drag-and-drop feature available in the ruleflow editor allows you to easily draw transitions from element to element.

Related concepts

- Defining data
- Building decision models
- Building task models

Collaborating on decision services

Automation Decision Services provides enhanced collaboration workflows and greater control over decision files.
Automation Decision Services allows different users to collaborate on decisions across users, departments, and disciplines.

Collaboration between business users

Collaboration between business users is essential to the success of decision services. In Decision Designer, they can work together on a decision service by sharing and managing decisions in the same repository. Collaborators work locally in the application and can decide when to share their changes to make them available to others.

Working in a shared space not only makes it easier to track changes made by other users, it also simplifies the management of decision automation versions and the resolution of conflicts that are caused by competing changes to a file.

Collaboration between business users and IT users

Automation Decision Services relies on close collaboration between business users and IT users to simplify the implementation of decisions and reduce time to deployment.

Decision development and deployment activities are distributed between these two roles, as shown in the following table.

Table 1. User roles for the development of decision automations

Role	Activities	Description
Business user	Model Author Run Share Load	Business users have the following responsibilities: • Set up the data model that is used as the rules vocabulary. • Model decisions and author the business logic. • Embed machine learning models. • Run decisions to ensure that they yield the expected results. • Share changes to make their work available to collaborators. • Load changes made by collaborators.
IT user	Install Configure Manage	IT users have the following responsibilities: • Install and configure the environment for Automation Decision Services. • Configure credentials. • Configure and manage a CI/CD stack to automate builds and deployments. • Set up the build environment. • Manage user access to decision automations.

Infusing business decisions with machine learning

The data of your organization contains highly valuable information that you can use to help you make better decisions. You can integrate predictive models with decision models to gain insight into your data and make business decisions with greater accuracy.
Predictive analytics and machine learning provide valuable insights that can help businesses make more effective decisions. For example, machine learning algorithms can help identify at-risk customers, fraudulent claims, failing machines, competitive orders, and cost-effective suppliers. However, these insights must be acted upon to provide value to the organization. This can be achieved with the help of decision automation.

Using machine learning with decision modeling allows for better insights and predictions about business. Machine learning and rule-based decision modeling are complementary approaches to making decisions: while rule-based decision modeling takes a deterministic approach to making decisions, machine learning takes a probabilistic approach.

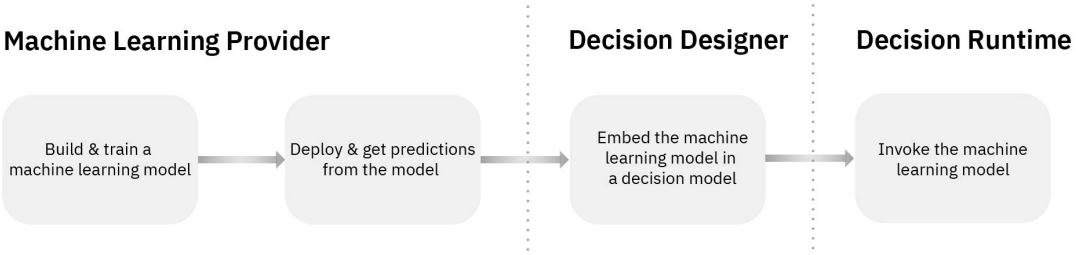
Take the example of a loan application. A decision model would tell you whether borrowers might pay off a new loan based on a set of known inputs, such as their income or their credit score. A machine learning model would give you wider insights into how much the borrowers are likely to repay or the probability of the borrowers defaulting.

The complementarity of rule-based decision modeling and machine learning allows you to get the best of both worlds: predictive insights from historical data and prescriptive business decisions based on company policies.

Automation Decision Services takes the complexity out of combining predictions and decisions, creating a sustainable, agile process that can be managed as business conditions change.

Remote machine learning models

You can use machine learning predictions from a variety of sources. Take advantage of Automation Decision Services' native integration with IBM Watson® Machine Learning to discover and import machine learning models directly into Decision Designer. Alternatively, you can connect to machine learning providers that are not naively supported by Automation Decision Services, such as Amazon SageMaker, Microsoft Azure Machine, Google Vertex AI, or custom services by using the IBM Open Prediction Service API.



Setting up the connection to a machine learning provider can easily be done from the Decision Designer interface. When a machine learning model is available in Decision Designer, you can prepare it for consumption with a step-by-step wizard and embed it into a decision model.

Local machine learning models

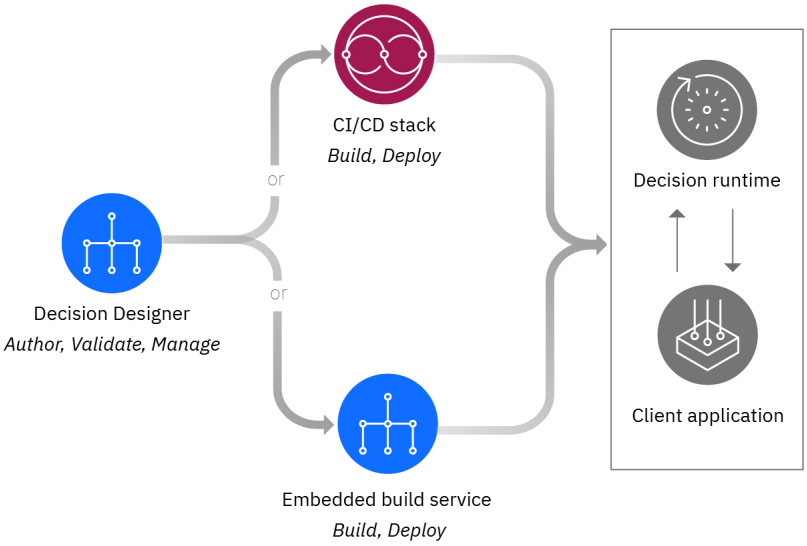
In Decision Designer, you can import ruleset and scorecard machine learning models in PMML (Predictive Model Markup Language) format and turn them into transparent business rules or decision tables. These rules can be easily understood and managed by business users. They are governed and executed directly in Automation Decision Services.

Related concepts

- Integrating machine learning

Deploying and executing decision services

When a decision service is ready for deployment, it can be securely deployed through a continuous integration and delivery (CI/CD) stack before being rolled out to production. Alternatively, you also can deploy decision services directly from Decision Designer.
Multiple deployment options are available to help you execute decisions at scale where you need them:



Building and deploying in a CI/CD stack
: To build and deploy decision services, the IT department can connect the Decision Designer web interface to the CI/CD stack in use in your organization. A built-in execution server, the decision runtime, enables decision services to be run with a simple REST API.

  If you do not have your own processes and tools in place already, Automation Decision Services provides a sample CI/CD stack that can be installed and set up easily mainly for evaluation purposes.

Building and deploying from Decision Designer
: As an alternative, you can easily deploy decision services from Decision Designer thanks to an embedded build and deploy service.

Related concepts

- Deploying decision services
- Executing decision services

Upgrading

Use the scripts provided in the GitHub repository to upgrade Automation Decision Services from the previous version.

Before you begin

The upgrading scripts are available in the /scripts folder in the following GitHub repository: https://github.com/icp4a/automation-decision-services-kubernetes.

Important: If the Decision Designer secret is now owned by the operator, you must introduce a new key **`sslKeystorePassword`**.

Procedure

1. Run the **`ads-upgrade-prereqs.sh`** script to upgrade IBM License Service and the certificate manager.
   The certificate manager is upgraded only if it's the one provided by IBM.

2. Run the **`ads-upgrade.sh`** script to upgrade the foundational services and the Automation Decision Services operator to the current version.
3. Before you edit the custom resources (CR) in the next step, make sure that the new key **`sslKeystorePassword`** is introduced for the Decision Designer secret.
4. When the **`ads-upgrade.sh`** script is finished, edit the **`spec.version`** parameter in the Automation Decision Services CR and change its value to 23.0.2 so that it reconciles to the current version.
5. Edit the CR to add new parameters or update existing parameters for new or updated features in the current version.
   For more information about the configuration parameters, see Configuration parameters.
   Tip: For more information about the new features and updates, see What's new.

Results

The Automation Decision Services CR status is `not ready` until all the Automation Decision Services dependencies and Automation Decision Services pods are ready. When all of them are ready, the status becomes `ready` again.

What to do next

Go to Completing the upgrade.

- **Completing the upgrade**
  After the operator upgrades your deployment of Automation Decision Services, you might need to perform some additional steps to complete the upgrade process.

Completing the upgrade

After the operator upgrades your deployment of Automation Decision Services, you might need to perform some additional steps to complete the upgrade process.

Procedure

Review the Environment section in Executing decision services with execution Java API to check the information about Java 17.

Installing

You can install Automation Decision Services to benefit from a comprehensive set of decision automation features.

Installation options

You can install Automation Decision Services in a production environment: OpenShift Container Platform (OCP) or Cloud Native Computing Foundation (CNCF) Certified Kubernetes.

For more information, see IBM Automation Decision Services Universal Base Image support statement for Kubernetes platforms.

Installation steps

Installing a production deployment involves the following steps:

1. Plan and prepare the cluster, including the prerequisites for Automation Decision Services.
2. Install the operator and generate the custom resource.
3. Edit, check and complete the configuration of the custom resource, and then apply it.
4. Do post-installation steps. You must assign a user role for each user. You also need to tune the logging level.

- **Planning to install Automation Decision Services**
  Before you install Automation Decision Services, you need to properly plan and prepare the cluster.
- **Preparing to install Automation Decision Services**
  Before you install Automation Decision Services, you must prepare your environment to make sure that you have everything that you need. The prerequisites include defining security and storage.
- **Installing Automation Decision Services**
  Installation of Automation Decision Services uses an operator, which is a Kubernetes feature that makes it simpler to install and update without having to worry about the underlying cloud provider.
- **Completing post-installation tasks for Automation Decision Services**
  The Automation Decision Services application runs after the CR is applied. However, it is still not reachable from the outside of your cluster. After the Automation Decision Services container is deployed to the cluster, you need to take additional steps.
- **Uninstalling Automation Decision Services**
  To remove Automation Decision Services, delete the namespace that you used to install.

Planning to install Automation Decision Services

Before you install Automation Decision Services, you need to properly plan and prepare the cluster.

- **System requirements**
  Automation Decision Services containers are Red Hat and IBM certified. To use the Automation Decision Services images, the administrator must make sure that the target cluster has the capacity.
- **Storage considerations**
  To run stateful applications, developers need to store the persistent data in a managed storage that is backed by some physical storage. Volumes allow a state to persist across pods.
- **Logging considerations**
  By default, logging is enabled in your cluster. You need to collect and forward the standard output "stdout" logs from the containers to help you troubleshoot issues and improve their health and performance.

System requirements

Automation Decision Services containers are Red Hat and IBM certified. To use the Automation Decision Services images, the administrator must make sure that the target cluster has the capacity.

For each stage in your operations, you must allocate a cluster of nodes before you install the Automation Decision Services. Development, preproduction, and production are stages that are best run on different compute nodes. High-level resource objects are scoped within namespaces. Low-level resources, such as nodes and persistent volumes, are not in namespaces.

The Detailed system requirements page provides a cluster requirements guideline for Automation Decision Services.

The minimum cluster configuration and physical resources that are needed to run the Automation Decision Services include the following elements:

- Hardware architecture: Intel (amd64 or x86_64 the 64-bit edition for Linux® x86) on all platforms, Linux on IBM Z, or Linux on Power.
- Node counts: Dual compute nodes for nonproduction and production clusters. A minimum of three nodes is needed for medium and large production environments and large test environments. Any cluster configuration needs to adapt to the size of the projects and the workload that is expected.

A cluster where you want to install Automation Decision Services needs as a minimum:

- Master (3 nodes): 4 vCPU and 8 Gi memory on each node.
- Worker (3 nodes): 4 vCPU and 10 Gi memory on each node.

Based on your cluster requirement, you can pick a deployment profile (deployment_profile_size) and enable it during installation. Automation Decision Services provides **`small`**, **`medium`**, **`large`**, and **`extra-large`** deployment profiles. You can set the profile during installation, in an update, and during an upgrade.

The default profile is **`small`**. Before you install Automation Decision Services, you can change the profile to a desired value. You can scale up or down a profile anytime after installation.

You can also adjust a profile size for Decision Designer or the decision runtime separately. For example, you can have a **`medium`** profile size for Decision Designer, and an **`extra-large`** profile size for the decision runtime to enable Horizontal Pod Autoscaler (HPA) only on the decision runtime.

Note:
The default profile size for the foundational services is **`small`**, the same as Automation Decision Services.

- If you install Automation Decision Services with a **`medium`** profile, the foundational services profile size is initialized to **`medium`**.
- If you install Automation Decision Services with a **`large`** profile, then the foundational services profile size is also **`large`**.

When you scale up or down the Automation Decision Services profile size afterward, the profile size for the foundational services remains unchanged. After installation, any change to the Automation Decision Services profile size is not reflected to the foundational services profile size. You must adapt it manually.

For more information about the foundational services, see the installing instructions in the GitHub repository: https://github.com/icp4a/automation-decision-services-kubernetes.

By default, the spec.zen.scale_config parameter takes the same value as the spec.deployment_profile_size parameter unless spec.zen.scale_config is explicitly set.

The following table describes each deployment profile.

Table 1. Deployment profiles and estimated workloads

Profile	Description	Scaling (per 8-hour day)	Minimum number of worker nodes
Small (no HA)	For environments that are used by a single department with a few users; useful for application development.	• Processes 350,000 decisions • Supports failover	3
Medium	For environments that are used by a single department and by limited users.	• Processes 700,000 decisions • Supports HA and failover • Provides at least two replicas of most services, if configuring failover	3
Large	For environments that are shared by multiple departments and users.	• Processes 2,000,000 decisions • Supports HA and failover • Provides at least two replicas of most services, if configuring failover	3

You can use custom resource templates to update the hardware requirements of the services that you want to install.

The following sections provide the default resources for each capability.

- Operator default requirements
- Small profile hardware requirements
- Medium profile hardware requirements
- Large profile hardware requirements
- Extra large profile hardware requirements

Attention: The values in the hardware requirements tables were derived under specific operating and environment conditions. The information is accurate under the given conditions, but results that are obtained in your operating environments might vary significantly. Therefore, IBM cannot provide any representations, assurances, guarantees, or warranties regarding the performance of the profiles in your environment.

Operator default requirements

Table 2. Automation Decision Services operator default requirements

Component	CPU Request (m)	CPU Limit (m)	Memory Request (Mi)	Memory Limit (Mi)	Number of replicas	Pods are licensed for production/nonproduction
ibm-ads-operator	10	500	64	512	1	No

Small profile hardware requirements

Table 3. Automation Decision Services default requirements for a small profile

Component	CPU Request (m)	CPU Limit (m)	Memory Request (Mi)	Memory Limit (Mi)	Number of replicas	Pods are licensed for production/nonproduction	Ephemeral storage Request	Ephemeral storage Limit
ads-runtime	500	1000	2048	3072	1	Yes	300Mi	500Mi
ads-credentials	250	1000	800	1536	1	No	300Mi	600Mi
ads-embedded-build	500	2000	1024	2048	1	No	1.1Gi	1.4Gi
ads-download	100	300	200	200	1	No	300Mi	500Mi
ads-front	100	300	256	256	1	No	300Mi	500Mi
ads-gitservice	500	1000	800	1536	1	No	400Mi	600Mi
ads-parsing	250	1000	800	1536	1	No	300Mi	500Mi
ads-restapi	500	1000	800	1536	1	No	300Mi	1.2Gi
ads-run	500	1000	800	1536	1	No	300Mi	700Mi

Note: Automation Decision Services also creates some jobs that request 200 m CPU and 256 Mi Memory. The following jobs are created at the beginning of the installation and do not last long:

- ads-ltpa-creation
- ads-ads-runtime-zen-translation-job
- ads-designer-zen-translation-job

Medium profile hardware requirements

Table 4. Automation Decision Services default requirements for a medium profile

Component	CPU Request (m)	CPU Limit (m)	Memory Request (Mi)	Memory Limit (Mi)	Number of replicas	Pods are licensed for production/nonproduction	Ephemeral storage Request	Ephemeral storage Limit
ads-runtime	500	1000	2048	3072	2	Yes	2.1Gi	3Gi
ads-credentials	250	1000	800	1536	2	No	300Mi	600Mi
ads-embedded-build	500	2000	1024	2048	2	No	1.1Gi	1.4Gi
ads-download	100	300	200	200	2	No	300Mi	600Mi
ads-front	100	300	256	256	2	No	300Mi	600Mi
ads-gitservice	500	1000	800	1536	2	No	400Mi	700Mi
ads-parsing	250	1000	800	1536	2	No	300Mi	600Mi
ads-restapi	500	1000	800	1536	2	No	300Mi	1.2Gi
ads-run	500	1000	800	1536	2	No	300Mi	700Mi

Note: Automation Decision Services also creates some jobs that request 200m CPU and 256Mi Memory. The following jobs are created at the beginning of the installation and do not last long:

- ads-ltpa-creation
- ads-ads-runtime-zen-translation-job
- ads-designer-zen-translation-job

Large profile hardware requirements

Table 5. Automation Decision Services default requirements for a large profile

Component	CPU Request (m)	CPU Limit (m)	Memory Request (Mi)	Memory Limit (Mi)	Number of replicas	Pods are licensed for production/nonproduction	Ephemeral storage Request	Ephemeral storage Limit
ads-runtime	1000	2000	2048	3072	2	Yes	2.1Gi	2.6Gi
ads-credentials	250	2000	800	1536	2	No	300Mi	700Mi
ads-embedded-build	500	2000	1024	2048	2	No	1.1Gi	1.5Gi
ads-download	100	300	200	200	2	No	300Mi	700Mi
ads-front	100	300	256	256	2	No	300Mi	700Mi
ads-gitservice	500	2000	800	1536	2	No	400Mi	800Mi
ads-parsing	250	2000	800	1536	2	No	300Mi	700Mi
ads-restapi	500	2000	800	1536	2	No	300Mi	1.2Gi
ads-run	500	2000	800	1536	2	No	300Mi	1Gi

Note: Automation Decision Services also creates some jobs that request 200 m CPU and 256 Mi Memory. The following jobs are created at the beginning of the installation and do not last long:

- ads-ltpa-creation
- ads-ads-runtime-zen-translation-job
- ads-designer-zen-translation-job

Extra large profile hardware requirements

The extra large profile can be set on Decision Designer and the decision runtime.

The extra large profile is not extending the CPU and memory limits of the large profile. Rather, it enables a `HorizontalPodAutoscaler`:

- On the Decision Designer component, you can scale up to 5 pods of the parsing and run services.
- On the decision runtime component, you can scale up to 5 pods of the runtime service.

You can change the maximum number of replica counts for each pod. For more information, see Configuration parameters.
You must then provision enough worker nodes or cluster resources to provide enough resources for the **`HorizontalPodAutoscaler`** to increase required pod counts, depending on the pods activity.

Also, check the resources of the Large profile hardware requirements.

Storage considerations

To run stateful applications, developers need to store the persistent data in a managed storage that is backed by some physical storage. Volumes allow a state to persist across pods.

About this task

The following table lists the disk space requirements for production deployments. Ranges are for small to large environments.

To pull all of the images from the IBM Entitled Registry, you need 30 - 50 GB of disk space. The actual size does depend on the registry type that you are using because some registries require more storage than other types of registry.

The following table provides storage requirements for the Automation Decision Services and runtimes. Kubernetes access modes include Read Write Once (RWO), Read Write Many (RWX), and Read Only Many (ROX).

Table 1. Storage requirements

Product	Storage type	Disk space	Access mode	Number of persistent volumes for non-HA/HA	Posix compliance
Automation Decision Services	[optional] mongo_embedded: File or Block [optional] runtime storage: File	50 GB	RWO RWX	1 (non-HA) 1 (shared)	Posix compliance needed Posix compliance not needed
Foundational services	Note: For information about sizing and storage options for the foundational services, see the foundational services and Platform UI columns in Table 1 in Storage options.				

Persistent storage can be defined as static persistent volumes or dynamic storage classes. The storage classes and persistent volumes describe the type of storage to use, which is then configured for an application by users of the cluster.

Persistent volumes

A cluster administrator defines and creates a persistent volume (PV) by providing the cloud infrastructure with the details of the implementation of the storage. That storage can be a number of different types, including a Network File System (NFS) or a cloud-specific storage system.
For more information, see Persistent volumes.

File system security permissions are needed to secure the Kubernetes environment for the Automation Decision Services and allow workloads to access storage. Access modes describe how the nodes access the storage. Note some default storage classes support **`ReadWriteOnce`** (RWO). Automation Decision Services needs **`ReadWriteMany`** (RWX) for volume access.

Storage classes

A **`StorageClass`** object describes and classifies dynamically provisioned storage that can be requested on demand. The objects can also be used to manage and control access to the storage. Cluster administrators define and create the objects that users can request without needing to know all of the details about the underlying storage sources.
Installation needs two storage class names. One that is associated with the dynamic storage you plan to use for file-based **`ReadWriteMany`** (RWX), and another for block storage for **`ReadWriteOnce`** (RWO). Your storage must have sufficient space for the deployment. For more information, see Storage Classes.

Storage classes must be POSIX-compliant, such as when used with NFS or a Common Internet File System (CIFS).

For more information about storage class parameters, see Product Documentation for Red Hat OpenShift Container Storage. For example, to allow a deployment to be deleted and redeployed without losing the data and files that are created by a deployment use reclaimPolicy: Retain. For cloud platforms where a file system group owner is needed, use gidAllocate: "true" to request one.

The foundational services use the cluster default storage class that is expected to be a block storage class.

Platform UI (Zen) needs both a block storage class and a file storage class. For more information about these classes, see Storage options.

The decision runtime needs an RWX storage class, or its persistence layer can be configured to use an S3 API compatible storage. For more information about S3 AP compatible storage, see Connecting to S3 compatible storage.

To use a persistent volume or a pool of storage that is defined by a storage class, a persistent volume claim (PVC) is needed to consume the storage resources. A PVC is a claim for storage by a user that can include requests for a specific size and access modes.

- If static provisioning is used, the PV and PVC must be created in the cluster, and the PVC name is specified in the custom resource.
  You can create a pool of volumes that can be used by many different workloads. Leave it up to persistent volume claim to bind to one from the pool. You can create different pools of storage by using PV labels and PVC selectors.
- If dynamic provisioning is used, the PVC names that are specified in the custom resource are used when the claim is created.

Note: You can get the existing storage classes in the environment by running the following command:

`kubectl get storageclass`

Take note of the storage classes that you want to use for your deployment.

Important:

- External database must be used by Automation Decision Service to persist data.
  You must provision the database instances and make sure that they are accessible from the cluster, or reuse existing database instances. To improve performance, reduce as much as possible the latency between the applications or containers and the database server.
- The embedded MongoDB is not recommended for production use.
  If you use the embedded MongoDB, use a block storage class for the MongoDB storage unless it is bound to a node, because the MongoDB pod that uses the PV can restart another node. If you don't have a block storage class that meets this requirement, use an RWX storage class instead.

Logging considerations

By default, logging is enabled in your cluster. You need to collect and forward the standard output "stdout" logs from the containers to help you troubleshoot issues and improve their health and performance.

Logs and log persistence must be considered before you install Automation Decision Services.

Logs and log persistence

All containerized applications write to the standard output and standard error streams, which can be viewed on a pod level by using **`kubectl logs`** on the command line. For example, to view them in the Red Hat OpenShift Container Platform (OCP) web console, click Workload > Pod > Pod Details > Logs.

All Automation Decision Services pods produce logs.

However, if a container crashes, a pod is evicted, or a node dies, you probably still want to access the application logs. Therefore, logs need a separate storage location and a lifecycle that is independent of nodes, pods, or containers. For example, OpenShift Container Platform can provide a logging solution that is based on the EFK stack: Elasticsearch, Fluentd, and Kibana. Fluentd collects all the node and container logs and stores them in dedicated project indexes. Kibana is the centralized user interface where you can create visualizations and dashboards with the aggregated data.

You can customize the logs that are written to **`stdout`** by referring to the configuration parameters for the capability. For more information, see Configuration parameters.

MustGather

For more information about collecting data to diagnose issues in Automation Decision Services, see MustGather.

Preparing to install Automation Decision Services

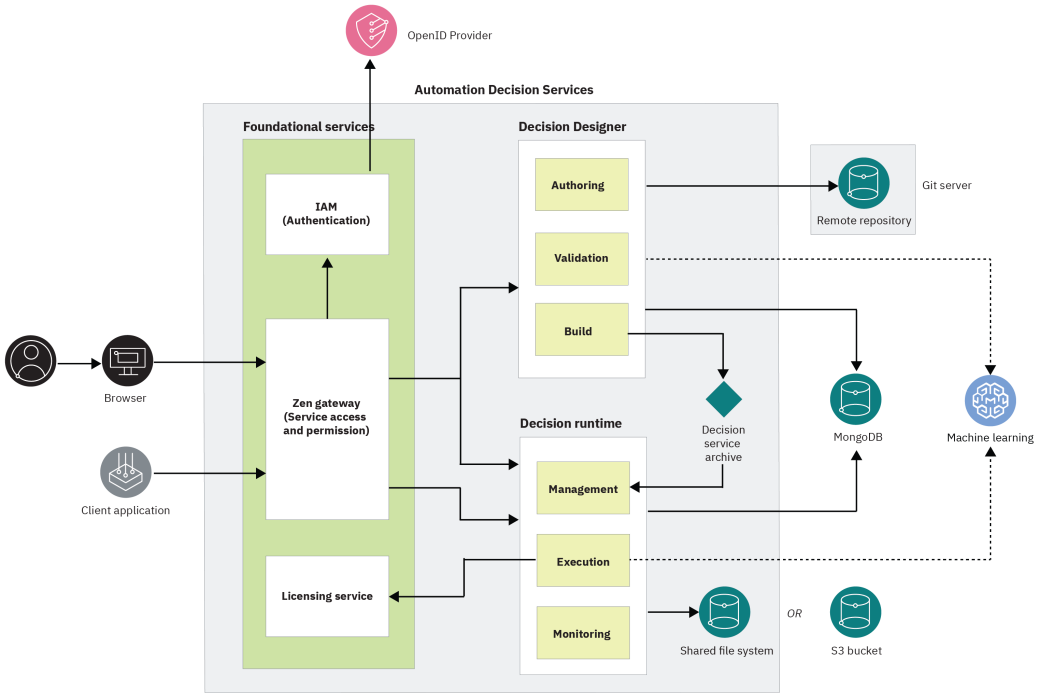
Before you install Automation Decision Services, you must prepare your environment to make sure that you have everything that you need. The prerequisites include defining security and storage.

Before you begin

You can install Automation Decision Services on different Kubernetes distributions.

You need to use Operator Lifecycle Manager (OLM) to install the Automation Decision Services operator along with its foundational services and Platform UI (Zen) dependencies.

Automation Decision Services uses the foundational services for licensing and authentication, and Zen for routing and authorization.



You prepare the following items:

- **Cluster**: You need to have a Kubernetes cluster to install Automation Decision Services. For more information about supported versions, see System requirements. You need to use the **kubectl** command with a version that is supported by your cluster.
- **Domain name**: You must control a DNS domain name (for example: `subdomain.my-company.com`) so that you can create DNS aliases (for example: `cp-console.subdomain.my-company.com` and `cpd-ads.subdomain.my-company.com`), and they can be resolved to your cluster load balancer.

About this task

Automation Decision Services comes with two main components: Decision Designer and a runtime that can be installed separately. They include the following services:

Table 1. Components and services

Component	Service	What to consider
Decision Designer	• Embedded MongoDB database (optional, shared with the decision runtime, disabled by default) • Embedded build service (automatically installed if the runtime service is installed) • Git service (file access backed by Git repositories) • Parsing service (compilation) • Run service (validation) • REST API (back end) • Credentials service	Before you install Decision Designer, you might need to provide: • ConfigMaps that contain a list of certificates to access Git servers, machine learning providers, and other servers, if the certificates are not trusted. • Admin secret
Decision runtime	• Embedded MongoDB database (optional, shared with Decision Designer, disabled by default) • Runtime service	Before you install the decision runtime, you might need to provide: • ConfigMaps with a list of certificates to access machine learning providers, if the certificates are not trusted. • Admin secret

The services require minimum CPU and memory resources. For more information, see System requirements.

Note: Important points about the MongoDB database:

- The embedded MongoDB database is provided for demo purposes only. It is NOT recommended for production.
- You must provide an external MongoDB database for production.
  An external MongoDB database allows high availability scenarios and finer control of the database configuration.
- You might want to customize the MongoDB database and the decision runtime persistent volumes.

For more information about supported databases, see Detailed system requirements.

What to do next

After you complete all the preparation steps, go ahead and create a production deployment.

- **Getting an IBM entitlement API key**
  To get access to the Automation Decision Servicesimages from the IBM Entitled Registry, you must have your IBM entitlement API key.
- **Preparing your cluster for an air gapped (offline) deployment**
  If your cluster is not connected to the internet, you can install Automation Decision Services in an air gap environment with the IBM Catalog Management Plug-in. Use either a bastion host, or a portable compute/storage device to transfer the images to your air gap environment.

- **Installing License Service and the certificate manager**
  The prerequisites script installs IBM License Service and the certificate manager. They must be installed before Automation Decision Services is installed.
- **Configuring MongoDB storage**
  Automation Decision Services requires a MongoDB database. You must specify an external MongoDB instance for production use.
- **Configuring Decision Designer**
  You can customize the Decision Designer sensitive configuration and TLS certificates.
- **Configuring the decision runtime**
  You can customize the decision runtime secret and persistent volume.

Getting an IBM entitlement API key

To get access to the Automation Decision Servicesimages from the IBM Entitled Registry, you must have your IBM entitlement API key.

About this task

Obtain the IBM entitlement API key that is associated with your My IBM account.

In OpenShift Container Platform (OCP), you can update the global pull secret for your cluster to ensure that all namespaces on your cluster have the necessary credentials to pull images.

Procedure

1. Log in to Container software library on My IBM with the IBMid and password that are associated with the entitled software.
2. On the Get entitlement key tab, select Copy key to copy the entitlement key to the clipboard.
   You need this key to create a Docker pull secret that is named `ibm-entitlement-key` into the namespace where you install Automation Decision Services, so that images can be pulled from the entitled registry. You can create it by using the following command:

```
kubectl create secret docker-registry ibm-entitlement-key \
                     --namespace <ads-namespace> \
                     --docker-server=cp.icr.io \
                     --docker-username=cp \
                     --docker-password=<entitlement-key>
```

3. If you plan to use the OpenShift Container Platform console to install, then you must create the secrets that you need in the appropriate namespaces.

Table 1. Image pull details for target namespace secrets

Field	Value
Name	`ibm-entitlement-key`
Authentication Type	Image Registry Credentials
Registry Server Address	`cp.icr.io`
Username	`cp`
Password	Your IBM Entitlement Key
Email	Optional

What to do next

You can now set up the cluster.

Preparing your cluster for an air gapped (offline) deployment

If your cluster is not connected to the internet, you can install Automation Decision Services in an air gap environment with the IBM Catalog Management Plug-in. Use either a bastion host, or a portable compute/storage device to transfer the images to your air gap environment.

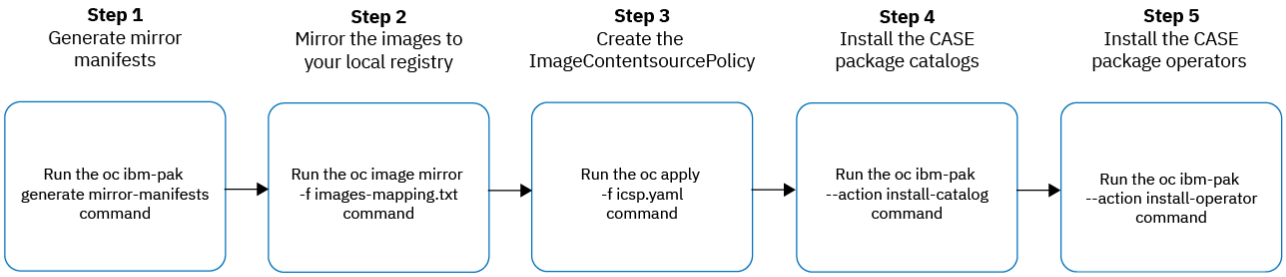
Before you begin

Restriction: An air gapped deployment is possible only for OpenShift Container Platform (OCP).
You can skip the procedure if you choose an online deployment.

About this task

The IBM Catalog Management Plug-in or `ibm-pak` for short, uses standard tooling for registry and cluster access. A CASE (Container Application Software for Enterprise) package specifies the composition of the product and the images that must be copied into an air-gapped environment.

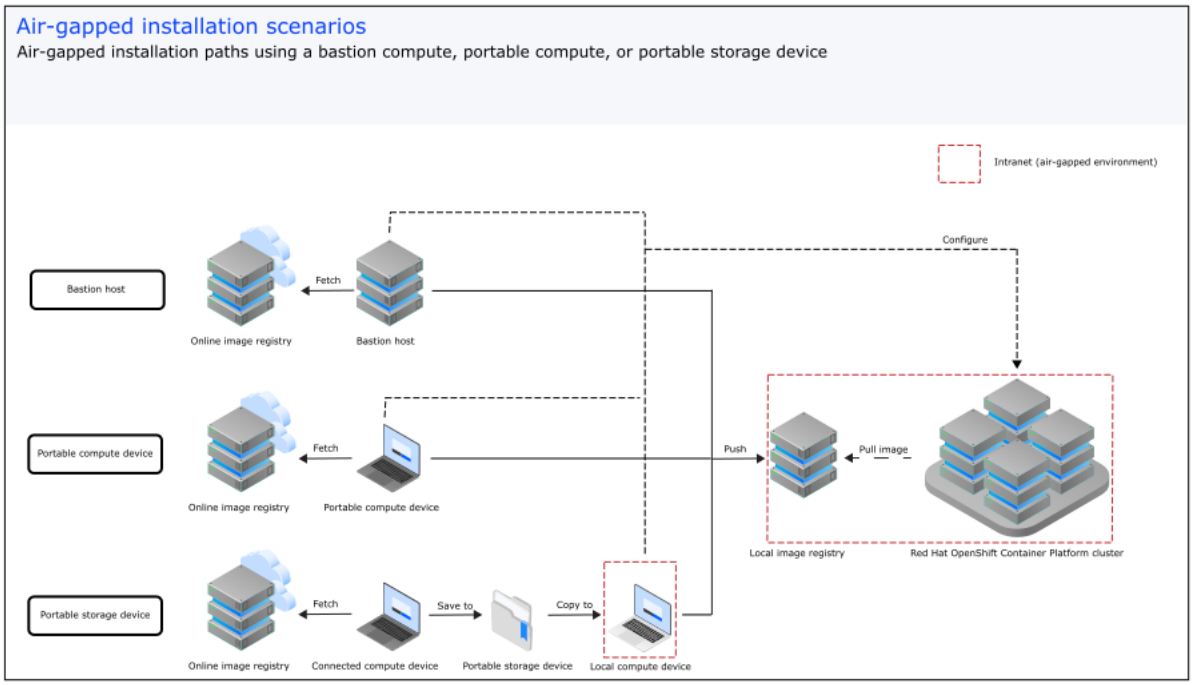
The following diagram shows the steps that are involved in installing with an air gap environment on OpenShift Container Platform.



It is common in production to have a cluster that cannot access the internet. In these cases, you can still install Automation Decision Services and OpenShift Container Platform (OCP) in an air-gapped (otherwise known as offline or disconnected) environment. An air-gapped installation uses the IBM operator catalog to mimic a typical online installation except that the Cloud Pak images are in your own registry. You can use a bastion host or a compute device (like a laptop), with or without portable storage (like an external hard disk drive) to transfer the images to an air-gapped network.

All of these scenarios use Container Application Software for Enterprises (CASE) files to mirror content from a source to a target. CASE is a specification that defines metadata and structure for packaging, managing, and unpacking containerized applications.

The following diagram provides an overview of the air-gapped installation options.



When you install Automation Decision Services in a Kubernetes namespace, then the foundational services are also installed in the same namespace exclusively for Automation Decision Services.

If you use multiple LDAPs with the Identity and Access Management (IAM) service, you must make sure that the usernames are unique across them. Your Automation Decision Services deployments can either use the same LDAP or make sure that your users are unique. For more information, see Configuring LDAP connection.

What to do next

Use the following steps to prepare your air-gapped installation. Follow the steps in order and use the **What to do next** sections to help you move to the next step.

- **Setting up a host to mirror images to a private registry**
  You can store everything that you need to install Automation Decision Services on a host that can be connected to the internet and use this host in an air gap environment.
- **Downloading the CASE files**
  Before mirroring the images, you must connect to the internet so that you can download the corresponding CASE files.
- **Mirroring images to a private registry**
  You must mirror all of the images that your deployment needs from the public image registries to a private registry. An `ImageContentSourcePolicy` is also needed to redirect the image pull requests from the public registries to the private registry.
- **Setting up a repeatable air-gap process**
  After you completed a CASE save, you can mirror the CASE as many times as you want to. Steps can be taken to air-gap a specific version of the Cloud Pak into development, test, and production stages.

Setting up a host to mirror images to a private registry

You can store everything that you need to install Automation Decision Services on a host that can be connected to the internet and use this host in an air gap environment.

Before you begin

You can use a bastion server, a portable compute device, or two compute devices with portable storage as your host.

Bastion host
: A bastion host is a server that is provisioned with a public IP address that is accessible through remote access Secure Shell (SSH). When configured, the bastion server acts as an intermediate server that allows a secure connection to the instances made available without a public IP address.

Portable compute device
: A portable compute device, such as a laptop, can be used to download images from the entitled registry to a portable image registry that is running locally on the device. You can then bring the device behind your firewall and copy the images from your portable registry on the device to the local private registry.

Portable storage device
: A portable storage device, such as a hard disk drive, can be connected to a compute device external to your firewall to download the images. The portable storage can then be connected to a device behind the firewall so that the images can be loaded to the local private registry.

No matter what medium you choose for your air-gapped installation, the host must satisfy the following prerequisites.

- The host must be able to access the OpenShift Container Platform (OCP) cluster, an internal image registry, and the internet.
- The host must be on a Linux® x86_64 or Mac platform with any operating system that the IBM Cloud Pak® CLI and the OpenShift Container Platform CLI support. If you are on a Windows platform, you must run the actions in a Linux® x86_64 VM or from a Windows Subsystem for Linux (WSL) terminal.

Procedure

1. Install the **oc** OCP CLI tool. For more information, see OCP CLI tools.
2. Install Podman on an RHEL machine. For more information, see Podman installation instructions.
3. Download and install the most recent version of the **IBM Catalog Management Plug-in**.
   a. Download the file based on the host operating system from IBM/ibm-pak-plugin/releases.
   b. Extract the binary file by entering the following command:

      ```
      tar -xf oc-ibm_pak-linux-amd64.tar.gz
      ```

   c. Run the following command to move the file to the /usr/local/bin directory:

      ```
      mv oc-ibm_pak-linux-amd64 /usr/local/bin/oc-ibm_pak
      ```

      Note: If you are installing as a non-root user, you must use **sudo**.
   d. You can confirm that **oc ibm-pak -h** is installed by running the following command:

      ```
      oc ibm-pak --help
      ```

      The plug-in usage is displayed.
4. Make sure that the following network ports are available on the host.
   - **icr.io:443** for the IBM Entitled Registry.
   - **quay.io:443** for foundational services.
   - **github.com** for CASE and tools.
   - **redhat.com** for OpenShift upgrades.

   Trouble: If the bastion host is unable to retrieve the source images from the public registries, you might need to allow specific access to these sites. A `HTTP 403 response` is an indication of such a parsing error. Docker and quay image registries might use proxies or mirror sites, so if you see images blocked check whether it is related to one of these image registries. If one of the registries is blocked, you must add that URL to the website *allowlist*. The following websites can be added to the *allowlist* to prevent pulling image errors.

   ```
   cp.icr.io/cp
   *.quay.io/opencloudio
   icr.io/cpopen
   ```

What to do next

You can now download the CASE files. For more information, see Downloading the CASE files.

Downloading the CASE files

Before mirroring the images, you must connect to the internet so that you can download the corresponding CASE files.

About this task

Important: If your bastion host, portable compute device, or portable storage device must connect to the internet through a proxy, you must set environment variables on the machine that accesses the internet through the proxy server. For more information, see Setting up proxy environment variables.

Procedure

1. Connect your host to the internet and disconnect it from the local air-gapped network.
2. Download the Automation Decision Services images to your host.

a. From IBM Catalog Management Plug-in (ibm-pak) v1.4, you can download the CASE files from **cp.icr.io/cpopen**. You can view the current config of the plug-in by running the following command:

```
oc ibm-pak config
```

The output lists all the configured repositories. The default repository from where the CASE files are downloaded has an asterisk mark (*) against the **Name** field.

b. You can then run the following command to configure a repository that downloads the CASE files from the **cp.icr.io** registry (an OCI-compliant registry) before you run the **oc ibm-pak get** command.

```
oc ibm-pak config repo 'IBM Cloud-Pak OCI registry' -r oci:cp.icr.io/cpopen --enable
```

The command sets 'IBM Cloud-Pak OCI registry' as the default repository.

c. You can list all the available CASE files to download by running the following command:

```
oc ibm-pak list
```

To get more help about the list command, run the following command:

```
oc ibm-pak list --help
```

d. Get the ADS-case-list.yaml file from a branch of the following GitHub repository: https://github.com/icp4a/automation-decision-services-kubernetes, which corresponds to the Automation Decision Services version of interim fix that you want to install.

The YAML file lists all of the dependent case images with their respective pinned versions.

e. When you are ready to start the download of the CASE files, run the following command:

```
oc ibm-pak get -c file://<absolute path to file>/ADS-case-list.yaml
```

The *<absolute path to file>* needs to be a path starting from "/". For example, "/opt".

By default, the root directory that is used by the ibm-pak plug-in is ~/.ibm-pak. Therefore, by default, the Automation Decision Services CASE is downloaded to ~/.ibm-pak/data/cases/$CASE_NAME/$CASE_VERSION.

Tip: You can configure the root directory by setting the ***IBMPAK_HOME*** environment variable.

f. To list the versions of all the downloaded CASE files, you can run the following command:

```
oc ibm-pak list --downloaded
```

If ***IBMPAK_HOME*** is set, the downloaded CASE is located in $IBMPAK_HOME/.ibm-pak/data/cases/$CASE_NAME/$CASE_VERSION. The logs files can be found in $IBMPAK_HOME/.ibm-pak/logs/oc-ibm_pak.log.

Results

Your host is now configured with the Automation Decision Services CASE files.

What to do next

You can now go ahead and mirror the images to your local registry. For more information, see Mirroring images to a private registry.

Mirroring images to a private registry

You must mirror all of the images that your deployment needs from the public image registries to a private registry. An **ImageContentSourcePolicy** is also needed to redirect the image pull requests from the public registries to the private registry.

About this task

Tip: If the image fails to load from the private registry, the request uses the public registry. The public registry request always fails in an air gap environment, so these errors might be misleading if you focus on the public registry request.
The running pods show the public image addresses even though the private registry is being used.

Choose the type of host that you want to set up. Your choice determines how you mirror the images to the private registry. A portable compute device can be used with or without the use of a portable storage device.

Important: You must set up an air gap environment for testing, quality assurance (QA), and user acceptance testing (UAT). These environments can help to ensure that the software is error-free before you roll the same tested operator version or interim fix out to a production environment. You must use the same bastion host, portable compute device, or portable storage device to mirror all of the tested images in all your deployment environments. When testing is complete in your test/QA/UAT environments and you are ready to roll the deployment out to production do not mirror the images again. Mirroring the images again pulls the latest versions of IBM Automation foundation and foundational services. These images might be different from what you tested in your test, QA, UAT, or pre-production environments.

- **Option 1: Mirroring images to a private registry with a bastion server**
  When you mirror the images, you must run the steps on the host that is connected to both the private registry and the Red Hat OpenShift Container Platform (OCP) cluster.
- **Option 2: Mirroring images to a private registry with a portable compute or a storage device (file system)**
  If your cluster is not connected to the internet, you can install Automation Decision Services in your cluster by using a file system. The portable storage and compute scenarios are enabled with a file system.

Option 1: Mirroring images to a private registry with a bastion server

When you mirror the images, you must run the steps on the host that is connected to both the private registry and the Red Hat OpenShift Container Platform (OCP) cluster.

Before you begin

Verify that you completed the prerequisites, prepared a host, set the environment variables, and created registry namespaces if the registry does not allow automatic creation of top-level namespaces.

Procedure

1. Generate the required mirror manifests.
   a. Define the environment variable *$TARGET_REGISTRY* by running the following command.

   ```
   export TARGET_REGISTRY=<target-registry>
   ```

   The *<target-registry>* refers to the registry (hostname and port) where the images are mirrored to and accessed by the OCP cluster. For example: `172.16.0.10:5000`.

   b. Create the following environment variables with the installer image name and the version.

   ```
   export CASE_NAME=ibm-ads
   export CASE_VERISON=1.1.0
   ```

   Note: Releases with interim fixes are packaged in archives with a new minor version. The version numbers follow the `release.major.minor` standard. For example, the first interim fix for 23.0.2 is packaged in the CASE version `1.1.0`.

   c. Run the following command to generate mirror manifests to be used when mirroring the image to the target registry. The *$TARGET_REGISTRY* refers to the registry where the images are mirrored to and accessed by the OpenShift Container Platform cluster.

   ```
   oc ibm-pak generate mirror-manifests $CASE_NAME $TARGET_REGISTRY \
     --version $CASE_VERSION
   ```

   Note: After an upgrade, you can remove images from the previous version from the local registry by including an alternative namespace to copy the images to instead of the default path. For example, the following CASE command copies the images to a `locationX` namespace.

   ```
   oc ibm-pak generate mirror-manifests $CASE_NAME $TARGET_REGISTRY/locationX \
     --version $CASE_VERSION
   ```

   The command generates the files images-mapping.txt and image-content-source-policy.yaml in ~/.ibm-pak/data/mirror/$CASE_NAME/$CASE_VERSION.

   Tip: If you are using a Red Hat® Quay.io registry and need to mirror the images to a specific organization in the registry, you can set the target to that organization. Specify the organization name in the `generate mirror-manifests` command:

   ```
   export ORGANIZATION=<your-organization> \
   oc ibm-pak generate mirror-manifests $CASE_NAME $TARGET_REGISTRY/$ORGANIZATION \
      --version $CASE_VERSION
   ```

   Restriction: Currently, you cannot select the images to mirror by their target architecture because image registries do not support sparse manifests (manifests that reference image digests outside of the package).
   The `~/.ibm-pak` directory structure is built over time as you save CASEs and mirror. The following tree shows an example of the `~/.ibm-pak` directory structure:

   ```
   /root/.ibm-pak
   ├── config
   │   └── config.yaml
   ├── data
   │   ├── cases
   │   │   └── ibm-ads
   │   │       └── 1.1.0
   │   │           ├── component-set-config.yaml
   │   │           ├── ibm-ads-1.1.0.tgz
   │   │           ├── xxxxx

   │   └── mirror
   │       └── ibm-ads
   │           └── 1.1.0
   │               ├── catalog-sources.yaml
   │               ├── image-content-source-policy.yaml
   │               └── images-mapping.txt
   └── logs
       └── oc-ibm_pak.log
   ```

   A new directory ~/.ibm-pak/mirror is created when you issue the `oc ibm-pak generate mirror-manifests` command. The mirror directory stores the image-content-source-policy.yaml and images-mapping.txt files.

   You can use the following command to list all the images in your mirror manifest and the publicly accessible registries from where the images are pulled from.

   ```
   oc ibm-pak describe $CASE_NAME \
     --version $CASE_VERSION \
     --list-mirror-images
   ```

   Tip: Make a note of the registries section at the end of output, so you can check that you can login to the registries. You need to be able to login to these registries to pull and mirror the images to your local target registry.
2. Authenticate the registries.
   You must store authentication credentials for all the Automation Decision Services source Docker registries. The following registries require authentication:

   - `cp.icr.io`

- `registry.redhat.io`
- `registry.access.redhat.com`

Run the following command to configure credentials for all target registries that require authentication. You must run the command separately for each registry.

```
export REGISTRY_AUTH_FILE=<path to the file that has the auth credentials generated on podman login>
podman login cp.icr.io
podman login <TARGET_REGISTRY>
```

Important: When you log in to `cp.icr.io`, you must specify the user as `cp` and the IBM entitlement key as the password. For example:

```
podman login cp.icr.io
Username: cp
Password: xxxxxxxxxxxxxxxxxxx
Login Succeeded!
```

The password can be copied from the IBM container library. You can add `--tls-verify=false` to the command, if you see "`cert error`" messages.

If you export `REGISTRY_AUTH_FILE=~/.ibm-pak/auth.json`, and then run the `podman login` command, you can see that the file is populated with registry credentials.

If you use `docker login`, the authentication file is typically located in `$HOME/.docker/config.json` on Linux or `%USERPROFILE%/.docker/config.json` on Windows. After you run the `docker login` command, you can export `REGISTRY_AUTH_FILE` to point to that location. For example, on Linux you can run the following command:

```
export REGISTRY_AUTH_FILE=$HOME/.docker/config.json
```

3. Mirror the images to the target registry.
The following steps must be completed on the host that is connected to both the local Docker registry and the OpenShift Container Platform cluster.

   a. Mirror the images by running the following command:

   ```
   oc image mirror -f ~/.ibm-pak/data/mirror/$CASE_NAME/$CASE_VERSION/images-mapping.txt \
     --filter-by-os '.*' -a $REGISTRY_AUTH_FILE \
     --insecure \
     --skip-multiple-scopes \
     --max-per-registry=1 \
     --continue-on-error=true
   ```

   The command does not produce any console logs for about 6 - 8 minutes as it prepares the list from the CASE package. If you want, you can add verbose (`-v`) to the command with possible values of 3, 9, 99, and so on.

   The following command can be used to see all the available options.

   ```
   oc image mirror --help
   ```

   Based on the number and size of the images to mirror, the `oc image mirror` command can take a considerable amount of time. If you are running the command on a remote machine, run the command in the background with the `nohup` POSIX command so that it does not stop if the user logs out. The following command starts the mirroring process in the background and writes the log to a my-mirror-progress.txt file.

   ```
   nohup oc image mirror -f ~/.ibm-pak/data/mirror/$CASE_NAME/$CASE_VERSION/images-mapping.txt \
     --filter-by-os '.*' -a $REGISTRY_AUTH_FILE \
     --insecure \
     --skip-multiple-scopes \
     --max-per-registry=1 \
     --continue-on-error=true > my-mirror-progress.txt 2>&1 &
   ```

   You can view the progress of the mirror command by running the following command on the remote machine:

   ```
   tail -f my-mirror-progress.txt
   ```

   b. Update the global image pull secret for your OpenShift cluster to have authentication credentials in place to pull images from your *$TARGET_REGISTRY* as specified in the image-content-source-policy.yaml file. For more information, see Updating the global cluster pull secret.

   c. Create the `ImageContentsourcePolicy`.
   Note: Before you run the command in this step, you must be logged in to your OpenShift cluster. You do not need to be a cluster administrator to run the mirroring commands. Using the `oc login` command, log in to the Red Hat OpenShift Container Platform cluster where you plan to pull the mirrored images. You can identify your specific oc login by clicking the user drop-down menu in the Red Hat OpenShift Container Platform console, then clicking Copy Login Command.
   Run the following command to create `ImageContentsourcePolicy`.

   ```
   oc apply -f ~/.ibm-pak/data/mirror/$CASE_NAME/$CASE_VERSION/image-content-source-policy.yaml
   ```

   d. Verify that the `ImageContentsourcePolicy` resource is created.

   ```
   oc get imageContentSourcePolicy
   ```

   e. Verify your cluster node status.

   ```
   oc get MachineConfigPool -w
   ```

   After the `ImageContentsourcePolicy` and global image pull secret are applied, the configuration of your nodes is updated sequentially. Wait until all the `MachineConfigPools` are updated before you move on to the next step.

   f. Create a project for the CASE commands (`cp4ba` is an example) by running the following commands:
   Note: Before you run the command in this step, you must be logged in to your OpenShift cluster.

   ```
   export NAMESPACE=ads
   oc new-project $NAMESPACE
   ```

   g. **Optional:** If you use an insecure registry, you must add the target registry to the cluster `insecureRegistries` list.

   ```
   oc patch image.config.openshift.io/cluster \
     --type=merge -p '{"spec":{"registrySources":{"insecureRegistries":["'${TARGET_REGISTRY}'"]}}}'
   ```

What to do next

You can now go ahead and install License Service and the certificate manager. For more information, see Installing License Service and the certificate manager.

Option 2: Mirroring images to a private registry with a portable compute or a storage device (file system)

If your cluster is not connected to the internet, you can install Automation Decision Services in your cluster by using a file system. The portable storage and compute scenarios are enabled with a file system.

Before you begin

Verify that you completed the prerequisites, prepared a host, set the environment variables, and created registry namespaces if the registry does not allow automatic creation of top-level namespaces.

Procedure

1. Generate the required mirror manifests.
   a. Define the environment variable *$TARGET_REGISTRY* by running the following command.

      ```
      export TARGET_REGISTRY=<target-registry>
      ```

      The *<target-registry>* refers to the registry (hostname and port) where the images are mirrored to and accessed by the OCP cluster. For example: `172.16.0.10:5000`.

   b. Create the following environment variables with the installer image name and the version.

      ```
      export CASE_NAME=ibm-ads
      export CASE_VERSION=1.1.0
      ```

      Note: Releases with interim fixes are packaged in archives with a new minor version. The version numbers follow the `release.major.minor` standard. For example, the first interim fix for 23.0.2 is packaged in the CASE version `1.1.1`.
   c. Run the following command to generate mirror manifests to be used when mirroring the image to the target registry. The *$TARGET_REGISTRY* refers to the registry where the images are mirrored to and accessed by the OCP cluster.

      ```
      oc ibm-pak generate mirror-manifests $CASE_NAME file://ads-images \
         --version $CASE_VERSION \
         --final-registry $TARGET_REGISTRY/ads
      ```

      Where `file://ads-images` indicates to the plug-in that the images are first mirrored to a local file system (to the ads-images directory) on the machine that runs the `oc image mirror` command. The `final-registry $TARGET_REGISTRY/ads` argument generates a mapping file that is used by the `oc image mirror` commands. The final URL in your target registry includes the new `ads` namespace in the mirrored namespace path. The namespace path can be multi-level if your target registry supports it.

      Note: After an upgrade, you can remove images from the previous version from the local registry by using an alternative namespace to copy the images to. For example, to a `locationX` namespace.
      The `generate mirror-manifests` command generates the following files in `~/.ibm-pak/data/mirror/$CASE_NAME/$CASE_VERSION`:

      - images-mapping-to-filesystem.txt
      - images-mapping-from-filesystem.txt
      - image-content-source-policy.yaml
      - catalog-sourcesxxx.yaml

   Tip: If you are using a Red Hat® Quay.io registry and need to mirror the images to a specific organization in the registry, you can set the target to that organization. Specify the organization name in the `generate mirror-manifests` command:

   ```
   export ORGANIZATION=<your-organization> \
   oc ibm-pak generate mirror-manifests $CASE_NAME file://ads-images \
      --version $CASE_VERSION \
      --final-registry $TARGET_REGISTRY/$ORGANIZATION
   ```

   If you do not know the value of the final registry where the images are to be mirrored, you can provide a placeholder value of `TARGET_REGISTRY`. For example:

   ```
   oc ibm-pak generate mirror-manifests $CASE_NAME file://ads-images \
      --version $CASE_VERSION \
      --final-registry TARGET_REGISTRY
   ```

   Note: `TARGET_REGISTRY` used without any environment variable expansion is just a plain string that must be replaced later with the actual image registry URL when it is known to you.
   Restriction: Currently, you cannot select the images to mirror by their target architecture because image registries do not support sparse manifests (manifests that reference image digests outside of the package).
   The `~/.ibm-pak` directory structure is built over time as you save CASEs and mirror. The following tree shows an example of the `~/.ibm-pak` directory structure:

   ```
   /root/.ibm-pak
   ├── config
   │   └── config.yaml
   ├── data
   │   ├── cases
   │   │   └── ibm-ads
   │   │       └── 1.1.0
   │   │           ├── component-set-config.yaml
   ```

```
│   │           ├── ibm-ads-1.1.0.tgz
│   │           ├── xxxxxx
│   └── mirror
│       └── ibm-ads
│           └── 1.1.0
│               ├── catalog-sources.yaml
│               ├── image-content-source-policy.yaml
│               ├── images-mapping-from-filesystem.txt
│               └── images-mapping-to-filesystem.txt
└── logs
    └── oc-ibm_pak.log
```

A new directory ~/.ibm-pak/mirror is created when you issue the **oc ibm-pak generate mirror-manifests** command. The mirror directory stores the image-content-source-policy.yaml and images-mapping.txt files.

You can use the following command to list all the images in your mirror manifest and the publicly accessible registries from where the images are pulled from.

```
oc ibm-pak describe $CASE_NAME \
   --version $CASE_VERSION \
   --list-mirror-images
```

Tip: Make a note of the registries section at the end of output, so you can check that you can log in to the registries. You need to be able to log in to these registries to pull and mirror the images to your target registry.

2. Authenticate the registries.
You must store authentication credentials for all the Automation Decision Services source Docker registries. The following registries require authentication:

   - **cp.icr.io**
   - **registry.redhat.io**
   - **registry.access.redhat.com**

Run the following command to configure credentials for all target registries that require authentication. You must run the command separately for each registry.

```
export REGISTRY_AUTH_FILE=<path to the file that has the auth credentials generated on podman login>
podman login cp.icr.io
podman login <TARGET_REGISTRY>
```

Important: When you log in to **cp.icr.io**, you must specify the user as **cp** and the IBM entitlement key as the password. For example:

```
podman login cp.icr.io
Username: cp
Password: xxxxxxxxxxxxxxxxxxxx
Login Succeeded!
```

The password can be copied from the IBM container library.

If you export **REGISTRY_AUTH_FILE=~/.ibm-pak/auth.json**, and then run the **podman login** command, you can see that the file is populated with registry credentials.

If you use **docker login**, the authentication file is typically located in **$HOME/.docker/config.json** on Linux or **%USERPROFILE%/.docker/config.json** on Windows. After you run the **docker login** command, you can export **REGISTRY_AUTH_FILE** to point to that location. For example, on Linux you can run the following command:

```
export REGISTRY_AUTH_FILE=$HOME/.docker/config.json
```

3. Mirror the images to the file system.
   a. Mirror images to the TARGET_REGISTRY.

   ```
   oc image mirror \
     -f ~/.ibm-pak/data/mirror/$CASE_NAME/$CASE_VERSION/images-mapping-to-filesystem.txt \
     -a $REGISTRY_AUTH_FILE \
     --filter-by-os '.*' \
     --insecure \
     --skip-multiple-scopes \
     --max-per-registry=1
   ```

   The command creates a **v2** folder in the current directory where all the images are copied. For example, if **file://ads-images** is used as the input to generate the mirror-manifests, then the command creates a subdirectory **ads-images** under **v2** and copies the images in this folder.

   The following command can be used to see all the available options.

   ```
   oc image mirror --help
   ```

   You can use the **continue-on-error** parameter to continue the mirroring even if an error occurs.

   Based on the number and size of the images to mirror, the **oc image mirror** command can take a considerable amount of time. If you are running the command on a remote machine, run the command in the background with the **nohup** POSIX command so that it does not stop if the user logs out. The following command starts the mirroring process in the background and writes the log to a my-mirror-progress.txt file.

   ```
   nohup oc image mirror -f ~/.ibm-pak/data/mirror/$CASE_NAME/$CASE_VERSION/images-mapping-to-filesystem.txt \
     --filter-by-os '.*' -a $REGISTRY_AUTH_FILE \
     --insecure \
     --skip-multiple-scopes \
     --max-per-registry=1 \
     --continue-on-error=true > my-mirror-progress.txt 2>&1 &
   ```

   You can view the progress of the mirror command by running the following command on the remote machine:

   ```
   tail -f my-mirror-progress.txt
   ```

   b. Move the following items to your file system:
      - The v2 directory.
      - The **auth** file referred to by the **$REGISTSRY_AUTH_FILE** variable.

- The `~/.ibm-pak/data/mirror/$CASE_NAME/$CASE_VERSION/images-mapping-from-filesystem.txt` file.

4. Mirror the images to the target registry from your file system.
   The following steps to copy the images from the file system to the `$TARGET_REGISTRY` must be completed on your file system. Your file system must be connected to both the local Docker registry and the OpenShift Container Platform cluster.

   Important: If you used the placeholder value of `TARGET_REGISTRY` for the `--final-registry` parameter when you generated the mirror manifests, replace the string in the images-mapping-from-filesystem.txt file with the registry where you want to mirror the images. For example, if you want to mirror the images to myregistry.com/mynamespace then replace `TARGET_REGISTRY` with myregistry.com/mynamespace.

   a. Mirror the images from the v2 directory to the target registry by running the following command:

   ```
   oc image mirror \
     -f ~/.ibm-pak/data/mirror/$CASE_NAME/$CASE_VERSION/images-mapping-from-filesystem.txt \
     -a $REGISTRY_AUTH_FILE \
     --from-dir=${v2_dir} \
     --filter-by-os '.*' \
     --insecure \
     --skip-multiple-scopes \
     --max-per-registry=1
   ```

   The `$v2_dir` refers to the parent directory on the file system where you copied the v2 directory. The command does not produce any console logs for about 6 - 8 minutes as it prepares the list from the CASE package. If you want, you can add verbose (`-v`) to the command with possible values of 3, 9, 99, and so on.

   b. Update the global image pull secret for your OpenShift cluster to have authentication credentials in place to pull images from your *$TARGET_REGISTRY* as specified in the image-content-source-policy.yaml file. For more information, see Updating the global cluster pull secret.

   c. Create the `ImageContentsourcePolicy`.
   Note: Before you run the command in this step, you must be logged in to your OpenShift cluster. You do not need to be a cluster administrator to run the mirroring commands. Using the `oc login` command, log in to the Red Hat OpenShift Container Platform cluster where you plan to pull the mirrored images. You can identify your specific oc login by clicking the user drop-down menu in the Red Hat OpenShift Container Platform console, then clicking Copy Login Command.
   Run the following command to create `ImageContentsourcePolicy`.

   ```
   oc apply -f ~/.ibm-pak/data/mirror/$CASE_NAME/$CASE_VERSION/image-content-source-policy.yaml
   ```

   d. Verify that the `ImageContentsourcePolicy` resource is created.

   ```
   oc get imageContentSourcePolicy
   ```

   e. Verify your cluster node status.

   ```
   oc get MachineConfigPool -w
   ```

   After the `ImageContentsourcePolicy` and global image pull secret are applied, the configuration of your nodes is updated sequentially. Wait until all the `MachineConfigPools` are updated before you move on to the next step.

   f. Create a project for the CASE commands (`ads` is an example) by running the following commands:
   Note: Before you run the command in this step, you must be logged in to your OpenShift cluster.

   ```
   export NAMESPACE=ads
   oc new-project $NAMESPACE
   ```

   g. **Optional:** If you use an insecure registry, you must add the target registry to the cluster `insecureRegistries` list.

   ```
   oc patch image.config.openshift.io/cluster /
     --type=merge -p '{"spec":{"registrySources":{"insecureRegistries":["'${TARGET_REGISTRY}'"]}}}'
   ```

What to do next

You can now go ahead and install License Service and the certificate manager. For more information, see Installing License Service and the certificate manager.

Setting up a repeatable air-gap process

After you completed a CASE save, you can mirror the CASE as many times as you want to. Steps can be taken to air-gap a specific version of the Cloud Pak into development, test, and production stages.

About this task

The following steps show you how to save the CASE files to multiple registries (per environment), and use the same saved CASE cache (~/.ibm-pak/$CASE_NAME/$CASE_VERSION) to mirror the CASE without repeating the save process. As a result of not needing to save the CASE each time, you do not have to worry about introducing newer versions of the dependencies to the CASE cache.

Procedure

1. Run the following command to save the CASE to ~/.ibm-pak/data/cases/$CASE_NAME/$CASE_VERSION.

   ```
   oc ibm-pak get $CASE_NAME \
      --version $CASE_VERSION
   ```

   This CASE can be used as an input for the mirror manifest generation.

2. Run the `oc ibm-pak generate mirror-manifests` command to generate the image-mapping.txt file.

```
oc ibm-pak generate mirror-manifests $CASE_NAME $TARGET_REGISTRY \
   --version $CASE_VERSION
```

3. Add the image-mapping.txt to the `oc image mirror` command.

```
oc image mirror -f ~/.ibm-pak/data/mirror/$CASE_NAME/$CASE_VERSION/images-mapping.txt \
   --filter-by-os '.*' -a $REGISTRY_AUTH_FILE \
   --insecure \
   --skip-multiple-scopes \
   --max-per-registry=1
```

Installing License Service and the certificate manager

The prerequisites script installs IBM License Service and the certificate manager. They must be installed before Automation Decision Services is installed.

Before you begin

You must have the cluster admin privileges to run the script.

Use a bastion server as your host. You must run the script from a bastion server in a session where you are connected to the targeted cluster.

Bastion host
: A bastion host is a server that is provisioned with a public IP address that is accessible through remote access Secure Shell (SSH). When configured, the bastion server acts as an intermediate server that allows a secure connection to the instances made available without a public IP address.

About this task

The prerequisites script (**`ads-install-prereqs.sh`**) is available in the following GitHub repository: https://github.com/icp4a/automation-decision-services-kubernetes.

The script does the following actions:

- Checks if Operator Lifecycle Manager (OLM) is installed in your cluster. If not, the script installs OLM. If you are using an Red Hat OpenShift cluster, OLM is already installed by default.
- Checks if a certificate manager is already installed in your cluster:
  - If it is already installed, Automation Decision Services uses the existing certificate manager.
  - If it is not installed, the script installs the certificate manager (**`ibm-certificate-manager`**) cluster-wide.
- Looks for License Service:
  - If it is found, Automation Decision Services uses the existing License Service.
  - If it is not found, the script installs it in your cluster by using the namespace that is set as an argument defaulting to **`ibm-licensing`**.

Procedure

Run the **`ads-install-prereqs.sh`** script:

```
./scripts/ads-install-prereqs.sh [-h] -a [-n licensing-namespace]
```

Description of the options:

- **`-a`**: Accept integration license. For more information, see Licensing in the IBM Cloud Pak for Integration documentation.
- **`-n`**: Namespace where the licensing operator is installed. The default value is **`ibm-licensing`**.

Results

When the script finishes running, you have everything that you need in your cluster to run the **`ads-install.sh`** script to install Automation Decision Services.

Configuring MongoDB storage

Automation Decision Services requires a MongoDB database. You must specify an external MongoDB instance for production use.

About this task

Instantiating an external MongoDB is highly recommended for production use. You have to provide the connection details to your external MongoDB inside a secret.

Restriction: You can use the embedded MongoDB database for demo purposes only. The embedded MongoDB is not for production use.
For more information about the supported versions of the MongoDB database, look under Supported Software in the detailed system requirements.

Important: If the external MongoDB presents a certificate that is not signed by an official Certificate Authority (CA), it must be handled as an untrusted TLS certificate.

Procedure

1. To configure an external MongoDB instance, create a mongo admin secret .
   The default name of this secret is **`ibm-dba-ads-mongo-secret`**. You can specify another name that you will add to the custom resource file at a later stage in the installation through the parameter ads_configuration.mongo.admin_secret_name.

Example MongoDB secret:

```
apiVersion: v1
kind: Secret
metadata:
  name: "ibm-dba-ads-mongo-secret"
type: Opaque
stringData:
  gitMongoUri: "mongodb+srv://sampleDbUser:sampleDbPassword@mongodb0.example.com/ads-git?
retryWrites=true&w=majority&authSource=admin"
  mongoUri: "mongodb+srv://sampleDbUser:sampleDbPassword@mongodb1.example.com/ads?
retryWrites=true&w=majority&authSource=admin"
  mongoHistoryUri: "mongodb+srv://sampleDbUser:sampleDbPassword@mongodb1.example.com/ads-history?
retryWrites=true&w=majority&authSource=admin"
  runtimeMongoUri: "mongodb+srv://sampleDbUser:sampleDbPassword@mongodb1.example.com/ads-runtime-archive-metadata?
retryWrites=true&w=majority&authSource=admin"
```

The parameters of the secret are explained in the following table.

Table 1. MongoDB secret configuration parameters

Parameter	Description
gitMongoUri	Required when installing Decision Designer. The connection string URI that is related to the Git Service. The database name that follows the third '/' character (ads-git in the sample) identifies the database that is used by the service to store its own data.
mongoUri	Required when installing Decision Designer. The connection string URI that is related to the Designer REST-API and the Run services. The database name that follows the third '/' character (ads in the sample) identifies the database that is used by the services to store their own data. The Credentials service and the Build service also use mongoUri.
mongoHistoryUri	Required when installing Decision Designer. The connection string URI that is related to temporary data of the REST API. The database name that follows the third '/' character (ads-history in the sample) identifies the database that is used by the service to store its own data.
runtimeMongoUri	Required when installing the decision runtime. The connection string URI that is related to decision runtime service. It honors the **`authSource`** option to specify the database for MongoDB credentials. The database name that follows the third '/' character (**`ads-runtime-archive-metadata`** in the sample) identifies the database that is used by the service to store its own data.

See the MongoDB documentation on Connection String URIs.

2. Optional: If you decide to use the embedded MongoDB, you must configure its persistent volume.
   If the **`mongo.persistence.enabled`** parameter is true, you can use dynamic provisioning or create a persistent volume.

   Note: The embedded MongoDB persistent volume requires a fully POSIX-compliant storage. Make sure to use a storage class that meets this requirement.

   Option 1: Using dynamic provisioning
   With **`mongo.persistence.use_dynamic_provisioning`** set to true, you must specify a **`mongo.persistence.storage_class_name`** according to the guidelines in Using storage class on GitHub. Use a ReadWriteOnce (RWO) class. Setting **`storage_class_name`** to **`null`** means using the default storage class.
   Warning: The **`mongo.persistence.storage_class_name`** parameter cannot be changed after installation.
   Note: The storage class must not be bound to a single node as the MongoDB pod can restart on a different node.

   Option 2: Creating a persistent volume
   If you decide to set **`mongo.persistence.use_dynamic_provisioning`** to false in the custom resource file (at a later stage in the installation), you must provide a matching persistent volume. You can create one like this:

   ```
   kind: PersistentVolume
   apiVersion: v1
   metadata:
     name: ibm-dba-ads-mongo-pv
   spec:
     accessModes:
       - ReadWriteOnce
     capacity:
       storage: 1Gi
     persistentVolumeReclaimPolicy: Recycle
     # example setup for nfs server - for other setups, see the Kubernetes Docs on PersistentVolumes
     nfs:
       server: "1.2.3.4"
       path: "/data/ads/mongo"
   ```

   Make sure you grant permissions to the root group **`0`** because the user that runs the MongoDB pods belongs to this group.
   The persistent volume must deliver a minimum performance of 300 IOPS. On ROKS (IBM Cloud®), this level of performance is achieved by a volume of 30Gi bytes from the ibmc-file-gold-gid storage class, or by a volume of 150Gi byte from the ibmc-file-bronze-gid storage class. For details, see Storage class reference

   The persistent volume must support read and write access for a non-root user. The *-gid storage classes on IBM Cloud typically provide this support.

3. Optional: If the external MongoDB presents a certificate that is not signed by an official Certificate Authority (CA), it must be handled as an untrusted TLS certificate. For more information, see Configuring Decision Designer and Configuring the decision runtime.

Configuring Decision Designer

You can customize the Decision Designer sensitive configuration and TLS certificates.

About this task

The customizations described here are optional as default values are set if you do nothing.
Some of the customizations described here take place at alter stage in the installation, namely when Checking and completing your custom resource.

Procedure

1. Generate a secret for sensitive configuration.
   Sensitive configuration settings must not be exposed in the custom resource YAML file. The **ibm-ads-operator** generates the secret **ibm-dba-ads-designer-secret** automatically, but you can decide to create your own secret with your own credentials.

   Here is an example of **ibm-dba-ads-designer-secret**.

   ```
   apiVersion: v1
   kind: Secret
   metadata:
     name: ibm-dba-ads-designer-secret
   type: Opaque
   stringData:
     encryptionKeys: |
       <ENCRYPTION_KEYS>
     sslKeystorePassword: <SSL_KEYSTORE_PASSWORD>
   ```

   The parameters of the secret are explained in the following table.

   Table 1. **ibm-dba-ads-designer-secret** configuration parameters

Parameter	Description	Mandatory
encryptionKeys	A data structure that encapsulates the secret to generate an Advanced Encryption Standard (AES) symetric key for ciphering/deciphering the secrets. The data structure must be: `{"activeKey":"key1","secretKeyList":[{"secretKeyId":"key1", "value":"123344566745435"},{"secretKeyId":"key2", "value":"987766544365675"}]}` Where **activeKey** must exist in the **secretKeyList** and designates the secret to use for symetric key derivation. If the **activeKey** value does not exist, Decision Designer won't start.	Yes
sslKeystorePassword	A password used to generate SSL keystores.	Yes

   Important: **encryptionKeys** is used to encrypt and decrypt some sensitive values in the database. The active key is used to encrypt new values. If the active key has been compromised or needs to be replaced, an administrator can define another one and reference it as the active key. Secrets that have been encrypted by older keys continue to be decrypted provided that the key ID is preserved. But it is recommended to re-encrypt them from Decision Designer, and then remove the old keys from **encryptionKeys**.
2. Take care of untrusted TLS certificates.
   If Decision Designer is supposed to interact with servers like Git, Machine Learning (ML), MongoDB, or other servers whose TLS certificates are not signed by an official Certificate Authority (CA), you must gather these certificates and configure Decision Designer to establish trust with these servers.

   a. Get the TLS certificate of a running service.

   ```
   openssl s_client -connect <hostname>:<port> -servername <hostname> < /dev/null | sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p'
   ```

   b. Create a ConfigMap to hold the TLS certificates of the Git servers,. (You will set this ConfigMap to the **decision_designer.git_servers_certs** parameter of the custom resource file, at a later stage in the installation.)

   ```
   apiVersion: v1
   kind: ConfigMap
   metadata:
     name: ibm-dba-ads-designer-git-certificates
   data:
     my_git_server1.crt: |
       -----BEGIN CERTIFICATE-----
       ...
       -----END CERTIFICATE-----
     my_git_server2.crt: |
       -----BEGIN CERTIFICATE-----
       ...
       -----END CERTIFICATE-----
   ```

   c. Create a ConfigMap to hold the TLS certificates of the ML providers. (You will set this ConfigMap to the **decision_designer.ml_providers_certs** parameter of the custom resource file, at a later stage in the installation.)

   ```
   apiVersion: v1
   kind: ConfigMap
   metadata:
     name: ibm-dba-ads-designer-ml-providers-certificates
   data:
     my_ml_provider1.crt: |
       -----BEGIN CERTIFICATE-----
            ...
       -----END CERTIFICATE-----
     my_ml_provider2.crt: |
       -----BEGIN CERTIFICATE-----
            ...
       -----END CERTIFICATE-----
   ```

   d. Create a ConfigMap to hold the TLS certificates of the external MongoDB or any other server. (You will set this ConfigMap to the **decision_designer.other_trusted_certs** parameter of the custom resource file, at a later stage in the installation.)

   ```
   apiVersion: v1
   kind: ConfigMap
   metadata:
     name: ibm-dba-ads-designer-mongo-other-certificates
   data:
     mongo_certificate.crt: |
   ```

```
    -----BEGIN CERTIFICATE-----
        ...
    -----END CERTIFICATE-----
  another_certificate.crt: |
    -----BEGIN CERTIFICATE-----
        ...
    -----END CERTIFICATE-----
```

Note: The key for each certificate entry must be a valid filename with a `.crt` extension.
For more information about Decision Designer configuration parameters, see Configuration parameters.

Configuring the decision runtime

You can customize the decision runtime secret and persistent volume.

About this task

The customization steps described here are optional as the installation applies the default configuration. However, if you change any parameter value inside the secret, you must update the secret.

Procedure

1. Generate a runtime secret for sensitive configuration.
   Sensitive configuration settings must not be exposed in the custom resource YAML file.

   The **ibm-ads-operator** generates some of the settings automatically but you might want to define your own configuration.

   For example, you can create the following **ibm-dba-ads-runtime-secret**:

```
apiVersion: v1
kind: Secret
metadata:
  name: ibm-dba-ads-runtime-secret
type: Opaque
stringData:
  decisionServiceUsername: "drs"
  decisionServicePassword: "drsPassword"
  decisionServiceManagerUsername: "drsManager"
  decisionServiceManagerPassword: "drsManagerPassword"
  decisionRuntimeMonitorUsername: "drsMonitor"
  decisionRuntimeMonitorPassword: "drsMonitorPassword"
  deploymentSpaceManagerUsername: "depManager"
  deploymentSpaceManagerPassword: "depManagerPassword"
  encryptionKeys: |
    <ENCRYPTION_KEYS>
  sslKeystorePassword: "ssl-keystore-password"
```

   The parameters of the secret are explained in the following table.

Table 1. `ibm-dba-ads-runtime-secret` configuration parameters

Parameter	Description	Mandatory
decisionServiceUsername	The user name to authenticate with the decision runtime server for executing decision services.	Yes
decisionServicePassword	The user password to authenticate with the decision runtime server for executing decision services.	Yes
decisionServiceManagerUsername	The user name to authenticate with the decision runtime server for managing the decision archives. Not affected by **decision_runtime.authentication_mode**.	Yes
decisionServiceManagerPassword	The user password to authenticate with the decision runtime server for managing the decision archives. Not affected by **decision_runtime.authentication_mode**.	Yes
decisionRuntimeMonitorUsername	The user name to authenticate with the decision runtime server for monitoring the runtime server. Not affected by **decision_runtime.authentication_mode**.	Yes
decisionRuntimeMonitorPassword	The user password to authenticate with the decision runtime server for monitoring the runtime server. Not affected by **decision_runtime.authentication_mode**.	Yes
deploymentSpaceManagerUsername	The user name to authenticate with the decision runtime server for managing the deployment spaces. Not affected by **decision_runtime.authentication_mode**.	Yes
deploymentSpaceManagerPassword	The user password to authenticate with the decision runtime server for managing the deployment spaces. Not affected by **decision_runtime.authentication_mode**.	Yes
encryptionKeys	A data structure that encapsulates the secret to generate an Advanced Encryption Standard (AES) symetric key for ciphering/deciphering the secrets. The data structure must be: `{"activeKey":"key1","secretKeyList":[{"secretKeyId":"key1", "value":"123344566745435"},{"secretKeyId":"key2", "value":"987766544365675"}]}` Where **activeKey** must exist in the **secretKeyList** and designates the secret to use for symetric key derivation. If the **activeKey** value does not exist, the decision runtime won't start.	Yes
sslKeystorePassword	A password used to generate SSL keystores.	Yes

2. Take care of untrusted TLS certificates.
   a. Get the TLS certificate of a running service.

```
openssl s_client -connect <hostname>:<port> -servername <hostname> < /dev/null | sed -ne '/-BEGIN CERTIFICATE-/,/-END CERTIFICATE-/p'
```

b. Create a ConfigMap that holds the TLS certificates of the services that are accessed by the decision runtime, namely the MongoDB instance and the optional ML servers. (You will set this ConfigMap to the `decision_runtime_service.tls.certs_config_map_name` parameter of the custom resource file, at a later stage in the installation.)

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: runtime-tls-config
  labels:
    webapp: runtime
data:
  decision_storage.crt: |
    -----BEGIN CERTIFICATE-----
        ...
    -----END CERTIFICATE-----
```

Note: The key for each certificate entry must be a valid filename with a `.crt` extension.

For more information about the decision runtime configuration parameters, see Configuration parameters.

3. Optional: Configure the runtime service persistent volume. Skip this step if you choose S3 compatible storage.
By default, dynamic provisioning is applied.

Option 1: Using dynamic provisioning

With `decision_runtime_service.persistence.use_dynamic_provisioning` set to true, you must specify a `decision_runtime_service.persistence.storage_class_name` according to storage classes guidelines. Use a ReadWriteMany (RWX) class.
Setting `storage_class_name` to `null` means using the default storage class.
Warning: The `decision_runtime_service.persistence.storage_class_name` parameter cannot be changed after installation.

Option 2: Creating a persistent volume

If you decide to set `decision_runtime_service.persistence.use_dynamic_provisioning` to false in the custom resource file (at a later stage in the installation), you must provide a matching persistent volume. You can create one like this:

```
kind: PersistentVolume
apiVersion: v1
metadata:
  name: ibm-dba-ads-runtimeservice-pv
spec:
  accessModes:
    - ReadWriteMany
  capacity:
    storage: 1Gi
  persistentVolumeReclaimPolicy: Recycle
  # example setup for nfs server - for other setups, see the Kubernetes Docs on PersistentVolumes
  nfs:
    server: "1.2.3.4"
    path: "/data/ads/runtime"
```

Make sure you grant permissions to the root group `0` because the user that runs the runtime service pods belongs to this group.
The persistent volume must deliver a minimum performance of 300 IOPS. On ROKS (IBM Cloud®), this level of performance is achieved by a volume of 30Gi bytes from the ibmc-file-gold-gid storage class, or by a volume of 150Gi byte from the ibmc-file-bronze-gid storage class. For details, see Storage class reference

The persistent volume must support read and write access for a non-root user. The *-gid storage classes on IBM Cloud typically provide this support.

- **Configuring the event emitter for an external Kafka instance**
To gain insights into the decision-making process of your applications, you can configure the decision runtime to emit events that contain technical and business data to an external Kafka instance.
- **Connecting to S3 compatible storage**
The decision runtime supports two types of decision service archive storage systems: the file system for Kubernetes volumes and storage that is compatible with the S3 API.

Configuring the event emitter for an external Kafka instance

To gain insights into the decision-making process of your applications, you can configure the decision runtime to emit events that contain technical and business data to an external Kafka instance.

About this task

Configure the custom resource (CR) YAML file to emit events to an external Kafka instance.

For more information about emitted data, see Emitting decision execution events.

Procedure

1. The sample configuration that targets a Kafka instance that is deployed in the same Kubernetes cluster as Automation Decision Services:

```
decision_runtime:
    event_emitter:
      enabled: true
      kafka_topic: adsruntime
      kafka_bootstrap_servers: kafkatest-kafka-bootstrap.strimzi.svc:9093
      kafka_connection_secret_name: kafka-auth  # properties kafka-username and kafka-password
```

```
    kafka_security_protocol: SASL_SSL
    kafka_sasl_mechanism: SCRAM-SHA-512
```

When you use a Simple Authentication Security Layer (SASL) authentication mechanism that requires credentials (**SCRAM-SHA-512** or **PLAIN**), the Kubernetes secret that is referenced by the **kafka_connection_secret_name** property must contain two keys: **kafka-username** and **kafka-password**.

When you target a Kafka instance over Transport Layer Security (TLS) (**kafka_security_protocol: SASL_SSL**), the TLS certificate that establishes trust to this instance can be included in one of the following certificates:

- Cloud Pak TLS trust certificate list (**spec.shared_configuration.trusted_certificates**)
- Decision runtime trust certificates for Automation Decision Services (**spec.ads_configuration.decision_runtime.decision_runtime_service.tls.certs_config_map_name**)
- Secret that is referenced by the **kafka_connection_secret_name** as the **kafka-server-certificate** property

For more information about the **event_emitter.*** parameters, see Decision runtime parameters.

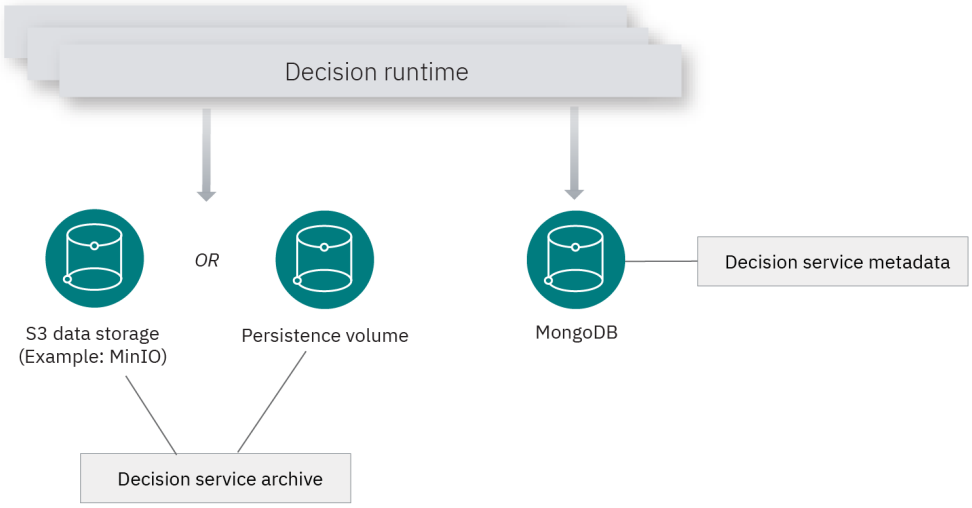
2. Configure the server certificate. The certificate is provided in PEM format in a Kubernetes ConfigMap, which is similar to other certificates such as the ones for the decision service archive repository and machine learning providers.
For more information, see Configuring the decision runtime.

Connecting to S3 compatible storage

The decision runtime supports two types of decision service archive storage systems: the file system for Kubernetes volumes and storage that is compatible with the S3 API.

You can configure the decision runtime to store and manage the decision service archives in the following storage systems:

- File system that is based on a Kubernetes RWX persistent volume
- Data storage that is compatible with the S3 API



Consider using the storage compatible with the S3 API when you have more advanced requirements for scalability, security, data management, and reliability to store decision archives. For example, replications of data on several nodes might be necessary to ensure high availability when a major issue occurs. In this case, the use of the storage compatible with the S3 API might be more appropriate and less costly to operate compared to a shared file system.

The storage compatible with the S3 API is not embedded in Automation Decision Services. It can be installed on premises, or you can use a SaaS offering. Consider an offer that suits your needs.

The metadata that are associated to decision services archives are stored in MongoDB. Sensitive metadata can be encrypted.

When you configure the decision runtime storage, the value of the storage type (**fs** or **s3**) in the **archive_storage_type** parameter cannot be changed after decision archives are deployed in the decision runtime.

For more information about the parameter and the storage types, see Configuration parameters.

To use an S3 server, you must configure several parameters:

- Type of supported decision service archive storage (S3)
- URL of the S3 server
- Bucket/container name
- Region: The region parameter is useful only for MinIO, when the region is specified at installation of the MinIO server.
- Region storage: The region storage is useful only for IBM Cloud Object Storage S3 API, not for MinIO.
- Kubernetes secret name that contains the S3 server credentials for nonanonymous connection

For example, you can set the following parameters:

```
spec:
  (...)
  decision_runtime:
```

```
  (...)

    archive_storage_type: "s3"
    s3:
      bucket_name: "mybucketname"
      region: "us-east-1"
      server_url: "https://theminioserverurl"
      secret_name: "theminiosecretname"
```

For more information about the decision runtime parameters for S3, see Decision runtime parameters.

Connecting to the storage server

Consider the following points when you connect to your storage server.

Table 1. Storage servers

Item	MinIO server	IBM Cloud Object Storage S3 API	Amazon Simple Storage Service (Amazon S3) server
Authentication	To create the Kubernetes secret, you must specify MinIO server credentials in the secret parameters with properties names: **s3Username** and **s3Password**. The name of the secret is stored in the **s3.secret_name** property. If no secret is specified, then the decision runtime tries to connect anonymously to the MinIO server.	You must first create HMAC credentials in the Service credentials menu to connect to the IBM Cloud Object Storage S3 API securely. For more information about service credentials, see Service credentials. To have an authenticated connection, you must create a secret that stores the HMAC credentials. The name of this secret must be stored in the **s3.secret_name** property. The secret must have two properties with names: • **s3Username**: Stores the value for **access_key_id** for HMAC. • **s3Password**: Stores the value for **secret_access_key** for HMAC. If no secret is specified, the decision runtime tries to connect anonymously to the IBM Cloud Object Storage S3 API.	Specify your AWS access key ID and password attached to the account that accesses your S3 bucket. The AWS access key ID and password must be specified in the secret parameters with properties names: **s3Username** and **s3Password**. The name of the secret is stored in the **s3.secret_name** property.
Bucket	If the bucket does not exist, it is automatically created at the first **POST** operation of a decision service archive to the MinIO server.	If the bucket does not exist, it is automatically created at the first **POST** operation of a decision service archive to the S3 server.	Specify your bucket name.
Type of region	If the region is specified when the MinIO server is installed, you must also specify the same region to connect to the MinIO server. If the region is not specified, you can specify any value. For the MinIO server, the region storage value is optional.	The type of region depends on the following items: • Type of storage that you want to use (**cross-region**, **regional**, or **single-site**) • Type of region (**eu**, **us**, ...) • Type of connection (**public**, **private**, or **direct**) You can find all the region endpoints when you create the HMAC credentials. For IBM Cloud Object Storage S3, you can find information about the endpoints as JSON objects in the following URL: https://control.cloud-object-storage.cloud.ibm.com/v2/endpoints (select the **JSON** tab for better readability).	Specify the region where your S3 bucket is accessible.
Type of region storage	--	The type of region storage depends on the type of storage of your bucket and the type of region. For example, if you want to create a bucket with the type of storage **standard** with the type of region **eu-de**, your type of region storage is **eu-de-standard**.	Specify the same value as the bucket region.
Server URL	--	Example URL: **https://s3.eu-de.cloud-object-storage.appdomain.cloud** For more information, see the s3.server_url parameter in Decision runtime parameter.	Example URL: **https://s3.eu-west-3.amazonaws.com** Note: This example is not the URL for the S3 bucket. The bucket name must not be present in this URL.

Installing Automation Decision Services

Installation of Automation Decision Services uses an operator, which is a Kubernetes feature that makes it simpler to install and update without having to worry about the underlying cloud provider.

Before you begin

Prepare your environment. For more information, see Preparing to install Automation Decision Services.

- **Installing the Automation Decision Services operator**
  The Automation Decision Services installation script installs the Automation Decision Services operator and its dependencies.
- **Checking and completing your custom resource**
  A custom resource YAML is a configuration file that describes an instance of a deployment that is created by the operator and includes parameters to install

Automation Decision Services. You can use a sample for minimal custom resources (CR) that is available on GitHub, and author and customize it for your needs.

- **Applying the custom resources for Automation Decision Services**
  What you apply in the custom resources (CR) for Automation Decision Services depends on your target Kubernetes cluster. You apply certain choices available in the storage classes.
- **Custom resource statuses**
  Automation Decision Services has a status value in the custom resource (CR) instance that is used by the operator.

Installing the Automation Decision Services operator

The Automation Decision Services installation script installs the Automation Decision Services operator and its dependencies.

Before you begin

Use a bastion server as your host. You must run the script from a bastion server in a session where you are connected to the targeted cluster.

Bastion host
: A bastion host is a server that is provisioned with a public IP address that is accessible through remote access Secure Shell (SSH). When configured, the bastion server acts as an intermediate server that allows a secure connection to the instances made available without a public IP address.

About this task

The installation script (**`ads-install.sh`**) is available in the following GitHub repository: https://github.com/icp4a/automation-decision-services-kubernetes.

This script installs the Automation Decision Services operator and its dependencies in a targeted namespace.

Procedure

Run the **`ads-install.sh`** script:

```
./scripts/ads-install.sh [-h] [-a] -n <ads_namespace> [-d <domain_name>]
```

Description of the options:

Table 1. Options

Option	Description	Purpose
`-a`	Accept license.	
`-n` ***`<ads_namespace>`***	Namespace where Automation Decision Services is installed.	The **`-n`** switch is to set the namespace where you are aiming to install Automation Decision Services.
`-d` ***`<domain_name>`***	Domain name where the Automation Decision Services URL is available. This option is mandatory unless you are using Red Hat OpenShift where it is ignored.	The **`-d`** switch is to pass the domain name. It is used by the foundational services to derive the Platform UI (Zen) console URL and the Cloud Pak console URL where your Automation Decision Services instance is reachable from a user browser. These URLs are also declared internally in authentication mechanism, and must be recognized before the installation. On Red Hat OpenShift, the same domain name as the one that is used in the Red Hat OpenShift admin console URL is used by default. Therefore, you do not need to provide one.

Results

When the script finishes running, you have everything that you need in your cluster to instantiate Automation Decision Services custom resources (CR) into the target namespace.

Checking and completing your custom resource

A custom resource YAML is a configuration file that describes an instance of a deployment that is created by the operator and includes parameters to install Automation Decision Services. You can use a sample for minimal custom resources (CR) that is available on GitHub, and author and customize it for your needs.

About this task

Samples for minimal CR and fully customized CR are available in the following GitHub repository: https://github.com/icp4a/automation-decision-services-kubernetes. You can check a sample CR file, minimal or fully customized, and update it.

For more information about the configuration parameters that are used in the CR, see Configuration parameters.

- **Configuring Automation Decision Services**
  Before you install, configure the custom resource YAML file for your Automation Decision Services deployment.

Configuring Automation Decision Services

Before you install, configure the custom resource YAML file for your Automation Decision Services deployment.

Before you begin

Make sure that you follow the instructions in Preparing to install Automation Decision Services where you configure important settings such as secrets, certificates, and persistent volumes.

Procedure

1. Specify which components to deploy and how to access them from outside the cluster.
   To do so, set the following parameters in the custom resource YAML file:

Table 1. Basic Automation Decision Services configuration parameters

Parameter	Description	Default
spec.decision_designer.enabled	A flag to control whether Decision Designer is deployed or not.	false
spec.decision_runtime.enabled	A flag to control whether the decision runtime is deployed or not.	false
spec.deployment_profile_size	A flag to control the default CPU/memory resources and replica count to be applied to Automation Decision Services services and pods. Possible values are: **small**, **medium**, **large**, and **extra-large**.	small

```
  spec:
    # Actually overloaded by other values below
    deployment_profile_size: "small"

    decision_designer:
      enabled: true
      deployment_profile_size: "medium"

    decision_runtime:
      enabled: true
      deployment_profile_size: "large"
```

   The subsequent steps 2-6 are optional.

2. Optional: If you do not want to keep the default values of the Decision Designer container services, set the resources requests and limits.
   The Decision Designer containers include:
   - `git_service`
   - `parsing_service`
   - `run_service`
   - `credentials_service`
   - `embedded_build_service`
   - `rest_api`

   Specify the values of the following parameters for each container.

Table 2. Decision Designer containers configuration parameters

Parameter	Description
image.repository	The registry and namespace where the container images are located.
image.tag	The image tag name of the container.
replica_count	The number of desired replica of the container.
resource.limits.memory	Specifies the memory limit for the container.
resource.limits.cpu	Specifies the CPU limit for the container.
resource.requests.memory	Specifies the memory request for the container.
resource.requests.cpu	Specifies the CPU request for the container.

   For example:

```
decision_designer:
  rest_api:
    image:
      repository: "cp.icr.io/ads-runtime"
    replica_count: "4"
    resources:
      limits:
        memory: "2G"
        cpu: "2000m"
      requests:
        memory: "512"
        cpu: "500m"
```

   For more information about the default resource values of the Decision Designer container services, see System requirements.

   For more information about the Decision Designer configuration parameters, see Decision Designer parameters.

3. Optional: If you do not want to keep the default values of the decision runtime container service, set the resources requests and limits as well as the horizontal autoscaling.
   Specify the values of the following parameters.

Table 3. Decision runtime container configuration parameters

Parameter	Description
image.repository	The registry and namespace where the container images are located.
image.tag	The image tag name of the container.

Parameter	Description
image.digest	The image digest name of the container, if you prefer to use the digest instead of the tag. Digest has priority over tag.
replica_count	The number of desired replica of the container. Ignored if **autoscaling.enabled** is set to true. Default value is 2.
resource.limits.memory	Specifies the memory limit for the container. Default value is 2Gi.
resource.limits.cpu	Specifies the CPU limit for the container. Default limit is 2.
resource.requests.memory	Specifies the memory request for the container. Default is 512Mi.
resource.requests.cpu	Specifies the CPU request for the container. Default is 1.
autoscaling.enabled	Specifies whether horizontal pod autoscaling is enabled or not. Default value is false in the **small**, **medium**, and **large** modes, true in the **extra-large** installation mode.
autoscaling.min_replicas	Minimum number of replicas. Default value is 2.
autoscaling.max_replicas	Maximum number of replicas. Default value is 5.
autoscaling.target_cpu_average_utilization	Determines when a new pod is created, based on the value of **resource.requests.cpu**. Default value is 160, which means that a new pod is created when the CPU usage is more than 1.6 the value of **resource.requests.cpu**.

For example, if you do NOT want to use horizontal pod autoscaling:

```
decision_runtime:
  decision_runtime_service:
    image:
      repository: "cp.icr.io/ads-runtime"
    replica_count: "4"
    autoscaling:
      enabled: false
    resources:
      limits:
        memory: "2Gi"
        cpu: "2000m"
      requests:
        memory: "512Mi"
        cpu: "500m"
```

For example, if you DO want to use horizontal pod autoscaling:

```
decision_runtime:
  decision_runtime_service:
    image:
      repository: "cp.icr.io/ads-runtime"
    autoscaling:
      enabled: true
      min_replicas: 2
      max_replicas: 5
      target_cpu_average_utilization: 160
    resources:
      limits:
        memory: "2Gi"
        cpu: "2000m"
      requests:
        memory: "512Mi"
        cpu: "1000m"
```

For more information about the default resource values of the decision runtime container service, see System requirements.

For more information about the decision runtime configuration parameters, see Decision runtime parameters.

4. Optional: You can specify additional labels for the Automation Decision Services pods.
For example:

```
spec:
  decision_runtime:
    labels:
       key: value
    decision_runtime_service:
      labels:
        key: value    # overrides decision_runtime.labels.key
        key2: value2  # addition specific to decision_runtime_service
        ..

  decision_designer:
    labels:
      key:value

  mongo:
    labels:
      key: value
```

- The **labels** section under the **decision_designer** element adds labels to all deployments and pod that compose Decision Designer (**rest-api**, **git-service**, **run-service**, and **credentials-service**).
- The **labels** section under the **decision_runtime** element adds labels to all deployments and pod that compose the decision runtime (**decision-runtime-service**).

You can also specify labels in the more specialized sections for each deployment, such as **git_service**, and **decision_runtime_service**. These specialized labels override the ones in the general sections.

For more information about the syntax for the individual label keys and values, see the Kubernetes documentation.

Restriction: Labels that use the prefixes `kubernetes.io`, `icp4a.ibm.com`, and `ads.ibm.com` are reserved and forbidden in these `labels` sections. Same for the `release` label.

5. Optional: Set any other configuration parameters in the custom resource YAML file. Refer to Preparing to install Automation Decision Services where you might have defined important settings such as secrets, certificates, and persistent volumes.
   For a complete list of configuration parameters, see Configuration parameters.

Applying the custom resources for Automation Decision Services

What you apply in the custom resources (CR) for Automation Decision Services depends on your target Kubernetes cluster. You apply certain choices available in the storage classes.

Procedure

1. Run the following command to verify that Automation Decision Services has a valid `yaml` file, and it conforms to the expected grammar:

   ```
   kubectl apply -f <ads_cr_path> --validate='strict'
   ```

   Remember: Running the command does not prevent certain errors such as unintentional duplications of a section.
2. When you finish applying your CR, you can view the progress of the installation by checking your CR status.
   For example, you can run the following command:

   ```
   kubectl get ads ads-standalone -n ads -o yaml -o go-template='{{range .status.conditions}}CONDITION: {{.type}}{{"\n"}}
   STATUS: {{.status}}{{"\n"}}  MESSAGE: {{.message}}{{"\n"}}{{end}}'
   CONDITION: Running
     STATUS: False
     MESSAGE:
   CONDITION: Ready
     STATUS: True
     MESSAGE:
   CONDITION: MongoInstalled
     STATUS: True
     MESSAGE:
   CONDITION: DesignerInstalled
     STATUS: True
     MESSAGE:
   CONDITION: RuntimeInstalled
     STATUS: True
     MESSAGE:
   ```

   Note: Running indicates whether the Automation Decision Services operator is reconciling. Its status is `True` if it is processing a CR change, or it was fired by a Kubernetes event it watches (like a secret modification). False is expected when no reconciliation is ongoing.
   You might have some expected transient errors (unknown type errors) reported in the status while the foundational services operators and custom resource definitions (CRD) are pulled and started. Then, `ZenService` starts and the Automation Decision Services status reports its progress.

   For more information about the CR statuses, see Custom resource statuses.

   Tip:
   The Automation Decision Service status is set to ready when all Automation Decision Services components have at least one pod ready. It can take about 30 minutes for the first start before it's ready.

   When you run the following command, you can wait until the Automation Decision Services installation status becomes ready:

   ```
   kubectl wait ads ads-standalone -n ads --for=condition=Ready=True --timeout=30m
   ```

Results

When all the underlying layers are ready, Decision Designer and the decision runtime are installed (you can choose to install one of them or both).

What to do next

Go to Completing post-installation tasks for Automation Decision Services.

Custom resource statuses

Automation Decision Services has a status value in the custom resource (CR) instance that is used by the operator.

The Automation Decision Services CR contains a status section with a list of conditions.

For example:

```
  status:
    conditions:
    - lastTransitionTime: "2023-05-23T07:38:01Z"
      message: ""
      reason: NotRunning
      status: "False"
      type: Running
    - lastTransitionTime: "2023-05-22T12:58:54Z"
      message: ""
      reason: Ready
```

```
    status: "True"
    type: Ready
  - lastTransitionTime: "2023-05-22T12:58:54Z"
    message: ""
    reason: Installed
    status: "True"
    type: MongoInstalled
  - lastTransitionTime: "2023-05-22T12:58:54Z"
    message: ""
    reason: Installed
    status: "True"
    type: DesignerInstalled
  - lastTransitionTime: "2023-05-22T12:58:54Z"
    message: ""
    reason: Installed
    status: "True"
    type: RuntimeInstalled
  endpoints:
  - caSecret:
      key: ca.crt
      secretName: iaf-system-automationui-aui-zen-cert
    name: adsDesignerUI
    scope: External
    type: UI
    uri: https://cpd-ads.<subdomain.domain>/ads
  - caSecret:
      key: ca.crt
      secretName: iaf-system-automationui-aui-zen-cert
    name: adsRuntimeSwaggerUI
    scope: External
    type: UI
    uri: https://cpd-ads.<subdomain.domain>/ads/runtime/api/swagger-ui
  observedGeneration: 83
  operatorInfos:
  - name: commitID
    value: 914cef8481991e57673f9161ea805e896845088c
  versions:
  - name: operator
    version: 23.0.2
  - name: ads
    version: 23.0.2
```

Table 1. `conditions` custom resource status

Type	Description
`Running`	Indicates whether the Automation Decision Services operator is reconciling. Its status is `True` if it is processing a CR change, or it was fired by a Kubernetes event it watches (like a secret modification). `False` is expected when no reconciliation is ongoing.
`Ready`	Indicates whether the Automation Decision Services topology is ready to be used. If its status is `False`, a message is displayed to explain why it is not ready in the message field.
`MongoInstalled`	Indicates whether the embedded MongoDB is used, and all prechecks and installation steps are done.
`DesignerInstalled`	Indicates whether Decision Designer is used, and all prechecks and installation steps are done.
`RuntimeInstalled`	Indicates whether the decision runtime is used, and all prechecks and installation steps are done.

Table 2. `endpoints` custom resource status

Name	Description
`adsDesignerUI`	Provides the entry point URL to access the Automation Decision Services instance.
`adsRuntimeSwaggerUI`	Provides the decision runtime Swagger UI URL to interact graphically with the decision runtime.

Table 3. `operatorInfos` custom resource status

Name	Description
`commitID`	ID to identify at the exact operator code source level. It can be used when you contact the support team for diagnosing an issue.

Table 4. `versions` custom resource status

Name / version	Description
`operator` version	The current installed operator version.
`ads` version	The Automation Decision Services version that is applied to the cluster.

Completing post-installation tasks for Automation Decision Services

The Automation Decision Services application runs after the CR is applied. However, it is still not reachable from the outside of your cluster. After the Automation Decision Services container is deployed to the cluster, you need to take additional steps.

Before you begin

Note: If you want to set up a build environment for Automation Decision Services, see Setting up your build environment.

- **(Optional) Creating Kubernetes Ingresses**
  You create ingresses to log in to the Automation Decision Services home page. If you are using OpenShift Container Platform (OCP), you do not need to create Ingresses.

- **Accessing the Automation Decision Services home page and the Zen console**
  You can log in to the Automation Decision Services home page or the Platform UI (Zen) console for the first time as the `cpadmin` user that is configured by default. After configuring an identity provider, you can allow more users to log in.
- **Configuring an identity provider**
  You configure Identify and Access Management (IAM) against your identity provider to allow more users to log in to the Platform UI (Zen) console and the Automation Decision Services home page.
- **Managing user permissions**
  You can manage various user permissions for Automation Decision Services in Platform UI (Zen). These permissions are used to access the administration page (`admin-platform`) or control access to decision runtime REST API endpoints and manage decision service archives and their metadata.
- **Tuning the logging level**
  To fine-tune and make it easier to analyze the logs, you might want to adjust the log trace of Decision Designer and the decision runtime services.

(Optional) Creating Kubernetes Ingresses

You create ingresses to log in to the Automation Decision Services home page. If you are using OpenShift Container Platform (OCP), you do not need to create Ingresses.

Before you begin

If you are using OpenShift Container Platform, skip this procedure and go to Configuring an identity provider.

About this task

The script (`ads-generate-ingresses.sh`) is available in the following GitHub repository: https://github.com/icp4a/automation-decision-services-kubernetes.

This script generates a file that contains ingresses Kubernetes descriptors for the foundational services and licensing service. This file is used on a cluster with the ingress-nginx contoller. If it's not the case, you need to review it and adapt it with the equivalent annotations for your ingress controller technology.

Procedure

1. Run the `ads-generate-ingresses.sh` script:

   ```
   ./scripts/ads-generate-ingresses.sh [-h] -n <ads_namespace> [-o output-file]
   ```

   Description of the options:
   - `-n <ads_namespace>`: Namespace where Automation Decision Services is installed.
   - `-o output-file`: A file where the Kubernetes manifests is generated. Default is a temporary file.
2. Apply the ingresses by running the following command:

   ```
   kubectl apply -f <your_ingress_file>
   ```

Accessing the Automation Decision Services home page and the Zen console

You can log in to the Automation Decision Services home page or the Platform UI (Zen) console for the first time as the `cpadmin` user that is configured by default. After configuring an identity provider, you can allow more users to log in.

Before you begin

By default, an admin user is created for `mgmt-ingress` with its details in the `platform-auth-idp-credentials` secret.

Procedure

1. Use the following commands to get credentials:

   ```
   kubectl get secret platform-auth-idp-credentials -n ads -o jsonpath='{.data.admin_username}' | base64 --decode
   kubectl get secret platform-auth-idp-credentials -n ads -o jsonpath='{.data.admin_password}' | base64 --decode
   ```

2. The URL of the Automation Decision Services home page is available in the `uri` section for `adsDesignerUI` in the `endpoints` custom resource (CR) status. For more information about the CR status, see Custom resource statuses.
   Example of the `endpoints` CR status:

   ```
   endpoints:
   - caSecret:
       key: ""
       secretName: ""
     name: adsDesignerUI
     scope: External
     type: UI
     uri: https://ads-cpd.<subdomain.domain>/ads
   - caSecret:
       key: ""
       secretName: ""
     name: adsRuntimeSwaggerUI
     scope: External
     type: UI
     uri: https://ads-cpd.<subdomain.domain>/ads/runtime/api/swagger-ui
   observedGeneration: 1
   ```

3. The URL of the Zen console is similar to the Automation Decision Services home page URL. You need to remove /ads from the Automation Decision Services home page URL.
   Example URL of the Automation Decision Services home page:

   `https://ads-cpd.<subdomain.domain>/ads`

   In this case, example URL of the Zen console:

   `https://ads-cpd.<subdomain.domain>`

   Tip: You can also access the Zen console by clicking the navigation menu in the upper left of the portal page in the Automation Decision Services home page, and selecting Home.

Configuring an identity provider

You configure Identify and Access Management (IAM) against your identity provider to allow more users to log in to the Platform UI (Zen) console and the Automation Decision Services home page.

Procedure

To configure an identity provider, check IM for your product platform users in the foundational services documentation.

Managing user permissions

You can manage various user permissions for Automation Decision Services in Platform UI (Zen). These permissions are used to access the administration page (**admin-platform**) or control access to decision runtime REST API endpoints and manage decision service archives and their metadata.

Before you begin

Sign in to the Zen console as an administrator.
For more information about managing users in the Zen console, see Managing users in the IBM Cloud Pak Platform UI documentation.

About this task

No user has any user permission by default. Therefore, the administrator must assign a user role that contains appropriate user permissions to each user.

A role is a container of permissions. You create a role and add permissions to the role, and then assign the role to users or user groups.

Tip: You can also assign predefined roles available in the Zen console to users or user groups. In this case, you can skip Step 2 in the Procedure section.

Procedure

1. Click Manage users in the administrator's UI.
2. Create a role and add permissions to it.
   a. In the Roles tab on the Access control page, click New role.
   b. In the Details section, enter a name of the role and its description. Click Next.
   c. In the Permissions section, expand IBM Cloud Pak for Business Automation and select permissions to add to the role. Click Next.

Table 1. Permissions for Automation Decision Services

User permissions	Description
Administer platform for decision services	Users with this permission can manage credentials for decision services in Decision Designer. You can access the administration page (**admin-platform**) in Decision Designer to perform various tasks as an administrator.
Execute decision services	Users with this permission can execute decisions and invoke related endpoints. Examples of what related endpoints can do: • List the operations of a decision service. • Generate an OpenAPI specification for a decision service. • Retrieve an example payload for a decision service. • Generate the schemas of the input and output for a decision service.
Manage deployed decision services	Users with this permission can manage the decision service archives and associated metadata by using the create, retrieve, update, and delete operations on their respective storage service. Note: Users must have this permission to run, build, and deploy decision services in Decision Designer.
Manage deployment spaces	Users with this permission can manage deployment spaces.
Monitor decision runtime	Users with this permission can take a snapshot of the state of the decision runtime on demand.

For more information about the permissions for decision runtime, see User permissions and authentication modes.

   d. Verify the information in the Summary section, and then click Create.
3. Assign the role to users or user groups.
   For example, to assign the role to a user:
   a. In the Users tab on the Access control page, select a user.
   b. Click Assign roles on the user's page.

c. Select the role that you want to assign on the Assign roles page, and then click Assign 1 role.
You can assign the role that you created in the previous step, or you can assign a predefined role.

Table 2. Predefined roles

Predefined role	Description	Associated permission
Decision Designer Platform Administrator	Users with this role can manage credentials for decision services in Decision Designer. You can access the administration page (`admin-platform`) in Decision Designer to perform various tasks as an administrator.	Administer platform for decision services
Decision User	Users with this role can perform actions that are allowed with the Execute decision services permission.	Execute decision services
Deployed Decision Manager	Users with this role can perform actions that are allowed with the Manage deployed decision services permission.	Manage deployed decision services
Decision Runtime Monitor	Users with this role can perform actions that are allowed with the Monitor decision runtime permission.	Monitor decision runtime
Decision Runtime Deployment Spaces Manager	Users with this role can perform actions that are allowed with the Manage deployment spaces permission.	Manage deployment spaces

Results

Now the new role is assigned to a user. You can click View assigned permissions to check a list of permissions that the user has.

Tuning the logging level

To fine-tune and make it easier to analyze the logs, you might want to adjust the log trace of Decision Designer and the decision runtime services.

About this task

In the Custom Resource (CR) file of the operator, you can set the `log_trace_specification` parameter for the following Decision Designer services:

- Credentials service
- Git service
- Parsing service
- Run service
- Runtime service
- REST API

The value that you provide for this parameter determines the level of detail that is returned in the log traces of the Decision Designer services. Follow the syntax of the `traceSpecification` attribute described here Logging and Trace.

Procedure

1. In the CR file, set the logging level for each Decision Designer service.

   Credentials service

   ```
   spec:
     decision_designer:
        credentials_service:
           log_trace_specification:
   ```

   The following table lists common logging values for this service.

Value	Description
*=info	All loggers are set at the info level

   Git service

   ```
   spec:
     decision_designer:
        git_service:
           log_trace_specification:
   ```

   The following table lists common logging values for this service.

Value	Description
*=info	All loggers are set at the info level

   Parsing service

   ```
   spec:
     decision_designer:
        parsing_service:
           log_trace_specification:
   ```

   The following table lists common logging values for this service.

Value	Description
*=info	All loggers are set at the info level

Run service

```
spec:
  decision_designer:
     run_service:
        log_trace_specification:
```

The following table lists common logging values for this service.

Value	Description
*=info	All loggers are set at the info level

Runtime service

```
spec:
  decision_runtime:
     decision_runtime_service:
        log_trace_specification:
```

The following table lists common logging values for this service.

Value	Description
*=info	All loggers are set at the info level

REST API

```
spec:
  decision_designer:
     rest_api:
        log_trace_specification:
```

The following table lists common logging values for this service.

Value	Description
*=info	All loggers are set at the info level
*=info:org.apache.httpcomponents=finer	All HTTP traffic that goes through the HTTP client is logged with a finer level

2. To see the updated logs, wait for the pods to restart automatically after the operator reconciliation.

Uninstalling Automation Decision Services

To remove Automation Decision Services, delete the namespace that you used to install.

Before you begin

If you need to back up your data, make sure that you take the necessary steps to reinstall the deployment. For example, make a copy of the custom resource (CR) that you used in the environment. Make copies of the security definitions that are used to protect the configuration data in the environment. Make copies of the persistent volumes (PV) and persistent volume claims (PVC) in the environment.

About this task

Uninstalling and cleaning up the cluster involves several separate high-level tasks.

1. Uninstalling the Automation Decision Services deployment.
2. Uninstalling the operators.
3. Deleting PVs and PVCs. If you do delete the PVs/PVCs, the persistent data along with configuration and log files are also deleted. Delete the PVCs before you delete the PVs.
   Note: The PVs are deleted if it is in dynamic provisioning mode.
4. Dropping the databases if the embedded MongoDB is used.
5. Deleting the secrets created by the Automation Decision Services deployment.
   If you do delete all of the secrets that you created for Automation Decision Services, then you must create them again if you want to install another Automation Decision Services deployment.
6. Deleting the Automation Decision Services namespace.
7. Removing the foundational services.

To uninstall the Automation Decision Services deployment, use the following steps.

Procedure

1. To uninstall Automation Decision Services and its dependencies, run the ads_uninstall.sh script:

   ```
   ./scripts/ads-uninstall.sh -n ads
   ```

   where `ads` is the namespace where your Automation Decision Services is installed.
   The following items are removed:
   - Automation Decision Services instance in the `ads` namespace
   - `ads` namespace
2. Verify that the namespace is deleted.

Administering

As an administrator, you can perform various tasks.

- **Configuring credentials for a Maven repository manager**
  You need to configure certain credentials to access Maven repository managers where your external libraries are stored. When it is configured, business experts can import external libraries into Decision Designer at the decision service level, and start using the library in a decision service.
- **Configuring global machine learning providers**
  A global machine learning provider can be used for all decision automations that are defined on its instance of Automation Decision Services, and can be used by all users. Before you import samples that use predictive models, you must configure the sample machine learning provider available by default as a global machine learning provider.
- **Connecting to a remote repository automatically**
  A remote Git repository (remote repository) is created on a Git provider. By default, all decision automations that are created on the Automation Decision Services platform are not connected to a remote repository. However, you can configure the platform to automatically create a remote repository and connect new decision automations to it.
- **Setting up your build environment**
  To build decision services either from your local machine with a command line, or from a continuous integration and delivery (CI/CD) infrastructure, you need Java™ and Maven. You also need to download specific files if you want to use the execution Java API for decision services.
- **Configuring for disaster recovery**
  It is important to have a disaster recovery plan (DRP) that describes how work can be resumed quickly and effectively after a disaster.
- **Viewing metering information**
  You can view the metering information based on the execution count. You can query the information anytime.

Configuring credentials for a Maven repository manager

You need to configure certain credentials to access Maven repository managers where your external libraries are stored. When it is configured, business experts can import external libraries into Decision Designer at the decision service level, and start using the library in a decision service.

Before you begin

Note: You must have the Administer platform for decision services permission to access the administration page. This permission is associated with the Decision Designer Platform Administrator predefined role. For more information, see Manage user permissions.

Procedure

1. Go to the administration page in Decision Designer by adding `/admin-platform` at the end of the URL of Decision Designer in your browser:

   `https://<decision_designer_url>/admin-platform`

2. Open the Maven configuration tab.
3. To add a Maven repository manager, click New.
4. In the Repository URL field, enter the URL of the Maven repository.
   For example:

   `https://<Maven_repository_manager>/repository/maven-public`

   Note: You must use **`HTTPS`** in the URL. Do not use **`HTTP`**.
5. In the Authentication type field, select Credentials or Header from the drop-down menu.
   - If you select Credentials, enter the username and the password to access the Maven repository manager.
   - If you select Header, enter the header name and the header value that are required to authenticate to the Maven repository manager.
     For example, in GitLab, the header name is **`Private-Token`**, and the header value is your personal access token.
6. Click Save.
7. Optional: To edit or delete the configuration, click the overflow menu next to the Maven configuration, and select Edit or Delete.

What to do next

A business expert can now import an external library. For more information, see Importing and external library into Decision Designer.

Configuring global machine learning providers

A global machine learning provider can be used for all decision automations that are defined on its instance of Automation Decision Services, and can be used by all users. Before you import samples that use predictive models, you must configure the sample machine learning provider available by default as a global machine learning provider.

Before you begin

Note: You must have the Administer platform for decision services permission to access the administration page. This permission is associated with the Decision Designer Platform Administrator predefined role. For more information, see Manage user permissions.
You can have global or local machine learning providers for your decision automations:

Table 1. Machine learning providers

Machine learning provider	Description
Global machine learning providers	A global machine learning provider is accessible from all of your decision automations that are defined in its instance of Automation Decision Services and all users. You can configure global machine learning providers in the Global providers tab on the administration page.
Local machine learning providers	A local machine learning provider is local to your decision automation, and accessible only from your specified decision automation. You can configure local machine learning providers on the Settings page for your decision automation in Decision Designer. For more information, see Managing local machine learning providers.

Three remote machine learning providers are supported in Automation Decision Services: IBM Watson® Machine Learning, Amazon SageMaker, and IBM® Open Prediction Service.

An embedded machine learning provider is also supported. This provider allows you to import any type of Predictive Model Markup Language (PMML) file and run it directly in your automation.

For more information about these providers, see Managing local machine learning providers.

About this task

You can create, update, or delete global machine learning providers in the Global providers tab on the administration page.

In the Global providers tab, a global machine learning provider called Sample Machine Learning Provider is provided for you to test decision services that use predictive models. For example, some samples use this machine learning provider to import a predictive model markup language (PMML) description in their decision services.

By default, no URL is set for Sample Machine Learning Provider. You need to set it up for the machine learning provider of your choice.

If you want to use an out-of-box sample that uses predictive models, you must configure this provider before you import it. For more information about importing samples, see Creating decision services.

Note: Only global machine learning providers are listed in the Global providers tab.

Procedure

1. To create a global machine learning provider, retrieve the URL of the provider of your choice.

Table 2. Machine learning provider options

Provider	Credentials
Watson™ Machine Learning	Enter the following service credentials to authenticate with your Watson Machine Learning service instance: • API key • Space ID • Authentication URL • URL This information can be found on Watson Studio. The Authentication URL is populated with a default value automatically.
SageMaker	Enter the following service credentials to authenticate with your SageMaker instance: • Region • Access key ID • Access key value
Open Prediction Service	Enter the URL of your Open Prediction Service instance. You can get the URL while you install your instance. For more information about installing Open Prediction Service instances, see the instructions on GitHub.
Embedded Machine Learning	No credentials needed.

2. Go to the administration page by adding `/admin-platform` at the end of the URL of Decision Designer in your browser:

   `https://<decision_designer_url>/admin-platform`

3. Open the Global providers tab, and then click New.
4. Select the provider type in the drop-down list.
5. Enter the name and description for the global machine learning provider.
6. Complete the required fields depending on the provider type. You must use **`HTTPS`** in the URL field, not **`HTTP`**.
7. Click Test connection to verify the connection.
8. When it is successfully connected, click Save.

   To update or delete a provider, click the overflow menu ⋮ next to the provider name, and then click Edit or Delete.

9. Optional: To configure Sample Machine Learning Provider:
   a. Click the overflow menu ⋮ next to the provider name, and select Edit.
   b. Enter the description and the URL. See step 1 on how to get the URL. You must use **`HTTPS`** in the URL.
   c. Click Test connection to verify the connection. Click Save.

   Restriction: You cannot delete Sample Machine Learning Provider. You can only update it.

What to do next

A business expert can now connect to the machine learning provider. For more information, see Managing local machine learning providers.

Connecting to a remote repository automatically

A remote Git repository (remote repository) is created on a Git provider. By default, all decision automations that are created on the Automation Decision Services platform are not connected to a remote repository. However, you can configure the platform to automatically create a remote repository and connect new decision automations to it.

Before you begin

Note: You must have the Administer platform for decision services permission to access the administration page. This permission is associated with the Decision Designer Platform Administrator predefined role. For more information, see Manage user permissions.
When you configure the automatic connections, the following things are possible:

- When a user creates a decision automation, it is connected to a remote repository automatically. This is applied for all users.
- When the automatic connection fails, the decision automation is still created but it is not connected to a remote repository.
- When the automatic connection fails, you can still connect to another remote repository manually. For more information, see Connecting to a remote repository manually.

Restriction: You can configure one Git provider only. This Git provider is used for all users.

Procedure

1. Go to the administration page in Decision Designer by adding `/admin-platform` at the end of the URL of Decision Designer in your browser:

   `https://<decision_designer_url>/admin-platform`

2. Open the Git provider configuration tab.
3. To add a Git provider configuration, click New.
4. From the Git provider pull-down menu, select a Git provider.
   You can select one of the following providers:
   - GitHub
   - GitLab
   - Gitea
   - Bitbucket
   - Azure Repos
   - AWS CodeCommit
5. Complete the required fields for the Git provider you selected.

Table 1.

Field	Mandatory or optional	Description
Service URL	Mandatory	The URL for your Git provider API. The current default paths for the supported Git providers: • GitHub: For **`https://github.com`**, the URL for the provider API is **`https://api.github.com`**. This path is automatically set in the UI. • GitLab: **`https://gitlab.com/api/v4`** • Gitea: **`https://my-server.com/api/v1`** • Bitbucket: **`https://my-server.com/api/2.0`** • Azure Repos: The path is provided in the UI. • AWS CodeCommit: **`https://git-codecommit.<region>.amazonaws.com/v1/repos/`** Tip: You might need to use a different URL and path for specific cases; for example: • If you are using your own instance of GitHub; that is, if you are not using **`https://github.com`**, the URL for the provider API is **`https://my-server.com/api/v3`**. • For **`https://bitbucket.org`**, the URL for the provider API is **`https://api.bitbucket.org/2.0`** The **`https`** protocol must be used. Certificates must be properly recognized by the Git service pods.
Git provider name	Mandatory	A unique name of your Git provider. A Git provider name cannot contain the following characters: • Unicode control characters or surrogate characters • Any of the following characters: • / : \ ~ & % ; @ ' " ? < > | # $ * } { , + = [ ] • characters starting with an underscore (_) • characters starting or ending with a period (.)

<table>
<tr><th>Field</th><th>Mandatory or optional</th><th>Description</th></tr>
<tr><td>Repository details</td><td>What you need to select or enter in this field varies depending on the Git provider you selected.

Not required for AWS CodeCommit.</td><td>

Project name: Your project name. This field is available and mandatory only for Azure Repos.
Organization name: Your organization name.
Note: GitLab has two types of namespaces: a personal namespace, and a group or subgroup namespace. For more information, see the GitLab documentation.

<table>
<tr><th>Git provider</th><th>Mandatory or optional</th></tr>
<tr><td>GitHub</td><td>Optional</td></tr>
<tr><td>GitLab</td><td>Optional</td></tr>
<tr><td>Gitea</td><td>Optional</td></tr>
<tr><td>Bitbucket</td><td>Mandatory</td></tr>
<tr><td>Azure Repos</td><td>Mandatory</td></tr>
<tr><td>AWS CodeCommit</td><td>The field is not available if you selected this Git provider.</td></tr>
</table>

Visibility of the Git repository: Select Public or Private. The default is Private.

</td></tr>
<tr><td>Git provider API</td><td>What you need to select or enter in this field varies depending on the Git provider you selected.</td><td>The type and values of credentials to access the Git provider API.
<table>
<tr><th>Git provider</th><th>Description</th></tr>
<tr><td>GitHub</td><td>a. Select Credentials or Access token in Authentication type.
b. Enter username and password for Credentials or enter access token for Access token.
To obtain the access token or application password, go to the following location: Settings > Developer Settings > Personal Access Tokens</td></tr>
<tr><td>Gitea</td><td>a. Select Credentials or Access token in Authentication type.
b. Enter username and password for Credentials or enter access token for Access token.
To obtain the access token or application password, go to the following location: Settings > Applications > Manage Access Tokens</td></tr>
<tr><td>GitLab</td><td>Enter an access token.

To obtain the access token or application password, go to the following location: Preferences > Access Tokens > Personal Access Tokens</td></tr>
<tr><td>AzureRepos</td><td>Enter an access token.

To obtain the access token or application password, go to the following location: User settings > Personal Access Tokens</td></tr>
<tr><td>Bitbucket</td><td>Enter the username and password.

To obtain the access token or application password, go to the following location: Personal settings > App passwords</td></tr>
<tr><td>AWS CodeCommit</td><td>Enter the username and password.

a. Go to Identity and Access Management (IAM): https://<region>.console.aws.amazon.com/iamv2/home?region=<region>#/home.
b. Go to Access Management > Users, and select your user.
c. Verify that you have the AWSCodeCommitFullAccess permission. If you don't have it, add one.
d. Go to Security credentials > Create access key, and select Application running outside AWS.
e. When the access key is created, use the access key as username, and the secret access key as password.</td></tr>
</table>
</td></tr>
<tr><td>Git credentials</td><td>Mandatory</td><td>The type and values of credentials to access the Git repositories.

Use one of them:

Username and password
Access token

For AWS CodeCommit:
a. Go to Identity and Access Management (IAM): https://<region>.console.aws.amazon.com/iamv2/home?region=<region>#/home
b. Do one of the following steps:

Go to Access Management > Security credentials > SSH public keys for AWS CodeCommit, and upload the SSH public key.
Go to HTTPS Git credentials for AWS CodeCommit, and generate credentials.

</td></tr>
</table>

6. Click Save.

Setting up your build environment

To build decision services either from your local machine with a command line, or from a continuous integration and delivery (CI/CD) infrastructure, you need Java™ and Maven. You also need to download specific files if you want to use the execution Java API for decision services.

Before you begin

The Detailed system requirements page provides information about supported Java SDK.

For the execution Java API, check the Environment section in Executing decision services with executing Java API.

- **Using Decision Designer as a Maven repository**
  You need the Java™ libraries and Maven plug-ins to build decision service archives or execute decision services with the execution Java API. They cannot be deployed to the Maven central repository. You can configure Maven to connect to Decision Designer, and use Decision Designer as a Maven repository.
- **Deploying to a Maven repository**
  You need Maven plug-ins to build and deploy decision services. You need certain JAR files if you want to use the execution Java™ API. These dependencies are designed to be pulled from Maven.

Using Decision Designer as a Maven repository

You need the Java™ libraries and Maven plug-ins to build decision service archives or execute decision services with the execution Java API. They cannot be deployed to the Maven central repository. You can configure Maven to connect to Decision Designer, and use Decision Designer as a Maven repository.

About this task

To run the command in the command line, you must allow your local Maven to connect to Decision Designer to download the necessary dependencies.

Procedure

1. Configure Maven to connect to Decision Designer.
   In the local $HOME/.m2/settings.xml file, add a repository:

```
<repositories>
    <repository>
        <id>ads-maven</id>
        <name>ADS maven repository</name>
        <url>https://<zen_hostname>:<zen_port>/ads/static/maven-releases</url>
        <snapshots>
            <enabled>false</enabled>
        </snapshots>
    </repository>
</repositories>
```

   If you need to use the Maven plug-ins, add the following item as well:

```
<pluginRepositories>
    <pluginRepository>
        <id>ads-maven</id>
        <name>ADS maven repository</name>
        <url>https://<zen_hostname>:<zen_port>/ads/static/maven-releases</url>
        <snapshots>
            <enabled>false</enabled>
        </snapshots>
    </pluginRepository>
</pluginRepositories>
```

   Tip:
   <zen_hostname> is part of the URL to access Automation Decision Services.

   For example, if the URL to access Automation Decision Services is **`https://cpd-cp4a.<custom>.com/ads`**, then *<zen_hostname>* is **`cpd-cp4a.<custom>.com`**.

   The default value of *<zen_port>* is 443. You can use the default value if you can't identify another value in the URL.

   The Maven command or the underlying JVM must trust the IBM Cloud Pak Platform UI (Zen) certificate, if it is self-signed.

2. Configure the authentication to the server with a Zen API key:

```
<server>
 <id>ads-maven</id>
  <configuration>
    <httpHeaders>
      <property>
        <name>Authorization</name>
        <value>ZenApiKey %YOUR API KEY HERE%</value>
      </property>
    </httpHeaders>
  </configuration>
</server>
```

   For more information about the Zen API key, see Authorizing HTTP requests by using the Zen API key.

Results

When the local settings.xml file is configured, the **mvn clean install** command in a Maven project uses the Java libraries from Decision Designer.

Deploying to a Maven repository

You need Maven plug-ins to build and deploy decision services. You need certain JAR files if you want to use the execution Java™ API. These dependencies are designed to be pulled from Maven.

About this task

Automation Decision Services provides Maven plug-ins to generate decision libraries and decision service archives.

In a shell script, use the following URL to determine the value of *<DECISION_DESIGNER_SERVER_URL>*:

```
https://$(oc get route cpd --no-headers | awk {'print $2'})/ads
```

Table 1 shows the Maven plug-ins that are available from *<DECISION_DESIGNER_SERVER_URL>*. You can see files that are available from *<DECISION_DESIGNER_SERVER_URL>*/download/.

Table 1. Maven plug-ins

Purpose	Plug-in
To compile decision service archives	• build-maven-plugin-VERSION.jar • build-maven-plugin-VERSION.pom
To compile decision service archives and the external libraries of Automation Decision Services	• foundation-VERSION.jar
To be used by the external libraries feature of Automation Decision Services	• build-import-maven-plugin-VERSION.jar • build-import-maven-plugin-VERSION.pom • annotations-VERSION.jar
To execute decision service archives while performing unit tests	• engine-compact-runtime-VERSION.jar
To be used for machine learning You need them to use predictive models.	• ml-integration-runtime-VERSION.jar • ml-integration-runtime-VERSION.pom
To create JSON-based decision test suites	• engine-de-api-VERSION.jar
A Maven archetype that is used to create Maven projects of external libraries For more information, see Creating a POM file.	maven-archetype-external-library-VERSION.jar

You must download the following files for your applications that use the execution Java™ API for Automation Decision Services. The JAR files must be added in the class path in your applications.

For more information about the API, see Executing decision services with execution Java API.

Table 2. JAR files for execution Java API

Purpose	JAR
To use execution Java API for decision services	• execution-api-VERSION.jar • execution-api-VERSION.pom • engine-de-api-VERSION.jar

Procedure

1. To access the Maven plug-ins, get a Zen API key from a Zen user (with **zen** user role).
   For more information, see Authorizing HTTP requests by using the Zen API key.

2. Download the Maven plug-ins and Java™ libraries.
   Use the file *<DECISION_DESIGNER_SERVER_URL>*/index.json to determine which version of each plug-in to use, or to automate the download and install process.

   You can use a download method of your choice.

   For example with cURL, run the following command:

   ```
   curl -k -s -H "Authorization: ZenApiKey $(printf "<ZEN_USERNAME>:<ZEN_APIKEY>" | base64)"
   <DECISION_DESIGNER_SERVER_URL>/build-maven-plugin_<VERSION>.jar -o build-maven-plugin_<VERSION>.jar
   ```

   The following example shows the cURL command to download JAR files for execution Java API:

   ```
   curl -k -s -H "Authorization: ZenApiKey $(printf "<ZEN_USERNAME>:<ZEN_APIKEY>" | base64)"
   <DECISION_DESIGNER_SERVER_URL>/execution-api_<EXECUTION_API_VERSION>.jar -o execution-api_<EXECUTION_API_VERSION>.jar
   curl -k -s -H "Authorization: ZenApiKey $(printf "<ZEN_USERNAME>:<ZEN_APIKEY>" | base64)"
   <DECISION_DESIGNER_SERVER_URL>/engine-de-api_<ENGINE_API_VERSION>.jar -o engine-de-api_<ENGINE_API_VERSION>.jar
   ```

3. If you do not have a Maven repository server, or if you want to build decision archives on your local machine, install the Maven artifacts in your local Maven installation.
   To install the foundation-VERSION.jar dependency used by the Maven plug-ins, extract GAV parameters from index.json and install as follows.

   ```
   mvn install:install-file -DgroupId=<GROUP_ID> -DartifactId=<ARTIFACT_ID> -Dversion=<VERSION> -Dpackaging=jar -Dfile=
   <PATH_TO_JAR_FILE>
   ```

   For machine learning and Maven plug-ins, a POM file is provided. Install them as follows.

   ```
   mvn install:install-file -DpomFile=<ML_OR_PLUGIN_POM_FILE> -Dfile=<ML_OR_PLUGIN_JAR_FILE>
   ```

4. If you have a Maven repository server as part of your CI/CD infrastructure, deploy the Maven artifacts.
   If you use a binary repository manager, like Sonatype Nexus or JFrog Artifactory, you might want to deploy the Maven plug-ins to the remote Maven repository of the binary repository manager. Use the credentials for *<REPOSITORY_ID>* set in your ${user.home}/.m2/settings.xml file.

```
<settings>
  ...
  <servers>
    <server>
      <id><REPOSITORY_ID></id>
      <username><ARTIFACTS_SERVER_USERNAME></username>
      <password><ARTIFACTS_SERVER_PASSWORD></password>
    </server>
  ...
  </servers>
  ...
</settings>
```

For each artifact without a POM file, run the following command:

```
mvn deploy:deploy-file -DgroupId=<GROUP_ID> -DartifactId=<ARTIFACT_ID> -Dversion=<VERSION> -Dpackaging=jar -DrepositoryId=
<REPOSITORY_ID> -Durl=<ARTIFACTS_SERVER_URL> -Dfile=<PATH_TO_JAR_FILE>
```

For each artifact with a POM file, run the following command:

```
mvn deploy:deploy-file -DpomFile=<POM_FILE> -DrepositoryId=<REPOSITORY_ID> -Durl=<ARTIFACTS_SERVER_URL> -Dfile=<JAR_FILE>
```

Configuring for disaster recovery

It is important to have a disaster recovery plan (DRP) that describes how work can be resumed quickly and effectively after a disaster.

About this task

Make regular backups of each environment in your multiple-zone clusters. The shorter the time in between two backups the less data you can potentially lose.

- **Backing up Automation Decision Services**
  If you use Automation Decision Services, back up the secrets and your MongoDB collections.
- **Restoring Automation Decision Services**
  When needed, restore the secrets, the MongoDB collections, and the Decision Designer solutions that you backed up for Automation Decision Services.

Backing up Automation Decision Services

If you use Automation Decision Services, back up the secrets and your MongoDB collections.

About this task

By default, the database connection is encapsulated in a secret that is named **ibm-dba-ads-mongo-secret**. The secret **ibm-dba-ads-designer-secret** encapsulates the **encryptionKeys** key, which is used to cipher and decipher secrets in Decision Designer. The secret **ibm-dba-ads-runtime-secret** encapsulates the **encryptionKeys** key, which is used to cipher and decipher secrets in the decision runtime.

Procedure

1. Make a copy of the secrets. To retrieve the secrets, run the following commands:

   ```
   kubectl --namespace $ADS_RELEASE_NAMESPACE get secret ibm-dba-ads-designer-secret -o yaml
   ```

   ```
   kubectl --namespace $ADS_RELEASE_NAMESPACE get secret ibm-dba-ads-runtime-secret -o yaml
   ```

2. If you previously changed the name of secrets in the custom resource (CR), configure the names with the **ads_configuration.decision_designer.admin_secret_name** and **ads_configuration.decision_runtime.admin_secret_name** parameters.
3. Synchronize your decision services with a remote Git repository to minimize risk of data loss.
4. Decision Designer stores its data in multiple collections in three different databases that are defined by the **mongoUri**, **gitMongoUri**, and **runtimeMongoUri** properties of the MongoDB admin secret. The default name of this secret is **ibm-dba-ads-mongo-secret**.
   For more information, see **mongo.admin_secret_name** in Configuration parameters.

   The following Decision Designer collections can be restored from the database specified by the **mongoUri** parameter:
   - **BranchProtection**
   - **build-service-manager.ChangeLog**
   - **Credentials**
   - **credentials-service.ChangeLog**
   - **IndexContent**
   - **JobHistory**
   - **Library**
   - **MachineLearningProvider**
   - **MachineLearningProviderPerSolution**
   - **MavenConfig**
   - **parsing-service.ChangeLog**
   - **parsing-srv-buildFS.files**
   - **Permission**
   - **rest-api.ChangeLog**
   - **run-service.ChangeLog**
   - **run-srv-buildFS.files**

- `SecretRef`
- `Secrets`
- `ShareRequest`

The following Decision Designer collections can be restored from the database specified by the `gitMongoUri` parameter:

- `Branch`
- `fs.chunks`
- `fs.files`
- `git-fs.files`
- `GitProvider`
- `git-service.ChangeLog`
- `ImportReport`
- `Packs`
- `Repo`
- `Resource`
- `Revision`
- `StagingResource`
- `Tag`
- `UserStaging`

The database specified by the `mongoHistoryUri` parameter contains non critical data that cannot be restored.

The decision runtime stores the decision archives either in a Persistent Volume (default) or in an S3 object storage bucket (referenced by the parameter `ads_configuration.decision_runtime.archive_storage_type`). The metadata are stored in the MongoDB database referenced by the `runtimeMongoUri` parameter. The decision archives store and the MongoDB database must be backed up simultaneously when no decision service management requests take place (no decision archive deployment, for example). All collections of the MongoDB runtime database must be backed up and restored.

5. Back up the MongoDB data.
   a. Get the value of the secrets and the name of the MongoDB data.

```
DESIGNER_ENCRYPTION_KEYS=$(kubectl get secret --namespace $ADS_RELEASE_NAMESPACE ibm-dba-ads-designer-secret -o
jsonpath="{.data.encryptionKeys}" | base64 --decode)
RUNTIME_ENCRYPTION_KEYS=$(kubectl get secret --namespace $ADS_RELEASE_NAMESPACE ibm-dba-ads-runtime-secret -o
jsonpath="{.data.encryptionKeys}" | base64 --decode)
MONGO_URI=$(kubectl get secret --namespace $ADS_RELEASE_NAMESPACE ibm-dba-ads-mongo-secret -o jsonpath="
{.data.mongoUri}" | base64 --decode)
GIT_MONGO_URI=$(kubectl get secret --namespace $ADS_RELEASE_NAMESPACE ibm-dba-ads-mongo-secret -o jsonpath="
{.data.gitMongoUri}" | base64 --decode)
RUNTIME_MONGO_URI=$(kubectl get secret --namespace $ADS_RELEASE_NAMESPACE ibm-dba-ads-mongo-secret -o jsonpath="
{.data.runtimeMongoUri}" | base64 --decode)
echo "DESIGNER_ENCRYPTION_KEYS: $DESIGNER_ENCRYPTION_KEYS"
echo "RUNTIME_ENCRYPTION_KEYS: $RUNTIME_ENCRYPTION_KEYS"
```

   b. If you use the embedded MongoDB instance instead of an external instance (which is recommended in production), you must forward the MongoDB data port and adapt the `MONGO_URI`, `GIT_MONGO_URI`, and `RUNTIME_MONGO_URI` parts of the previous step.

```
# Get ADS mongo service name
export ADS_MONGO_SERVICE=$(kubectl get services --namespace $ADS_RELEASE_NAMESPACE -l 'app.kubernetes.io/name=ads-
mongo' --output=jsonpath='{.items[0].metadata.name}')
MONGO_URI="${MONGO_URI/@$ADS_MONGO_SERVICE*:/@localhost:}"
GIT_MONGO_URI="${GIT_MONGO_URI/@$ADS_MONGO_SERVICE*:/@localhost:}"
RUNTIME_MONGO_URI="${RUNTIME_MONGO_URI/@$ADS_MONGO_SERVICE*:/@localhost:}"

kubectl port-forward --namespace $ADS_RELEASE_NAMESPACE svc/$ADS_MONGO_SERVICE 27017:27017 &
```

   c. Create directories for the files that you want to back up.

```
mkdir adsbackup
```

   d. Back up the collections to these directories by using the `mongodump` tool.

```
for col_name in Credentials JobHistory Library MachineLearningProvider MachineLearningProviderPerSolution \
                Permission \ credentials-service.ChangeLog credentials-service.DbVersion \ run-service.ChangeLog run-
service.DbVersion \
                rest-api.ChangeLog rest-api.DbVersion ; do
    mongodump --uri "$MONGO_URI" --tlsInsecure --out=adsbackup/ads-db --collection="$col_name"
done

for col_name in Branch ImportReport Repo Resource Revision StagingResource Tag UserStaging \
                fs.chunks fs.files git-service.ChangeLog git-service.DbVersion  ; do
    mongodump --uri "$GIT_MONGO_URI" --tlsInsecure --out=adsbackup/ads-git-db --collection="$col_name"
done

mongodump --uri "$RUNTIME_MONGO_URI" --tlsInsecure --out=adsbackup/ads-runtime
```

Restoring Automation Decision Services

When needed, restore the secrets, the MongoDB collections, and the Decision Designer solutions that you backed up for Automation Decision Services.

Procedure

1. Run the following command to make sure that you have the secrets `ibm-dba-ads-mongo-secret`, `ibm-dba-ads-designer-secret` (if you installed Decision Designer), and `ibm-dba-ads-runtime-secret` (if you installed the decision runtime server).

```
kubectl get --namespace $ADS_RELEASE_NAMESPACE secrets
```

If you do not have the secrets, you must restore them and make sure that the **encryptionKeys** value corresponds to the time of the MongoDB backup to be restored.

Note: You might need a MongoDB administrator password to completely re-create embedded MongoDB databases. To obtain the generated password, you can use the following commands.

```
export MONGODB_USER=$(kubectl get secret --namespace $ADS_RELEASE_NAMESPACE ibm-dba-ads-mongo-secret -o jsonpath="
{.data.mongoUser}" | base64 --decode)
export MONGODB_PASSWORD=$(kubectl get secret --namespace $ADS_RELEASE_NAMESPACE ibm-dba-ads-mongo-secret -o jsonpath="
{.data.mongoPassword}" | base64 --decode)
```

2. Get the value of the secrets and the name of the new MongoDB service.

```
MONGO_URI=$(kubectl get secret --namespace $ADS_RELEASE_NAMESPACE ibm-dba-ads-mongo-secret -o jsonpath="{.data.mongoUri}"
| base64 --decode)
GIT_MONGO_URI=$(kubectl get secret --namespace $ADS_RELEASE_NAMESPACE ibm-dba-ads-mongo-secret -o jsonpath="
{.data.gitMongoUri}" | base64 --decode)
RUNTIME_MONGO_URI=$(kubectl get secret --namespace $ADS_RELEASE_NAMESPACE ibm-dba-ads-mongo-secret -o jsonpath="
{.data.runtimeMongoUri}" | base64 --decode)
```

3. If you want to restore into the embedded MongoDB instance instead of into an external instance (which is recommended in production), you must forward the MongoDB service port and adapt the **MONGO_URI**, **GIT_MONGO_URI**, and **RUNTIME_MONGO_URI** parts of the previous step.

```
# Get ADS mongo service name
export ADS_MONGO_SERVICE=$(kubectl get services --namespace $ADS_RELEASE_NAMESPACE -l 'app.kubernetes.io/name=ads-mongo' -
-output=jsonpath='{.items[0].metadata.name}')
MONGO_URI="${MONGO_URI/@$ADS_MONGO_SERVICE*:/@localhost:}"
GIT_MONGO_URI="${GIT_MONGO_URI/@$ADS_MONGO_SERVICE*:/@localhost:}"
RUNTIME_MONGO_URI="${RUNTIME_MONGO_URI/@$ADS_MONGO_SERVICE*:/@localhost:}"

kubectl port-forward --namespace $ADS_RELEASE_NAMESPACE svc/$ADS_MONGO_SERVICE 27017:27017 &
```

4. To restore the Decision Designer collections that you backed up, you can use the **mongorestore** command.
In the following command sample, the folders correspond to a backup from a deployment that uses the embedded MongoDB instance with its default database names (**ads-db**, **ads-git-db**, **ads-runtime-archive-metadata**). Adapt the paths to the names of the databases of your original Automation Decision Services instance.
The **${MONGO_URI}** and **${GIT_MONGO_URI}** variables refer to the variables defined in substep c above and correspond to the URI for the new Automation Decision Services instance.

```
mongorestore --uri "${MONGO_URI}" --tlsInsecure  adsbackup/ads-db/ads-db
mongorestore --uri "${GIT_MONGO_URI}" --tlsInsecure  adsbackup/ads-git-db/ads-git-db
```

5. To restore the collections of the decision runtime server, you can use the following **mongorestore** command.

```
mongorestore --uri "${RUNTIME_MONGO_URI}" --tlsInsecure adsbackup/ads-runtime/ads-runtime-archive-metadata
```

Viewing metering information

You can view the metering information based on the execution count. You can query the information anytime.

Procedure

You can query the metering data by navigating to the following URL:

```
https://licensing.subdomain.my-company.com/products?token=<token>
```

The **token** is in a secret in the Automation Decision Services namespace. It can be retrieved with the following command:

```
kubectl get secret -n <ads_namespace> ibm-licensing-bindinfo-ibm-licensing-token -o jsonpath='{.data.token}' | base64 --decode
```

The execution count is reported in an interval that is specified in the decision_runtime.ils.flush_interval parameter. The default is 10 minutes. For more information about the parameter, see Decision runtime parameters.

Results

You get a result similar to the following sample when you query the metering data:

```
[{"name":"IBM Automation Decision Services","id":"34a9f36c66b043bfb29784e5d5c014cf","metricPeakDate":"2023-06-
02","metricName":"MONTHLY_API_CALL","metricQuantity":1}]
```

Learning resources

Get hands-on experience with Automation Decision Services. These resources give you a head start in automating your decisions through samples and tutorials that cover the main development and deployment activities.

- **Getting started**
  You create, share, and deploy a decision service in Automation Decision Services. You define its logic through a decision model, and run it by using test data. You also share it for collaborative development, and use it in a runtime environment through a Swagger UI.
- **Samples and tutorials in GitHub**
  Find samples and tutorials for Automation Decision Services in GitHub. They include a set of decision models for the getting started tutorial, and code for integrating machine learning.

Getting started

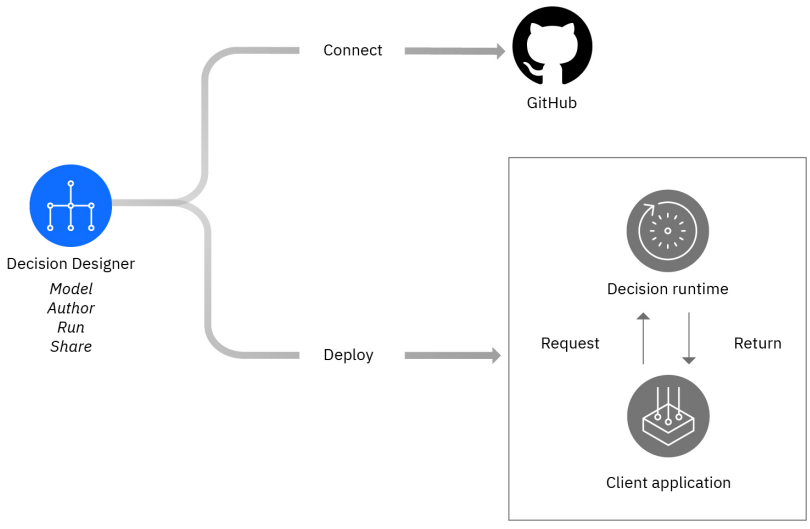
You create, share, and deploy a decision service in Automation Decision Services. You define its logic through a decision model, and run it by using test data. You also share it for collaborative development, and use it in a runtime environment through a Swagger UI.

- **Before you start**
  Learn some of the basic concepts and prerequisites before you do the tutorial.
- **Task 1: Making a decision service**
  You create a decision service that contains a decision model. When you run the model, it produces a greeting.
- **Task 2: Connecting to a Git repository and sharing a decision service**
  You create an operation to expose your decision service, and a Git repository to connect to your project. Then, you share your decision service, and look at selecting collaborators for your project.
- **Task 3: Adding nodes**
  You add two nodes for the weather to the decision model. When you run the model, it produces a greeting and weather advice.
- **Task 4: Creating a data model**
  You add a data model to enumerate the temperatures cold, warm, and hot. When you run the model, it displays a greeting and advice for a cold day.
- **Task 5: Using the interaction policy**
  You add a storm alert by updating the data and decision models. When you run the decision model to test the storm alert, it produces a greeting and a warning to stay home.
- **Task 6: Deploying and running a decision**
  You share your latest changes. Then, you deploy your decision service, and run it in the runtime environment.

Next >

Before you start

Learn some of the basic concepts and prerequisites before you do the tutorial.
Automation Decision Services includes Decision Designer, which is a development environment for creating decision services. In this tutorial, you use the editor to define and run a decision service. Then, you connect your decision automation to a repository in GitHub, and share your changes. You also look at assigning collaborators to your decision automation. After you further define your decision service, you share more changes, and then deploy the service to a runtime environment to run it in a Swagger UI:



You start by creating a *decision automation* to hold a decision service. A *decision service* uses decision artifacts to define a business decision:

- *Decision model*: Contains a diagram that has nodes for decisions and data input. The decision nodes hold the logic that processes the information from the input data nodes. The diagram directs the flow of data among the nodes (see Creating a decision diagram). Predictive and function nodes use other models defined in the decision service (see Calling other models).
- *Task model*: Chains together tasks, and specifies the conditions for running them.
- *Predictive model*: Applies data from a machine learning model.
- *Data model*: Defines the custom data types that are used in the decision service.
- *External libraries*: Contain data types and functions that are used in models in the decision service.
- *Decision operations*: Define entry points to models.

This tutorial covers the creation and use of a decision service that is based on a decision model. It does not cover external libraries, or predictive or task models. You can find samples that use the models and external libraries in the GitHub repository Automation Decision Services samples.

Learning objectives

- Create a decision automation and a decision service.
- Create a decision model in the service.
- Define the decision logic that is used in the model.
- Run the decision model by using test data.

- Create a Git repository and connect your decision automation to it.
- Share your decision service.
- Select collaborators for your decision automation.
- Create a data model for the decision service.
- Declare a dependency between the decision model and the data model.
- Create an operation.
- Create a version of your decision automation.
- Deploy your decision service.
- Run your service in the decision runtime.

The GitHub repository Automation Decision Services samples includes a decision service that is based on this tutorial. The decision service has a separate decision model for each task in the tutorial. Use the service to tour the product, or run it alongside the tutorial to work through the tasks. To import the decision service, see Creating decision services.

Time required

About 50 minutes

Audience

Anyone who wants to learn how to use Automation Decision Services.

Prerequisites

You need the following services:

- Decision Designer: A web-based environment for developing decision services.
- GitHub account: You must have a GitHub account to complete this tutorial. If you do not have one, check whether your organization has a GitHub account or visit GitHub.com.

Next >
< Previous | Next >

Task 1: Making a decision service

You create a decision service that contains a decision model. When you run the model, it produces a greeting.

About this task

In this task, you...

- Create a decision automation.
- Create a decision service inside the automation.
- Create a decision model inside the service.
- Define the data and logic of the model.
- Create a test data set.
- Run the model on test data.

When defined, the model takes a name as input, and produces a greeting that includes the name. If no name is entered, the model produces a default value.

Step 1: Creating a decision automation

About this task

In this step, you create a decision automation. Users can work collaboratively in decision automations through a shared Decision Designer repository.

Procedure

1. On the Automation Decision Services homepage, click Create.
2. Enter a name for your decision automation in the Name field.
   The automation does not accept the name of an existing one. You must use a unique name, for example, a name that uses your initials: <YourInitials> Getting Started. The project name in this tutorial is My Getting Started.
3. Enter the following description:

   ```
   My decision automation for the getting started tutorial.
   ```

4. Click Create. Your new decision automation opens in Decision Designer.

Step 2: Creating a decision service

About this task

In this step, you create a decision service to model and define a decision (see Creating decision services).

Procedure

1. Click New decision in your automation to create a decision service.
   From the New decision wizard, you can:
   - Create decision services from scratch.
   - Import existing decision services.
   - Import samples and tutorials:
     - Discovery tutorials help you get started in Automation Decision Services.
     - Industry samples allow you to discover decision services tailored for specific business applications.
     - Localized samples show you an example of a simple decision service in the different languages supported by Automation Decision Services.
2. Make sure Create decision service is selected and enter My Service in the Name field.
3. Expand Language, and select English (United States) in the menu.
   Tip: This tutorial is done in English, but you can make decision services in different languages. To get the results of this tutorial in another language, see Automation Decision Services samples.
4. Enter the following description:

   ```
   My getting started decision service.
   ```

   Note: The group ID is optional. If none is entered, a default one is used. The group ID is used to uniquely identify a decision service.
5. Click Create.
   Your decision service opens. It shows the following tabs:
   - Models: Create and manage decision, task, and predictive models that define decisions.
   - Data: Create and manage data models that define custom data types.
   - Decision operations: Create and manage decision operations that define entry points for running the decision, predictive, and task models.

Step 3: Creating a decision model

About this task

You define the flow of a decision through a diagram in a model. The primary parts of the diagram are nodes:

- Decision nodes: Contain the logic that processes the input data.
- Input nodes: Provide the data that is needed to make the decision.
- Function nodes: Provide values that are computed in other decision models. (Function nodes are not used in this tutorial.)
- Prediction nodes: Provide values that are computed in predictive models. (Prediction nodes are not used in this tutorial.)

Data enters through input data nodes, and is processed by rules in decision nodes. The rules define the logic of the decision. They are expressed in business rules and decision tables. The rules determine the output of the model (see Creating a decision diagram).

In this step, you create a model that has a diagram with a decision node and an input node.

Procedure

1. In the Models tab, click Create.
   The Decision model type is selected by default.
2. Enter My Model in the Name field. You can add a description, but it is optional.
3. Click Create. A basic model opens. It contains a diagram that has a decision node and an input node.
   In addition to the Modeling tab, Decision Designer shows four other tabs:
   - Error report: Checks the model for errors.
   - Run: Runs the model on test data.
   - View history: Shows the version of the model. You create the first version when you share the decision service.
   - Dependencies: Shows the decision artifacts and external libraries the model depends on. You can declare new dependencies in this tab.

   Your changes are automatically saved, and the information in these tabs is recomputed in each save.

   You can use the navigation breadcrumbs to move back through the interconnected interfaces.

   To the right of the breadcrumbs is a toolbar with a group of icons:

   From left to right, the icons have the following functions:

   - Status shows that your changes are saved automatically. The status icon is updated for each save.
   - Edit opens a window for editing the details of the model.
   - Undo reverts the last change.
   - Redo reinstates the last reverted change.
   - Navigation history helps you navigate easily between artifacts.
   - Connected to shows the URI of the Git repository the decision automation is connected to.
   - Load changes shows how many incoming changes can be loaded.
   - Share changes shows a blue dot when you have changes to share with your collaborators.
   - Deploy is a feature for deploying your decision service.

   The appearance of the toolbar varies depending on the page you are viewing.

Step 4: Defining the nodes

About this task

You start defining the nodes in a model by giving them descriptive names and output types. If you create a decision node that produces a message, for example, you might name it Message and set its output to string. For a node that outputs a price, you might name it Price and set its output to integer (see Modeling data in Automation Decision Services).

In this step, you define the names and output types of the nodes in your model. You define the logic of the model in the next step.

Procedure

1. Select the Input node.
2. In the right panel, change the Node name field to Name.
3. Leave string selected in the Output type field.
4. Select the Decision node.
5. Change the Node name field to Daily advice. Leave string as the output type.

Step 5: Defining the decision logic

About this task

You use rules to define the logic that is used by decision nodes. A rule applies conditions to data, and when the data meets the conditions, the rule associates an action with the conditions. A rule might check data for a specific value, for example, and if the data contains the value, the rule outputs an action that is related to the value.

You express rules in business rules and decision tables:

- Business rules contain a rule statement that has a condition and an action.
- Decision tables group together business rules that use the same rule statement but with different variables.
- Default rules apply an action when no other condition can be met.

When you write a rule, you use a language that is natural in form (see Rule language). The rules include variables that are derived from the names of the nodes in the model (see Rule structure).

In this step, you add a business rule and default value to the Daily advice node to process the data from the Name node.

Procedure

1. With the Daily advice decision node selected, open the Logic tab, and then click the Add button and select Business rule.
2. Enter Advice rule in the Name field, select Name in the list of criteria, and click Create.
   The editor displays an error icon until you define the rule.
3. Define the rule for processing input data.
   In the rule editor, you see the default rule:

   ```
   if
   Name is <a string>
   then
   set decision to <a string> ;
   ```

   You change the rule to output a greeting that includes a name when one is provided:
   a. Click the first **<a string>**.
   b. Browse the completion menu, and click **defined**.
   c. Click outside the menu to close it.
   d. Click the second **<a string>**.
   e. Use string interpolation to complete the output by adding **`Hello { Name }!`** to compose the greeting. Add a space just after **Hello** to insert a space before the name in the output message.
   Note: String interpolation assists you in authoring rules for decision services (see String interpolation).
   You now have the following greeting rule:

   ```
   if
       Name is defined
   then
       set decision to `Hello { Name }!` ;
   ```

   The rule states that if a name is entered as input, the output of the model shows a greeting that includes the name.

   Tip: Alternatively, you can copy and paste the greeting rule into the rule editor.
4. In the right column, click the Add button and select Default rule to add a rule that displays a message when no name is entered.
5. Replace the default rule with the following rule statement:

   ```
   set decision to "Sorry, we don't have enough information to provide a response." ;
   ```

   Now, if no name in entered, the output of the model is the message in the output-default-setting rule.

   You must enclose a string of characters in quotation marks to display the string as a message.

6. Click Back to the diagram in the upper right corner of the editor.
   Now, data flows from the Name node up to the Daily advice node, which adds the data to an output message. When no data is entered, the default statement becomes the output message.

Step 6: Running the model

About this task

You run the model against representative test data, and use the results to further develop the model (see Running models).

In this step, you run your model twice. You enter test data, and run the model to produce a greeting. Then, you run the model without test data to see whether it returns the default message.

Procedure

1. Open the Run tab. You define your test data set in the side column of the tab window.
2. Click Add test data set.

You can provide the data in a friendly form or in its underlying JSON format. You can click the  switch button to switch between the two edition modes. In this example, you enter your data in a form.

3. Click name in the form, and enter `Jamie` in the input field.
   The form uses the JSON representation of the name of an input node. In this instance, Name becomes name. For more information, see Modeling data in Automation Decision Services.
4. Click Run. You get the following result:

   ```
   Hello Jamie!
   ```

5. Click the dots next to name to open the overflow menu, and click Delete in the menu to remove the name from the test data.
6. Click Run again. You get the following result:

   ```
   Sorry, we don't have enough information to provide a response.
   ```

7. When you finish running the model, click the trash bin icon at the top of the data set column to delete the test data set. You create a new test data set in the next task.

What to do next

In the next task, you connect to a Git repository and share your decision service.

< Previous | Next >
< Previous | Next >

Task 2: Connecting to a Git repository and sharing a decision service

You create an operation to expose your decision service, and a Git repository to connect to your project. Then, you share your decision service, and look at selecting collaborators for your project.

About this task

In this task, you...

- Create an operation.
- Create a Git repository.
- Connect your decision automation to the Git repository.
- Share your decision service.
- Look at how to select collaborators.

Step 1: Creating an operation

About this task

To expose a decision service, you must define an operation that is used to call the service. The operation includes a name and a reference to a decision model (see Creating decision operations).

In this step, you create the operation.

Procedure

1. Click My service in the breadcrumbs at the top of the page to open your decision service.
2. Click the Decision operations tab, and then Create.
3. Under Source model, select My Model. Keep My-Model in the Operation name field. The input and output parameters are not editable.
4. Click Create.
   The decision operation My-Model is created. You see it used in task 6 to run your decision service.

   The top bar shows that you have a new change to share. Now, you must connect your project to a Git repository to share your changes.

Step 2: Creating a Git repository

About this task

In this step, you create a GitHub repository that you connect to in the next step. As explained in the prerequisites of this tutorial, you must have a GitHub account to do this step.

Note: Decision Designer might be configured to connect to a Git repository automatically (see Connecting to a remote repository automatically). In this case, you can see the URI of the repository Decision Designer is connected to when you hover over the connection icon in the toolbar on the decision automation page. You can skip steps 2 and 3 if Decision Designer is connected to a repository, or you can do the steps to connect Decision Designer to a different repository.

Procedure

1. Open GitHub in your browser, and use your GitHub credentials to log in.
2. Click the + button at the upper right part of the page and select New repository.
3. Give the repository a unique name, and add the following description:

   ```
   Git repository for the Automation Decision Services getting started decision service.
   ```

4. Click the Create repository button to finish creating the repository.

Note: After you create the repository, make sure that it is empty. The repository must not contain a readme, .gitignore, or license file.
5. Click the Copy button to copy the HTTPS URI. It has the following format:
   https://github.com/<yourAccountName>/<yourRepoName>.git
6. Open your profile for your GitHub account in the upper right corner of the page.
7. Click Settings > Developer settings > Personal access tokens > Tokens (classic).
8. Click Generate new token. Then, select Generate new token (classic).
9. Enter a note to remind you of the reason for the token, and select repo to give full control of the repository to Automation Decision Services.
10. Click Generate token at the bottom of the page. Copy the generated access token before closing this page.

Step 3: Connecting your project to the Git repository

About this task

In this step, you connect to the Git repository from Decision Designer. For different ways to define this connection, see Connecting to a remote repository automatically.

Note: If your Decision Designer automatically connects to a repository, you can do step 3 to connect Decision Designer to the repository created in step 2.

Procedure

1. Click My Getting Started in the breadcrumbs to open your decision automation.
2. Check the status of your Git repository connection in the upper right corner of Decision Designer. The connection icon shows if the decision automation is connected to a remote Git repository.
3. Click the connection icon to define or update a connection.
   Make sure Create or update credentials for the project is selected.
4. Enter the Git URI, and keep the Username and password type selected.
5. Enter your GitHub username and the personal access token that you generated in step 2.
6. Click Connect. When your decision automation connects to your GitHub repository, Decision Designer displays the following message:

   ```
   The remote Git repository is connected successfully.
   ```

   Your decision service is added to the repository as part of the connection.

7. Refresh your repository page in GitHub. The initial commit is empty. You must share your changes to see the files in your decision service.
8. Click My Getting Started in the breadcrumbs to return to your decision service. The connection icon shows that your decision automation is now connected to the Git repository.

Step 4: Sharing your decision service

About this task

In this step, you share your decision service within your instance of Decision Designer (see Sharing changes).

Procedure

1. Click the Share changes tab to open it. You see two artifacts: decision service and decision model. These changes are from the decision service that you assembled in the previous task.
2. Keep My Service selected, and click Share.
3. Enter the following comment, and then click Share:

   ```
   First getting started version.
   ```

   A message says that your changes have been successfully shared. You can now select collaborators for your decision automation.

Step 5: Choosing your collaborators

In this step, you see how to add collaborators to your decision automation. Because you are working alone, you do not complete this procedure.

Procedure

1. Click Decision Automations in the breadcrumbs.
2. Open the overflow menu inside the box of the MyGetting Started decision automation and select Edit details.
3. Open the User management tab. You see that you have administrative permissions on your decision automation.
4. Click the Add permission button to add collaborators, and select the appropriate permissions:
   - Admin for full access
   - Edit for export, publish, preview, and edit
   - Read for export and preview
5. Because you are doing this tutorial alone, click Cancel to abandon this operation.
6. Click MyGetting Started to return to your decision automation.

What to do next

In the next task, you add nodes to your decision model to provide weather advice.

< Previous | Next >
< Previous | Next >

Task 3: Adding nodes

You add two nodes for the weather to the decision model. When you run the model, it produces a greeting and weather advice.

About this task

In this task, you...

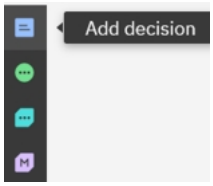
- Add an input node for the rain forecast, and a decision node for weather advice to the model.
- Add a decision table, and define the logic for the weather advice.
- Run the model to check your changes.

Step 1: Adding nodes to the model

About this task

In this step, you add a decision node and an input node for weather advice to the model.

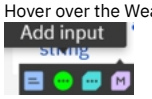
Procedure

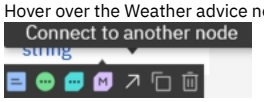
1. Click My service to open your decision service. Then, click My Model to open your decision model.
2. Click the Add decision button to create a decision node:


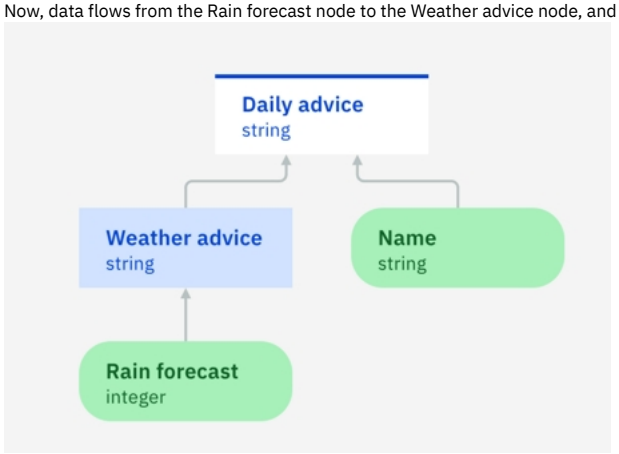
3. In the right panel, change the Node name field of the new node to Weather advice. Leave string as the output type.
4. Hover over the error icon to see its message:

   ```
   The node 'Weather advice' must have one or more links to or from other nodes.
   ```

   The message is also shown in the Error report tab. Each node must be connected to at least one other node.

5. Hover over the Weather advice node, and click the Add input button:


   Tip: Depending on your system, you might have to select the node before you can hover over it to see the toolbar.
6. In the right panel, change the Node name field to Rain forecast.
7. Expand the Output type field, and select integer from the menu.
8. Hover over the Weather advice node, and click the Connect to another node button:


9. Drag the line to the Daily advice node, and click the node to complete the connection.
   Now, data flows from the Rain forecast node to the Weather advice node, and decisions flow from the Weather advice node to the Daily advice node:


Step 2: Adding a decision table

About this task

A decision table groups business rules that share the same logic but have different values. Each row in a decision table represents a complete business rule. The decision table has separate columns for condition values and action values. When data enters a decision table, it is matched to values in condition columns and the associated action value is used in the output of the decision table (see Working with decision tables).

In this step, you add a decision table to the Weather advice node.

Procedure

1. Select the Weather advice node.
2. Open the Logic tab, click the Add button, and select Decision table.
3. Enter `weather table` in the Name field of the new table, select Rain forecast for the condition columns, and then click Create.

4. Enter the following values in the decision table. To add a value to a cell, double-click the cell.
   Tip: You can resize the decision table by dragging its borders.

Rain forecast		Weather advice
min	max	
0	20	Nice day. Enjoy the weather!
20	80	Risk of rain. Might need an umbrella.
80	100	Rainy day. Take an umbrella.

5. Right-click `100` in the decision table, and select Change operator. By default, the operator is set to [..[. Select [..] to change the operator. Now, the value is included in the conditions.
6. Click Back to the diagram to close the editor. Now, the model outputs a greeting, and advice from the weather decision table.
7. Select the Daily advice node.
8. In the Logic tab, open the Advice rule and make the following changes.
   Tip: As with `Hello`, add a space after `!` to leave a gap before the weather advice in the output.

```
if
    Name is defined
then
    set decision to `Hello { 'Name' }! { 'Weather advice' }` ;
```

   Now, you run the model to see the results.

Step 3: Running the model

About this task

In this step, you run your changes as explained in Task 1: Making a decision service.

Procedure

1. Open the Run tab.
2. Add the following test data:
   Again, the form uses the JSON representation of the name of an input node. Here, Name and Rain forecast become name and rainForecast. For more information, see Modeling data in Automation Decision Services.
   - name: Robin
   - rainForecast: 87
     Tip: For rainForecast, you can type the number or select it with the arrow buttons.
3. Click Run. You get the following message:

```
Hello Robin! Rainy day. Take an umbrella.
```

4. Delete the test data for the next task.

What to do next

In the next task, you add a data model to your decision service to enumerate temperatures.
< Previous | Next >
< Previous | Next >

Task 4: Creating a data model

You add a data model to enumerate the temperatures cold, warm, and hot. When you run the model, it displays a greeting and advice for a cold day.

About this task

In this task, you...

- Create a data model for weather.
- Enumerate a data type for temperatures in the data model.
- Add a composite data type for weather to the data model.
- Declare a dependency.
- Update the logic in the decision model.
- Run the model to check your changes.

You create a data model to define custom data types for your service. Basic data types, including *string* and *integer*, are predefined by default in Automation Decision Services. However, organizations often need custom data types that match their operations. For example, a car rental company might need special types such as *car*, *fleet* and *rental price*. These types are defined in a data model, and can be used in multiple artifacts in a decision service (see Creating a data model).

Step 1: Creating a data model and enumerating a data type

About this task

In this step, you create a data model and enumerate the values for a custom data type (see Working with enumeration types).

Tip: You can complete the data model by using external libraries. This operation is not covered in this tutorial. However, you can find it in the External Library tutorial in GitHub repository Automation Decision Services samples.

Procedure

1. Click My Service in the navigation breadcrumbs to return to the main page of your decision service.
2. Open the Data tab.
3. Click Create to create a data model.
4. Enter Data in the Name field, and click Create.
   The data model editor opens.
5. In the data model editor, click the + button next to Data types, and click Enumeration type in the menu.
6. Change the name of the new enumeration type from new enum to temperature.
   Take a moment explore the editor. The first part displays the language, the documentation, and the different verbalizations of the vocabulary elements. You can collapse this part by clicking the arrow to the right of the type name.
7. In the Values section, click the Add button twice to add two more values.
8. Change the names of the values to `cold`, `warm`, and `hot`.
   Your changes are automatically saved. The enumeration type temperature is now defined by these values.

Step 2: Adding a composite data type

About this task

In this step, you add a composite data type that uses the enumeration type temperature. You define a composite data type by using a set of attributes (see Working with composite types).

Procedure

1. In the data model editor, click the + button next to Data types, and click Composite type in the menu.
2. Change the name of the new composite type from new type to weather.
3. In the Attributes section, click the Add button twice to create two attributes.
4. Enter the following names and types for the attributes:

Name	Type
`temperature`	temperature
`rain forecast`	integer

   The weather data type is now defined by the temperature and the rain forecast.

Step 3: Declaring a dependency

About this task

In this step, you declare a dependency between the decision model and the data model to be able to use the types you defined in step 1 and 2 in the decision model.

Procedure

1. Click the Navigation history button in the top right toolbar, and click My Model to open it.
2. Open the Dependencies tab.
3. Click the Add button to declare a new dependency.
4. Select Data in the table and click Add.
   The custom data types are now ready to be used in the decision model.

Step 4: Changing the logic

About this task

In this step, you update the decision model to use the data types that you defined in your data model.

Procedure

1. Return to the Modeling tab to work on your decision model.
2. Change the name of the Rain forecast node to Weather, and change its output type to weather.
   Because these changes are automatically reflected in the dependent node, the Weather advice node shows an error. You resolve the error by updating the weather table to use the new data type.
3. In the Weather advice node, open Logic > weather table.
4. Double-click the header of the Rain forecast column, and change the name to Rain forecast %.
5. Right-click the header again, and click Define column in the menu. Enter the following condition in the rule editor, and then click OK:

   ```
   the rain forecast of 'Weather' is at least <min> and less than <max>
   ```

   The table no longer displays an error.

6. Right-click the header again, and click Insert column > Condition after in the menu.
7. Change the name of the new column to Temperature.
8. Right-click the header of the new column, and click Define column in the menu. Enter the following condition, and then click OK:

   ```
   the temperature of Weather is <a temperature>
   ```

9. Double-click the first cell in the Temperature column, and select cold from the menu.
10. Double-click the first cell in the Weather advice column, and enter a new message:

    ```
    Cold day. Take a coat.
    ```

11. Hover over the warning icon. It displays the following message:

    ```
    Lines 1 to 1 have gaps - Missing values: hot, warm
    ```

You defined three values for the enumeration type temperature. One value is declared in the table, but the editor warns you that two values are undeclared: **hot** and **warm**. Because the model still runs, the editor shows a warning and not an error.

You decide to ignore the warning for now, and disable the gap checking. You can return to the table to make additional changes after deploying the decision service in task 6.

12. Right-click the Temperature column header, and deselect Check gap in the menu to hide the warning. The gaps do not prevent the operation of the model.
13. Click Back to the diagram to close the editor.
    The table now provides advice for a cold day.

Step 5: Running the model

About this task

In this step, you run the model to test your changes.

Procedure

1. Open the Run tab.
2. Add the following test data:
   - name: Jamie
   - rainForecast: 0
   - temperature: cold
3. Click Run. You get the following message:

   ```
   "Hello Jamie! Cold day. Take a coat."
   ```

4. Change the test data:
   - name: Robin
   - rainForecast: 90
   - temperature: warm
5. Click Run. You get the following message:

   ```
   "Hello Robin! Rainy day. Take an umbrella."
   ```

6. Delete the test data for the next task.

What to do next

In the next task, you create a rain alert by adding an attribute to the composite type and defining the alert in a rule. You also change the interaction policy.
< Previous | Next >
< Previous | Next >

Task 5: Using the interaction policy

You add a storm alert by updating the data and decision models. When you run the decision model to test the storm alert, it produces a greeting and a warning to stay home.

About this task

In this task, you...

- Add a boolean attribute to the composite type in the data model.
- Create a rule to use the new attribute in the decision model.
- Set the order of the rules to apply the new rule first.
- Run the decision model to check your changes.

Step 1: Adding an attribute

About this task

In this step, you add an attribute to the composite type in the data model.

Procedure

1. Click the Navigation history button in the top right bar, and then click Data to return to the data model.
2. In the left panel, select weather.
3. In the Attributes section, click the Add button to create an attribute.
4. Change the name of the attribute to storm alert, and set its type to boolean.

Step 2: Modifying the decision logic

About this task

In this step, you add a business rule to use the storm alert attribute in the Weather advice node, and then set the sequence for running the rules in the node.

Rules are run in a specific order. By default, they are sequentially run in the order in which they are listed in a decision node. However, you can supersede the default behavior by setting the order for running the rules (see Choosing an interaction policy).

Procedure

1. Use the Navigation history button to return to My Model and its Modeling tab.
2. Select the Weather advice node, and open the Logic tab.
3. Click the Add button and select Business rule.
4. Enter storm rule in the Name field, and select **`Weather is storm alert`** in the list of criteria.
5. Click Create. The rule opens in the rule editor.
   The Error report tab shows an error. You fix the error when you define the rule.
6. Enter the following rule statement:

```
if
    Weather is storm alert
then
    set decision to "Stay home! There is a storm alert." ;
```

7. Click Back to the diagram.
8. To set the order for applying the rules, expand Rules are applied in sequence in the right panel, and select First rule applies in the menu.
9. Click the dots next to the storm rule in the Logic tab, and select Move to top in the menu to apply the rule first.
   Now, the storm rule runs first. When the storm alert notice is selected in the list of test data, a storm warning is added to the output message.

Step 3: Running the model

About this task

In this step, you run the model to check your changes.

Procedure

1. Open the Run tab.
2. Add the following test data:
   - name: Jamie
   - rainForecast: 100
   - stormAlert: True (Select the box.)
   - temperature: cold
3. Click the Rename test data set button at the top of the tab window.
4. Enter `Jamie-storm` as the new name of the data set, and press Enter.
5. Click Run. You get the following message:

```
"Hello Jamie! Stay home! There is a storm alert."
```

6. Add another set of test data:
   - name: Robin
   - rainForecast: 90
   - stormAlert: False (Do not select the box.)
   - temperature: cold
7. Rename the data set `Robin-rain`.
8. Click Run. You get the following message:

```
"Hello Robin! Rainy day. Take an umbrella."
```

What to do next

Your decision service is now complete. In the next task, you deploy and execute it by using the Automation Decision Services runtime.

< Previous | Next >
< Previous

Task 6: Deploying and running a decision

You share your latest changes. Then, you deploy your decision service, and run it in the runtime environment.

About this task

In this task, you...

- Share your latest changes.
- Create a version of your project.
- Deploy your decision service.
- Run the service in the runtime environment.

Tip: With assistance from your IT developers, you can use a CI/CD stack to share and build your decision service.

Step 1: Sharing your decision service

About this task

In this step, you share your decision service.

Procedure

1. Click the Share changes button in the top right toolbar.
2. Select My Service, Data, and My model to share all the changes .
3. Click Share, and enter the following comment:

   ```
   Changes from tasks 3, 4 and 5.
   ```

4. Click Share to share your changes in your Git repository. Now, you can deploy your decision service.

Step 2: Deploying your project

About this task

In this step, you deploy your project to the decision runtime.

Tip: In this tutorial, we build and deploy in Decision Designer, and a custom CI/CD stack can be used (see Building and deploying from Decision Designer).

Procedure

1. Open the Deploy tab. You see that no version exists, and you have a suggestion to create a new version from the changes that you shared in the previous steps.
2. Click Create version.
3. Enter the name `1.0.0` and the following description, and then click Create.

   ```
   Completed getting started version
   ```

   The version name should have a semantic meaning (see Creating versions).

4. Expand **`1.0.0`**. You see that it is not deployed.
5. Click Deploy at the end of the row for My Service.
6. Click Deploy in the confirmation window, and wait for the deployment to finish.
   You can see the decision ID in the **`Decision ID`** row. It has the following form:

   ```
   <Your User ID>/my_getting_started/my_service/myServiceDecisionService/1.0.0/myServiceDecisionService-1.0.0.jar
   ```

   Optionally, you can check the logs by clicking the View logs button in the Deployment status part.

Step 3: Running your decision service in the runtime

About this task

In this step, you use the Swagger UI tool to access the REST API that is generated by the runtime to run your decision service archive. For more information, see Swagger UI and Decision services from the embedded build service

Procedure

1. In the Deploy tab, click {...} at the end of the row for the My Service deployment.
   The Swagger UI tool opens the REST API of your decision service archive.
2. In the My Decision/My Service part, expand **`POST /My-Model/execute`**, and click Try it out.
3. Enter the following schema in the request body:

   ```
   {
     "name": "Jamie",
     "weather": {
       "rainForecast": 90,
       "stormAlert": true,
       "temperature": "cold"
     }
   }
   ```

   Click Execute. You get the following response:

   ```
   Hello Jamie! Stay home! There is a storm alert.
   ```

Results

You have completed the Automation Decision Services getting started tutorial. You created a decision service, defined its logic, and shared it. Then after making an input-output operation and deploying the decision service, you used a Swagger UI to run an archive of the decision service in an embedded runtime.

What to do next

To continue learning about the main features, including the business rules, decision tables, diagrams, rule policies, and data and task models, see the Training sample in the library of samples in Decision Designer. For more information about the library, see the section Import a sample in Creating decision services. To get hands-on experience with machine learning, external libraries, or decision service execution, see Automation Decision Services samples.

< Previous

Samples and tutorials in GitHub

Find samples and tutorials for Automation Decision Services in GitHub. They include a set of decision models for the getting started tutorial, and code for integrating machine learning.
Click Automation Decision Services samples to go to the GitHub repository. Its resources are organized into the following directories:

- Archives: Sample decision services that you can run directly in an Automation Decision Services runtime.

- Decision services: Sample decision services that you can import directly into Decision Designer, which is the development environment in Automation Decision Services. You must import the projects (see the section Importing a sample in Creating decision services).
- Samples: Tutorials and samples that you can use in Automation Decision Services. The tutorials provide detailed step-by-step instructions for specific features in Automation Decision Services, including Decision Designer and the external libraries. The samples provide code to use in Automation Decision Services.

The repository of decision services includes the Training sample, which has several decision services to introduce the main features in Automation Decision Services: diagrams, business rules, decision tables, rule policies, and data and task models. The sample includes a series of decision models that gradually increase in complexity in defining a decision service.

Developing decision services

Decision Designer provides a collaborative environment for managing and implementing decisions.

- **Managing decisions**
  In Decision Designer, you can create and organize decision services into decision automations and control access to individual decision automations.
- **Creating decision artifacts**
  In Decision Designer, you create all the artifacts you need to define your decision.

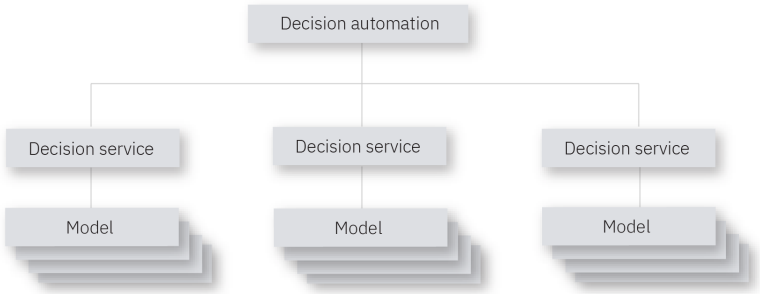
Managing decisions

In Decision Designer, you can create and organize decision services into decision automations and control access to individual decision automations.

- **Decision automations and decision services**
  A decision automation contains one or more decision services. Each decision service can hold several decision artifacts.
- **Connecting to a remote repository manually**
  After the IT user sets up a remote Git repository (remote repository) and configures its credentials to access it, you can connect a decision automation to this remote repository in Decision Designer.
- **Working collaboratively on decision services**
  Users who have access to the same decision automation can collaborate on the decision services that it contains.
- **Managing branches**
  Automation Decision Services allows branching so that you can manage the evolution of decision services over time.
- **Creating versions**
  You can create versions for your decision automation in Decision Designer. When you create a version of your decision automation, it can be later used to build and deploy a decision service.

Decision automations and decision services

A decision automation contains one or more decision services. Each decision service can hold several decision artifacts.



Decision automation

A decision automation is a container for Decision Designer users to organize one or more related decision services. When you open a decision automation, Decision Designer automatically loads all the decision services it contains.

Each decision automation must be connected to a remote Git repository (remote repository) to share the decision automation collaboratively. When you create a decision automation, you associate it with a branch that is synchronized with the repository. A branch contains a version of all the files and their change history. You can create several branches.

To understand branches, take as an example the operations of a bank. A bank might reassess offerings such as loans and corporate policies several times a year. You might want to create a branch each time a new policy or a new loan rate goes into effect.

Decision service

After you create your decision automation, you can create decision services in Decision Designer. A decision service contains decision artifacts that implement a decision:

Decision model
: The representation of a decision. It contains a diagram that describes a decision, its subdecisions, the data that is required to make these decisions, and how they are related to each other. It includes rules and decision tables that describe the decision logic.

Data model
: The representation of business concepts in the form of data types. Use data models to describe all the data that you need to make your decisions.

Predictive model
: The representation of a prediction based on a machine learning model. Predictions are consumed by decision models and task models.

Task model
: The implementation of a decision. It contains rule artifacts and a ruleflow to organize their execution.

Continuing the bank example, you might have a decision service for mortgages, and another one for student loans. The decision service for mortgages might contain a decision model for fixed rate mortgages and another one for adjustable rate mortgages. This decision service might also include a data model that represents, for example, the customers and their personal data, financial documents, and available mortgage options.

- **Creating decision automations**
  You work on decision services in decision automations. Each decision automation is linked to a remote Git repository (remote repository), and team members can work collaboratively through the repository.
- **Managing access to decision automations**
  Administrators can manage user permission to access decision automations.
- **Creating decision services**
  Decision services are where you create and verify your decisions. You can create an empty decision service, start from a sample decision service, or import a previously exported decision service.
- **Managing decision services**
  You can upgrade, duplicate, and export decision services.
- **Editing group IDs**
  A group ID is a unique identifier of a decision service, and can be used by IT users later. You can edit group IDs in Decision Designer.

Creating decision automations

You work on decision services in decision automations. Each decision automation is linked to a remote Git repository (remote repository), and team members can work collaboratively through the repository.

About this task

You create a decision automation.

Procedure

1. Log in your instance of Platform UI (Zen).
2. Click the navigation menu in the upper left of the portal page.
3. Expand Design, and click Automation Decision Services. Decision Designer opens.
4. Click Create.
5. Enter a name and purpose for the decision automation, and click Create. The decision automation page opens.

What to do next

- When you create a decision automation, a branch is also created by default. You can also create another branch after your decision automation is connected to a remote repository, and make it the current branch of the decision automation. Branches isolate changes specific to an objective, for example, a product release, a promotional period, or a trial version that has a limited delivery scope. For more information about creating another branch, see Creating branches.
- You can add collaborators who can access to your decision automation and assign them permission. For more information, see Managing access to decision automations.

Managing access to decision automations

Administrators can manage user permission to access decision automations.

About this task

You manage users who access your decision automation, and assign each of them a permission type. Multiple users can have the admin permission for the same decision automation.

Procedure

1. Click the overflow menu ⋮ in the decision automation tile, and then click Edit details.
2. In Edit decision automation, click User management.
3. Click Add permission, and add a user in the Name field.
4. Select a permission type of the user from the pull-down menu in the Permission field.

Table 1. Permission types

Permission type	Description

Permission type	Description
Read	Users with the read permission can do the following actions: • Open the decision automation. • Load changes. • Work on decision services in the decision automation. Restriction: You cannot share your changes with collaborators. Changes that you made are saved only in your personal copy.
Edit (Write)	Users with the edit permission can do everything that the read permission allows them to do, plus the following actions: • Configure on the Settings page in Decision Designer. • Connect to a remote Git repository. • Add machine learning providers. • Import external libraries. • Share changes. • Create, delete, or deploy a version.
Admin	Users with the admin permission can do everything that the edit permission allows them to do, plus the following actions: • Edit or remove the decision automation. • Manage user access and permission for the decision automation. However, you cannot remove yourself as a collaborator of the decision automation. Another user who has the admin permission for the decision automation can do it for you. • Merge and share changes on protected branches.

5. Click Save.

Creating decision services

Decision services are where you create and verify your decisions. You can create an empty decision service, start from a sample decision service, or import a previously exported decision service.

Before you begin

You need to have a decision automation where you create your decision service. For more information about decision automations, see Creating decision automations.

Procedure

1. Click the decision automation name in the tile in Decision Designer.
   Check the tabs on the decision automation page:

Table 1. Tabs on the decision automation page

Tab	What you can do
Decision services	Displays the list of decision services. • Create or delete decision services. • Rename the decision service or edit the description. • Import decision services • Import samples
Load changes	You see an icon with a blue dot when you have incoming changes that you need to load for the decision automation. • Check whether your collaborators made changes in the decision automation and shared them. • Retrieve the changes that have been shared by your collaborators. If there is any conflict between your changes and your collaborators' changes on the same decision artifact, you can resolve it here.
Share changes	• View the decision artifacts that you haven't shared yet. • Share decision services or decision artifacts that you created, updated, or deleted with your collaborators. • Revert some or all of your changed decision artifacts that are not shared to a previous state by undoing the changes you made.
Branches	Manage branches: • Create new branches based off existing ones. • Create merge requests and merge branches to incorporate changes from a source branch to a target branch. • Delete branches.
View history	• View the changes that are shared with your collaborators. • Revert to a version that was shared in the past. • Create a version of your decision automation. The version that you created can be later used to build and deploy a decision service.
Deploy	• Build and deploy decision services.

2. In the Decision services tab, click New decision to create a decision service. Choose whether to create an empty decision service or to create a decision service based on an exported decision decision service or a sample:

Table 2. Options available to create a decision service

Option	Procedure

Option	Procedure
Create a decision service from scratch	a. Click Create decision service. b. Enter a name for the decision service. c. Select a language. d. (Optional) Add a description. e. (Optional) Edit the group ID of the decision service. You need to follow a naming convention when you enter the group ID. For more information about the naming convention, see the Unique Package Names section in this documentation. Contact your administrator if you are not sure what group ID to enter. For more information about group IDs, see Editing group IDs. f. Click Create.
Import an existing decision service	You can import existing decision services, as well as decision modeling projects from IBM Operational Decision Manager and IBM Decision Composer. You can import .zip and .dproject files. These files can contain multiple decision services. a. Select Import decision service. b. Click Browse to select a file on your machine. c. Click Import.
Import a sample	A collection of samples is available in the Discovery tutorials and Industry samples, and Localized samples tabs. The samples under Discovery tutorials are designed to help get you started in Automation Decision Services. The samples under Industry samples allow you to discover decision services tailored for specific business applications. The samples under Localized samples show you an example of a simple decision service in the different languages supported by Automation Decision Services. a. Select the sample that you want to import. b. Click Import.

What to do next

The decision service is created. You can now do the following actions:

- Start working on decision artifacts for the decision service. For more information, see Creating decision artifacts.
- Share your work with collaborators. For more information, see Working collaboratively on decision services.

Managing decision services

You can upgrade, duplicate, and export decision services.

Upgrading

When you open a decision automation in Decision Designer, decision services that need to be upgraded are tagged with the Upgrade required tag.

1. Click the Upgrade button.
2. Click Upgrade.
   The results of the upgrade are displayed directly in the Upgrade decision service view.
3. When the upgrade is completed, close the Upgrade decision service view.

Duplicating

You might need to make changes in an existing decision service but want to make these changes in a separate decision service instead. You can create a copy of the existing decision service and work on it.

1. Open the Decision services tab on your decision automation page in Decision Designer.
2. Click the overflow menu for your decision service, and then select Duplicate.
3. Click the overflow menu for the duplicated decision service, and then select Edit details.
4. Update the name, description, and group ID for the duplicated decision service, and then click Save.
   Note: You need to follow a naming convention when you update the group ID. For more information about group IDs, see Editing group IDs.

Exporting

The Export feature creates a .zip file with all the decision services contained in a decision automation.

To download the content of an automation, click the Export icon in the menu bar at the top of your automation.

Editing group IDs

A group ID is a unique identifier of a decision service, and can be used by IT users later. You can edit group IDs in Decision Designer.

About this task

IT users can use a group ID to identify a decision service in their decision automations. Each decision service links to Maven dependencies that control how the decision service is consumed. For more information, see **`mavenGroupId`** in Decision service metadata.

You assign a group ID when you create a decision service. You can change the group ID later.

Artifact IDs for your decision artifacts are derived directly from their business names, and cannot be modified.

Procedure

1. Open the Decision services tab on your decision automation page in Decision Designer.
2. Click the overflow menu ⋮ for your decision service, and then select Edit details.
3. Update the group ID, and then click Save.
   Note:
   You need to follow a naming convention when you update the group ID. For more information about the naming convention, see the Unique Package Names section in this documentation. Contact your administrator if you are not sure what group ID to enter.
4. Go to the Share changes tab, and share your change.
   When you update the group ID for a decision service, group IDs for all decision artifacts such as decision models and data models in the decision service are also updated automatically.

What to do next

The IT users can now verify in the remote repository that the group ID is updated in the pom.xml file for the decision service, in .designproject.

Connecting to a remote repository manually

After the IT user sets up a remote Git repository (remote repository) and configures its credentials to access it, you can connect a decision automation to this remote repository in Decision Designer.

Before you begin

Tip: See Connecting to a remote repository automatically:

- To learn which Git providers are supported.
- You are an IT user or administrator, and want to automatically create a remote repository and connect your decision automations to it.

Your decision automation must be connected to a remote repository to use the build system of your organization, which allows you to build, deploy, and execute decision services. Your decision automation must be connected to the remote repository to share your work with collaborators. When you create a branch for your decision automation in addition to a default branch, your decision automation must be connected to the remote repository as well. The remote repository can also be used for keeping backups of decision source files.

There are two ways to connect to the remote repository:

- You can set up a remote repository to share changes with your collaborators. To establish a connection with a remote repository, make sure to obtain the URI of the repository from the IT or administrator to complete the procedure.
- If you set up your own private remote repository and connect a decision automation to it, the credentials that are related to the repository must have write access. In this case, you know the credentials and can configure them without asking the IT or administrator.

Verify the following conditions before you connect a decision automation to a remote repository:

- If the decision automation already contains decision artifacts, the remote repository must be empty.
- If the remote repository is not empty or already initialized, the decision automation must be empty.

When both the decision automation and the remote repository already contain data before connecting to each other, an error occurs and the decision automation can't be connected to the repository.

Procedure

1. Click the Settings ⚙ or Connect 🔗 icon on your decision automation page. The Remote Git repositories tab opens on the Settings page.
   When the Connect button is shown as connected, the decision automation is already connected to a remote repository.
2. Enter the URI for the remote repository in the Repository URI field.
   Examples:

   `https://myGitServer.com/<user>/<repository_name>.git`

   `git@myGitServer.com:<user>/<repository_name>.git`

   `ssh://git@myGitServer.com:<port>/<user>/<repository_name>.git`

   Including the SSH protocol is useful if you need to change the default SSH port.
   Important: In the SSH URI for AWS CodeCommit, it must contain **`SSH key ID`** that is linked to Upload SSH public key in the AWS Identity and Access Management (IAM) user interface. See the example URI:

   `ssh://<ssh_key_id>@myGitServer.com:<port>/<user>/<repository_name>.git`

Tip: Example URIs for Azure Repos:

`https://<organization_name>@dev.azure.com/<organization_name>/<project_name>/_git/<repository_name>`

`git@ssh.dev.azure.com:v3/<organization_name>/<project_name>/<repository_name>`

3. Select either Use existing credentials or Create or update credentials for the decision automation.
   Note: When you update the URI in Step 2, you must select Create or update credentials for the decision automation.
   a. If you selected Use existing credentials, click Connect.
   b. If you selected Create or update credentials for the decision automation, enter credential information in the Username & password or SSH key tab, and then click Connect.

Table 1. Different ways to enter Git credentials

Tab	Description
Username & password	Enter the username and the password for the Git URI. Restriction: The password that you provide to connect to a GitHub repository must be a personal access token. It cannot be the password that you use to log in to GitHub.
SSH key	Enter the SSH key for the Git URI in the SSH key field. Restriction: You can use only RSA or OpenSSH format.

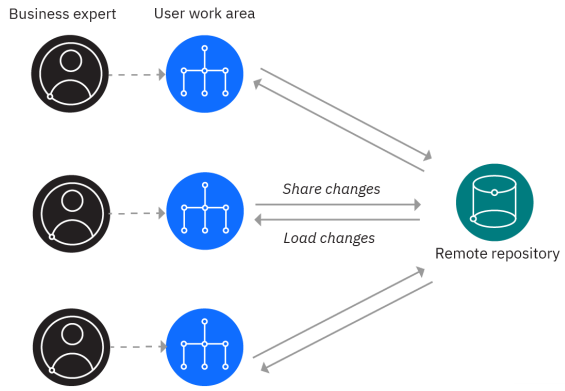
Results

The decision automation is now connected to the remote Git repository. Changes in the decision artifacts are pushed to this repository every time they are shared with collaborators.

Working collaboratively on decision services

Users who have access to the same decision automation can collaborate on the decision services that it contains.
Several collaborators can work on the same decision services. To make your changes available to others, you must share them.

When you make a change, it is automatically saved to your personal copy and visible to you only. Your decision automation must be connected to a remote repository to share your work with collaborators.



Sharing your work

Before you share changes, you must be up to date with the changes shared on the remote repository. Then, you might need to resolve conflicts. You can share when there is no more conflict. You can also revert to the state of your decision artifacts before you made changes, or you can restore your decision artifacts to a previous state that was shared with your collaborators.

Table 1. Tasks for sharing your work

Task	What to do	Where to find information
Create decision artifacts	Model and create decision artifacts.	Defining data Building decision models Building task models Integrating machine learning
Revert changes	Revert the changes that you made before you publish them.	Reverting changes

Task	What to do	Where to find information
Load changes	Check whether your collaborators made changes in the same decision automation. You see an icon with a blue dot when there are incoming changes that you need to load for the decision automation. If there is any conflict between the changes that you made and your collaborator shared, you must resolve it first. Then, you load the changes that are shared by your collaborators.	Loading changes Resolving conflicts
Run decisions	If you retrieved changes from a collaborator, check the behavior of your updated decision model. Changes that do not conflict from a file perspective might conflict from a semantic one. You can run test data to validate this behavior before sharing.	Running models
Share changes	Share your changes. Your changes are now available to your collaborators. If there is any conflict between the changes that you made and your collaborator shared, you must resolve it first. Then, you share your changes.	Sharing changes Resolving conflicts
Restore changes	Restore your artifact to a previous state that was shared by you or your collaborators.	Restoring changes

- **Loading changes**
  If a collaborator shares changes after you start making changes, you must manually retrieve the shared changes. You cannot share your changes until you retrieve their changes.
- **Resolving conflicts**
  When you work on a decision service and a collaborator shares changes on the same decision service before you do, there might be conflicts. You must resolve these conflicts.
- **Sharing changes**
  You share your changes so that your collaborators can retrieve them.
- **Reverting changes**
  You can revert some or all of your changed decision artifacts that are not shared to a previous state by undoing the changes you made.
- **Restoring changes**
  You can restore your decision artifacts to a previous state that was shared.

Loading changes

If a collaborator shares changes after you start making changes, you must manually retrieve the shared changes. You cannot share your changes until you retrieve their changes.

Procedure

1. Open the Load changes tab from your decision automation page to see whether someone shared changes.
   You see an icon with a blue dot in the tab when there are incoming changes that you need to load.
2. View the details of the changes that are shared by your collaborators.
3. To retrieve changes, click Load changes, and then click Load.

Resolving conflicts

When you work on a decision service and a collaborator shares changes on the same decision service before you do, there might be conflicts. You must resolve these conflicts.

Procedure

1. Open the Load changes tab from your decision automation page to see if there is any conflict.
   Expand the list and click Select version to see the side-by-side view of the conflicting changes.
2. Select a version to keep. The version that you do not select will be lost. Click Select version.
   Tip: If you want to change the version that you want to keep, click Update and select another version.
3. Click Load changes to resolve conflicts. Click Load.
4. If you chose to keep your version, go to the Share changes tab and share your changes.

Sharing changes

You share your changes so that your collaborators can retrieve them.

Before you begin

- If there are incoming changes, you must retrieve them first. Resolve conflicts if there is any.
- If you retrieved changes from a collaborator, check the behavior of your updated decision model. For more information about running your decision models, predictive models, and task models, see Running models.

Changes that do not conflict from a file perspective might conflict from a semantic one. For example, if you retrieved changes on a data model that your decision model is using and you didn't change this data model, there is no conflict between the files. But the results of your decision model might change. You must run test data to validate this behavior before sharing.

Procedure

1. Open the Share changes tab from your decision automation page to check the changes that have not been shared yet.
   You can share all or some of your changes. For example, share only some of your changes when the rest is not ready to be shared yet, or to associate different messages with different changes.

   Expand the list and click View details to see the details of the changes that you made.
2. Select the decision service and artifacts that you want to share, and click Share.
   Enter a message that describes the changes that you made so that others understand what you intended to do when they see your changes.
3. Click Share.

Results

The changes can be now shared with your collaborators.
Note: To the IT users:
When you make a change in the decision artifact, the corresponding pom.xml file for the artifact is automatically updated after you share it.

Reverting changes

You can revert some or all of your changed decision artifacts that are not shared to a previous state by undoing the changes you made.

About this task

You revert your changes that have not been shared.

If your changes have been shared, you can restore decision artifacts to a previous state in the View history tab. For more information, see Restoring changes.

Procedure

1. Open the Share changes tab from your decision automation page.
2. Select the artifacts that you want to revert from the list, and then click Revert changes.
   When you revert the artifacts to their previous state, any change that you have not shared yet is lost.
3. Click Revert.

Restoring changes

You can restore your decision artifacts to a previous state that was shared.

Procedure

1. Open the View history tab from your decision automation page to review the artifacts that have been shared.
2. Select the change set that you want to restore, and click Restore.
3. Select the decision services and artifacts that you want to restore, and then click Restore.
   Before you click Restore, you can review the details of the changes by clicking View details when this option is available.
4. Click Restore again in the confirmation window.
   The changes are now restored locally.
5. Go to the Share changes tab and share the changes that you restored.

Managing branches

Automation Decision Services allows branching so that you can manage the evolution of decision services over time.

- **Creating branches**
  When you create a decision automation, a branch is created by default. You can create another branch after your decision automation is connected to a remote repository, and make it the current branch of the decision automation.
- **Merging branches**
  You can merge branches to combine changes from one branch into another.
- **Protecting branches**
  Create branch protection rules to safeguard branches from inadvertent actions and undesirable changes.

Creating branches

When you create a decision automation, a branch is created by default. You can create another branch after your decision automation is connected to a remote repository, and make it the current branch of the decision automation.

Procedure

1. Open the Branches tab from your decision automation page.
2. Choose one the following options to create a branch:

Option	Procedure
Start from the Create branch menu	a. Click the Create branch button. b. Enter a branch name. c. In the Branch from section, use the drop-down list to choose which existing branch you want to base your new branch off. d. Click Create.
Start from the branch you want to copy	a. Click the Create new branch from icon next to the branch you want to base your new branch off. b. Enter a branch name. c. Click Create.

Results

You automatically switch to the newly created branch. If you want to switch to another branch, click the Switch branch icon in the toolbar at the top of your decision automation.

Merging branches

You can merge branches to combine changes from one branch into another.

Before you begin

The merge process cannot be started if there are unshared changes on your current branch. You must share or revert them before you can start to merge changes.

About this task

The merging process consists of the following steps:

- Select the branch that you want to merge into the current branch.
- Preview the changes that you are going to integrate into the current branch.
- Select the changes to apply to the current branch in case of merge conflict. Merge conflicts may arise when the two branches that you want to merge have competing changes.
- Share your changes with your collaborators to finalize the merge process.

You can use merge requests to notify your collaborators that a branch is ready for merging. To create a merge request, click the Start merge request icon next to the name of the branch that is ready for merging.

Procedure

1. Open the Branches tab from your decision automation page.
2. Click the Merge branch into icon next to name of branch you want to merge into the current branch.
3. If there is no merge conflict, you can compare the changes between the two branches before clicking Apply changes.
   If there are merge conflicts, you must resolve them before you can merge:
   a. Click the arrow next to the name of your decision service to see all the conflicting changes. Complete the two following steps for each conflict.
   b. Click View details to open the differences viewer and review the changes.
   c. Close the differences viewer and click Select version and select the version that you want to keep.
   Warning: Changes from the version that you do not select will be lost
   d. When you are done resolving the merge conflicts, click Apply changes.
4. The Share changes tab automatically opens.
   The merged changes are loaded and ready for review in your decision automation. The merge process will only be finalized after you share your changes. Before finalizing the merge, you can go back to your decision services to run your models and make sure everything still works as expected. If needed, you can make adjustments to your decision services. You can go back to the Share changes tab at any time by clicking View changes in the top banner.
   - If you are satisfied with the merged changes, click Finalize merge in the Share changes tab. You can enter a comment to describe the changes that you made. After you click Share, your changes become available to your collaborators and the merge operation is visible in the View history tab.
   - If you are not satisfied with the merged changes and want to revert them, click Cancel merge in the Share changes tab.

Protecting branches

Create branch protection rules to safeguard branches from inadvertent actions and undesirable changes.

About this task

While role permissions let you manage which users have write and read access to decision automations, it might not be enough to protect your important branches of development. When a branch is protected, only users with the admin permission can share the changes they made in this branch or merge changes from another branch into the protected branch. Users who do not have the admin permission can continue to work locally in protected branches.

You can create a branch protection rule for a specific branch, all branches, or any branch that matches a name pattern specified with fnmatch syntax:

- To protect a specific branch, enter the name of the branch. For example, to protect the `main` branch, create a branch rule with the following pattern: `main`.
- To protect any branch that contains a specific string, enclose that string in asterisks. For example, to protect any branch containing the word `prod`, create a branch rule with the following pattern: `*prod*`.
- To protect all current and future branches in your decision automation, use the wildcard character `*`.

Procedure

1. Click the Settings icon in the toolbar at the top of your decision automation page to open the Settings menu.
2. Open the Branch protection rules tab.
3. Click Add +.
4. Under Pattern, type the branch name or pattern that you want to protect.
   Click the arrow next to the pattern name to see to the list of branches the pattern applies to.

Creating versions

You can create versions for your decision automation in Decision Designer. When you create a version of your decision automation, it can be later used to build and deploy a decision service.

Before you begin

You can create versions when you have the following permission types for your decision automation:

- Admin - Administrative privileges
- Edit - Write permission

For more information about permissions, see Managing access to decision automations.

About this task

Versions correspond to tags in Git. A version is a snapshot of the decision automation and records a point-in-time of the decision services within the decision automation; therefore, it cannot be changed.

Procedure

1. Open the View history tab from your decision automation page.
2. Click Version next to the changes.
3. Enter a name and description of the version, and click Create.
   The version name must be unique.

   Note:
   If you do not follow the semantic versioning format, you cannot select and execute the last semantic version of the decision service.

   For more information about the semantic versioning and specifying semantic versions for decision services, see Executing the last semantic version.

What to do next

You can open the Deploy tab, and then build and deploy decision services. For more information, see Building and deploying from Decision Designer.

Creating decision artifacts

In Decision Designer, you create all the artifacts you need to define your decision.

- **Defining data**
  After identifying the data involved in the decision-making process, you can model this data by creating data models. Advanced users can also create and import external libraries to enrich their data models.
- **Building decision models**
  Decision models allow you to capture business decisions in a decision diagram, an easily understood representation that includes the data that is needed to make these decisions.
- **Building task models**
  Task models allow you to implement your business logic with rules, and control the flow execution of these rules in a ruleflow.
- **Integrating machine learning**
  Use machine learning to make decisions based on the analysis of past data.
- **Authoring the decision logic**
  You must author the decision logic that drives each decision. You express the decision logic with business rules and decision tables.

- **Declaring and managing dependencies**
  Declaring dependencies lets you reuse decision artifacts within a decision service.
- **Creating decision operations**
  In a decision operation, you define an entry-point to a model. A decision service must contain at least one decision operation to be deployed.
- **Enabling automatic refactoring**
  When automatic refactoring is enabled, common updates are automatically propagated to other decision artifacts. This feature is available as a technology preview.

Defining data

After identifying the data involved in the decision-making process, you can model this data by creating data models. Advanced users can also create and import external libraries to enrich their data models.

- **Creating a data model**
  Data models provide a user-defined view of data that represents real-world entities. You can either create data models from scratch or import data model definition files.
- **Working with external libraries**
  Automation Decision Services allows you to import and use external libraries in your decision services. External libraries allow you to enrich your data model and extend rule authoring with custom data types and functions from Java™ libraries.

Creating a data model

Data models provide a user-defined view of data that represents real-world entities. You can either create data models from scratch or import data model definition files. You create data models from the Data tab inside a decision service. You can create multiple data models in a decision service.

For a complete example of how to handle data, see the Training sample available on GitHub. Alternatively, you can import the Training sample from the Samples library in Decision Designer. For more information about the Samples library, see Creating decision services.

- **Modeling data in Automation Decision Services**
  You use the data model editor to define the data you need to make your decision.
- **Working with composite types**
  You create composite types, and define their list of attributes.
- **Working with enumeration types**
  You create enumeration types, and define the list of values they can take.
- **Importing a data model definition file**
  You can automatically populate a data model with composite types and enumeration types that are imported from a JSON schema file.
- **Using data models in other decision artifacts**
  To use your vocabulary in another decision artifact, you must create a dependency between the decision artifact and the data model.
- **Default verbalization**
  Automation Decision Services applies a default verbalization to all data model elements. You can edit the default verbalization of attributes in the data model editor.

Modeling data in Automation Decision Services

You use the data model editor to define the data you need to make your decision.
A data model is a collection of data types that represents the data you need to feed your decision. In this data model, you use built-in data types and custom data types as building blocks to model the structure of the real-life data that you want to represent.

For example, in a decision that computes the discount rate to apply to the purchases of a loyal customer, you would create a data model that represents the customer details and the content of its shopping cart.

The data types that you define in a data model are used to assign a type to each input data node and decision node in your decision models. Assigning a type to a node determines:

- What information the node can store and send.
- The possibilities when authoring the decision logic.

You can create multiple data models in a decision services. There are two ways to build a data model:

- Create a data model from scratch and define each data type you need manually.
- Import a data model definition file that contains per-defined data types. See Importing a data model definition file.

- Data types
- Verbalization
- JSON name

Data types

Each term that you define in a data model is associated to a data type. This data type can either be a built-in type, or a custom type.

Automation Decision Services supports the following built-in data types:

Table 1. List of supported built-in data types in Automation Decision Services

Data type	Description
Boolean	The logical values true or false

Data type	Description
calendar duration	A duration expressed by units of time, for example `P6M10D` for a period of 6 months and 10 days
date	A specific calendar date, for example `01/01/2019`
date & time	The combination of date and time, for example `01/01/2019 12:30:00`
day of week	A day of the week
integer	A whole number, for example `15` or `-8`
month	A month of the year, for example `January`
number	A numeric value, for example `15` or `-8.5`
string	A set of characters, for example `"Hello world"`
time	A specific time value, for example `12:30:00`
time period	A specific time period that is characterized by a start time and an end time, for example `2019-07-14T01:01:01Z/2019-07-15T01:01:01Z`
year	A year, for example `2019`

Two custom types are available in Automation Decision Services:

Composite type
: A composite type is a composite of other existing data types. You might create a composite type whose attributes include built-in types, or other custom types. The attributes of a composite type describe the characteristics of the real-world entity that it represents.

  A car, for example, can be characterized by its brand, its ID, and its category. To represent it, you can create a composite type car that contains the following attributes:

  - brand, of built-in type string
  - ID, of built-in type number
  - car category, of custom type car category

  For more information about composite types, see Working with composite types.

Enumeration type
: An enumeration type is a data type that contains a specific set of values. The data that the enumeration represents can only take a value from this list.

  The custom type car category, for example, can be described as an enumeration that can only take one the three following values: compact, full size, or premium.

  For more information about enumeration types, see Working with enumeration types.

Automation Decision Services also allows you to enrich your data model with custom data types and functions from imported external libraries. For more information about building and importing external libraries, see Working with external libraries.

Verbalization

For each element that you define in the data model, Automation Decision Services applies a default verbalization. This default verbalization defines how these elements are referenced in business rules and decision tables.

The verbalization depends on the type of element, for example whether it is a simple type or a composite type with several attributes. You can see examples of this default verbalization in the data model editor when you open the details of an element. For more information about the default verbalization that is applied to attributes, see Default verbalization.

Automation Decision Services allows you to edit the default verbalization directly in the data model editor.

Each attribute comes with a set of automatically generated expression and action phrases:

- An expression accesses the data associated with a composite type, for example `{name} of {this}`
- An action modifies the data associated with a composite type, for example `set the name of {this} to {name}`

You might want to edit the default verbalization of expression and action phrases to make them easier to understand or avoid wordiness.

Additionally, each data type and attribute that you define has an associated plural form and a list of associated articles in both its singular and plural forms:

- A definite and an indefinite article for both data types and attributes
- A quantitative article for data types only

You can change this default verbalization to specify, for example, the plural of a term that has an irregular form.

JSON name

Each element that you define in a data model has an associated JSON name. This JSON name is used when validating or executing a decision service. Each data type must have a unique JSON name. Each attribute must have a unique JSON name within its enclosing data type.

The JSON name is derived directly from the element name, but uses the camel case convention. The JSON name can be edited.

Restriction: To be used in automation services, JSON names must:

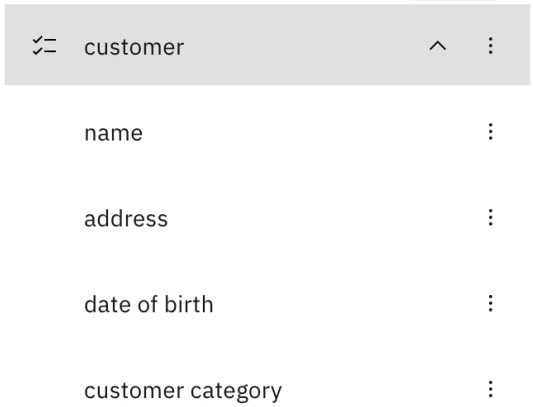
- Have a maximum length of 64 characters.
- Be valid JavaScript identifiers.
- Not be Java reserved words.
- Not be JavaScript reserved words.

For a JSON name to be considered a valid JavaScript identifier, it must:

- Only contain letters, digits, `$`, and `_`.
- Not start with a digit.
- Not contain any space.

Working with composite types

You create composite types, and define their list of attributes.
A composite data type is composed of a name and a list of attributes that describe the characteristics of an object. It lets you regroup related data under one entity. In the following example, you see a composite type called `customer` with its associated attributes:



Attributes can be built-in data types or other custom data types.

Attributes can also be identified as a list. You use lists when your decision acts on more than one item of something, and you need to make a distinction between the items. For example, if your decision is about checking in baggage on a plane, you can identify `baggage` as a list, since more than one of piece of baggage may be checked in.

- **Creating composite types**
  You create composite types in the data model editor, and edit their details.
- **Adding attributes**
  You add attributes to describe the structure of the composite type.

Creating composite types

You create composite types in the data model editor, and edit their details.

Procedure

1. Click the Add button + next to Data types and select Composite type from the drop-down list.
2. Enter a suitable name for the composite type. It is recommended to use a singular form for the type name.
3. If necessary, you can update the details of the composite type.
   - Define a documentation property using the Documentation text field. This documentation will be visible in the prediction menu in the rule editor.
   - Edit the JSON name associated with the composite type.
   - Change the plural form or the articles associated with the composite type. To do so, click the placeholder for the article or plural form that you want to edit and edit it directly in the pop-up window that opens. You can revert to the default verbalization by clicking **Reset**.
   - Select the gender of the composite type if your decision service is in French. The masculine gender is applied by default.

What to do next

You can now define the attributes of the composite type. See Adding attributes.

Adding attributes

You add attributes to describe the structure of the composite type.

Procedure

1. Click the Add button + in the Attributes panel.
2. Enter a suitable name for the attribute. It is recommended to use a singular form for the attribute name.
3. Associate the attribute with a data type that you select from the Type drop-down list.
4. If necessary, edit the details of the attribute. To access the full details of the attribute, click the Edit button next to the attribute, or click the attribute in the Data types panel.
   - Define a documentation property using the Documentation text field. This documentation will be visible in the prediction menu in the rule editor.
   - Edit the JSON name associated with the attribute.
   - Change the plural form or the articles associated with the attribute. To do so, click the placeholder for the article or plural form that you want to edit and edit it directly in the pop-up window that opens. You can revert to the default verbalization by clicking **Reset**.

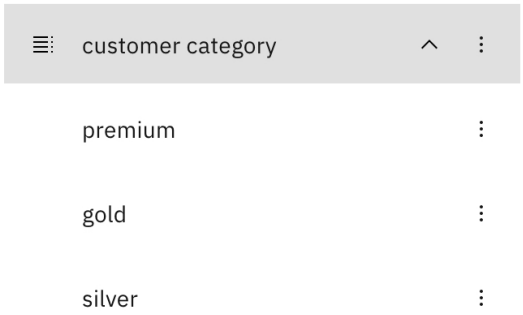
- Edit the default verbalization of expressions and actions associated with the attribute. For more information about the default verbalization applied to attributes, see Default verbalization.
- Identify the attribute as a list by checking List. When an attribute is identified as a list, it can have multiple values.
- Select the gender of the attribute if your decision service is in French. The masculine gender is applied by default.

5. Repeat for each attribute of the composite type.

Working with enumeration types

You create enumeration types, and define the list of values they can take.
An enumeration is composed of a name and a list of values. The data that the enumeration represents can take only a value from this list.

In the following example, you see an enumeration called **`customer category`** with its possible values:



- **Creating enumeration types**
  You create enumeration types in the data model editor, and edit their details.
- **Adding values**
  You define the values the enumeration type can take.
- **Extracting values from an external data source**
  You can populate enumeration types with values extracted from an Excel file.

Creating enumeration types

You create enumeration types in the data model editor, and edit their details.

Procedure

1. Click the Add button + next to Data types and select Enumeration type from the drop-down list.
2. Enter a suitable name for the enumeration type. It is recommended to use a singular form for the type name.
3. If necessary, you can update the details of the enumeration type.
   - Define a documentation property using the Documentation text field. This documentation will be visible in the prediction menu in the rule editor.
   - Edit the JSON name associated with the enumeration type.
   - Change the plural form or the articles associated with the enumeration type. To do so, click the placeholder for the article or plural form that you want to edit and edit it directly in the pop-up window that opens. You can revert to the default verbalization by clicking **Reset**.
   - Select the gender of the enumeration type if your decision service is in French. The masculine gender is applied by default.

What to do next

You can now define the values of the enumeration type. See Adding values.

As an alternative, you can populate the enumeration type with values extracted from an Excel file. See Extracting values from an external data source.

Adding values

You define the values the enumeration type can take.

Procedure

1. Click the Add button + in the Values panel.
2. Enter a suitable name for the value.
3. If necessary, edit the details of the value. To access the full details of the value, click the Edit button next to the value, or click the value in the Data types panel.
   - Define a documentation property using the Documentation text field. This documentation will be visible in the prediction menu in the rule editor.
   - Edit the JSON name associated with the value.

4. Repeat for each value of the enumeration type.

Extracting values from an external data source

You can populate enumeration types with values extracted from an Excel file.
Automation Decision Services allows you to import sets of values stored and managed in an Excel file outside of Decision Designer. The Excel file is stored as an external data source in the data model. You can modify the Excel file and then upload the changes to Decision Designer.

The Excel file must have the following structure:

Value column
: You must create a value column to enter the values you need.

Verbalization column
: You must create a verbalization column to define the verbalization of each value. The verbalization is the name displayed for the value when editing a rule.

Documentation column
: The documentation column is optional. You can create a column to enter documentation on a value.

Sheets
: You can have several sheets in the same Excel file. Each worksheet corresponds to a set of values.

An Excel file template is available in the External data sources tab in your data model.

- **Importing an Excel file**
  You import the Excel file where your values are stored.
- **Mapping values to an enumeration type**
  You map the enumeration type to values extracted from an Excel file.
- **Updating values**
  When your values change, you must update them in Decision Designer.

Importing an Excel file

You import the Excel file where your values are stored.

Procedure

1. In your data model, click the External data sources tab.
2. Click  Upload an Excel file and click Browse to select the Excel file.
3. Click Import file.

What to do next

You can now map the Excel file to an enumeration type to populate it with values.

Mapping values to an enumeration type

You map the enumeration type to values extracted from an Excel file.

Procedure

1. In your data model, open the Modeling tab.
2. Create an enumeration type, or open an existing one.
3. Click Select file under External data source.

   Note: If you are mapping an existing enumeration type, click Expand to access the Select file menu.
4. In the Excel file field, select the Excel file that you uploaded.
5. In the Sheet field, select the sheet that contains the values that you want to map to your type.
6. Select the Table with header check box if you have created a header in your Excel file.
   Selecting this check box displays the name of the columns in the drop-down lists instead of the default column letters.
7. In the Value, Verbalization, and Documentation fields, select the corresponding columns in your Excel file.
8. Click Save.

Results

The values are displayed as read-only in the data model editor. The values that the enumeration type contained before you mapped it are now tagged as deprecated and can be deleted.

Updating values

When your values change, you must update them in Decision Designer.

Procedure

1. In your data model, open the External data sources tab.
2. Select the Excel file that you want to update.
3. Click Update file and click Browse to select the new version of the Excel file.
   Note: The name of the new file must match the name of the original file.
4. Click Import file.
5. Under Mapped types, check if there are any changes to apply to the enumeration types mapped with values from this file.
   The list shows any new or deprecated value.
6. Select the enumeration types that you want to update and click Apply changes.

Importing a data model definition file

You can automatically populate a data model with composite types and enumeration types that are imported from a JSON schema file.

About this task

The following JSON schema formats are supported:

- Swagger 2.0
- OpenAPI 2.0
- OpenAPI 3.0
- OpenAPI 3.1
- JSON schema draft 04
- JSON schema draft 06
- JSON schema draft 07
- JSON schema draft 2019-09
- JSON schema draft 2020-12

The following example shows a Swagger 2.0 definition file that contains a definitions element. Inside the definitions element, two data types are described:

- A composite type **person** with multiple attributes.
- An enumeration type **status** with three values, **Married**, **Single**, and **Divorced**.

```
{
  "swagger": "2.0",
  "info": {
    "title": "Swagger definition file",
    "description": "Simple definition file"
  },
  "definitions": {
    "Person": {
      "type": "object",
      "properties": {
        "firstName": {
          "type": "string"
        },
        "dateOfBirth": {
          "type": "string",
          "format": "date-time"
        },
        "dateOfLoanStart": {
          "type": "string",
          "format": "date"
        },
        "numberOfChildren": {
          "type": "integer"
        },
        "income": {
          "type": "number"
        },
        "noBankrupcy": {
          "type": "boolean"
        },
        "lastName": {
          "type": "string"
        },
        "status": {
          "type": "string",
          "enum": [
            "Married",
            "Single",
            "Divorced"
          ]
        },
        "descendants": {
          "type": "array",
          "items": {
```

```
            "$ref": "#/definitions/Person"
          }
        }
      }
    }
  }
}
```

You can annotate schemas to preserve the data types and attributes that you want to import. The following tags are available:

- **`x-ibm-decision-adapt-type-name`**: when set to **`false`**, the name of the data type is preserved.
- **`x-ibm-decision-adapt-property-name`**: when set to **`false`**, the name of the attribute is preserved.

Take the following example:

```
"RadioNetworkControllerRole": {
    "x-ibm-decision-adapt-type-name" : false,
    "x-ibm-decision-adapt-property-name" : false,
    "enum": [
      "D_NRC",
      "S_NRC"
    ],
    "type": "string"
  }
```

Without the **`x-ibm-decision-adapt-type-name`** tag set to **`false`**, the data type name would appear as **`radio network controller role`** once imported into Decision Designer.

Without the **`x-ibm-decision-adapt-type-name`** tag set to **`false`**, the two attributes would appear as **`D NRC`** and **`S NRC`** once imported into Decision Designer.

Procedure

1. Open the Data tab in your decision service and click Create.
2. Enter a name for the data model.
3. Drag your definition file into the drop target area.
   Alternatively, you can click the drop target area and navigate to the definition file on your local system.
4. Click Create.

Results

Your data model opens. You can now start to use the data types that it contains in your decision service. For more information, see Using data models in other decision artifacts.

Using data models in other decision artifacts

To use your vocabulary in another decision artifact, you must create a dependency between the decision artifact and the data model.
This action can be done at the creation of the decision artifact, or at any moment from the Dependencies tab. You can create dependencies between a decision artifact and multiple data models.

Creating a dependency automatically

You can automatically create a dependency between a data model and the following decision artifacts: decision models, task models, and predictive models.

In the Create model wizard, select your data model under Select the data model you want to use as a data source. The data models contained in your decision service are listed in alphabetical order. The first data model in the list is automatically selected. You can select another one from the drop-down list.

When you select a data model at the creation of a decision artifact, a dependency is automatically created between the artifact and the data model. This dependency is then visible in the Dependencies tab.

Creating a dependency manually

This option can be used to create a dependency between a data model and any other decision artifact.

1. From your decision artifact, open the Dependencies tab.
2. Click Add +.
3. Select the data model that you want to use and click Add.

Once you have created the dependency, the data types from your data model are available in your decision artifact.

Default verbalization

Automation Decision Services applies a default verbalization to all data model elements. You can edit the default verbalization of attributes in the data model editor.
The default verbalization is defined for a specific locale.

For examples of localized data models, see the Brazilian Portuguese, Chinese, French, Italian, Japanese, German, and Spanish samples available on GitHub. You can choose the version of Automation Decision Services that you are using from the branch menu.

Alternatively, you can import localized samples from the Samples library in Decision Designer. For more information about the Samples library, see Creating decision services.

- **Default verbalization in Brazilian Portuguese**
  The following table shows the default verbalization of data model elements in Brazilian Portuguese.
- **Default verbalization in Chinese**
  The following table shows the default verbalization of data model elements in Chinese.
- **Default verbalization in English**
  The following table shows the default verbalization of data model elements in English.
- **Default verbalization in French**
  The following table shows the default verbalization of data model elements in French.
- **Default verbalization in German**
  The following table shows the default verbalization of data model elements in German.
- **Default verbalization in Italian**
  The following table shows the default verbalization of data model elements in Italian.
- **Default verbalization in Japanese**
  The following table shows the default verbalization of data model elements in Japanese.
- **Default verbalization in Spanish**
  The following table shows the default verbalization of data model elements in Spanish.

Default verbalization in Brazilian Portuguese

The following table shows the default verbalization of data model elements in Brazilian Portuguese.

Table 1. Default verbalization in Brazilian Portuguese

Data model element	Default verbalization
Types For example: `Cliente`	Class name in lowercase For example: `cliente`
Constructors Constructor types are generated for composite types For example: A composite type **`Cliente`** with two attributes **`nome`** and **`apelido`**	*um novo <type> [no qual <attribute> seja <value> (, <attribute> seja <value>)*]* For example: `um novo cliente no qual o nome seja "John", o apelido seja "Doe"`
Attributes For example: `idade`	Expression *{XXX} {this, PARTITIVE_ARTICLE}* For example: `{idade} {this, PARTITIVE_ARTICLE}`
	Action *atribuir a/o XXX {this, PARTITIVE_ARTICLE} a {XXX}* `atribuir a idade {this, PARTITIVE_ARTICLE} a {idade}`
Boolean attributes For example: `boolean aceite`	**Expressions** Affirmative *{this} é XXX* For example: `{this} é aceite` Negative *{this} não está XXX* For example: `{this} não está aceite`

Data model element	Default verbalization
	Actions Affirmative *{this} fica XXX* For example: `{this} fica aceite` Negative *{this} não fica XXX* For example: `{this} não fica aceite` Conditional *{this} fica XXX somente se {XXX}* For example: `{this} fica aceite somente se {aceite}`
List attributes For example: A list of: **endereço**	*adicionar {0} à lista uns/umas XXX {this, PARTITIVE_ARTICLE}* For example: `adicionar {0} à lista uns endereços {this, PARTITIVE_ARTICLE}` *remover {0} da lista uns/umas XXX {this, PARTITIVE_ARTICLE}* For example: `remover {0} da lista uns endereços {this, PARTITIVE_ARTICLE}` *limpar XXX {this, PARTITIVE_ARTICLE}* For example: `limpar os endereços {this, PARTITIVE_ARTICLE}`
Number and integer attributes For example: `int desonto`	*adicionar {number} ao/à XXX {this, PARTITIVE_ARTICLE}* For example: `adicionar {desconto} ao desconto {this, PARTITIVE_ARTICLE}` *subtrair {number} do/da XXX {this, PARTITIVE_ARTICLE}* For example: `subtrair {desconto} do desconto {this, PARTITIVE_ARTICLE}`

Default verbalization in Chinese

The following table shows the default verbalization of data model elements in Chinese.

Table 1. Default verbalization in Chinese

Data model element	Default verbalization
Types For example: 客户	Class name For example: 客户
Constructors Constructor types are generated for composite types For example: A composite type **客户** with two attributes **名** and **姓**	*新建<类>[符合条件：当前<属性>是<数值>(,当前<属性>是<数值>)*]* For example: 新建客户符合条件：当前名是"王"，当前姓是"小明"
Attributes For example: 年龄	Expression *{this} 的{XXX}* For example: `{this}` 的 `{年龄}`
	Action *设置{this} 的当前{XXX} 为{0}* 设置 `{this}` 的当前年龄为 `{年龄}`

Data model element	Default verbalization
Boolean attributes For example: 布尔类型 已接受	**Expressions** Affirmative *{this} 是 XXX* For example: `{this}` 是 当前已接受 Negative *{this} 不是 XXX* For example: `{this}` 不是 当前已接受
	Actions Affirmative *{this} 的 XXX 设置为真* For example: `{this}` 的当前已接受设置为真 Negative *{this} 的 XXX 设置为假* For example: `{this}` 的当前已接受设置为假 Conditional *只有 {XXX} 时 {this} 的 XXX 设置为真* For example: 只有 `{已接受}` 时 `{this}` 的当前已接受设置为真
List attributes For example: A list of: **地址**	*在 {this} 的这些 XXX 中增加 {0}* For example: 在 `{this}` 的这些 地址中增加 `{0}` *从 {this} 的这些 XXX 中删除 {0}* For example: 从 `{this}` 的这些 地址中删除 `{0}` *删除 这些 XXX 的 {this}* For example: 删除 这些 地址 的 `{this}`
Number and integer attributes For example: 整数类型 折扣	*加 {number} 到 {this} 的属性 XXX 里* For example: 加 `{number}` 到 `{this}` 的属性当前折扣里 *{number} 到 {this} 的属性 XXX 里* For example: 减 `{number}` 到 `{this}` 的属性当前折扣里

Default verbalization in English

The following table shows the default verbalization of data model elements in English.

Table 1. Default verbalization in English

Data model element	Default verbalization
Types For example: `Customer`	Class name in lowercase For example: `customer`

Data model element	Default verbalization
Constructors Constructor types are generated for composite types For example: A composite type **`Customer`** with two attributes **`first name`** and **`last name`**	*a new <type> [where <attribute> is <value> (, <attribute> is <value>)*]* For example: `a new customer where the first name is "John", the last name is "Doe"`
Attributes For example: `age`	Expression *{XXX} of {this}* For example: `{age} of {this}`
	Action *set the {XXX} of {this} to {0}* `set the discount of {this} to {discount}`
Boolean attributes For example: `boolean accepted`	**Expressions** Affirmative *{this} is XXX* For example: `{this} is accepted` Negative *{this} is not XXX* For example: `{this} is not accepted`
	Actions Affirmative *{this} gets XXX* For example: `{this} gets accepted` Negative *{this} does not XXX* For example: `{this} does not get accepted` Conditional *{this} gets XXX only if {XXX}* For example: `{this} does gets accepted only if {accepted}` The *make it {0} that {this} is XXX* syntax is now deprecated.
List attributes For example: `address List`	*add {0} to the XXX of {this}* For example: `add {0} to the addresses of {this}` *remove {0} from the XXX of {this}* For example: `remove {0} from the addresses of {this}` *clear the XXX of {this}* For example: `clear the addresses of {this}`

Data model element	Default verbalization
Number and integer attributes For example: `int discount`	*add {number} to the XXX of {this}* For example: `add {number} to the discount of {this}` *subtract {number} from the XXX of {this}* For example: `subtract {0} from the discount of {this}`

Default verbalization in French

The following table shows the default verbalization of data model elements in French.

Table 1. Default verbalization in French

Data model element	Default verbalization
Types For example: `client`	For example: `client`
Constructors Constructor types are generated for composite types For example: A composite type **`Client`** with two attributes **`prénom`** and **`nom`**	*un nouveau <type> [tel que <attribute> est <value> (, <attribute> est <value>)*]* For example: `un nouveau client tel que le prénom est "John", le nom est "Doe"`
Attributes For example: `age`	Expression *{XXX} {this, PARTITIVE_ARTICLE}* For example: `{age} {this, PARTITIVE_ARTICLE}`
	Action *assigner le/l'/la XXX {this, PARTITIVE_ARTICLE} à {XXX}* `assigner l'age {this, PARTITIVE_ARTICLE} à {age}`
Boolean attributes For example: `boolean accepté`	**Expressions** Affirmative *{this} est XXX* For example: `{this} est accepté` Negative *{this} n'est pas XXX* For example: `{this} n'est pas accepté`
	Actions Affirmative *que {this} soit XXX* For example: `que {this} soit accepté` Negative *que {this} ne soit pas XXX* For example: `que {this} ne soit pas accepté` Conditional *que {this} soit XXX seulement si XXX* For example: `que {this} soit accepté seulement si {accepté}` The *mettre à {XXX} que {this} est XXX* syntax is now deprecated.

Data model element	Default verbalization
List attributes For example: A list of: **adresse**	*ajouter {0} à la liste des XXX {this,PARTITIVE_ARTICLE}* For example: `ajouter {0} à la liste des adresses {this, PARTITIVE_ARTICLE}` *retirer {0} de la liste des XXX {this, PARTITIVE_ARTICLE}* For example: `retirer {0} de la liste des adresses {this, PARTITIVE_ARTICLE}` *effacer les XXX {this, PARTITIVE_ARTICLE}* For example: `effacer les adresses {this, PARTITIVE_ARTICLE}`
Number and integer attributes For example: `int remise`	*ajouter {XXX} à la/a l'/au XXX {this, PARTITIVE_ARTICLE}* For example: `ajouter {remise} à la remise {this, PARTITIVE_ARTICLE}` *retirer {XXX} de la/de l'/du XXX {this, PARTITIVE_ARTICLE}* For example: `retirer {remise} de la remise {this, PARTITIVE_ARTICLE}`

Default verbalization in German

The following table shows the default verbalization of data model elements in German.

Table 1. Default verbalization in German

Data model element	Default verbalization
Types For example: `Kunde`	For example: `Kunde`
Constructors Constructor types are generated for composite types For example: A composite type **Kunde** with two attributes **Vorname** and **Nachname**	*ein neuer <type> [wobei <attribute> ist <value> (, <attribute> ist <value>)*]* For example: `ein neuer Kunde wobei der Vorname ist "John", der Nachname ist "Doe"`
Attributes For example: `Alter`	Expression *{XXX} von {this}* For example: `{Alter} von {this}`
	Action *setze das/der/die XXX von {this} auf {XXX}* `setze das Alter von {this} auf {Alter}`
Boolean attributes For example: `boolean akzeptiert`	**Expressions** Affirmative *{this} ist XXX* For example: `{this} ist akzeptiert` Negative *{this} ist nicht XXX* For example: `{this} ist nicht akzeptiert`

Data model element	Default verbalization
	Actions Affirmative *{this} wird XXX* For example: `{this} wird akzeptiert` Negative *{this} wird nicht XXX* For example: `{this} wird nicht akzeptiert` Conditional *{this} wird genau dann XXX wenn {XXX}* For example: `{this} wird genau dann akzeptiert wenn {akzeptiert}`
List attributes For example: A list of: **`Adresse`**	*füge {0} zu der/die/das XXX von {this} hinzu* For example: `füge {0} zu die Adressen von {this} hinzu` *entferne {0} aus der/die/das XXX von {this}* For example: `entferne {0} aus die Adressen von {this}` *lösche der/die/das XXX von {this}* For example: `lösche die Adressen von {this}`
Number and integer attributes For example: `int Rabbat`	*füge {XXX} zu der/die/das XXX von {this} hinzu* For example: `füge {Rabatt} zu der Rabatt von {this} hinzu` *ziehe {XXX} von der/die/das XXX von {this} ab* For example: `ziehe {Rabatt} von der Rabatt von {this} ab`

Default verbalization in Italian

The following table shows the default verbalization of data model elements in Italian.

Table 1. Default verbalization in Italian

Data model element	Default verbalization
Types For example: `cliente`	For example: `cliente`
Constructors Constructor types are generated for composite types For example: A composite type **`cliente`** with two attributes **`nome`** and **`cognome`**	*un nuovo <type> [dove <attribute> è <value> (, <attribute> è <value>)*]* For example: `un nuovo cliente dove il nome è "John", il cognome è "Doe";`
Attributes For example: `età`	Expression *{XXX} {this, PARTITIVE_ARTICLE}* For example: `{età} {this, PARTITIVE_ARTICLE}`
	Action *cambia il/l'/la XXX {this, PARTITIVE_ARTICLE} a* `cambia l'età {this, PARTITIVE_ARTICLE} a {età}`

Data model element	Default verbalization
Boolean attributes For example: `boolean accettato`	**Expressions** Affirmative *{this} è XXX* For example: `{this} è accettato` Negative *{this} non è XXX* For example: `{this} non è accettato`
	Actions Affirmative *{this} verrà XXX* For example: `{this} verrà accettato` Negative *{this} non verrà XXX* For example: `{this} non verrà accettato` Conditional *{this} verrà XXX soltanto se {XXX}* For example: `{this} verrà accettato soltanto se {accettato}`
List attributes For example: A list of: **`indirizzo`**	*aggregare {0} in il/l'/la XXX {this,PARTITIVE_ARTICLE}* For example: `aggregare {0} nell'indirizzo {this, PARTITIVE_ARTICLE}` *togliere {0} di gli/i XXX {this,PARTITIVE_ARTICLE}* For example: `togliere {0} degli indirizzi {this, PARTITIVE_ARTICLE}` *cancella gli/i XXX {this, PARTITIVE_ARTICLE}* For example: `cancella gli indirizzi {this, PARTITIVE_ARTICLE}`
Number and integer attributes For example: `int sconto`	*aggiunge {XXX} a il/l'/la/lo XXX {this, PARTITIVE_ARTICLE}* For example: `aggiunge {sconto} allo sconto {this, PARTITIVE_ARTICLE}` *sostrae {XXX} da il/l'/la/lo XXX {this, PARTITIVE_ARTICLE}* For example: `sostrae {sconto} dallo sconto {this, PARTITIVE_ARTICLE}`

Default verbalization in Japanese

The following table shows the default verbalization of data model elements in Japanese.

Table 1. Default verbalization in Japanese

Data model element	Default verbalization
Types For example: 顧客	Class name For example: 顧客

Data model element	Default verbalization
Constructors Constructor types are generated for composite types For example: A composite type **顧客** with two attributes **名前** and **苗字**	*新規の <タイプ> [ここで <属性> は <値> (, <属性> は <値>)*]* For example: 新規の顧客 ここで 名前は "太郎", 苗字は "山田"
Attributes For example: 年齢	Expression *{this}の{XXX}* For example: {this}の{年齢}
	Action *{this}の{XXX}を{0}とする* {this}の年齢を{年齢}とする
Boolean attributes For example: boolean 承認	**Expressions** Affirmative *{this}はXXXする* For example: {this}は承認する Negative *{this}はXXXしていない* For example: {this}は承認していない
	Actions Affirmative *{this}をXXXする* For example: {this}を承認する Negative *{this}をXXXしない* For example: {this}を承認しない Conditional *{XXX}の場合のみ{this}をXXXする* For example: {承認}の場合のみ{this}を承認する
List attributes For example: 住所	*{this}のXXXに{0}を追加* For example: {this}の住所に{0}を追加 *{this}のXXXから{0}を削除* For example: {this}の住所から{0}を削除 *{this}のXXXをクリアする* For example: {this}の住所をクリアする

Data model element	Default verbalization
Number and integer attributes For example: `int` 割引	*{this}のXXXに{number}を追加する* For example: `{this}`の割引に`{`割引`}`を追加する *{this}のXXXから{number}を引く* For example: `{this}`の割引から`{`割引`}`を引く

Default verbalization in Spanish

The following table shows the default verbalization of data model elements in Spanish.

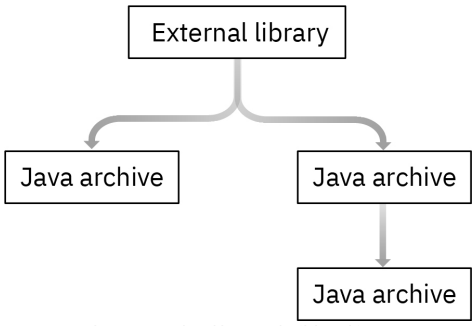
Table 1. Default verbalization in Spanish

Data model element	Default verbalization
Types For example: `Cliente`	Class name in lowercase For example: `cliente`
Constructors Constructor types are generated for composite types For example: A composite type **`Cliente`** with two attributes **`nombre`** and **`apellido`**	*un nuevo <type> [en que <attribute> es <value> (, <attribute> es <value>)*]* For example: `un nuevo cliente en que el nombre es "John", el apellido es "Doe"`
Attributes For example: `edad`	Expression *{XXX} {this, PARTITIVE_ARTICLE}* For example: `{edad} {this, PARTITIVE_ARTICLE}`
	Action *cambiar el/la XXX {this, PARTITIVE_ARTICLE} a {XXX}* `cambiar la edad {this, PARTITIVE_ARTICLE} a {edad}`
Boolean attributes For example: `boolean aceptado`	**Expressions** Affirmative *{this} es XXX* For example: `{this} es aceptado` Negative *{this} no es XXX* For example: `{this} no es aceptado`
	Actions Affirmative *{this} será XXX* For example: `{this} será aceptado` Negative *{this} no será XXX* For example: `{this} no será aceptado` Conditional *{this} será XXX solo si {XXX}* For example: `{this} será aceptado solo si {aceptado}`

Data model element	Default verbalization
List attributes For example: A list of: **dirección**	*agregar {0} en los/las XXX {this, PARTITIVE_ARTICLE}* For example: `agregar {0} en las direcciones {this, PARTITIVE_ARTICLE}` *quitar {0} de los/las XXX {this, PARTITIVE_ARTICLE}* For example: `quitar {0} de las direcciones {this, PARTITIVE_ARTICLE}` *vaciar XXX {this, PARTITIVE_ARTICLE}* For example: `vaciar las direcciones {this, PARTITIVE_ARTICLE}`
Number and integer attributes For example: `int descuento`	*añadir {number} a el/la XXX {this, PARTITIVE_ARTICLE}* For example: `añadir {descuento} a el descuento {this, PARTITIVE_ARTICLE}` *sustraer {number} de el/la XXX {this, PARTITIVE_ARTICLE}* For example: `sustraer {descuento} de el descuento {this, PARTITIVE_ARTICLE}`

Working with external libraries

Automation Decision Services allows you to import and use external libraries in your decision services. External libraries allow you to enrich your data model and extend rule authoring with custom data types and functions from Java™ libraries.
External libraries provide the business vocabulary that is used to author rules. An external library depends on one or more Java archives. These Java archives can have dependencies on other Java archives, as shown in the following diagram:

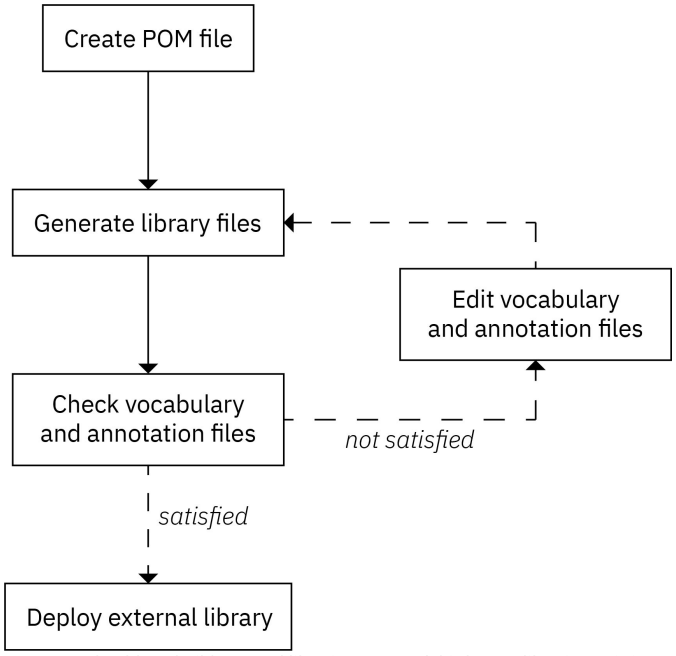


For a complete example of how to build and integrate an external library in a decision service, see the Using an external library detailed tutorial available on GitHub.

- **Building an external library**
  You create an external library that can be imported into Decision Designer. The process for creating a library is based on Maven.
- **Importing an external library into Decision Designer**
  Before you can use an external library in a decision service, you need to make it available at the decision automation level.
- **Updating an external library**
  When a new snapshot version of an external library is available in the Maven repository, you must update it in Decision Designer.
- **Using an external library in a decision service**
  You create a dependency between decision artifacts and the external library to be able to access the vocabulary the library contains.

Building an external library

You create an external library that can be imported into Decision Designer. The process for creating a library is based on Maven.
The following diagram describes the step-by-step process for creating an external library:



Note: You should not build external libraries on top of third-party libraries. It is instead recommended to use custom Java projects and classes to expose the business methods and APIs you need.

- **Creating a POM file**
  The Project Object Model (POM) file contains information about the external library and configuration details that are used by Maven to build the library.
- **Generating the external library files**
  You execute the Automation Decision Services plug-ins to generate the external library files.
- **Editing the generated files**
  If you are not satisfied with the generated library, you can annotate the source code or edit the vocabulary files.
- **Method restrictions**
  Methods contained in external libraries must meet some restrictions to be used in business rules.

Creating a POM file

The Project Object Model (POM) file contains information about the external library and configuration details that are used by Maven to build the library.
Automation Decision Services provides a Maven archetype that you can use to easily build POM files.

Note: For more information about setting up the Maven archetype plug-in in Automation Decision Services, see Setting up your build environment.
You can either run the archetype in interactive mode or in non-interactive mode. In the interactive mode, you are prompted to enter values for each required parameter after running the archetype. In the non-interactive mode, you must specify all the required parameters directly in the command line.

Important: You must use JDK 8.0+ and Maven 3 to run the Maven archetype.
When the POM file is generated, you need to update it to configure the Automation Decision Services plug-ins.

To see a complete sample of a POM file, see the Using an external library tutorial available on GitHub.

Generating a POM file in interactive mode

1. Open a command prompt and run the following command:

```
mvn archetype:generate \
-DarchetypeGroupId=com.ibm.decision \
-DarchetypeArtifactId=maven-archetype-external-library \
-DarchetypeVersion=11.0.8
```

2. When prompted, enter values for the following parameters:
   - `groupId`
   - `artifactId`
   - `version`
   - `dependencies`
   - `locales`
   - `importMethods`

Generating a POM file in non-interactive mode

Open a command prompt and run the following command:

```
mvn archetype:generate \
-DarchetypeGroupId=com.ibm.decision \
-DarchetypeArtifactId=maven-archetype-external-library \
-DarchetypeVersion=11.0.8 \
-DgroupId=<groupID> \
-DartifactId=<artifactId> \
-Dversion=<version> \
-Ddependencies=<dependencies> \
-Dlocales=<locales> \
-DimportMethods=<true|false> \
-B
```

Parameter details

Table 1. Archetype identifiers

Parameter	Details
`archetypeGroupId`	The `archetypeGroupId`, `archetypeArtifactId`, and `archetypeVersion` parameters define the coordinates of the archetype to be used as a template.
`archetypeArtifactId`	
`archetypeVersion`	

Table 2. External library identifiers

Parameter	Details
`groupId`	The `groupId`, `artifactId`, and `version` parameters define the Maven coordinates of the external library. Maven coordinates uniquely identify a project. These parameters will be used to import the external library into Decision Designer.
`artifactId`	
`version`	
`dependencies`	Your external library might depend on another project to build correctly, such as your own Java™ library or a third party library. The `dependencies` parameter lets you declare these dependencies. It accepts a list of values separated by commas. For example: `GroupId1:ArtifactId1:Version1,GroupId2:ArtifactId2:Version2,GroupId3:ArtifactId3:Version3`
`locales`	The `locales` parameter defines the locales that the external library supports. It accepts a list of values separated by commas. For example: `en_US, fr_FR` The following locales are supported in Decision Designer: • `en_US` - American English (default locale) • `de_DE` - German • `es_ES` - Spanish • `fr_FR` - French • `it_IT` - Italian • `ja_JP` - Japanese • `pt_BR` - Brazilian Portuguese • `zh_CN` - simplified Chinese
`importMethods`	When the `importMethods` parameter is set to true, Java class methods are automatically verbalized. When set to false, Java class methods are not verbalized.

Configuring the Automation Decision Services plug-ins

You manually configure the Automation Decision Services plug-ins in the <build> section of the generated external library.

The import-maven-plugin generates the business object model, external annotation files, and vocabulary file from the classes contained in the Java libraries listed in the <dependencies> section.

The import-maven-plugin can be configured by using the following parameters:

Parameter	Details
`skip`	Set to false by default. When set to true, the execution of the import-maven-plugin is skipped and the files are not generated.
`locales`	The `locales` parameter defines the locales that the external library supports. It accepts a list of values separated by commas. For example: `en_US, fr_FR`. For more information about supported locales, see `Table 2. External library identifiers` above.
`importMethods`	When the `importMethods` parameter is set to true, Java class methods are automatically verbalized. When set to false, Java class methods are not verbalized.

Parameter	Details
`filter`	Optional. This parameter allows you to manually define which classes and packages to import or ignore. By default, only classes and packages from direct dependencies are imported and verbalized. Direct dependencies are those defined in the POM file. If a filter is specified, all classes and packages from both direct and indirect dependencies are checked before being potentially imported and verbalized. You use the <includes> parameter to list the classes and packages to import, and the <excludes> parameter to list the ones to exclude. You can use either the fully qualified name of a class or the fully qualified name of a package followed by .* or .**, as shown in the following example: `<filter>` `  <includes>` `  <include>org.apache.commons.math3.complex.Complex</include>` `  </includes>` `  <excludes>` `  <exclude>org.apache.commons.**</exclude>` `  </excludes>` `</filter>` Use .* to select the content of a package, and .** to select a package and all its subpackages. Note: In addition to the **`<include>`** and **`<exclude>`** filters, you can also use the **`@NotBusiness`** annotation to narrow down the scope of the verbalization generation. For more information about the **`@NotBusiness`** annotation, see the Annotations API Java reference under Reference.
`referenceFolder`	Allows you to specify the name of the reference folder. The reference folder contains information that doesn't have to be generated again, such as external annotations and vocabulary files.

What to do next

When you have created the POM file, you are ready to generate the external library files.

Generating the external library files

You execute the Automation Decision Services plug-ins to generate the external library files.

Procedure

1. Open your command-line interface and navigate to the directory that contains the pom.xml file.
2. Run the **`mvn install`** command.
   The command runs the import-maven-plugin and the build-maven-plugin.

Results

The import-maven-plugin creates the following directories and files:

- A BOM directory that contains the model.b2xa and model.bom files. These files cannot be edited manually. However, you can annotate the model.bom file.
- A vocabulary file for each locale that the external library supports. You can annotate the vocabulary files, or manually edit them after moving them to the <referencefolder>/bom directory.
- A deployment directory. This directory and the files it contains do not need to be checked or edited.
- A resources directory that contains pure function annotations that are deduced from the code.
- A reference directory that is empty. You can use this directory to add external annotations and copy a vocabulary file that you would like to edit.

The build-maven-plugin creates the following files:

- A deployable JAR that contains the library files. After the JAR file is deployed, the external library is ready to be imported into Decision Designer.
- A markdown file that contains the external library documentation. The documentation shows each imported type, along with the verbalization of its constructors, attributes, and methods. The name of this file depends on the local of the external library, for example Complex_en_US.md.

What to do next

If you are satisfied with the generated external library, you can deploy it and import it into Decision Designer.

Use the **`mvn clean deploy`** command to deploy the external library into your archive repository.

If you are not satisfied with the generated files, you can:

- Manually edit the vocabulary files.
- Annotate the Java™ source code, if possible.
- Add external annotations.
- Add custom BOM types and members.

Editing the generated files

If you are not satisfied with the generated library, you can annotate the source code or edit the vocabulary files.
All the files you want to edit need to be located in the reference folder of the library. In the reference folder, you can:

- Copy any vocabulary file that you would like to edit.

- Copy an existing external annotation file, or create an external annotation file.
- Create BOM and BOM-to-XOM mapping extension files.

When you have finished editing the generated files, you need to run the **mvn install** command again, as explained in Generating the external library files.

- **Editing the vocabulary**
  You can edit the vocabulary files. This allows you to edit the default verbalization of vocabulary elements, and document vocabulary elements.
- **Adding annotations**
  You can annotate the source code or add an external annotation file. This allows you, for example, to declare members as pure or impure functions, or exclude members from being verbalized.
- **Adding custom BOM types and members**
  You can add members to a class to extend your external library.

Editing the vocabulary

You can edit the vocabulary files. This allows you to edit the default verbalization of vocabulary elements, and document vocabulary elements.

- **Vocabulary**
  The vocabulary comprises terms and phrases that are attached to the elements of the business object model (BOM). You use the vocabulary to create business rules and decision tables.
- **Editing terms**
  You can edit the default verbalization of a term.
- **Documenting terms**
  You can define a documentation property for each BOM element.

Vocabulary

The vocabulary comprises terms and phrases that are attached to the elements of the business object model (BOM). You use the vocabulary to create business rules and decision tables.
Automation Decision Services verbalizes BOM elements to make them visible in business rules and decision tables. When you build an external library, a vocabulary file is automatically created. This file contains a verbalization that is generated by default for BOM elements. This default verbalization can be customized.

When you create a BOM entry, you can choose to verbalize the business elements to create a vocabulary. A default verbalization is then applied to all attributes, getters, setters, and static references in the BOM entry. You can create a code-like verbalization of all other methods by updating the vocabulary file.

Business rules and decision tables use the vocabulary. If you do not verbalize a business element, that element is not part of the vocabulary, and therefore not visible from business rules and decision tables.

You can translate the vocabulary, and use several vocabularies in different languages on top of the same business object model. The locale of your decision service determines which translation is used.

- **Vocabulary elements**
  A vocabulary element is defined by the nature of its corresponding business element. A vocabulary element can be a business term, a phrase, or a constant.
- **Placeholders**
  Placeholders represent a return type, a declaring class, or a method argument. Placeholders can be filled automatically or manually depending on the placeholder type.
- **Default verbalization in English**
  Automation Decision Services applies a default verbalization to all business elements. You can add new phrases or edit the default verbalization directly in the vocabulary file.
- **Vocabulary properties**
  In the vocabulary file, you define properties, such as precedence and sortIndex to specify terms and phrases that verbalize the boot BOM.
- **Vocabulary errors and warnings**
  If you have vocabulary ambiguities, missing placeholders, or problems on business elements, the object model validator raises errors and warnings on terms and phrases at editing or build time.

Vocabulary elements

A vocabulary element is defined by the nature of its corresponding business element. A vocabulary element can be a business term, a phrase, or a constant.
Whether a vocabulary element is a business term, a phrase, or a constant depends on the nature of its corresponding business element.

Business term

A business term is a key concept handled in business rules. It is the verbalization of a class or nested class in the BOM.

For example, **customer** is a business term.

Vocabulary

```
# org.samples.Customer
org.samples.Customer#concept.label = customer
```

Rule example

```
definitions
   set 'standard customer' to a customer from <relation> ;
   set 'loyal customers' to all customers in <collection>

      where each customer is loyalty program member ;
if
   there are 1000 customers
```

Attention:
Do not use the characters " and **>** in term labels. They cause warnings and ambiguity problems when used in rules.

Expression

An expression is a phrase that associates two business elements. It can be the verbalization of:

- A method that has a non void return type
- The getter part of an attribute

For example, **`{name} of {this}`** is an expression phrase for the attribute **`Customer.name`**.

Vocabulary

```
org.samples.Customer.name#phrase.navigation = {name} of {this}
```

Rule example

```
if
   the name of customer contains "Smith"
```

You can also use predicate phrases in rules. A predicate phrase is a specific type of expression that verbalizes a non void method that returns a **`boolean`** or **`java.lang.Boolean`**. A predicate phrase has an implicit subject. For example, **`{this} is loyalty program member`** is a predicate phrase for the attribute **`Customer.loyaltyProgramMember`**.

Vocabulary

```
org.samples.Customer.loyaltyProgramMember#phrase.navigation = {this} is loyalty program member
```

Rule example

```
if
   customer is loyalty program member
```

Action

An action phrase applies an action to an object. It can be the verbalization of:

- A method that has a void return type
- The setter part, the adder part, the remover part, or the clearer part of an attribute

For example, **`in {this}, display the message : {0}`** is an action phrase for the method **`Session.displayMessage(String)`**.

Vocabulary

```
org.samples.Session.displayMessage(java.lang.String)#phrase.action = in {this}, display the message : {0}
```

Rule example

```
then
   in 'current session' , display the message : "Hello!" ;
```

Constant

A constant is the verbalization of the public static final attribute of a class with the same type as this class.

For example, **`Minivan`** is a constant for the public static final attribute **`CarGroup.Minivan`**.

Vocabulary

```
org.samples.CarGroup.Compact#instance.label = Compact
org.samples.CarGroup.Minivan#instance.label = Minivan
```

Rule example

```
if
   'my car group' is one of { Compact , Minivan }
```

Placeholders

Placeholders represent a return type, a declaring class, or a method argument. Placeholders can be filled automatically or manually depending on the placeholder type. Vocabulary phrase templates contain placeholders. Placeholders represent gaps in phrases that can be filled in either automatically or manually when editing rules. Placeholders are identified by curly brackets **`{}`**.

There are three types of placeholder:

- Subject placeholders, which are completed automatically
- **{this}** placeholders and argument placeholders, which you complete manually when editing rules

To set a different text for an argument placeholder, use the syntax **{x, "my text"}** in the phrase template.

Subject placeholder

Subject placeholders are shown in navigation phrases. The subject is a business term that you can edit in the vocabulary file. The subject corresponds to the return type of a member.

The subject placeholder is optional. When used in rules, the subject placeholder automatically takes the appropriate number and article for the context. In the placeholder text, the subject needs to be specified using the singular form, without article.

{this} placeholder

The **{this}** placeholder represents the declaring class of a member. You cannot use this placeholder in the verbalization of static members.

For example, for an attribute **lastName** of type **String** in class **Customer**, you can specify the following phrase template:

```
{last name} of {this}
```

Argument placeholder

Argument placeholders represent the arguments of methods. They are represented by the index of the argument.

For example, the action phrase **in {this}, display the message : {0}** is the verbalization of the method **Session.displayMessage(String)**. **{0}** is the argument placeholder for a **String**.

Vocabulary

```
org.samples.Session.displayMessage(java.lang.String)#phrase.action = in {this}, display the message : {0}
```

Example

```
then
   in 'current session' , display the message : <a string> ;
```

Default verbalization in English

Automation Decision Services applies a default verbalization to all business elements. You can add new phrases or edit the default verbalization directly in the vocabulary file.
The following table shows the default verbalization of vocabulary elements.

Table 1. Default verbalization

Java element	Default verbalization
Classes For example: `org.acme.Customer`	Class name in lowercase For example: `customer`
Constructors See Constructors For example: `@BeanConstructor` `Customer(String firstName, String lastName);`	*a new <type> [where <attribute> is <value> (, <attribute> is <value>)*]* For example: `a new customer where the first name is "John", the last name is "Doe"`
Getters (fields or methods of type **getXXX** without arguments) For example: `int getAge()` or `int age` or `finale String name;`	*{XXX} of {this}* For example: `{age} of {this}` or `{name} of {this}`

Java element	Default verbalization
Boolean getters (fields or methods of type **isXXX** without arguments that return a Boolean) For example: `boolean isAccepted()` or `boolean accepted`	Affirmative *{this} is XXX* For example: `{this} is accepted` Negative *{this} is not XXX* For example: `{this} is not accepted`
Setters (non final fields or void methods of type **setXXX** with one argument) For example: `void setDiscount(int discount)` or `int discount`	*set the XXX of {this} to {0}* For example: `set the discount of {this} to {discount}`
Boolean setters (non final fields or void methods of type **setXXX** with one argument of type Boolean) For example: `void setAccepted(boolean b)` or `boolean accepted`	Affirmative *{this} gets XXX* For example: `{this} gets accepted` Negative *{this} does not get XXX* For example: `{this} does not get accepted` Conditional *{this} gets XXX only if {XXX}* For example: `{this} gets accepted only if {accepted}` The *make it {0} that {this} is XXX* syntax is now deprecated.
Collections See Collections For example: `List<Address> getAddresses()` `@CollectionAttribute("addresses")` `void addAddress(Address address)` `@CollectionAttribute("addresses")` `void removeAddress(Address address)` `@CollectionAttribute("addresses")` `void clearAddresses()`	For example: `add {0} to the addresses of {this}` or `remove {0} from the addresses of {this}` or `clear the addresses of {this}`
Numbers (fields, or getter or setter methods) For example: `int getDiscount()` or `void setDiscount(int discount)` or `int discount`	For example: `add {number} to the discount of {this}` or `subtract {number} from the discount of {this}`
All other methods For example: `void displayMessage(String)`	Code-like verbalization, that is, the method signature, without subjects. For example: `{this}.displayMessage({0})`
Static references (public static final attribute of a class with same type as this class) For example: `Minivan`	Name of the static reference For example: `Minivan`

Vocabulary properties

In the vocabulary file, you define properties, such as precedence and sortIndex to specify terms and phrases that verbalize the boot BOM.
Vocabulary properties are defined in a vocabulary file (_*<locale>*.voc) that stores the verbalization data for the BOM. You have one vocabulary file per locale.

Property	Element	Value	Description
sortIndex	Static reference	Number	The sort Index property specifies the order in which the constants verbalized in the vocabulary are displayed in the rule editor. If the sorting is locale-specific, you must specify it in the vocabulary file. The first day of the week can vary depending on the country. In the following example, you set **`Day of week Monday`** in the first position: **`ilog.rules.brl.DayOfWeek.Monday#label = Monday`** **`ilog.rules.brl.DayOfWeek.Monday#sortIndex = 1`**
precedence	Sentence	Formatted string	You can set the precedence property on phrases (sentences) to help the rule language parser disambiguate expressions, such as arithmetic or Boolean expressions. The value of the precedence property is composed of three parts separated by a colon: • Description of the operator • Associativity • Level of precedence The following example indicates the precedence for the sentence **`plus`** on a number: **`ilog.rules.brl.Number.plus(ilog.rules.brl.Number)#sentence.operator = {this} + {0}`** **`ilog.rules.brl.Number.plus(ilog.rules.brl.Number)#sentence.operator.precedence = 1-1:left:100`** In this example, the precedence values have the following meaning: • **`1-1`**: The indices of the words that represent the operator in the text property. • **`left`**: The associativity of the operator. The operator is left associative. The possible values are **`nonassoc`**, **`left`** or **`right`**. • **`100`**: The level of priority.

Vocabulary errors and warnings

If you have vocabulary ambiguities, missing placeholders, or problems on business elements, the object model validator raises errors and warnings on terms and phrases at editing or build time.
Vocabulary elements, such as terms, phrases, articles and constants, might be made up of one or more words, called lexical units. These lexical units must comply with specific lexical rules. Vocabulary elements must be non ambiguous. You cannot verbalize two classes with the same term, and you cannot verbalize two members with the same phrase.

Terms

Terms must not conflict with the elements of the business rule language and the System BOM. For example, if a word in a term can be interpreted as a number, you get an error.

Ambiguity errors are reported locally when editing a vocabulary in the same vocabulary file, and reported globally at build time if the ambiguity comes from duplicate terms in two separate files.

Phrases

In phrases, you can use each type of placeholder only once.

You get a warning in the following circumstances:

- If the subject placeholder is not used in a navigation phrase
- If the **`{this}`** placeholder is missing

You can set a preference to ignore these warnings.

Errors in phrases can also come from their related business element:

- An action phrase for a read-only or final attribute
- A navigation phrase for a write-only attribute

Table 1. Examples of vocabulary errors

Message	Description
Character > is forbidden.	A vocabulary element uses a forbidden character. This error is reported when editing the vocabulary.
Character > in term for class balsample.Book is forbidden.	A vocabulary element uses a forbidden character. This error is reported at build time.
Term book is duplicated.	The term is already used for another class in the current vocabulary file.

Message	Description
Placeholder "author" is missing in "balsample.Book: author"	The subject placeholder (**author**) is missing in the phrase associated to the member **balsample.Book.author**.
Duplicate verbalization: [balsample.Book: author] and [balsample.Book: title]	The members **balsample.Book.author** and **balsample.Book.title** have the same verbalization.
Action phrase not allowed for "balsample.Book: author" (member is read-only)	The attribute is read-only and must not be verbalized as an action phrase.
Navigation phrase not allowed for "balsample.Book: author" (member is write-only)	The attribute is write-only and must not be verbalized as a navigation phrase.

Delimiters in numbers

Commas and spaces are used to separate parts of rules and expressions. In certain locales, they also serve as delimiters in numbers:

- A comma can be used to group three digits, as in **10,000**.
- A space can also be used to group three digits, as in **10 000**.
- A comma can be used as a decimal mark, as in **3,14159265**.

Follow these instructions to avoid problems when expressing numbers:

- When possible, do not use digits in verbalizations.
- When using digits, avoid space and comma separators between digits, especially if those characters are valid group delimiters or decimal markers in the locale of the vocabulary.
- If a verbalization must contain digits separated with spaces or commas, surround the numbers with double quotation marks, for example, **"3,14159265"**.

Editing terms

You can edit the default verbalization of a term.

About this task

For each data type and function that is defined in the vocabulary file, a default verbalization is applied. This default verbalization defines how data types and functions are referenced in business rules.

You edit terms to change, for example, the singular or plural form of a term.

Procedure

To edit a term:

1. Copy the vocabulary file that you want to edit in the <referencefolder>/bom folder of your external library.
2. Open the vocabulary file in a text editor.
3. Look for the term that you want to edit and change it as required.
   For example, if you want to change the plural form for the term **child**, you would update its definition as follows:

   ```
   @child#plural = children
   org.samples.Child#concept.label = child
   ```

   Note: Only the verbalization part of a vocabulary element is editable, that is the part on the right side of the equal sign.
4. When you have finished, save your changes.

Documenting terms

You can define a documentation property for each BOM element.

About this task

You document business terms to provide guidance to policy managers.

Procedure

To document a term:

1. Copy the vocabulary file that you want to edit in the <referencefolder>/bom folder of your external library.
2. Open the vocabulary file in a text editor.
3. Look for the term that you want to document and add the required description or comment using the following syntax:

   ```
   element.signature#documentation = <documentation>
   ```

   For example:

   ```
   org.samples.Customer.name#phrase.navigation.documentation = Represents the name of a customer

   org.samples.Session.DisplayMessage(java.lang.String)#phrase.action.documentation = Displays a message
   ```

4. When you have finished, save your changes.

Adding annotations

You can annotate the source code or add an external annotation file. This allows you, for example, to declare members as pure or impure functions, or exclude members from being verbalized.
To annotate your source code, you must declare a dependency to the annotations JAR file provided with Automation Decision Services. You can determine which version of the annotations JAR file to use by using the index.json file, as described in Deploying to a Maven repository.

To avoid conflicts when building the external library, the dependency should be declared as **<scope>provided</scope>**. The same requirement applies if you use Jackson annotations.

The updated POM file should look like the following example:

```
<dependency>
  <groupId>com.ibm.decision</groupId>
  <artifactId>annotations</artifactId>
  <version>version from index.json</version>
  <scope>provided</scope>
</dependency>
```

- **Annotations**
  You can use the Automation Decision Services annotations API to add annotations to your external libraries.
- **External annotation files**
  External annotations are specified in XML files, outside of the source code.

Annotations

You can use the Automation Decision Services annotations API to add annotations to your external libraries.
Note: Only a selection of annotions is presented in this section. The complete annotations API reference manual is available in the Reference section.

- **Collections**
  Automation Decision Services supports different patterns of collection usage.
- **Pure functions**
  The **@PureFunction** annotation is used to mark methods as pure.
- **Impure functions**
  The **@ImpureFunction** annotation is used to mark methods and constructors as impure.
- **Constructors**
  The **@BeanConstructor** annotation is used to declare constructors as valid.
- **Value types**
  The **@ValueType** annotation is used to declare value types.

Collections

Automation Decision Services supports different patterns of collection usage.

Managed collections

Managed collections are exposed as read-only by getter methods.

There are different methods that you can use to update the content of a managed collection: setter, adder, remover, and clearer. You need to use the **@CollectionAttribute** annotation with those last three methods, as shown in the following example:

```
public class Customer {
  private List<Address> addresses;
  public List<Address> getAddresses() {
    return addresses == null ? Collections.emptyList() : Collections.unmodifiableList(addresses);
  }
  public void setAddresses(List<Address> addresses) {
    if (addresses == null)
      this.addresses = null;
    else
      this.addresses = new ArrayList<>(addresses);
  }
  @CollectionAttribute("addresses")
  public void addAddress(Address address) {
    if (addresses == null)
      addresses = new ArrayList<>();
    addresses.add(address);
  }
  @CollectionAttribute("addresses")
  public void removeAddress(Address address) {
    if (addresses != null)
    addresses.remove(address);
  }
  @CollectionAttribute("addresses")
  public void clearAddresses() {
    addresses = null;
```

```
  }
}
```

Fully accessible collections

Fully accessible collections are directly returned by getter methods.

Methods are not necessary to update the content of a fully accessible collection. You must use the `@MutableCollection` annotation to access the whole range of verbalized ways available to update the content of an accessible collection, as shown in the following example:

```
public class Client {
  private List<Address> addresses = new ArrayList<>();
  @MutableCollection
  public List<Adress> getAdresses() {
    return addresses;
  }
}
```

Pure functions

The `@PureFunction` annotation is used to mark methods as pure.
A method is considered a pure function if:

- The method always returns the same outputs for the same inputs.
- The state at the start of the method call and the state at the end of the method call are the same: they contain the same set of objects, assign the same value to each attribute, and assign the same value to each variable.
  Two values are considered the same if:
  - `v == w` evaluates to true, where `v` and `w` are primitive values.
  - `v.equals(w)` evaluates to true, where `v` and `w` are two objects and are neither arrays nor collections.
  - Two collections contain the same elements in the same order.
  - Two arrays have the same size and contain the same elements in the same order.

When classes are imported to create the external library, the source code is reviewed to automatically detect pure functions. The result of this code review is stored in the resources/extannotations folder. However, complex pure functions may not be automatically detected and should be manually annotated. Pure functions can be declared in the source code, or in an external annotation file.

A method needs to be marked as a pure function to be called in the expression part of a rule, as explained in Method restrictions.

Impure functions

The `@ImpureFunction` annotation is used to mark methods and constructors as impure.
Functions that are not pure are considered impure. Impure functions can be called in the rule actions.

Impure functions can have the following characteristics:

- Have a side effect.
- Store or return a reference to a mutable value.
- Depend on external state, such as the system clock.

These characteristics are listed in an `@ImpurityKind` enumeration.

Some methods that are considered impure do not need the `@ImpureFunction` annotation because they follow a specific naming convention. That is the case of the setter, adder, remover, and clearer methods. For more information about these methods, see Method restrictions.

Constructors

The `@BeanConstructor` annotation is used to declare constructors as valid.
Because constructors initialize new objects, they cannot be considered as pure functions. The `@BeanConstructor` annotation allows you to declare constructors as valid so that they can be used in business rules and decision tables.

A `@BeanConstructor` adds an object to a set of existing objects, and initializes the attribute values of this new object. It must meet the following requirements:

- The attribute values of the existing objects must not be modified by the constructor call.
- Each variable must have the same value at the start and at the end of the constructor call.

The following example shows a `@BeanConstructor` that creates a new object `Borrower` and initializes its attribute values:

```
    @BeanConstructor
    public Borrower(String firstName,
                    String lastName,
                    Borrower spouse,
                    long yearlyIncome,
                    RiskLevel risk,
                    ZonedDateTime birthTime,
                    Address address) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.yearlyIncome = yearlyIncome;
```

```
        this.spouse = spouse;
        this.risk = risk;
        this.birthTime = birthTime;
        this.address = address;
    }
```

Value types

The **@ValueType** annotation is used to declare value types.
Value types must:

- Be immutable and final.
- Have implementations of hashcode and equals methods.

The value type fields must:

- Be final.
- Be of value type.
- Be provided when the object is created.

When an external library is imported into Decision Designer, an automatic check is performed on value types to ensure that they fulfill these requirements.

The following example shows a class, **Address**, annotated with **@ValueType**:

```
@ValueType
public final class Address {

    public final String city;
    public final String street;
    public final String number;

    @BeanConstructor
    @JsonCreator(mode = JsonCreator.Mode.PROPERTIES)
    public Address(@JsonProperty("city") String city,
                   @JsonProperty("street") String street,
                   @JsonProperty("number") String number) {
        this.city = city;
        this.street = street;
        this.number = number;
    }

    @NotVerbalized
    @Override
    public boolean equals(Object object) {
        if (object instanceof Address) {
            Address address = (Address) object;
            return (Objects.equals(this.city, address.city) &&
                    Objects.equals(this.street, address.street) &&
                    Objects.equals(this.number, address.number));
        }
        return false;
    }

    @NotVerbalized
    @Override
    public int hashCode() {
        return Objects.hash(city, street, number);
    }
}
```

Assigning **Address** as a type to a data type in your data model would allow you to check whether two objects have the same values. When checking whether two objects of type **Address** are equal in the rule language, this test would succeed if the two objects have the same **city**, **street**, and **number**.

External annotation files

External annotations are specified in XML files, outside of the source code.
When you generate the external library files, pure function annotations can be automatically created in the resources/extannotations folder. You can have one external annotation file per package.

You can copy an external annotation file into the reference folder of the library to edit it, or create a new external annotation file in this same folder.

External annotation file format

External annotation files contain a **<root>** element and at least one **<item>** element, as shown in the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
  <item name="org.apache.commons.math3.complex.Complex org.apache.commons.math3.complex.ComplexField getField()">
    <annotation name="ilog.rules.bom.annotations.NotBusiness"/>
```

```
  </item>
</root>
```

<item> element

The **<item>** element can represent a class, a class member (field, method, or constructor), or a parameter. It contains a **name** attribute.

The following table describes the format to use for each type of item:

Item type	name format	Example
Class	**name="*<class name>*"** The class name is the name of the class as returned by **Class.getName()**. The separator for a nested class is **$**.	**<item name="stuff.Customer">**
Field	**name="*<class name>* *<canonical name of the field type>* *<field name>*"** The canonical name of a class is the name of the class as returned by **Class.getCanonicalName()**. The separator for a nested class is **.**.	**<item name="stuff.Customer java.lang.String name">**
Constructor	**name="*<class name>* *<short class name>(<canonical name of the parameter types>)*"** The short class name is the class name without the namespace. Each parameter type is separated by a comma and a space (', ').	**<item name="stuff.Customer Customer(java.lang.String)">**
Method	**name="*<class name>* *<method name>(<canonical name of the parameter types>)*"** Each parameter type is separated by a comma and a space (', ').	**<item name="stuff.Customer void addAge(int, java.lang.String)">**
Constructor parameter	**name="*<constructor information>* *<number of parameters>*"**	**<item name="stuff.Customer Customer(java.lang.String) 0">**
Method parameter	**name="*<method information>* *<number of parameters>*"**	**<item name="stuff.Customer void addAge(int,java.lang.String) 1">**

<annotation> element

The **<annotation>** element represents an annotation. It contains a **name** attribute. The value of the **name** attribute is the canonical name of the annotation type.

You can define annotations without parameters (or that use default parameters), annotations with parameters, and nested annotations.

Annotation type	Example	Comments
Annotation without parameters	**<annotation name="ilog.rules.bom.annotations.NotBusiness"/>**	
Annotation with parameters	**<annotation name='ilog.rules.bom.annotations.BoundedIntDomain'> <val name="min" val="0"/> <val name="max" val="32"/> </annotation>**	Each **<val>** element represents a parameter. The **name** attribute specifies the name of the parameter. This attribute is optional if the parameter name is **value**.
Nested annotation	**<annotation name="ilog.rules.bom.annotations.VocabularyLabels"> <val> <annotation name='ilog.rules.bom.annotations.VocabularyLabel'> <val val="le vv"/> <val name="locale" val="fr"/> </annotation> <annotation name='ilog.rules.bom.annotations.VocabularyLabel'> <val val="the vv"/> <val name="locale" val="en"/> </annotation> </val> </annotation>**	The **<val>** parent element contains a list of **<annotations>** elements.

Adding custom BOM types and members

You can add members to a class to extend your external library.

About this task

Adding members to a class requires the manual creation of two files: a BOM file and a BOM-to-XOM mapping file.

- A BOM file (**extension.bom**) is a text file that defines part of the business object model (BOM).
- A BOM-to-XOM mapping file (**extension.b2xa**) is an XML file that stores the mapping between the BOM and the execution object model (XOM). The XOM is the model against which you run rules.

In the BOM-to-XOM mapping file, you can provide the implementation for BOM methods, and attribute getters and setters. You use the Advanced Rule Language (ARL) for BOM-to-XOM mapping. For more information about writing rules using the ARL, see ARL syntax.

What to do next

The following topics provide guidance for mapping business elements to the XOM:

- **Defining custom members and their implementation**
  You can map different business elements to the XOM. These elements include attributes and method calls.
- **Understanding BOM-to-XOM mapping**
  When you build an external library, a BOM is automatically created from the Java™ compiled classes contained in the XOM.

Defining custom members and their implementation

You can map different business elements to the XOM. These elements include attributes and method calls.
To define and implement custom members:

1. Create a BOM file named extension.bom in the reference folder of the external library.
2. Add the following boilerplate code to the BOM file:

```
#loadGetterSetterAsProperties
package <package_name>;
```

3. Declare a class inside the BOM file.
   You can copy a class declaration from the BOM file that was automatically generated in the external library.
4. Declare the member variable.
5. Create a BOM-to-XOM mapping file named extension.b2xa in the reference folder of the external library.
6. Add the following boilerplate code to the BOM-to-XOM file:

```
<b2x:translation xmlns:b2x="http://schemas.ilog.com/JRules/1.3/Translation"
                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                 xsi:schemaLocation="http://schemas.ilog.com/JRules/1.3/Translation ilog/rules/schemas/1_3/b2x.xsd">

</b2x:translation>
```

7. Add the member implementation inside the BOM-to-XOM file.

The following sections provide specific details about adding computed attributes and custom methods.

Adding a computed attribute

You can add a BOM read-only attribute and define how it is computed.

For example, you have a class that is named **ShoppingCart** in the XOM, and the class contains two methods that return the number of items in the shopping cart and the total value of the items:

```
// XOM class
public class ShoppingCart  {
  public double getValue() ...
  public int getNumberOfItems() ...
}
```

To add a computed attribute:

1. Open the extension.bom file and declare the class and member variable as shown in the following example:

```
#loadGetterSetterAsProperties
package ads.samples.externalLibrary;

class ShoppingCart
        extends com.ibm.ia.StringProvider
{
    readonly double averageItemPrice;
}
```

   The variable must always be prefixed with the **readonly** modifier.
2. Open the extension.b2xa file and add the member implementation as shown in the following example:

```
<b2x:translation xmlns:b2x="http://schemas.ilog.com/JRules/1.3/Translation"
                 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                 xsi:schemaLocation="http://schemas.ilog.com/JRules/1.3/Translation ilog/rules/schemas/1_3/b2x.xsd">
    <class>
        <businessName>ads.samples.externalLibrary.ShoppingCart</businessName>
        <executionName>ads.samples.externalLibrary.ShoppingCart</executionName>
        <attribute>
            <name>averageItemPrice</name>
            <getter language="arl"><![CDATA[
                if (this.getNumberOfItems() == 0 )
                    return 0;
                return this.getValue()/this.getNumberOfItems();
                ]]></getter>
        </attribute>
    </class>
</b2x:translation>
```

Where:

- The name of the attribute is the same as the one used in the extension.bom file.
- The body of the getter returns a value.

The *businessName* and *executionName* values can be copied directly from the model.bom file.

Adding a custom method

You can add a custom method and define how it is implemented.

For example, you have a class **Customer** in the XOM, and the class contains a method that returns the age of the customer:

```
// XOM class
public class Customer {
  public int getAge() ...
}
```

To add a custom method:

1. Open the extension.bom file and declare the class and member variable as shown in the following example:

   ```
   #loadGetterSetterAsProperties
   package ads.samples.externalLibrary;

   class Customer
           extends com.ibm.ia.StringProvider
   {
       boolean isOlderThan(int arg)
                   #pureFunction;
   }
   ```

   The variable must always be annotated with the **#pureFunction;** tag.
2. Open the extension.b2xa file and add the member implementation as shown in the following example:

   ```
   <b2x:translation xmlns:b2x="http://schemas.ilog.com/JRules/1.3/Translation"
                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                    xsi:schemaLocation="http://schemas.ilog.com/JRules/1.3/Translation ilog/rules/schemas/1_3/b2x.xsd">
       <class>
           <businessName>ads.samples.externalLibrary.Customer</businessName>
           <executionName>ads.samples.externalLibrary.Customer</executionName>
           <method>
               <name>isOlderThan</name>
               <parameter type="int"/>
               <body language="arl"><![CDATA[
                return this.getAge() > arg;
               ]]></body>
           </method>
       </class>
   </b2x:translation>
   ```

   The name and type of the parameter must be the same as the ones you used in the extension.bom file.

Understanding BOM-to-XOM mapping

When you build an external library, a BOM is automatically created from the Java™ compiled classes contained in the XOM.

Table 1. Business elements originating from a Java XOM

Java XOM element	Becomes element in the BOM...
Nongeneric public class	Class with the same name
Class that implements the **java.util.Collection** interface For example: **MyCollection implements java.util.Collection**	Class with a collection domain to be recognized as a collection when verbalized For example: **public class MyCollection implements java.util.Collection { domain 0, * ; }**
Public constructor	Constructor with the same parameters
Public attribute	Attribute with the same name and return type
Final attribute	Read-only attribute with the same name and return type
Public static final attributes whose types are the current class. For example: **public class Color { private Color(String name) {…} public static final Color red = new Color("red"); public static final Color blue = new Color("blue"); public static final Color green = new Color("green"); }**	Enumerated domain of static references of the class For example: **public class Color { domain { static red, static blue, static green } public static final readonly Color red; public static final readonly Color blue; public static final readonly Color green; }**

Java XOM element	Becomes element in the BOM...
Public method that does not follow the JavaBeans convention for property accessors (`void setFoo(PropertyType value)` and `PropertyType getFoo()`) For example: `public interface Customer` `{` `public boolean register(java.sql.Connection db);` `}`	Method with equivalent parameters For example: `public interface Customer` `{` `public abstract boolean register(java.sql.Connection arg);` `}`
Public method that follows the JavaBeans convention, with a get method and no set method For example: `public interface Customer` `{` `public int getAge();` `}`	Read-only attribute For example: `public interface Customer` `{` `public readonly int age;` `}`
Public method that follows the JavaBeans convention, with only a set method For example: `public interface Customer {` `public void setBirthDate(Date date);` `}`	Write-only attribute For example: `public interface Customer` `{` `public writeonly java.util.Date birthDate;`
Public method that follows the JavaBeans convention, with both a get method and a set method For example: `public interface Customer {` `public String getName();` `public void setName(String name);` `}`	Attribute For example: `public interface Customer` `{` `public java.lang.String name;` `}`

Method restrictions

Methods contained in external libraries must meet some restrictions to be used in business rules.
The rule language supports method calls as expressions and actions.

Expression
: Expressions can be used in both rule conditions and rule actions.

  Expressions can consist of a call to a method or a call to a constructor. If the expression consists of a call to a method, the method must be a pure function.

Action
: Actions are used in rule actions only. Actions modify the value of variables and attributes.

  If the action consists of a call to a method, the method must be a modifier (setter, adder, remover, or clearer).

Expressions and actions need to meet the restrictions that are described in this topic to be properly evaluated.

- `get` method
- `set` method
- `add` method
- `remove` method
- `clear` method

`get` method

A getter returns the value of an attribute for a class or object.

It must start with the `get` prefix. It can also start with the `is` prefix with Boolean values.

A `get` method must be pure. It can be a static method or an instance method.

`set` method

A setter assigns or updates the value of an attribute for a class or object.

It must start with the `set` prefix and has one parameter, a value.

A `set` method can be a static method or an instance method. It must meet the following requirements:

- At the end of the method call, the value of the attribute must equal the given value.
- The other attributes of the class or object must have the same value at the start and at the end of the method call.
- The attributes of other classes or objects must have the same value at the start and at the end of the method call.
- Each variable must have the same value at the start and at the end of the method call.

`add` method

An adder adds a value to a collection-valued attribute for a class or object.

It must start with the **add** prefix and has one parameter, a value.

An **add** method can be a static method or an instance method. It must meet the following requirements:

- At the end of the method call, the collection must contain the new value in addition to the values it already contained before the method call.
- The other attributes of the class or object must have the same value at the start and at the end of the method call.
- The attributes of other classes or objects must have the same value at the start and at the end of the method call.
- Each variable must have the same value at the start and at the end of the method call.

`remove` method

A remover removes a value from a collection-valued attribute for a class or object.

It must start with the **remove** prefix and has one parameter, a value.

A **remove** method can be a static method or an instance method. It must meet the following requirements:

- At the end of the method call, the collection must contain the same values that it contained before the method call, except for the value that was removed.
- The other attributes of the class or object must have the same value at the start and at the end of the method call.
- The attributes of other classes or objects must have the same value at the start and at the end of the method call.
- Each variable must have the same value at the start and at the end of the method call.

`clear` method

A clearer clears all the values from a collection-valued attribute for a class or object.

It must start with the **clear** prefix. It does not have any parameter.

A **clear** method can be a static method or an instance method. It must meet the following requirements:

- At the end of the method call, the collection must contain no value.
- The other attributes of the class or object must have the same value at the start and at the end of the method call.
- The attributes of other classes or objects must have the same value at the start and at the end of the method call.
- Each variable must have the same value at the start and at the end of the method call.

The **add**, **remove**, and **clear** methods must be annotated with **@CollectionAttribute**, as explained in Collections.

Importing an external library into Decision Designer

Before you can use an external library in a decision service, you need to make it available at the decision automation level.

Before you begin

Before you can import an external library, the administrator needs to configure credentials for the Maven repository where your external libraries are stored. For more information, see Configuring credentials for a Maven repository manager.

Procedure

1. Click the Settings icon in the menu bar at the top of your decision automation to open the Settings menu.
2. Open the External libraries tab and click Import .
3. Enter the coordinates of the external library that you want to import into the decision automation: its groupId, artifactId, and version.
   These are declared in the **<dependencies>** section of the project POM file.
   Note: You can import both specific and snapshot versions of external libraries.
4. Click OK.

Results

The external library is now available in the list of libraries you imported. Open the library to access the list of custom functions and data types it contains.

What to do next

You can now create a dependency between the external library and the decision service in which you want to use its vocabulary.

Updating an external library

When a new snapshot version of an external library is available in the Maven repository, you must update it in Decision Designer.

About this task

When you update an external library, the snapshot version and its dependencies are downloaded. After you update it, the snapshot version of the external library is used when a decision service is executed.

Procedure

1. Click the Settings icon in the menu bar at the top of your decision automation page to open the Settings menu.
2. Open the External libraries tab and click Update.
3. Click Update again in the Update external library window.

What to do next

You might need to update your data models, decision models, and predictive models that use the updated external library depending on what is updated in the snapshot version.

Using an external library in a decision service

You create a dependency between decision artifacts and the external library to be able to access the vocabulary the library contains.

About this task

The vocabulary defined in your external library can be used:

- In data models, to define the type associated with an attribute.
- In decision models and predictive models, to define the output types of diagram nodes.
- In task models, to define the type of variables in variable sets.
- In decision models, task models, and predictive models to write rules.

Procedure

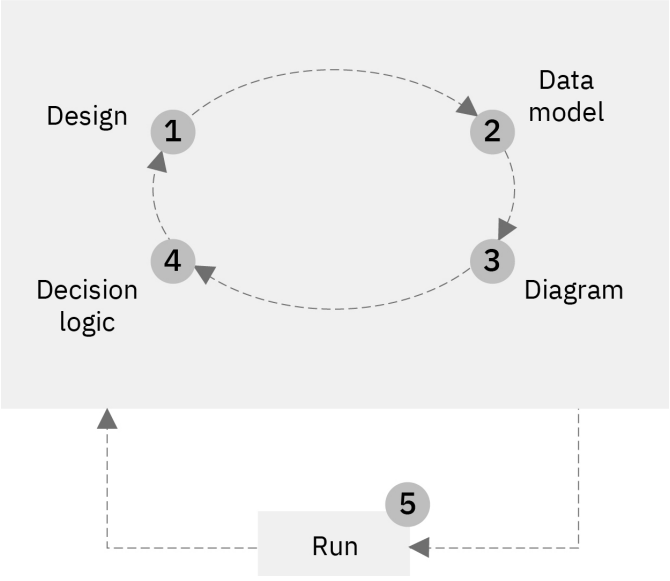
1. From your decision artifact, open the Dependencies tab.
2. Click Add +.
   The list of decision artifacts and external libraries available to be used as dependencies is displayed. For an external library to be available, it must be in the same locale as the decision service you are working in. If the external library you want to use is not in the list, see Importing an external library into Decision Designer.
3. Select the external library that you want to use and click Add.

What to do next

You can now use the custom functions and data types defined in the external library in your decision artifacts. You can access the list of custom functions and data types at any time by opening the external library from your decision service.

Building decision models

Decision models allow you to capture business decisions in a decision diagram, an easily understood representation that includes the data that is needed to make these decisions.
The modeling process is made of five steps that are performed iteratively until you have a complete and coherent model:



1. Design the decision that you want to automate. Think about how you can break your decision into smaller decisions to make your model simpler to manage and identify the data that is needed to make this decision.
2. Create a data model and populate it with data types to represent the data that is necessary to make your decision.
3. Create a decision model and build a diagram that represents your decision, its subdecisions, and the data that influences them.
4. Add the decision logic to define how each decision is made by creating business rules and decision tables.
5. Create test data sets to run your decision and make sure it works as expected.

- **Designing a decision**
  Successful decision modeling projects begin by fleshing out all the elements that compose your decision: the diagram, the data model, and the decision logic.
- **Creating a decision diagram**
  To implement your decision, you create a decision diagram.

Designing a decision

Successful decision modeling projects begin by fleshing out all the elements that compose your decision: the diagram, the data model, and the decision logic.
Diagrams provide an abstract, high-level representation of how decisions and the data that is required to make these decisions are structured and related to each other. Creating diagrams is an iterative process where you decompose the decision that you want to make. There are two approaches to decomposing decisions:

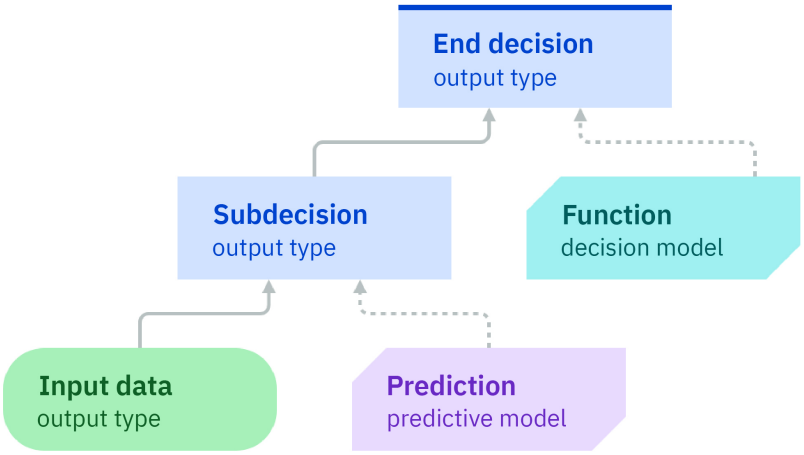
Decision-driven design
: The decision-driven approach works from an outcome that is positive to the business. You start with the decision that needs to be made and ask what data is needed for this decision. For example, a seller might want to determine the amount of discount they can offer a loyal customer. Starting from the decision "how much discount to offer", they then add node-by-node all of the data and other influencing decisions that they need to automate this decision.

Data-driven design
: The data-driven approach starts with the data that you have and works out what part of the business is affected by this data. It asks what are the business decisions that the data influences. For example, a business selling fresh produce might have some data on the weather. Starting from the weather data, they might create a decision model to help them decide whether it is worth setting up their stand at an open market given fewer people are likely to go if it rains.

Both of these approaches are valid but the decision-driven approach is the recommended one: by breaking the end decision into smaller decisions, you end up with a simpler model that is easier to understand.

Diagrams provide an abstract, high-level representation of how decisions and the data that is required to make these decisions are structured and related to each other. They are composed of a set of nodes that are used as building blocks to represent decisions in a graphical way:

- Decision nodes represent the end decision, that is the decision that you want to automate, and the subdecisions that the end decision depends on.
- Data nodes represent the data that is needed to make a decision.
- Function nodes encapsulate computations from other decision models.
- Prediction nodes encapsulate predictions that you can call directly from your decision model.



The decision logic is captured in the form of business rules and decision tables:

- Business rules are if-then statements that are written with a syntax close to natural language that can be readily understood by business experts. If-then statements associate a condition (if) with an action (then). When the condition is met, the rule action is triggered. You create business rules by using a wizard that lets you select the criteria that you want the rule to cover. You write rules by using a guided editor that lets you assemble statements and add missing variables.
- Decision tables represent decision logic as a table where each row corresponds to a business rule. You create decision tables by using a wizard that lets you select the conditions that you want to use in your rules. You use the special facilities that are provided by the decision table editor to work with decision tables.

Creating a decision diagram

To implement your decision, you create a decision diagram.
The first step when you create a decision model is to implement the architecture of your decision by building a dependency diagram. A diagram is composed of the following elements:

Decision nodes
: Decision nodes represent the decision that you want to automate and its supporting subdecisions. They contain a decision logic that defines how each decision is made.

Input data nodes
: Input data nodes represent the data that decisions depend on to determine their output. They are associated with a data type that defines the kind of data that they can hold.

Function nodes
: Function nodes represent values that are computed from other decision models.

Prediction nodes
: Prediction nodes represent values that are computed from predictive models.

Links
: Links represent:

  - The dependency relationship between the end decision and the subdecisions and input data it depends on.
  - The invocation of a function node or prediction node by a decision node.

For a complete example of how to work with diagrams, see the Training sample available on GitHub. Alternatively, you can import the Training sample from the Samples library in Decision Designer. For more information about the Samples library, see Creating decision services.

- **Defining input**
  You create input data nodes to represent the data that is needed to make a decision.
- **Adding decisions**
  You create decision nodes to represent decisions and subdecisions.
- **Calling other models**
  You can integrate the output of decision models and predictive models in other decision models that are stored in the same decision service.

Defining input

You create input data nodes to represent the data that is needed to make a decision.

- **Creating an input data node**
  Create an input data node and define the type of data it can hold.
- **Defining a default value**
  You can set a default value on an input data node to provide a fallback value if no value is available.

Creating an input data node

Create an input data node and define the type of data it can hold.

Procedure

1. Click the Add input icon  in the diagram toolbar.
2. Enter the required information to define your node:

Table 1. Input data node fields

View	Field	Description
Details	Node name	Enter a unique name for the data node.
	Input type	Associate the data with a type. The data type represents the kind of value that the node can store. For more information about data types, see Creating a data model.
	Description	Optionally provide a description for the data node.
	Output variable name	You can set a different output name for your input data. This output name can be used as a variable in the rules and decision tables. By default, the Same as data node name option is selected.
Default value		Optionally add a default value to your data. This default value is used for validation if no input is provided for the data node. For more information about default values, see Defining a default value.

Defining a default value

You can set a default value on an input data node to provide a fallback value if no value is available.

About this task

You can set default values for the following purposes:

- The data that is handled by the node is optional, and might not always be provided by the client application. For example, if the customer's country is US for almost all transactions, the client application might provide this information only if the country is different.
- For convenience, the validation feature uses the default value if no value is provided in the test data. Default values can reduce your data entry in the data sets that you create.

The following examples show how to set the default value on a data node:

- On a built-in data type:

```
set gender to "Female";
```

- On a custom data type:

```
set loan to a new loan where
    the amount is 100000,
    the monthly repayment is 688;
```

Procedure

To create a default value for an input data node:

1. Open the Default value tab of the node.
2. Click the + button, and select Default rule.
3. Define the default rule in the rule editor.

Adding decisions

You create decision nodes to represent decisions and subdecisions.

- **Creating a decision node**
  Create a decision node.
- **Adding the decision logic**
  You model the decision logic by defining how each decision's output is derived from its inputs.

Creating a decision node

Create a decision node.

Procedure

1. Click the Add decision icon  in the diagram toolbar.
2. Enter the required information to define your node:

Table 1. Decision node fields

View	Field	Description
Details	Node name	Enter a unique name for the decision node.
	Output type	Associate the decision with a type. The type assigned to a decision determines its output type. It can either be a simple built-in type, such as Boolean if the output of the decision is true or false, or a custom type. If you check the Ouput is a list option, the decision can have multiple outputs. For more information about output types, see Creating a data model.
	Description	Optionally provide a description for the decision node.
	Output variable name	You can set a different output name for your decision. This output name can be used as a variable in the rules and decision tables. By default, the Same as decision node name option is selected.
Logic		Add a business rule. A business rule is an if-then statement where the **`if`** is the condition and the **`then`** is the action of the rule. If the condition is met, the action is performed. For more information about business rules, see Working with business rules.
		Add a decision table. A decision table is a tabular representation of related rules, where each row forms a rule and each column represents one condition or one action. For more information about decision tables, see Working with decision tables.
		Define a default value. A default value is a fallback value that is used when no decision is made by the decision logic. For more information about default values, see Defining a default value.
	Rules are applied in sequence	Open the drop-down menu to select an interaction policy to apply to the decision tables and business rules. The interaction policy specifies how the result of a decision is evaluated when more than one rule matches. By default, the sequential policy is selected. For more information about interaction policies, see Choosing an interaction policy.

Adding the decision logic

You model the decision logic by defining how each decision's output is derived from its inputs.
Automation Decision Services provides two ways to define the decision logic of node: business rules and decision tables.

Business rules are conditional statements. Basic business rules use an if-then statement to state what action to perform when specific conditions are met. In the following example, the rule outputs a message encouraging people to stay home if there is a storm alert:

```
if
     Weather is storm alert
then
    set decision to "It would be wise to stay home. There is a storm alert." ;
```

More complex business rules can consist of up to four parts: definitions, if, then, and else.

Decision tables allow you to model complex logic. Their tabular layout helps you effectively document all the possible conditions and results of a decision node. In the following example, the decision table consists of two inputs, the condition columns **`Rain forecast`** and **`Temperature`**, and an output column, **`Weather advice`**. It gives a recommendation on what to bring depending on the weather: a coat, some water, or an umbrella.

Rain forecast		Temperature	Weather advice
0	20	cold	Cold day! Take a coat.
0	20	warm	Warm day! Enjoy.
0	20	hot	Hot day! Grab some water.
20	80		Cloudy day! Think of an umbrella.
80	100		Rainy day! Take an umbrella.

You can use input data and the output of subdecisions while defining the logic of a decision node.

- **Creating a business rule**
  You use the rule editor to create and work with rules and use the special facilities provided.
- **Creating a decision table**
  You use the decision table editor to create and work with decision tables and use the special facilities provided.
- **Defining a default value**
  You can set a default value on a decision node to provide a fallback value if no value is available.
- **Choosing an interaction policy**
  For each decision node, you select a rule interaction policy that applies to all the rules and decision tables in this decision node. Interaction policies define how rules interact with each other by governing their order of execution, and ensure that the decision logic produces a consistent result.

Creating a business rule

You use the rule editor to create and work with rules and use the special facilities provided.

About this task

Each rule exists in the context of a specific decision and this decision might require more than one rule to detail its decision logic. A rule uses the vocabulary of the input data that the decision is based on, and selects a value for the decision output.

Procedure

1. In a diagram, click the decision node that you want to edit and open the Logic tab.
2. Click the + button and select Business rule.
   The rule creation wizard opens.
3. Enter a name for the rule.
4. Optional: Select the criteria that you want to use in the rule. The criteria that you select define the rule conditions.
   The list of criteria depends on the inputs of the decision node.
   Note: Selecting criteria provides you with a more complete draft of the rule, but you can edit it entirely after the rule is created.
5. Click Create.

Results

You can now use the rule editor to complete the different parts of your rule.

For more information about how to build rules, see Working with business rules. The rule language reference manual is available in the Rule language section.

Example

For example, consider a decision that gives a discount that depends on the category of a customer and how much he spent.

When you create a rule, the rule template in the preview only contains the action that sets the decision, with the construct **`set decision to`**. You can generate a rule that is closer to the decision logic that you want to express by selecting the input data that the rule needs to make the decision. Here the input data is the customer category and the value of the shopping cart of the customer. When they are selected in the list of criteria, an **`if`** part with conditions that use these inputs is added to the rule template.

The conditions contain placeholders where you will indicate actual values. The types of these values and the default constructs depend on the type of data that you defined in your data model and on the node types in the diagram.

Similarly, the output of the decision is the amount of the discount. Therefore, the placeholder for the value of the decision indicates that is a number, based on the output type of the decision.

Now, let's say that the actual decision logic consists in giving a discount of 10 to Gold customer whose cart value is over 100. In the rule editor, the rule template can be modified to represent this decision logic. In the following example, the construct to evaluate the shopping cart value has been replaced with one that is more relevant to this case, and the placeholders for the shopping cart, category, and discount values have been filled with specific values.

```
if
   'shopping cart value' is more than 100
   and 'customer category' is "Gold"
then
   set decision to 10;
```

Creating a decision table

You use the decision table editor to create and work with decision tables and use the special facilities provided.

About this task

Decision tables comprise rows and columns, and each row corresponds to a rule. They are used to represent in tabular form all possible situations that a business decision may encounter, and to specify which action to take in each of these situations.

Procedure

1. In a diagram, click the decision node that you want to edit and open the Logic tab.
2. Click the + button and select Decision table.
   The decision table creation wizard opens.
3. Enter a name for the decision table.
4. Select the criteria that you want to use in the decision table. The criteria that you select define the condition columns of the table.
   The list of criteria depends on the inputs of the decision node.
   Note: Selecting criteria provides you with a more complete draft of the table, but you can edit it entirely after the table is created.
5. Click Create.

Results

You can now use the decision table editor to define the condition and action columns for the decision table, and to specify values for each row, that is, for each rule.

For more information about how to build decision tables, see Working with decision tables. The rule language reference manual is available in the Rule language section.

Example

For example, consider a decision that defines the minimum age requirement for renting a car depending on the state where the rental originates.

When you create a table, the table template in the preview only contains an action column. You can generate a table that is closer to the decision logic that you want to express by selecting the input data that the table needs to make the decision.

Here, the input data is the age of the driver and the state where the rental originates. When they are selected in the list of criteria, condition columns are automatically added to the table template.

You can drag column headers to reorganize the order of the condition columns.

Defining a default value

You can set a default value on a decision node to provide a fallback value if no value is available.

About this task

When you author the logic of a decision node, you create the rules that are required to reach a decision for each possible combination of input values. In practice, it is usually not possible to consider all possible cases, even for a simple decision.

An easy way to make the decision logic complete consists in specifying a default value for the decision. The default value applies if no other rule made a decision, and is always the last rule to execute.

This default value is just a fallback measure and should not prevent you from completing the decision logic in an informed and deliberate way.

You specify the default value as an action of the form:

```
set decision to <value>
```

If the node uses multiple values enabled by lists, use the form:

```
add <value1> to decision
add <value2> to decision
...
```

Procedure

To create a default value for a decision node:

1. Open the Logic tab of the node.
2. Click the + button, and select Default rule.
3. Define the default rule in the rule editor.
   For more information about how to build rules, see Working with business rules. The rule language reference manual is available in the Rule language section.

Choosing an interaction policy

For each decision node, you select a rule interaction policy that applies to all the rules and decision tables in this decision node. Interaction policies define how rules interact with each other by governing their order of execution, and ensure that the decision logic produces a consistent result.

Sequential policy

The sequential policy is the default policy, where rules execute in the order in which they are listed in a decision node. A rule can modify or overwrite decisions that were previously made by other rules. As a consequence, several rules might apply for a decision.

The first rule in a decision node can be used to initialize the decision. To use the first rule as an initialization rule, omit the condition part of the rule.

First rule applies policy

Rules execute in the order in which they are listed in the decision node. As soon as a rule applies and makes a decision, this decision is final for the impacted attributes, and the execution in the decision node stops. If a rule might apply several times, and is the first applicable rule, it will only apply once. Therefore, do not use this policy if the decision is a list that results from multiple instances of one or more rules.

Smallest and greatest value policies

All applicable rules execute and the order of execution does not matter. If you use the smallest value policy, the value of the decision is set to the minimum value available among the values that are obtained by the applicable rules. If you use the greatest value policy, it is set to the maximum value available.

You can use these policies for decisions of type number and integer, and that use the **`set decision to`** construct.

Collect policy

All applicable rules execute. If you use the collect policy, the values of all applicable rules are collected and returned as a list. The execution order of rules does not influence which values are collected, but it impacts the order in which the collected values appear in the list.

You can use the collect policy for decisions that are multi-valued, and that use the **`add ... to decision`** construct.

Sum policy

All applicable rules execute and the order of execution does not matter. If you use the sum policy, the values of all applicable rules are summed up.

You can use the sum policy for decisions of type number and integer, and that use the **`add ... to decision`** construct.

Important: If no rule is applicable, the value of the decision is null.
For a complete example of how to work with interaction policies, see the Training sample available on GitHub. Alternatively, you can import the Training sample from the Samples library in Decision Designer. For more information about the Samples library, see Creating decision services.

Calling other models

You can integrate the output of decision models and predictive models in other decision models that are stored in the same decision service.

Adding a function node

Function nodes allow you to encapsulate the output of a decision model, that you can reuse in other decision models. In a decision model, you create a dependency between a decision node and a function node so that the decision node makes a decision based on the output of the function node.

To call another decision model:

1. Click the Add function icon  in the diagram toolbar.
2. Open the drop-down menu and select the decision model that you want to use.
3. In the diagram, create a dependency between this node and the decision node that will use the result of the function: hover over the function node, click the Connect to another node button  , and select the decision node.
4. Write the decision logic of the decision node using the output of the function node. You can use the invocation example available in the Details tab of the function node.

When you write the decision logic of the decision node, use the output of the function node to make decisions. You can use the invocation example available in the Details tab of the function node to write the decision logic.

Adding a prediction node

Predictive models are meant to be reused in decision models to make decisions based on predictions. When you design the decision model, you must think about the part of the decision logic where you want to use the predictive model. Then, you configure the corresponding decision node to use the predictive model.

To call a predictive model:

1. Click the Add prediction icon  in the diagram toolbar.
2. Open the drop-down menu and select the predictive model that you want to use.
3. In the diagram, create a dependency between this node and the decision node that will use the result of the prediction: hover over the prediction node, click the Connect to another node button  , and select the decision node.
4. Write the decision logic of the decision node using the output of the prediction node. You can use the invocation example available in the Details tab of the prediction node.

Building task models

Task models allow you to implement your business logic with rules, and control the flow execution of these rules in a ruleflow.
For a complete example of how to work with task models, see the Training sample available on GitHub. Alternatively, you can import the Training sample from the Samples library in Decision Designer. For more information about the Samples library, see Creating decision services.

- **Orchestrating rules**
  In task models, you manage and build the items that comprise the business logic of your application: rule artifacts and ruleflows. Ruleflows orchestrate the execution of rules.
- **Creating task model artifacts**
  You create the artifacts you need to define your decision.
- **Creating functions**
  A function contains all the information that is needed to run a specific set of rules in a task model.

Orchestrating rules

In task models, you manage and build the items that comprise the business logic of your application: rule artifacts and ruleflows. Ruleflows orchestrate the execution of rules.
In a task model, you create and edit all the artifacts you need to define your decision: rule artifacts, ruleflows, and variable sets. You can organize and group related artifacts in folders and subfolders.

Rule artifacts

You create rule artifacts to define the decision logic that you want to implement. Two types of rule artifacts are available in task models: business rules and decision tables.

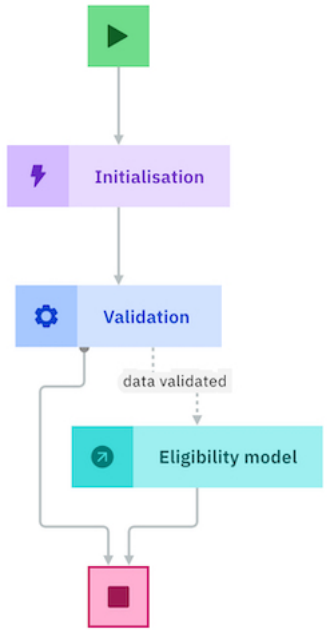
- Business rules are if-then statements that are written with a syntax close to natural language that can be readily understood by business experts. If-then statements associate a condition (if) with an action (then). When the condition is met, the rule action is triggered. You write rules by using a guided editor that lets you assemble statements and add missing variables.
- Decision tables represent decision logic as a table where each row corresponds to a business rule. You use the special facilities that are provided by the decision table editor to work with decision tables.

Ruleflows

With ruleflows, you can manage the flow of rule execution within your task model. A task model must contain at least one ruleflow, but can contain more.

Ruleflows are composed of a set of nodes that are used as building blocks:

- Start nodes and end nodes are graphical markers for the start and the end of the execution flow.
- Rule task nodes contain sets of business rules and/or decision tables.
- Action task nodes contain rule action statements.
- Subflow task nodes reference other ruleflows contained in the same task model.
- Function task nodes reference other models. The referenced model can be any other model contained in the same decision service.



Transitions connect rule tasks in a ruleflow and define the sequence of the ruleflow from one rule task to another.

Variable sets

Variable sets allow you to group related variables. These variables are internal to the task model and can be referenced in business rules and decision tables.

The variables can also be used as input and output parameters in functions.

Creating task model artifacts

You create the artifacts you need to define your decision.

- **Creating a folder**
  You create folders and subfolders to organize the artifacts your task model contains.
- **Creating a business rule**
  You use the rule editor to create and work with rules and use the special facilities provided.
- **Creating a decision table**
  You use the decision table editor to create and work with decision tables and use the special facilities provided.
- **Creating a variable set**
  You create a variable set to group variables that can be used across your task model.
- **Creating a ruleflow**
  You create a ruleflow to manage the flow of execution within your task model.

Creating a folder

You create folders and subfolders to organize the artifacts your task model contains.

About this task

Folders in a task model are similar to folders in a file system. A task model contains a root folder where all your artifacts are created by default. You can create other folders at the root level to better organize your artifacts.

A folder can contain multiple subfolders.

Procedure

1. In the task model toolbar, click the Add folder button .
2. Click the default name of the folder to edit it.

Results

The folder is ready to be used:

- To create a new artifact directly inside the folder, click the + button next to the folder name, and select the type of artifact you want to create.

- To move an existing artifact into the folder, click the More options button ⋮ next to the name of the artifact you want to move. Then, click Move or copy and select the folder that you want to move the artifact to.
- To copy an existing artifact into the folder, click the More options button ⋮ next to the name of the artifact you want to copy. Then, click Move or copy and select the folder that you want to copy the artifact to.

Creating a business rule

You use the rule editor to create and work with rules and use the special facilities provided.

About this task

A business rule defines the specific actions to take when certain conditions are met. A basic rule uses an if-then statement to associate a condition (**`if`**) with an action (**`then`**).

Procedure

1. Use one of the following options to create the rule:

Option	Procedure
Create a rule at the root of the task model	In the task model toolbar, click the Add business rule button  .
Create a rule inside a folder or subfolder	a. Open the folder. b. Click the + button, and select New business rule.

2. Click the default name of the rule to update it.
3. Start building the different parts of the rule. You can:
   - Type directly in the editing area.
   - Copy text from another editor or application, and paste it into the editing area.
   - Select predefined terms and phrases from a completion menu.

Results

You can now use the rule editor to complete the different parts of your rule.

For more information about how to build rules, see Working with business rules. The rule language reference manual is available in the Rule language section.

Example

For example, consider a decision that gives a discount that depends on the category of a customer and how much the spent.

When you create a rule, the rule template contains a simple expression, with the construct **`if <condition> then <action>`**. You can generate a rule that is closer to the decision logic that you want to express by selecting the input data that the rule needs to make the decision. Here the input data is the customer category and the value of the shopping cart of the customer. When they are selected in the list of criteria, an **`if`** part with conditions that use these inputs is added to the rule template.

The conditions contain placeholders where you will indicate actual values. The types of these values and the default constructs depend on the type of data that you defined in your data model and on the node types in the diagram.

Similarly, the output of the decision is the amount of the discount. Therefore, the placeholder for the value of the decision indicates that is a number, based on the output type of the decision.

Now, let's say that the actual decision logic consists in giving a discount of 10 to Gold customer whose cart value is over 100. In the rule editor, the rule template can be modified to represent this decision logic. In the following example, the construct to evaluate the shopping cart value has been replaced with one that is more relevant to this case, and the placeholders for the shopping cart, category, and discount values have been filled with specific values.

```
if
   'shopping cart value' is more than 100
   and 'customer category' is "Gold"
then
   set 'discount' to 10;
```

Creating a decision table

You use the decision table editor to create and work with decision tables and use the special facilities provided.

About this task

Decision tables comprise rows and columns, and each row corresponds to a rule. They are used to represent in tabular form all possible situations that a business decision may encounter, and to specify which action to take in each of these situations.

Procedure

1. Use one of the following options to create the decision table

Option	Procedure
Create a decision table at the root of the task model	In the task model toolbar, click the Add decision table button .
Create a decision table inside a folder or subfolder	a. Open the folder. b. Click the + button, and select New decision table.

2. Click the default name of the decision table to update it.
3. Start defining the condition and action columns for the decision table.

Results

You can now use the decision table editor to define the condition and action columns for the decision table, and to specify values for each row, that is, for each rule.

For more information about how to build data tables, see Working with decision tables. The rule language reference manual is available in the Rule language section.

Example

For example, consider a decision that defines the minimum age requirement for renting a car depending on the state where the rental originates.

When you create a table, the table template contains two empty condition columns and an empty action column. You can generate a table that is closer to the decision logic that you want to express by selecting the input data that the table needs to make the decision.

Here, the input data is the age of the driver and the state where the rental originates.

Creating a variable set

You create a variable set to group variables that can be used across your task model.

About this task

A variable set contains a set of variables that can be referenced in all the rules and decision tables in the task model. These variables can also be used as input and output parameters when you create a function for the task model.

Each variable in a variable set has the following associated information:

- A name to identify it.
- A type to specify the type of data it can contain. This type can be built-in or custom.
- An initial value.
- A verbalization to define how the variable is referenced in business rules and decision tables.

Procedure

1. In the task model toolbar, click the Add variable set button .
2. Click the default name of the variable set to edit it.
3. Click the + button next to the variable set name to add a new variable.
4. Edit the details of the variable:
   - Specify the name, type, and verbalization of the variable.
   - (Optional) Specify an initial value for the variable. For more information about initial values, see ARL task model examples.
   - (Optional) Identify the variable as a list by checking List. When a variable is identified as a list, it can have multiple values.
5. Repeat steps 3 and 4 for each additional variable.

Creating a ruleflow

You create a ruleflow to manage the flow of execution within your task model.

- **Building a ruleflow**
  You can use different ruleflow elements to create your ruleflow. A ruleflow specifies how tasks are chained together: how, when, and under what conditions they are executed.
- **Defining the ruleflow structure**
  You work in the ruleflow editor to create the elements you need to build your ruleflow.
- **Choosing an execution mode**
  You can choose an execution mode for each rule task node of a ruleflow.
- **Choosing a language**
  Two languages are available in the rule editor: the rule language and the Advanced Rule Language (ARL).

Building a ruleflow

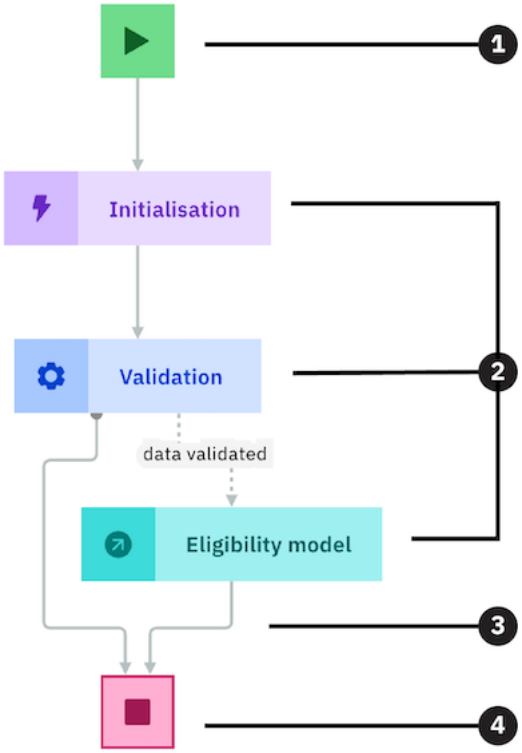
You can use different ruleflow elements to create your ruleflow. A ruleflow specifies how tasks are chained together: how, when, and under what conditions they are executed.

A task model must contain at least one ruleflow in order to be executed. A default ruleflow is created automatically when you create a task model. You can create other ruleflows as well. To create a ruleflow, click the Add ruleflow button in the task model toolbar.

When you click a ruleflow in the Artifacts tab, it opens in preview mode. The preview mode allows you to see the details of each node of ruleflow and easily access the artifacts it contains. You must click Open ruleflow to be able to edit a ruleflow.

The following diagram shows the main parts of a ruleflow:

- Start node 1
- Task nodes 2
- Transition 3
- End node 4



Start and end nodes

A start node and an end node are graphical markers for the start and end of a ruleflow. Every ruleflow has one start node and at least one end node.

You can specify actions to be executed at the start and end nodes. For example, you can define an action on the start node to reset the data used in the ruleflow. Actions defined for an end node also apply to any other end nodes in the ruleflow.

Task nodes

Between the start node and the end node, a ruleflow is made of task nodes that are linked by transitions. The task nodes contain the instructions for what to execute and in what order.

The following types of task node are available:

Rule task node
A rule task node contains a set of rules to be executed at that point in the ruleflow.

Depending on how the execution properties of a rule task are set, the rules might execute in order, or following a more complex logic.

Action task node
An action task node contains action statement to be executed. You define the actions of an action task in the same way that you define actions in business rules.

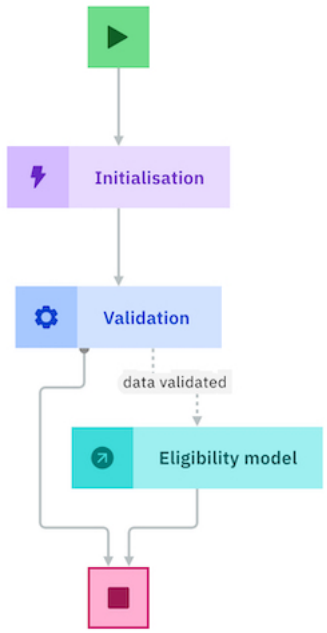
Function task node
A function task node references another model to be executed. The referenced model can be any task model, decision model, or predictive model contained in the decision service.

Subflow task node
A subflow task node references another ruleflow to be executed. The referenced ruleflow can be any other ruleflow in the task model.

Transitions

Transitions connect task nodes in a ruleflow and define the sequence of the ruleflow from one task node to another. Transitions are unidirectional and can have associated conditions.

These conditions determine whether a transition is part of the execution flow. For example, with the following condition on the transition between the validation and eligibility model tasks, the eligibility task can be performed only when the data is validated, otherwise the ruleflow ends.



Two types of elements can be added to a ruleflow to organize transitions: branches and forks. Branches allow you to organize conditional transitions, in the same way that you could start several conditional transitions from a task node. Forks allow you to create multiple, parallel paths in your ruleflow, if you need to execute rules simultaneously.

Initial and final actions

You can define initial actions and final actions on tasks. Initial actions apply before a task is processed and final actions apply after a task is processed. The execution sequence of a task node consists of execute its initial actions, then its body, and then its final actions.

You define initial and final actions in the same way as you define actions for an action task.

Initial and final actions are not mandatory and you can use them independently of each other.

Defining the ruleflow structure

You work in the ruleflow editor to create the elements you need to build your ruleflow.

- **Adding task nodes**
  Between the start node and the end node, the ruleflow is made of tasks linked by transitions. The tasks of a ruleflow contain the instructions for what to run and in what order.
- **Connecting nodes**
  There are a number of different ways to connect nodes in a ruleflow: transitions, branches, forks, and joins. The way you connect nodes impacts the execution flow.

Adding task nodes

Between the start node and the end node, the ruleflow is made of tasks linked by transitions. The tasks of a ruleflow contain the instructions for what to run and in what order.

- **Adding a rule task node**
  A rule task contains a set of rules to be executed at that point in the ruleflow.
- **Adding an action task node**
  An action task contains rule action statements to be executed as part of the ruleflow.
- **Adding a function task node**
  A function task references another model to be executed. The referenced model can be any other model in the decision service.
- **Adding a subflow task node**
  A subflow task references another ruleflow to be executed. The referenced ruleflow can be any other ruleflow in the task model.

Adding a rule task node

A rule task contains a set of rules to be executed at that point in the ruleflow.

Procedure

1. Add a rule task node to the ruleflow in one of the following ways:
   - Hover over the node or the transition after which you want to add the rule task node and click the Add rule task  icon in the toolbar.
   - Click the Add rule task  icon in the ruleflow toolbar and connect the node to the ruleflow.
2. (Optional) Click the default name of the node to edit it.
3. Configure the execution details of the rule task in the Details tab:

   Execution mode
   : The three following execution modes are available:
     - Fastpath (default)
     - Sequential
     - RetePlus

     The Fastpath mode is the default execution mode. Advanced users who want to achieve optimal performance or choose the execution mode that is better adapted for the current rule task can select another execution mode.
     For more information about execution modes, see Choosing an execution mode.

   Exit criteria
   : The rule task exit criteria specifies whether all rules or one rule instance executes before moving to the next node. You can select one of the following values:
     - Execute all rules: All the instances of all applicable rules execute.
     - Execute only one rule instance: Only the first instance of the first applicable rule executes.

   (Optional) Initial and final actions
   : Initial actions apply before the rule task is processed and final actions apply after the rule task is processed. You define initial and final actions in the same way as you define actions for an action task.

4. Open the Logic tab to select the artifacts the rule task must include:
   a. Click Select artifacts.
   b. Select the artifacts to add to the rule task in the left pane.
      These artifacts can be individual business rules or decision tables, or folders containing several rules and tables. It is also possible to select a specific business rule or decision table inside a folder that is also selected.

      When the artifacts for a rule task are selected at run time, the folders in the list of selected artifacts are expanded. The expansion starts at the top of the list. If a business rule or decision table is listed separately above its folder, it retains its execution position in the list. Each rule is selected only once.

      For example, consider a folder that contains business rules R1, R2, and R3:
      - If you add R2 and then the folder, the expanded list is R2, R1, and R3.
      - If you add the folder and then R2, the expanded list is R1, R3, and R2.

   c. In the right pane, use the up and down arrows next to each artifact to change the execution order.
   d. Click Done.
   e. (Optional) You can define a runtime rule selection filter to control which rules are evaluated in the rule task at run time.
      You write the selection filter like a condition in a rule. You can specify it with the rule language or the advanced rule language. For more information about writing selection filters with the advanced rule language, see ARL task model examples.

Adding an action task node

An action task contains rule action statements to be executed as part of the ruleflow.

Procedure

1. Add an action task node to the ruleflow in one of the following ways:
   - Hover over the node or the transition after which you want to add the action task node and click the Add action task  icon in the toolbar.
   - Click the Add action task  icon in the ruleflow toolbar and connect the node to the ruleflow.
2. (Optional) Click the default name of the node to edit it.
3. Click the Edit actions  next to Actions to edit the action task.
   Enter the actions for the task in the editor. For example:

   ```
   set the total of 'the rental agreement' to the price of 'the rental agreement' best offer
   + the coverage subtotal of 'the rental agreement';
   ```

   You can write the rule statement with the rule language or the advanced rule language. For more information about writing actions with the advanced rule language, see ARL task model examples.
4. (Optional) Define initial actions and final actions for the action task node.
   Initial actions apply before the rule task is processed and final actions apply after the rule task is processed. You define initial and final actions in the same way as you define actions for an action task.

Adding a function task node

A function task references another model to be executed. The referenced model can be any other model in the decision service.

Procedure

1. Add a function task node to the ruleflow in one of the following ways:
   - Hover over the node or the transition after which you want to add the function task node and click the Add function task  icon in the toolbar.
   - Click the Add function task  icon in the ruleflow toolbar.

   The Configure function task node wizard automatically opens.
2. In the Select function pane, select the model that you want to reference and click Next.
   A function contains all the information that is needed to run a model. Functions are created automatically for decision models and predictive models. You need to create functions manually for task models. For more information, see Creating functions.
3. Map the input and the output of the referenced model to the ruleflow:
   a. Under Input mapping, you can see the list of input parameters coming from the referenced model. Click the Edit value  icon next to each parameter to map it to task model variables.
      You can enter a simple expression to map input parameters, for example: `the score of 'the borrower'`.
   b. Under Output mapping, click Add mapping to start mapping task model variables to output parameters from the referenced model.
   c. Select a task model variable in the Select a task model variable drop-down list and click the Edit value  icon.
      You can enter a simple expression to map variables, for example: `'insurance decision'`.
   d. Repeat steps b and c for each task model variable that needs to be mapped to output parameters.

   You can use the rule language or the advanced rule language (ARL) to map the parameters.
   Note: There is no completion menu for output mapping in ARL. To access the value returned by a function, you must use the verbalized name of the function in backquotes. Take the example of a function compute pricing and category that returns an object with an attribute pricing. You should enter `` `computePricingAndCategory`.pricing `` to access the value of pricing.
4. Click Save.
   You can reopen the Configure function task node at any time by clicking Edit configuration.
5. (Optional) Click the default name of the node to edit it.
6. (Optional) Define initial actions and final actions for the function task node.
   Initial actions apply before the rule task is processed and final actions apply after the rule task is processed. You define initial and final actions in the same way as you define actions for an action task.

Adding a subflow task node

A subflow task references another ruleflow to be executed. The referenced ruleflow can be any other ruleflow in the task model.

Procedure

1. Add a subflow task node to the ruleflow in one of the following ways:
   - Hover over the node or the transition after which you want to add the subflow task node and click the Add subflow task  icon in the toolbar.
   - Click the Add subflow task  icon in the ruleflow toolbar and connect the node to the ruleflow.
2. (Optional) Click the default name of the node to edit it.
3. Select the ruleflow that you want to reference in the Subflow drop-down list.
4. (Optional) Define initial actions and final actions for the subflow task node.
   Initial actions apply before the rule task is processed and final actions apply after the rule task is processed. You define initial and final actions in the same way as you define actions for an action task.

Connecting nodes

There are a number of different ways to connect nodes in a ruleflow: transitions, branches, forks, and joins. The way you connect nodes impacts the execution flow.

- Transitions
- Branches
- Forks and joins

Transitions

Transitions connect nodes in a ruleflow and define the sequence of the ruleflow from one node to another. Transitions are unidirectional and can have associated conditions.

Transition conditions determine whether a transition is part of the execution flow. You define these conditions in the same way as conditions in a rule.

A single transition from one node to another does not require a condition. However, when several transitions originate from a node, you use transition conditions to define under what conditions a specific path is followed in the ruleflow.

To create a transition between to nodes:

1. Hover over the node that you would like to use as the start of the connection and click the Connect to another node icon  to begin drawing the connection.
2. Click the destination node.
3. (Optional) To specify a name for the transition, select the transition and edit the default name Transition in the details tab.
4. (Optional) To add conditions, click Edit conditions  next to Conditions in the details tab.
   You can either use the rule language or the advanced rule language (ARL) to type the condition statement.

Branches

A branch organizes conditional transitions. Several transitions can go to and from a branch.

All transitions created from a branch must have a condition, except the Else transition.

To organize conditional transitions:

1. Add a branch in one of the following ways:
   - Hover over the transition or the node where you want to add the conditional transitions and click the Add branch  icon in the toolbar.
   - Click the Add branch  in the ruleflow toolbar and connect it to the ruleflow.
2. Create transitions from the branch and add conditions to them, except for the **`Else`** condition.
   Transitions become dashed when you have added conditions to them.
3. (Optional) To add a name to the branch node, select the branch node, and click the default name **`Branch node`** to edit it.

Forks and joins

A fork splits the execution flow into several parallel paths. A join combines all the transitions created from a fork when the parallel paths are all completed.

The transitions from a fork to a join must not have conditions, because the ruleflow follows all paths in parallel between the fork and the join.

To create parallel paths in your ruleflow:

1. Add a fork to the ruleflow in one of the following ways:
   - Hover over the transition or the node where you want your ruleflow to execute several rules in parallel and click the Add fork  icon in the toolbar.
   - Click the Add fork  icon in the ruleflow toolbar and connect it to the ruleflow.
2. Create several transitions and nodes after the fork. You must create a least two transitions from the fork.
3. Add a join to the ruleflow in one of the following ways:
   - Hover over one of the nodes you created after the fork and click the Add join  icon in the toolbar.
   - Click the Add join  icon in the ruleflow toolbar.
4. Create transitions from all nodes originating from the fork to join the node. A path created from a fork must lead to a join.

Choosing an execution mode

You can choose an execution mode for each rule task node of a ruleflow.
By default, a rule task node is assigned the Fastpath execution mode when you create it. To achieve optimal performance, you might need to choose another execution mode that is better adapted for the rules in a particular rule task node.

To determine which execution mode to use on a rule task node, you must analyze the structure of the rules in the rule task node and what type of processing they do.

To make the best choice, answer the following questions:

- What type of application do your rules implement?
- What sort of test do you use in rule conditions?
- What types data types do your rules use?
- What is the effect of rule actions?

What type of application do your rules implement?

Depending on the purpose of the business logic that is defined in a rule task node, you choose a different execution mode.

Compliance and validation
: Compliance business rule applications are often used in underwriting, fraud detection, data validation, and form validation. Business rules in such applications generally have a yes or no result and provide some explanation on the decision.

  For compliance applications, preferably use the **sequential** or **Fastpath** execution modes.

Computation
: Computation business rule applications are often used for scoring and rating, contracts, and allocation. Business rules in such applications carry out different calculations on data that is responsible for providing a final value (or rating).

  For such applications, preferably use the **RetePlus** execution mode if inference is necessary, or use the **Fastpath** execution mode.

Correlation
: Correlation business rule applications are often used for billing. Business rules in such applications insert information.

  For correlation applications, preferably use the **RetePlus** execution mode if inference is necessary, or use the **Fastpath** execution mode.

Stateful session
: Stateful applications are often used in alarm filtering and correlation, GUI customization, and web page navigation.

  For such applications, preferably use the **RetePlus** execution mode without a ruleflow.

What sort of test do you use in rule conditions?

Depending on the type of conditions used in your rules, you choose a different execution mode.

Tests that require a working memory
: If rule conditions test the existence of an object in the working use the **RetePlus** or **Fastpath** execution modes.

  You may use the **Sequential** execution mode when using the working memory if the rule conditions are homogeneous. You can read more about homogeneous rules in the **What data types do your rules use?** section below.

Regular pattern for tests
: If the tests in rule conditions have the same pattern and order, such as the tests that are generated from decision tables, use the **Fastpath** execution mode.

What data types do your rules use?

Depending on the data types acted upon by your rules, you choose a different execution mode.

Homogeneous or heterogeneous rules?
: Bindings are heterogeneous when rules operate on different data types or on the same data types in different order.

Rule	Condition
Rule 1	`definitions` `set 'a customer' to a Customer;` `set 'a shopping cart' to a Shopping cart;`
Rule 2	`definitions` `set 'a customer' to a Customer;`
Rule 3	`definitions` `set 'a customer' to a Customer;` `set ' a car' to a Car;`
Rule 4	`definitions` `set 'a shopping cart' to a Shopping cart;` `set 'a customer' to a Customer;`

- Rule 1 and rule 2 are heterogeneous: they do not have the same number of bindings.
- Rule 1 and rule 3 are heterogeneous: they have the same number of bindings, but operate on different data types.
- Rule 1 and rule 4 are heterogeneous: they operate on the same data types, but in a different order.

If your rules define heterogeneous bindings, use the **RetePlus** or **Fastpath** execution modes.

For two rules to be homogeneous, both must begin with the same `definitions` part.

What is the effect of rule actions?

Depending of the types of effects generated by your rules on execution, you choose a different execution mode.

Rule chaining
: When rule actions cause modifications in the working memory or in the parameters, and when they do pattern matching on objects that have no relationship, like people and hobbies, there is chaining between your rules. This process is also known as inference.

  For example:

Rule 1	`if` `the category of 'the customer' is Silver` `and the amount of 'the shopping cart' is greater than 5000` `then` `promote 'the customer' to Gold ;`
Rule 2	`if` `the category of 'the customer' is Gold` `then` `set 'the discount' to 25 ;`

You can see there is chaining between these two rules because the first rule modifies an object, the customer, that is used in the second rule. The second rule may be executed after the first one.

Basically, if you know your rule actions cause the execution of other rules, use the **RetePlus** execution mode.

Summary

You can use the following table as a reference to make your decision when you choose an execution mode for a rule task node. The number of stars indicates the degree of performance.

In your rule task node:	Fastpath	Sequential	RetePlus
Compliance and validation application	★★★	★★★	★
Computation application with rule chaining	⦸	⦸	★★★
Computation application without rule chaining	★★★	★	★
Correlation application with rule chaining	⦸	⦸	★★★
Correlation application without rule chaining	★★★	★	★
Rule chaining	⦸	⦸	★★★
Tests on existence	★★★	⦸	★★★
Shared test patterns	★★★	★	★★
Heterogeneous bindings	★★★	⦸	★★★
Runtime rule selection that selects a few rules among many	★★	★★★	★★★
Numerous rules	★★★	★★★	★

Choosing a language

Two languages are available in the rule editor: the rule language and the Advanced Rule Language (ARL).
You can choose to write rules in the rule language or the Advanced Rule Language when you:

- Add action statements to start and end nodes.
- Define initial and final actions in task nodes.
- Write rule statements in action task nodes.
- Define a runtime rule selection filter in rule task nodes.
- Map input and output parameters in function task nodes.
- Add conditions on transitions.

You can easily switch from one language to the other in the rule editor by clicking the toggle button  .

The syntax of the rule language is close to that of the natural language. For more information about how to use the rule language, see Rule language.

The syntax of the ARL rule language is close to Java. For more information about how to write actions statements with the ARL rule language, see Advanced Rule Language (ARL).

Creating functions

A function contains all the information that is needed to run a specific set of rules in a task model.

About this task

In a function, you define:

- A reference to a ruleflow, which is going to be used as an entry point to run the task model. If a task model contains multiple rule flows, you can create several functions.
- The input and output parameters that are needed to run the task model. You create parameters from any of the variables that are available in the task model. For more information about variables, see Creating a variable set.

You need to create functions to:

- Call a task model from another one. For more information about composing task models, see Adding a function task node.
- Create a decision operation before you deploy your decision service. For more information about decision operations, see Creating decision operations.

Procedure

1. Open the Functions tab from your task model.
2. Click the + button next to Functions.
3. (Optional) Click the default name of the function to edit it.
4. Select the ruleflow that you want to use as an entry point to the task model.
5. Define input and output parameters for the function:
   a. Click Add in the Parameters section.
   b. Select the variable set that contains the variables you want to use as parameters in the left pane.
   c. Select the variables in the right pane, and set the direction of each parameter.
      The parameters can have three directions:

      `IN`
      The parameter value is provided as input to the ruleflow.
      `OUT`
      The parameter value is provided as output from the ruleflow when you run it.
      `IN` and **`OUT`**
      The parameter value is provided as input to the ruleflow and its value can be modified by the ruleflow and provided as output when you run it.

   d. Click Done.

Integrating machine learning

Use machine learning to make decisions based on the analysis of past data.

When to use machine learning

If your company has a large set of data of past decisions, data scientists can use this data to create a machine learning model. This model can then predict the outcome of new decisions based on this data. The accuracy of the prediction can vary depending on the size and range of the data set.

When or after you design and implement a decision model, you can enrich it by combining rules that describe decisions with machine learning that make predictions.

How to use machine learning in a decision model

Data scientists deploy machine learning models on a machine learning platform, such as Watson™ Machine Learning. Then, you must configure the access from a decision solution to the machine learning providers that contain the model deployments. These providers become available in Decision Designer and you can import deployments or serialized models from these providers to your decision services.

Alternatively, data scientists can also provide you with transparent machine learning models that can be imported into Decision Designer without any prior configuration of the platform.

When you import a machine learning model, it generates a predictive model template that contains all the elements to invoke the machine learning model. You complete this template so that the predictive model can consume your machine learning model.

Finally, you encapsulate the predictive model in a decision node in your decision model. When the decision model is executed, the machine learning model computes a prediction based on the inputs of the decision node that contains it.

For complete examples of how to create a predictive model and integrate it in a decision model, see our tutorials available on GitHub:

Machine learning tutorial - Full path
: Create a predictive model from scratch, and connect it to a machine learning model hosted on Watson Machine Learning.

Machine learning tutorial - Short path
: Connect an existing predictive model to a machine learning model hosted on Watson Machine Learning.

Machine learning tutorial - Quick path
: Import a machine learning model directly into Decision Designer.

Sample - Import PMML files to create transparent predictive models
: Import a ruleset model directly into Decision Designer.

- **Managing local machine learning providers**
  Connect to the machine learning providers where your deployed models are stored.
- **Creating a predictive model**
  Use predictive models to compute a prediction that you can use to make a decision. You create predictive models in decision services, to then use them in decision models.
- **Updating a predictive model**
  You can update a predictive model so that it references a newly trained machine learning model.
- **Integrating a predictive model**
  Integrate a predictive model in a decision model to enrich your decision making.

Managing local machine learning providers

Connect to the machine learning providers where your deployed models are stored.

Before you begin

You need the credentials of the machine learning provider you want to connect to. If you do not know where to find these credentials, contact your admin.

About this task

You can have global or local machine learning providers for your decision automations:

Table 1. Machine learning providers

Machine learning provider	Description
Global machine learning providers	A global machine learning provider is accessible from all of your decision automations that are defined in its instance of Automation Decision Services and all users. Your administrator can configure global machine learning providers on the Automation Decision Services administration page. For more information, see Configuring global machine learning providers.
Local machine learning providers	A local machine learning provider is local to your decision automation, and accessible only from your specified decision automation. You can configure local machine learning providers on the Settings page for your decision automation in Decision Designer, as described in the procedure below.

Three remote machine learning providers are supported in Automation Decision Services: IBM Watson® Machine Learning, Amazon SageMaker, and IBM® Open Prediction Service.

IBM Open Prediction Service is an extension framework that allows you to connect to machine learning providers that are not natively supported by Automation Decision Services. This includes custom machine learning services and third-party machine learning tools, such as Microsoft Azure Machine Learning. For more information about Open Prediction Service and how to install it, see the Open Prediction Service Hub repository.

An embedded machine learning provider is also supported. This provider allows you to import any type of Predictive Model Markup Language (PMML) file and run it directly in your automation.

Procedure

1. Click the Settings icon in the menu bar of your decision automation.
2. Open the Machine learning providers tab and click New.
3. Select the provider type in the drop-down list.
4. Enter a name for the provider and optionally add a description.
5. Complete the required fields depending on the provider type:

Provider	Credentials
Watson™ Machine Learning	Enter the following service credentials to authenticate with your Watson Machine Learning service instance: • API key • Space ID • Authentication URL • URL This information can be found on Watson Studio. The Authentication URL is populated with a default value automatically.
SageMaker	Enter the following service credentials to authenticate with your SageMaker instance: • Region • Access key ID • Access key value
Open Prediction Service	Enter the URL of your Open Prediction Service instance.
Embedded Machine Learning	No credentials needed.

6. Click Test connection to verify the connection.
7. When it is successfully connected, click Save.

What to do next

The machine learning models are now ready to be used in your decision services. You can now import the deployment of your choice to generate a predictive model template that contains all the elements to invoke a machine learning model.
Note: The IT user needs to configure machine learning providers for runtime before executing a decision service. For more information, see Configuring the decision runtime.

Creating a predictive model

Use predictive models to compute a prediction that you can use to make a decision. You create predictive models in decision services, to then use them in decision models.
For example, let's say you have a decision model that decides whether a loan that is requested by a customer should be approved. You have a machine learning model that can predict how likely it is that the customer reimburses the loan, based on a database of past loans. To use this prediction in your decision model, you must encapsulate it in a predictive model first. Then, you can inject this predictive model in your decision model.

Before you create a predictive model, you must have knowledge about the machine learning model:

- List the data that the model requires to make a prediction. For example, the prediction might be based on the age and monthly salary of the customer, and on the amount and duration of the loan.
- Find out the expected form of the prediction. It might be a number in the range 1 - 100, where 100 means that is it certain that the customer will reimburse the loan, and 1 means he will not.
- Verify the range of the values that the model was trained with. For example, if the age of the customer is used to make the prediction, verify the range of ages that the model was trained with. By ensuring that the input data that you provide to the predictive model is within this range, you increase the reliability of its predictions.

To create a predictive model, you generate a template based on a machine learning model. It contains several nodes that are automatically generated for you:

- An input data node that represents one of the input data types that the machine learning model requires to make a prediction.
- A decision node for the mapping of the input data. It contains rules that you write to map the input data types of the machine learning model to data types of your data model.
- A decision node that contains the rule to invoke the machine learning model.
- A decision node for the mapping of the output data. It contains rules that you write to map the output data type of the machine learning model to a data type of your data model.

A machine learning sample is available with Automation Decision Services to help you get started with predictive models. For more information, see Samples and tutorials in GitHub.

- **Configuring a predictive model**
  A wizard guides you through a step-by-step process to configure your predictive model.
- **Mapping input data types**
  You must map the input data types of the machine learning model to the data types of your data model.
- **Editing the invocation rule**
  The invocation rule invokes machine learning models hosted remotely. You can optionally edit the invocation rule that is generated by default.

- **Mapping output data types**
  You must map the data type of the output of the machine learning model to a data type from your data model.

Configuring a predictive model

A wizard guides you through a step-by-step process to configure your predictive model.
You can choose between two options to configure a predictive model:

Remote machine learning model
: Configure the invocation of a machine learning model that is hosted outside of Automation Decision Services. This option requires the configuration of machine learning providers by your administrator.

Local machine learning model
: Import a ruleset or a scorecard model from your file system directly into Decision Designer. This option does not require any prior configuration.

- **Invoking a machine learning model hosted remotely**
  You use the wizard to configure the invocation of a remote machine learning model.
- **Importing a transparent machine learning model**
  You use the wizard to upload ruleset models and scorecard models from your file system.

Invoking a machine learning model hosted remotely

You use the wizard to configure the invocation of a remote machine learning model.

About this task

Using the configuration wizard, you can select the deployment of a machine learning model hosted on a remote provider service. This deployment is used to generate a pre-filled predictive model. Automation Decision Services supports three remote providers: IBM Watson® Machine Learning, IBM Open Prediction Service, and Amazon SageMaker.

Alternatively, you can upload a serialized machine learning model directly into the wizard. Both IBM Watson Machine Learning and IBM Open Prediction Service support the upload of Predictive Model Markup Language (PMML) files. This option is also available with the embedded machine learning provider.

For more information about providers, see Managing local machine learning providers.

Procedure

1. Open your predictive model and click Configure in the details panel on the right to launch the predictive model configuration wizard.
2. Select Remote machine learning model and click Next.
3. Select the provider where the machine learning model that you want to use is stored from the drop-down list.
   The list of deployments that are available from this provider is displayed.
4. Select a deployment or upload the serialized model you need to make your prediction:

Option	Procedure
Select deployment	a. In the list of deployments available, click the arrow next to the machine learning model that you want to use to expand it. b. Select a deployment. Note: This option is not available with the embedded machine learning provider.
Upload serialized model	a. Click the Upload button above the list of deployments available. b. Drag your file into the drop target area or click the drop target area and navigate to the file on your local system.

5. Click Next to define the input parameters that are needed to make the prediction.
   - If an input schema was automatically generated by the provider, you can edit it if necessary. You can update or delete existing parameters, and add new ones.
   - If no input schema was generated, you can:
     - Work in the Form tab and click Add to add parameters.
     - Work in the JSON tab and directly enter a JSON schema.
     - Or use a JSON payload to generate a pre-populated schema by clicking Generate from payload and pasting a JSON payload. The generated schema is editable.
6. Click Next to test the model invocation.
   This step is optional and allows you to make sure that the model returns the expected results when called. To test your model:
   a. Enter values for each input parameter.
   b. Click Run.
   If the model does not return the expected results, you can go back to the previous step to edit the input schema as required.
7. Click Next to define the output values of the prediction. To define the output schema, you can:
   - Work in the Form tab and click Add to add values.
   - Work in the JSON tab and directly enter a JSON schema.
   - Use a JSON payload to generate a pre-populated schema by clicking Generate from payload and pasting a JSON payload. The generated schema is editable.
   - Or use the output of the invocation test to generate a pre-populated schema. The generated schema is editable. This option is available only if you tested the model invocation.
8. Click Apply.

What to do next

You can now map the input and output data from the machine learning model to data types available in your decision service.

Importing a transparent machine learning model

You use the wizard to upload ruleset models and scorecard models from your file system.

About this task

In ruleset models, historical data is used to generate a set of rules which, when executed, returns predictions. When you import a ruleset model into Decision Designer, the rules it contains can either be transformed into Automation Decision Services business rules or decision tables.

In scorecard models, metrics are used to compute a score reflecting a probability. This type of model is used, for example, to compute credit scores. When you import a scorecard model into Decision Designer, it is recommended to import it as decision tables.

Transparent machine learning models must have either the `.pmml` or the `.xml` file format.

Procedure

1. Open your predictive model and click Configure in the details panel on the right to launch the predictive model configuration wizard.
2. Select Local machine learning model and click Next.
3. Drag your file into the drop target area in the wizard.
   Alternatively, you can click the drop target area and navigate to the file on your local system.
4. Select the decision logic generation method that is best suited to the type of model you imported: business rules or decision tables. Then, click Next.
5. (Optional) The list of data types required by the machine learning model displays. Use the interactive mapping wizard to directly map these data types to data types from your data model:
   a. Click any source data type to start mapping it.
   b. Follow the mapping instructions: select a data type from your data model and an expression to map the source type to.
   c. If the source type contains attributes, map them to data types from your data model.
   Note: This step is optional. Mapping input data types can be done at any time following the procedure described in Mapping input data types.
6. Click Apply.

What to do next

You can now map the output data from the machine learning model to data types available in your decision service.

Mapping input data types

You must map the input data types of the machine learning model to the data types of your data model.

About this task

When you encapsulate the predictive model in a decision model, it exists as a function in the context of a dependency diagram. This diagram defines the input data types that are available to the predictive function. The input data types that are expected by the machine learning model might be different from these data types that are part of your data model. You must map these data types so that the machine learning model understands the data types that are used in your decision models.

When you configure the predictive model, a data model is also created. It contains the data types that the machine learning model expects as input, and the data type of its prediction.

Procedure

1. List the data types that are used by the machine learning model. If you do not have this information, you might need to work with the data scientist who created the machine learning model.
2. In your predictive model, define an input data node for each data type that the model needs to make the prediction.
   a. In the diagram editor, click Add input to create an input data node.
   b. Configure the node: give it a name and select a data type from the list.
   c. Link the input data node to the input mapping node.
3. Select the input mapping node and open the Logic tab.
4. Map the input data types by using rules and decision tables.
   - If the mapping is between two data types that are the same except for their name, the mapping is straightforward. In this case, you write a rule statement that looks like this:

     ```
     set the age of decision to the age of patient
     ```

     You can also write a rule statement that maps multiple data types:

     ```
     set decision to a new ML model input where
        the age is the age of patient ,
        the gender is the gender of patient ,
        the systolicbloodpressure is the systolic blood pressure of patient ,
        the diastolicbloodpressure is the diastolic blood pressure of patient ;
     ```

   - If the mapping is between different data types, you need to convert them.
     In the following example, the gender of the patient is an enumeration (male or female) in the data model but the machine learning model expects a number:

     ```
     if the gender of patient is female, then set the gender of decision to 0, else set the gender of decision to 1
     ```

The mapping can require a simple computation. In the following example, the sodium to potassium ratio is computed from a medical test result. It is defined as *the sodium level / the potassium level*:

```
set 'SP ratio' of decision to the sodium level / the potassium level
```

The mapping can require a more complex computation that you implement with a separate decision table in the input mapping node. For example, the blood pressure of a patient is an enumerated value that can be high, normal, or low. It can be computed from a medical test result, based on the systolic blood pressure and the diastolic blood pressure of the patient. Imagine the decision model can provide these two values, and the machine learning expects the computed blood pressure as input data type. In this case, the easiest way to map the data types is to create a decision table:

	Systolic blood pressure		Diastolic blood pressure		***Blood pressure quality***
	Min	**Max**	**Min**	**Max**	
1	< 90		< 60		LOW
2	90	120	40	80	NORMAL
3	> 120				HIGH
4			> 80		HIGH

What to do next

You can now optionally configure the invocation rule that invokes the machine learning model. Else, you can map the data type of the output of the machine learning model to a data type from your data model.

Editing the invocation rule

The invocation rule invokes machine learning models hosted remotely. You can optionally edit the invocation rule that is generated by default.

About this task

The invocation rule is generated by default when you create a predictive model. You can modify the rule to define conditions that must be satisfied in order to invoke the machine learning model.

For example, when an input data is out of the range within which the machine learning model was trained, the prediction becomes unreliable. To prevent the final decision model from using this prediction, you can add a condition that tests for the range of the value.

Procedure

1. Open the predictive model.
2. Select the invocation node, open the Logic tab, and click the invocation rule to open it. By default, the rule invokes the deployment that you imported. It looks like this:

   ```
   set decision to predict ( 'Ml model input' ) ;
   ```

3. Define the condition part of the invocation rule.
   For example, the following invocation rule returns a prediction only if the age of the patient is within the range 28-78. If the patient is younger or older than that, the rule returns null.

   ```
   if the age is more than 28 and the age is less than 78
   then
   set decision to predict ( 'ML model input' ) ;
   ```

Mapping output data types

You must map the data type of the output of the machine learning model to a data type from your data model.

About this task

The output of the machine learning model prediction is usually one of the following:

- A score, in the form of a number.
- A classification.

The format of the output of the prediction might be different from the business data types that can be used in the decision model. So, you must map this output to a data type from a data model.

Procedure

1. Open the predictive model.
2. Select the output mapping node and open the Logic tab.
3. Map the output data types by using a rule or decision table.
   If the prediction returns a score, you do not need to map it to a data type because it already corresponds to a predefined data type. In the case where no mapping is needed, you write a simple rule statement that retrieves the output of the machine learning model:

   ```
   set decision to the prediction of 'ML model invocation' ;
   ```

What to do next

You can run your predictive model to make sure it works as expected. For more information about the running feature, see Running models.

Updating a predictive model

You can update a predictive model so that it references a newly trained machine learning model.
Over time, machine learning models need to be retrained using new input data sets. After a machine learning model has been retrained, you might want to update any predictive model that references it so that it uses the newer version of the machine learning model.

This is done directly in the predictive model wizard so that you can easily update your predictive model without having to redo all the data mapping or rewrite any rule.

You can use one of the following options to open the wizard to select a newer version of a machine learning model:

- Click the Edit configuration button in the predictive model details panel.
- Click Edit configuration in the predictive model menu.
- Click Edit configuration in the ML model invocation node menu.

For more information about selecting machine learning models in the predictive model wizard, see Configuring a predictive model.

Integrating a predictive model

Integrate a predictive model in a decision model to enrich your decision making.

About this task

Predictive models are meant to be reused in decision models to make decisions based on predictions. When you design the decision model, you must think about the part of the decision logic where you want to use the predictive model. Then, you configure the corresponding decision node to use the predictive model.

Procedure

1. Open the decision model where you want to integrate the predictive model.
2. In the diagram toolbar, click Add prediction to create a prediction node.
3. Configure the node by selecting the predictive model that you want to use.
4. In the diagram, create a dependency between this node and the decision node that will use the prediction to make a decision: hover over the prediction node, click the Connect to another node button  , and select the decision node.
5. When you write the decision logic of the decision node, use the output of the predictive model to make your decision.
   For example, if you want to define interest rates based on the risk score computed by a predictive model, you can create a decision table in which the output of the predictive model is used to define a condition column:

	Loan duration		Predicted risk score		Annual interest rates
	Min	Max	Min	Max	
1	< 5		0	0.7	0.05
2	< 5		0.7	0.8	0.052
3	5	8	0	0.7	0.056
4	5	8	0.7	0.8	0.057
5	9	12	0	0.7	0.06
6	9	12	0.7	0.8	0.061

   Where the Predicted risk score condition column is defined as follows:

```
the risk score computed from
   Borrower being Borrower ,
   Loan being Loan ,
is at least <min> and less than <max>
```

   Tip: The invocation example available in the Details tab of the function node can help you write the decision logic.

What to do next

Run the decision model to verify that it behaves as expected with the prediction.

Authoring the decision logic

You must author the decision logic that drives each decision. You express the decision logic with business rules and decision tables.

Business rules

Business rules use if-then statements that express a set of conditions followed by the actions to take if the conditions are satisfied. You write rules in the rule language, which is easily understood because it approximates natural language.

You write rules using an editor that lists relevant statements. You assemble the rules by selecting statements and adding missing variables. The statements automatically include the data model vocabulary that you defined.

Decision tables

Decision tables represent decision logic as a table. Each row in a decision table corresponds to a complete rule.

The rows and columns of the tables show the possible situations that a business decision might encounter, and specify which action to take in each situation. You use decision tables to view and manage large sets of action rules with identical conditions.

- **Working with business rules**
  A business rule defines the specific actions to take when certain conditions are met.
- **Working with decision tables**
  Decision tables express sets of rules with similar conditions and actions, and help you spot problems such as overlaps and gaps among rules.

Working with business rules

A business rule defines the specific actions to take when certain conditions are met.
For a complete example of how to work with business rules, see the Training sample available on GitHub. Alternatively, you can import the Training sample from the Samples library in Decision Designer. For more information about the Samples library, see Creating decision services.

- **How business rules work**
  Learn the basics for writing business rules using a near-natural language based syntax.
- **Using the rule editor**
  The rule editor helps you to write the rules by providing an automatic completion mechanism to select the constructs and statements that make up the rule.

How business rules work

Learn the basics for writing business rules using a near-natural language based syntax.
A rule defines the specific actions to take when certain conditions are met. A basic rule uses an if-then statement to associate a condition (**if**) with an action (**then**). The rule states what action to perform when a condition is true, for example:

```
if
        the credit score is less than 200
then
        set decision to "Loan rejected: credit score too low.";
```

You write a rule as a sentence in a natural language. The rule is composed of business terms, operators, functions, and literal values. In the example, **the credit score** is a business term defined in the vocabulary of your data model, **is less than** is an arithmetic function, and **200** is a literal value.

Business applications call the rules to execute them and provide data values for the business terms. In the example, the rule must access data for the business term **the credit score of the borrower**.

To provide a complete statement, a rule can consist of four parts: definitions, if, then, and else.

The following example shows the four parts in a rule that decides on what discount to give to customers based on how much they spent:

```
definitions
        set cart to the shopping cart of customer;
if
        the value of cart is more than 200
then
        set discount to 15;
else
        set discount to 5;
```

Condition part

Definitions (**definitions**)
Use the **definitions** part to define variables for the rule.
The **definitions** part is optional.

Conditions (**if**)
Use the **if** part to specify the conditions for performing the actions in the **then** and **else** parts. In the example, the condition is **the value of cart is more than 200**.
The **if** part is optional. Rules without conditions perform their actions under all circumstances.

Action part

Actions (**then**)
Use the **then** part to define one or more actions to perform if the **if** part is true. The action in the example says to **set decision to "15% discount"** if the **if** part is true.
The **then** part is mandatory but the **then** keyword is optional if there is no condition in the rule. The rule must have at least one action.

Alternative actions (**else**)
Use the **else** part to define one or more actions to perform if the **if** part is false. The **else** part of the example says to **set decision to "5% discount"** if the **if** part is false.

The **else** part is optional. If the **if** part of a rule is false, and there is no **else** part, the rule does not execute an action.

Reusing input values

Rules in a decision node can't modify the input values that come from nodes that the decision depends on. When a rule tries to modify an input value, IBM Decision Composer creates a copy of the data so that the input value is never modified.

For example, consider the following rule where a person works from home and the address of this person is an input from a parent node:

```
set the address of decision to the address of 'the person' ;
```

IBM Decision Composer detects if a rule modifies the address of the company. As soon as there is one, the system creates a copy of the address of the person, so that this address remains unchanged.

Adding comments in rules

You can add comments to help you structure and document your rules.

One line comments and inline comments both use double dashes (--). The comment finishes at the end of the line. For example:

```
if
        the value of cart is more than 200 -- This is a comment
then
-- This is a comment
        set discount to 15;
```

- **Rule variables**
  You create a rule variable to define the scope of a rule.
- **Rule conditions**
  You use rule conditions to state when to apply the rule actions in the **then** and **else** parts of a rule.
- **Rule actions**
  Rule actions define what to do when the **if** part of the rule is true or false.

Rule variables

You create a rule variable to define the scope of a rule.
A rule variable is a name to which you assign a value. You define the variable in the **definitions** part of a rule, and you can use it only in the condition and action parts of the rule that declares the variable.

Rule variables can make your rules easier to build and understand by simplifying terms.

Making rules easier to understand

You might find parts of a rule difficult to understand. For example, the following rule uses the term **of** to chain together business terms:

```
if
        the value of the shopping cart of customer is less than 100
then...
```

You can shorten this chain of words by using a rule variable. For example, you can define a variable named **cart** to represent the business terms **shopping cart** and **customer**:

```
definitions
        set cart to the shopping cart of customer;
if
        the value of cart is less than 100
then...
```

One rule per rule variable

The scope of a variable is the rule that declares the variable. The name of the variable must be unique in the rule. After you define a variable, it can be used in any part of the rule.

In the following rule, the **definitions** part defines the variable **Smith**, which is then used in the **if** part of the rule.

```
definitions
        set Smith to a customer in customers;
if
        the category of Smith is Gold
then
        set discount to 10;
```

When rule variables do not match data

A rule can define a rule variable that data cannot satisfy when an application calls the decision. For example, you might define a variable named **'low risk borrowers'** for all the borrowers whose credit score is at least 200. However, when you run the rule against data from an application, the rule might not find any borrowers who match the rule variable. In this case, the rule variable behaves as a condition, and the rule cannot execute the actions in its **then** and **else** parts.

Restrictions on rule variable names

Do not use the following characters in the names of your variables:

Character	Description
TAB	Tabulation
\n	Line break
'	Single quotation mark
"	Double quotation mark
()	Opening and closing parentheses
/	Slash
,	Comma
;	Semicolon
’	Curly quotes

Single quotation marks

If the name of a rule variable contains one or more spaces, you must enclose the name in single quotation marks when you define or use the variable. If a variable name contains only one word, quotation marks are optional.

- **Types of rule variables**
  You can assign different types of values to your rule variables.

Types of rule variables

You can assign different types of values to your rule variables.
A rule variable can represent a constant, an expression, a business term, or a collection of business terms. You define a rule variable by giving it a name and a value. You choose the name, and the value can be text, a number, or an arithmetic expression. The value can also be a predefined business term that is already in your rule (for example, **customer**). Once you set a variable, you can use it in any part of the rule that declares the variable.

Rule variables that define constants

The simplest use of a rule variable is to declare a constant value.

For example, the variable **maxAmount** makes the following rule easier to understand, and ensures that the **if** and **then** parts of the rule use the same value:

```
definitions
        set maxAmount to 1000000;
 if
        the amount of loan is at least maxAmount
then
        set decision to "The loan cannot exceed" + maxAmount;
```

Restrictions on rule variables

You can further restrict a variable in the **definitions** part of a rule by using the operator **where**.

Example

In the following rule, **where** restricts the **loyal customer** variable to customers in the Gold category:

```
definitions
        set 'loyal customer' to a customer in customers
                where the category of this customer is Gold;
if
        the value of the shopping cart of 'loyal customer' is more than 200
then
        set decision to "super discount";
```

Example

The following rule declares the **senior Gold customer** variable to be a customer who is in the Gold category and at least 65 years old:

```
definitions
        set 'senior Gold customer' to a customer in customers
                where all of the following conditions are true:
                        - the category of this customer is Gold
                        - the age of this customer is at least 65;
```

Rule variables that refer to more than one occurrence of a business term

If a business term has more than one definition in a rule, you must define the different definitions.

If you have a rule that requires you to refer to more than one occurrence of a customer, you must define the other occurrences in the **definitions** part of the rule, for example:

```
definitions
        set applicant to a customer in customers;
        set 'loyal customer' to a customer in customers;
if
        all of the following conditions are true:
                - applicant is married
```

```
                - 'loyal customer' is insured
                - the address of 'loyal customer' is the address of applicant
then
        set decision to "high rating";
```

Variables are useful in rules that refer to relationships between two or more things of the same type.

For example, the following condition involves two different customers:

```
if
        the address of 'customer 1' is the address of 'customer 2'
```

The condition identifies and names two different customers: **`customer 1`** and **`customer 2`**. The business term "customer" varies, and you can define two customer variables to write a rule like the following one:

```
definitions
        set 'customer 1' to a customer in customers;
        set 'customer 2' to a customer in customers;
if
        the address of 'customer 1' is the address of 'customer 2'
        and 'customer 2' is insured
then
        set rating to the rating of 'customer 2';
```

Note: When a rule contains multiple occurrences of a business term, the rule fires multiple times to cover all the permutations. In the example, there are two customers, so the rule fires two times. If the rule had three customers, the rule would fire six times.

Rule variables that retrieve all the occurrences of a business term

You can use the operator **`all <...>`** to create a variable that retrieves a list of all the occurrences of a business term, for example:

```
definitions
        set 'gold customers' to all customers in customers
                where the category of each customer is gold;
        set 'junior gold customer' to a customer in 'gold customers'
                where the age of this customer is at most 15;
        set 'senior gold customer' to a customer in 'gold customers'
                where the age of this customer is at least 65;
```

The example creates three variables:

- **`gold customers`**: A list of the customers in the gold category.
- **`junior gold customer`**: A customer from the list of gold customers whose age is at most 15 years old.
- **`senior gold customer`**: A customer from the list of gold customers whose age is at least 65 years old.

Rule conditions

You use rule conditions to state when to apply the rule actions in the **`then`** and **`else`** parts of a rule.
You define conditions in the **`if`** part of a rule. A rule condition tests business terms, and produces an answer that is either true or false.

Conditions use data from the application that calls the decision, or data that is the result of a subdecision. Each conditional test compares different business terms or values in the data. Different comparison operators are available for text, numbers, dates, business terms, and sets of entries. All conditions are either true or false. For example, the action in the following rule sets a message that says to redirect a call to a member of the Gold team only if the customer category is Gold:

```
if
        the category of customer is Gold
then
        set message to "Redirect the call to a member of the Gold team";
```

The **`if`** part in this rule contains one condition, which can be either true or false. The rule applies the action statement in the **`then`** part only if the condition is true.

- **Conditions that compare business terms and values**
  You can build conditions that compare and manipulate numeric and Boolean (true or false) values, dates, business terms, and text strings.
- **Conditions that test for existence**
  You can build conditions that check whether one or more business terms of a certain type exist in a set of data.
- **Conditions that test for set membership**
  You can use conditions to test whether a business term belongs to a set.
- **Combinations of conditions**
  You can apply conditions to groups, and test nested groups.
- **Combination negation**
  You can set a rule to perform an action when a condition is not true.

Conditions that compare business terms and values

You can build conditions that compare and manipulate numeric and Boolean (true or false) values, dates, business terms, and text strings.
The following sections show how to use the business operators to compare data.

`starts with`

You can use the text operator **`starts with`** to determine whether a text string starts with a specific sequence of characters:

```
if
        the maintenance number of customer starts with "TX"
```

```
then
        set message to "redirect the call to call center A";
```

is more than

You can use the numerical operator **is more than** to compare two numerical values:

```
if
        the purchase value of the shopping cart is more than 100
then
        set discount to 10;
```

after <date> and **before <date>**

You can use the date operators **after <date>** and **before <date>** to compare two dates:

```
if
        the return date of 'rented car' is after 'pickup date' and before 'scheduled return date'
then...
```

Conditions that test for existence

You can build conditions that check whether one or more business terms of a certain type exist in a set of data.
The following definitions cover operators that test for specified data.

there are

The operator **there are** tests whether there is a specific number of business terms of a certain type in the data from the application that calls the decision.

The following condition checks for two customers:

```
if
   there are 2 customers in customers
then...
```

You can use the operator **where** to specify more conditions to apply to business terms.

The following rule condition is only true if there are 10 customers, and if each customer is a member of the Gold category:

```
if
   there are 10 customers in customers where the category of each customer is Gold,
then...
```

there are at least and **there are at most**

The following operators check whether data contains more or fewer instances of a business term than a specified number:

- **there are at least**: The condition is true if the number of occurrences of the business term is greater than or equal to the specified number.
- **there are at most**: The condition is true if the number of occurrences of the business term is less than or equal to the specified number.

The following condition checks that there are no more than three Gold customers:

```
if
        there are at most 3 customers in customers
                where the category of each customer is Gold,
then...
```

The following condition checks whether there are more Gold customers than Silver customers. The rule starts by defining the following variables:

- **silver customers**: The list of customers in the Silver category.
- **silver count**: The number of customers in the Silver category.

The rule then uses these variables to compare the number of Gold customers with the number of Silver customers:

```
definitions
        set 'silver customers' to all customers in customers
                where the category of each customer is Silver;
        set 'silver count' to the number of elements in 'silver customers';
if
        there are at least ('silver count' + 1) customers in customers
                where the category of each customer is Gold,
then...
```

there is at least one

The operator **there is at least one** checks that there is at least one instance of a business term in the data from the application that calls the decision. The data can contain more than one instance of the business term.

The following condition looks for at least one **customer** object:

```
if
        there is at least one customer in customers
then...
```

The following example checks for at least one senior Gold customer:

```
definitions
   set 'gold customers' to all customers in customers
      where the category of each customer is Gold;
if
   there is at least one customer in 'gold customers'
      where the age of this customer is at least 65,
then...
```

Conditions that test for set membership

You can use conditions to test whether a business term belongs to a set.
The following condition tests whether a customer is in the Silver, Gold, or Platinum category:

```
if
        'customer category' is one of {Silver, Gold, Platinum}
```

The following condition tests whether the rule variable **luxury car groups** contains the car group that is specified in the current rental agreement. The rule first defines the variable **luxury car groups** as the set of all car groups with a daily rate above a certain value:

```
definitions
        set 'luxury car groups' to all car groups in 'car groups'
                where the daily rate of each car group is at least 50;
if
        'luxury car groups' contain the car group of 'current rental agreement'
```

Combinations of conditions

You can apply conditions to groups, and test nested groups.
You can use logical operators to combine conditions in the **if** part of a rule.

The following **if** statement contains one condition:

```
if the category of customer is Gold
```

In the following **if** statement, the logical operator **and** links two conditions:

```
if the category of customer is Gold and the age of customer is at least 65
```

Precedence of `and` over `or`

When the logical operators **and** and **or** link conditions in the **if** part of a rule, the **and** operator takes precedence over the **or** operator.

Consider the following **if** statement:

```
if
        the age of customer is more than 60
        or the age of customer is less than 21
        and the category of customer is Student
```

The statement is true if either of the following conditions is true:

- The customer is older than 60
- The customer is younger than 21 and a student

Multiple conditions and parentheses

You can use parentheses to clarify the precedence of logical operators.

In the following **if** statement, parentheses group an age limit with a category:

```
if
        the age of customer is more than 60
        or (the age of customer is less than 21 and the category of customer is Student)
```

You can also use parentheses to change the interpretation of conditions.

In the following **if** statement, the condition in parentheses is met if the customer is a student older than 60 or younger than 21:

```
if
        (the age of customer is more than 60 or the age of customer is less than 21)
        and the category of customer is Student
```

Multiple conditions that use the same logical operator

You can group conditions and apply the same logical operator to all the conditions in the group by using these statements:

- **`all of the following conditions are true`**: This business term links all the conditions in the group with the logical operator **`and`**.
- **`any of the following conditions is true`**: This business term links all the conditions in the group with the logical operator **`or`**.

In the following rule, the action is performed only when both conditions are true:

```
if
        all of the following conditions are true:
                - the category of customer is Gold
                - a member of the Gold team is available
then
        set message to "Redirect the call to a member of the Gold team";
```

In the following example, the action is performed if the customer's category is Gold or the customer has been waiting for longer than five minutes:

```
if
        any of the following conditions is true:
                - the category of customer is Gold
                - the waiting time of customer is more than 5
then
        set message to "Redirect the call to a member of the Gold team";
```

Combination negation

You can set a rule to perform an action when a condition is not true.
You can use these operators to test whether one or more conditions are not true.

`it is not true that`
You use the operator **`it is not true that`** to negate a condition.

The following condition checks whether the customer is a member of the Gold category:

```
if
        the category of customer is Gold
then...
```

To check whether the customer is not a member of the Gold category, you can use the following statement:

```
if
        it is not true that the category of customer is Gold
then...
```

`none of the following conditions are true`
You can negate a group of conditions by using the operator **`none of the following conditions are true`**. The logical operator **`and`** links the conditions in the group:

```
if
        all the following conditions are true:
                - the category of customer is Gold
                - the age of customer is at most 15
then...
```

The following **`if`** statement is true only if all of the conditions are not true:

```
if
        none of the following conditions are true:
                - the category of customer is Gold
                - the age of customer is at most 15
then...
```

Rule actions

Rule actions define what to do when the **`if`** part of the rule is true or false.
You define actions in the **`then`** and **`else`** parts of a rule. A rule action consists of at least one action phrase that the rule executes when the **`if`** part of the rule is true. It can also contain action phrases to execute when the **`if`** part of the rule is false.

You define actions in the following parts of a rule:

- **`then`**: This part defines actions that the rule executes when the **`if`** part of the rule is true or when no **`if`** part exists. When there is no **`if`** part, the **`then`** keyword becomes optional.
- **`else`**: This optional part defines actions that are executed when the **`if`** part of the rule is false. You do not have to declare an **`else`** part for a rule.

The **`then`** part of a rule must define at least one action. If the **`then`** or **`else`** part contains multiple actions, the rule executes the actions in their listed order.

Action phrases must pick a value for the decision to make. The decision is either represented by the **`decision`** keyword, or by the name of the output of the decision.

Example of the **`then`** part

```
if
        the value of 'shopping cart' is more than $100
then
        set discount to 15;
```

Example of the **`else`** part

The following rule applies a change when the conditions are met (**`then`**), or a different change when the conditions are not met (**`else`**):

```
if
        'loan report' is approved
        and the loan grade of 'loan report' is one of { "A", "B", "C" }
then
        set decision to a new loan report status where the message is "Congratulations! Your loan has been approved";
else
        set decision to a new loan report status where the message is "Congratulations! Your loan has been approved";
```

- **Rule actions for lists of business terms**
  You can define actions that a rule applies for each item in a list.
- **Dependency of rule actions on rule variables**
  You cannot execute the actions of a rule if the variables of the rule cannot be instantiated by objects.

Rule actions for lists of business terms

You can define actions that a rule applies for each item in a list.
You use the operator **for each <item> in <list>** to apply actions to items in a list.

Example

In this example, the decision results in a new shopping cart that contains discounted items. If the category of one or more items in the initial shopping cart contains the word **special**, then all the items in the discounted shopping cart get a discount with an associated message. It means that the last two action statements apply to every item in the list of items of the new discounted shopping cart.

```
if
        there is at least one item in the items of  'initial shopping cart'
                where the category of this item  contains "special",
then
        set decision to a new shopping cart
                where the items are the items of 'initial shopping cart';
        for each item in the items of decision:
                - set the price of this item to the price of this item * (100 - 'the discount') / 100
                - set the message of this item to "A discount has been applied!";
```

Dependency of rule actions on rule variables

You cannot execute the actions of a rule if the variables of the rule cannot be instantiated by objects.
You define rule variables in the **definitions** part of a rule. If the variables of a rule cannot be instantiated by objects, the rule cannot execute and perform an action.

If a rule has variables and conditions, the rule can execute its **else** part only if the variables can be instantiated by objects and the conditions are not true.

The following rule always applies a discount because every customer receives a discount of at least 5 percent:

```
if
        all of the following conditions are true:
                - the value of the shopping cart of customer is more than 100
                - the category of customer is Gold
then
        set discount to 15
else
        set discount to 5
```

However, you can change the rule by using the **definitions** part to give a discount to only customers in the Gold category:

```
definitions
        set applicant to a customer in customers
                where the category of this customer is Gold;
if
        the value of the shopping cart of applicant is more than 100
then
        set discount to 15
else
        set discount to 5
```

Using the rule editor

The rule editor helps you to write the rules by providing an automatic completion mechanism to select the constructs and statements that make up the rule.
When you create or open a rule, the editor opens by default.

You can add terms and phrases to a rule by typing or pasting in text, or by selecting them from the completion menu. If you select a phrase that contains placeholders, you must replace each placeholder with a term, value, or other phrase.

Terms and phrases

The rule editor provides an editing area for building rules. You can add terms and phrases in the following ways:

- Type directly in the editing area.
- Copy text from another editor or application, and paste it into the editing area.
- Select predefined terms and phrases from a completion menu.

Completion menu

The data model defines the terms and phrases in the completion menu, and the ways you can combine them.

Placeholders

Placeholders indicate places in a term or phrase that you must complete. You must insert the name of a business term, a value, or some other phrase in order to construct a valid expression. Placeholders are identified with angle brackets. For example, the following phrase contains the placeholder **<a customer>**:

```
the age of <a customer>
```

To complete this phrase, click the placeholder. You can then either start typing a value or business term, or select a term or phrase from the completion menu. Completion menu options might contain more placeholders that you must also complete to construct a valid expression.

Comments

You can add comments to help you structure and document your rules.

One line comments and inline comments both use double dashes (--). The comment finishes at the end of the line. For example:

```
if
   insurance is null -- Condition to set a default value
then
-- Insurance rate as of 01/01/2017 is 0.02
   set insurance to insurance.of ( 0.02 , true ) ;
```

- **Creating rule parts with the completion menu**
  Use the completion menu to insert a term or a phrase into a rule.
- **Using punctuation to avoid ambiguity**
  Use commas or parentheses to avoid building rules that would otherwise have multiple interpretations.
- **Basic syntax checks**
  Decision Designer verifies the syntax of business rules.
- **Using shortcuts**
  Decision Designer lets you perform most tasks directly from the keyboard. This page lists out the keyboard shortcuts for Windows and Mac.

Creating rule parts with the completion menu

Use the completion menu to insert a term or a phrase into a rule.

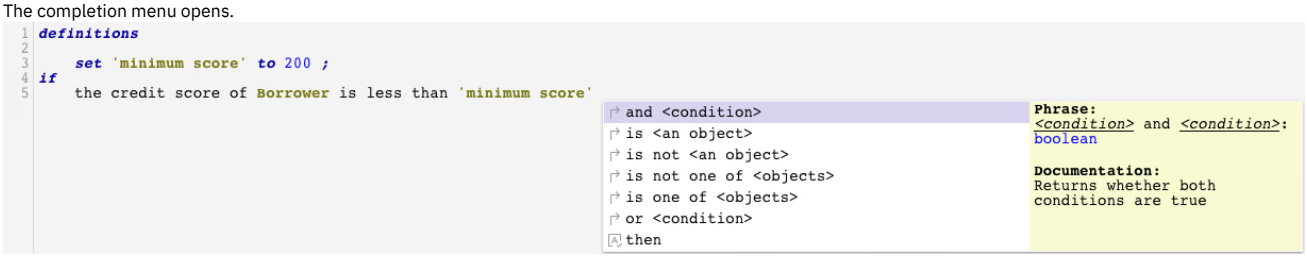
About this task

A rule can comprise of some or all of the following parts in this order: **definitions**, **if**, **then**, and **else**.

You can use the completion menu to create the different parts of a rule.

Procedure

1. Click the location in the rule where you want to insert the term or phrase, and then press Ctrl+Spacebar.
   The completion menu opens.



2. In the completion menu, select the term or phrase you want to insert, and then press Enter.

Results

The term or phrase is now inserted and you can continue building the rule statement.

Using punctuation to avoid ambiguity

Use commas or parentheses to avoid building rules that would otherwise have multiple interpretations.
Ambiguities occur when part of a rule has at least two interpretations that are semantically correct, or when there are two identical terms or phrases in the vocabulary.

For example, the following statement is ambiguous:

```
if
all of the following conditions are true :
   - the category of customer is Gold
   - the age of customer is less than 15
and
   the salary of customer is more than 100
```

The editor raises an ambiguity error because it cannot tell whether **the salary of the customer is more than 100** is associated with **the age of the customer is at most 15**, or whether it is the third condition of the rule.

To remove the ambiguity, you can add a comma at the end of the **the age of the customer is less than 15** condition statement:

```
if
all of the following conditions are true :
   - the category of customer is Gold
   - the age of customer is less than 15,
and
   the salary of customer is more than 100
```

An alternative way to remove ambiguity is to use parentheses to group expressions:

```
if
(all of the following conditions are true :
   - the category of customer is Gold
   - the age of customer is less than 15)
and
   the salary of customer is more than 100
```

Basic syntax checks

Decision Designer verifies the syntax of business rules.
When you edit a rule, Decision Designer displays any errors or warnings that result from checking the syntax. Decision Designer checks for the following syntax errors:

- Unrecognizable or invalid rule statement
- Ambiguous rule statement with more than one correct interpretation
- Repeated or wrong variable in the **`definitions`** part
- Incomplete rule with empty placeholders
- No **`if`** part in a rule with an **`else`** part

Using shortcuts

Decision Designer lets you perform most tasks directly from the keyboard. This page lists out the keyboard shortcuts for Windows and Mac.

Navigation

Keyboard shortcuts for navigating in the rule editor.

Navigation shortcuts

	Mac	Windows
Move to the previous line	↑ or CTRL + P	↑
Move to the next line	↓ or CTRL + N	↓
Move to the next character	→ or CTRL + F	→
Move to the previous character	← or CTRL + B	←
Move the cursor to the end of the current line	Fn + → or CTRL + A	END
Move the cursor to the beginning of the current line	Fn + ← or CTRL + E	HOME
Move the cursor to the beginning of the editor	CMD + ↑	CTRL + HOME
Move the cursor to the end of the editor	CMD + ↓	CTRL + END
Move to the next placeholder	CMD + →	ALT + →
Move to the previous placeholder	CMD + ←	ALT + ←
Open placeholder	CMD + ↓	ALT + ↓

Formatting

Keyboard shortcuts for formatting the content of the rule editor.

Formatting shortcuts

	Mac	Windows
Select the whole content of the editor	CMD + A	CTRL + A
Format text	CTRL + OPTION + F	CTRL + ALT + F
Delete the current line	CMD + D	CTRL + D
Delete the character before the cursor	Shift + Backspace or CTRL + H	Shift + Backspace
Delete the character after the cursor	Delete or CTRL + D	Delete
Indent the current line or selection	Tab	

Completion menu

Keyboard shortcuts for working in the completion menu.

Completion menu shortcuts

	Mac	Windows
Open the completion menu	CTRL + SPACE	

	Mac	Windows
Move to the next entry in the completion menu	↓	
Move to the previous entry in the completion menu	↑	
Move to the last entry in the completion menu	Fn + →	END
Move to the first entry in the completion menu	Fn + ←	HOME
Move to the next page in the completion menu	Fn + ↓	PAGE DOWN
Move to the previous page in the completion menu	Fn + ↑	PAGE UP
Select an element in the completion menu and automatically close the menu	RETURN or TAB	ENTER or TAB
Close the completion menu	ESC	

Working with decision tables

Decision tables express sets of rules with similar conditions and actions, and help you spot problems such as overlaps and gaps among rules.
For a complete example of how to work with decision tables, see the Training sample available on GitHub. Alternatively, you can import the Training sample from the Samples library in Decision Designer. For more information about the Samples library, see Creating decision services.

- **How decision tables work**
  Learn the basics for using decision tables to apply rules that have the same rule statement but different values.
- **Using the decision table editor**
  You use the decision table editor to edit rules in decision tables.

How decision tables work

Learn the basics for using decision tables to apply rules that have the same rule statement but different values.
A decision table contains rows and columns that work together to form rules. In the following table, each numbered row expresses a rule. The columns define the conditions (`Grade` and `Amount of loan`) and actions (`Insurance required` and `Insurance rate`) of the rules in the table.

	Grade	Amount of loan		*Insurance required*	*Insurance rate*
		Min	Max		
1	A	<100,000		*false*	
2		100,000	300,000	*true*	*0.001*
3		300,000	600,000	*true*	*0.003*
4		≥600,000		*true*	*0.005*
5	B	<100,000		*false*	
6		100,000	300,000	*true*	*0.0025*
7		300,000	600,000	*true*	*0.005*
8		≥600,000		*true*	*0.0075*

When an application calls a decision table, the rules are executed. If the conditions in a row are met, the rule that is formed by the row performs the actions in the row.

You can add rows to the decision table and enter values in their cells to create new rules. You can also define preconditions that apply to all the rules in a table. Error markers help you find overlaps and gaps in your rules.

- **Columns**
  Each column in a decision table represents a condition or an action.
- **Rows and cells**
  Each row in a decision table forms a rule.
- **Preconditions**
  You can pretest data before using it with a decision table.

Columns

Each column in a decision table represents a condition or an action.
The columns in a decision table hold the conditions and actions of the rules in the table. The top cell of each column identifies the object of a condition or the target of an action.

	Grade	Loan		Insurance	
		Min	Max	Rate	Required
1	A	<100,000			false
2		100,000	300,000	0,001	true

Each numbered row in the table forms a rule. The rule performs the actions in its row when the conditions in the same row are met. For example, the second row in the example states the following rule:

```
if
   all of the following conditions are true:
      - Grade is A
      - the amount of Loan is between 100,000 and 300,000,
then
   set decision to a new insurance where
      required is true,
      the rate is 0,001;
```

You can split a condition across columns in a decision table when a rule statement contains more than one value. For example, the following condition requires you to specify values for **<min>** and **<max>**:

```
if
   the age of Customer is between <min> and <max>
```

The following decision table expresses the condition in the `Age` column by using subcolumns for **<min>** and **<max>**:

Age group	Age	
	Min	Max
1	18	25
2	26	40

Rows and cells

Each row in a decision table forms a rule.
Each numbered row in a table forms a rule. The values in the cells of the row contain the conditions and actions of the rule:

	Grade	Loan		Insurance	
		Min	Max	Rate	Required
1	A	100,000	300,000	0,001	true

To make rows in a decision table easier to read, you can merge cells with conditions that are common to more than one rule. In the following table, the Grade A cells of rows 1 and 2 have been merged into one cell:

	Grade	Loan	
		Min	Max
1	A	100,000	300,000
2		300,000	600,000

You read the two rules as follows:

- Rule 1

  ```
  if
     all of the following conditions are true:
        - Grade is A
        - the amount of Loan is between 100,000 and 300,000,
  then...
  ```

- Rule 2

  ```
  if
     all of the following conditions are true:
        - Grade is A
        - the amount of Loan is between 300,000 and 600,000,
  then...
  ```

If a third row is added to the decision table, it can be shown in the following ways:

- It can have a different value for the first condition:

	Grade	Loan	
		Min	Max
1	A	100,000	300,000
2		300,000	600,000
3	B	600,000	900,000

- It can share the value for the first condition:

	Grade	Loan	
		Min	Max
1	A	100,000	300,000
2		300,000	600,000
3		600,000	900,000

Merged or split cells

In condition columns, you can merge or split the cells in subcolumns and change the operator. For example, the third row in the following table is also about `Age`, but it includes the operator for **`is more than`**, which takes only one value, instead of **`is between`**, which requires two values:

	Age	
	Min	Max
1	18	25
2	26	40
3	>40	

Partitioned cells

A partition is a group of cells in a condition column with a common cell next to them. In the following table, cells `A` and `B` in the `Grade` column each have a partition of cells in the `Loan` column.

	Grade	Loan		Insurance	
		Min	Max	Rate	Required
1	A	<100,000			false
2		100,000	300,000	0.001	true
3		300,000	600,000	0.003	true
4		≥600,000		0.005	true
5	B	<100,000			false
6		100,000	300,000	0.0025	true
7		300,000	600,000	0.005	true
8		600,000	800,000		
9		≥600,000		0.0075	true

Each numbered row in the table still forms a rule. Partitioning helps you compare rules with similar conditions, and find overlaps and gaps between values of the rules.

Empty cells

A row can contain empty cells. If an empty cell is in a condition column, the condition is always satisfied. If an empty cell is in an action column, the action cell is ignored.

In the following table, the first rule does not set an insurance rate.

	Grade	Loan		Insurance	
		Min	Max	Rate	Required
1	A	<100,000			false
2		100,000	300,000	0.001	true
3		300,000	600,000	0.003	true
4		≥600,000		0.005	true

Preconditions

You can pretest data before using it with a decision table.
Preconditions give you a way test data to determine whether it can be checked by a decision table. If the data from an application that calls a decision table does not satisfy the preconditions of the table, the rules in the table are not executed.

You can also use preconditions to modify incoming data, and declare variables for a decision table.

The preconditions of a decision table are managed outside the decision table, typically in a separate dialog that you access from the decision table. You state preconditions as rules, for example:

```
definitions
   set 'wealthy customer' to a customer
      where the average monthly balance of this customer
                                    is more than $1 000 000
if
   the state of residence of 'wealthy customer' is NY
```

These preconditions start by defining the variable **`wealthy customer`**. The preconditions then limit the scope of the rules in the decision table to customers who have an average monthly balance of $1 million, and reside in the state of New York.

The preconditions are tested before each rule in a table is executed. If an application presents data that passes the preconditions of a decision table, the data is tested by the preconditions before each rule in the decision table is executed. For example, if a table contains 10 rules, the preconditions can be tested 10 times.

Using the decision table editor

You use the decision table editor to edit rules in decision tables.

- **Columns**
  You can change the columns that define the conditions and actions for business rules in decision tables.
- **Rows**
  You can change the rows that form the rules in decision tables.
- **Cells**
  You can update table cells by entering changes directly or by using a rule editor.
- **Defining preconditions**
  You define preconditions to create variable definitions that are available throughout the decision table and conditions that must be satisfied before the decision table is executed.
- **Using decision table locking facilities**
  You can apply a variety of locks to protect your decision tables against unintentional changes.
- **Decision tables errors and warnings**
  Decision Designer checks for overlaps and gaps in decision table conditions, and indicates problems using visual cues.

Columns

You can change the columns that define the conditions and actions for business rules in decision tables.
A decision table contains a group of similar rules that use different conditions and actions. Each row in a table forms a rule, and each column in a decision table represents a condition or an action. With the decision table editor, you can add or remove columns through the column menu, and change the constraints that are applied by each column. When you add or remove a column, you change the rule statement that is used by all the rules in the table.

To open the column menu, right-click the top cell of the column that you want to change.

Table 1. Table of column menu commands

Command	Description
Define Column	Opens a rule editor to change the condition of the column. The rule editor includes a completion menu for building rule statements, and error checking that highlights errors in red and lists them at bottom of the editing area.
Format column	Opens an editor to set the formatting options for displaying values in a column (dates, numbers, and so on). You can select the format from the drop-down list options, and optionally edit the # symbols to personalize the format.
Check Gap	Looks for gaps between values in the cells of condition columns.
Check Overlap	Looks for overlapping values in the cells of condition columns.
Cut	Cuts a selected column. You can paste the column to another location.
Copy	Copies a selected column. You can paste the copy of the column to another location.
Paste	Pastes a copied or cut column to a selected location.
Insert Column	Inserts a condition or action column, depending on the current selection.
Delete	Removes the selected column and its part of the rule statement in the decision table. Deleting a column can disable a decision table.
Clear	Deletes the contents of the cells in the column.

- **Defining columns**
  You define a condition column by specifying a condition statement. You define an action column by specifying an action statement.
- **Sorting and filtering columns**
  You can sort and filter columns.

Defining columns

You define a condition column by specifying a condition statement. You define an action column by specifying an action statement.

About this task

A decision table must have at least one condition column and one action column. You can define condition and action columns using the decision table editor.

In a condition column, you enter a condition statement where you define what the comparison will be, but not the value of the comparison.

A condition statement is an incomplete rule language predicate expression whose placeholders are mapped to specific subcolumns. Placeholders are specific tokens enclosed within < and >. They can represent a concept from the data model (for example, **<an object>**, **<a loan>**, or **<a string>**), or empty text defined in the vocabulary phrase, such as **is between <min> and <max>**.

If you create a condition statement that has more than one placeholder, the required subcolumns are automatically created. If the expression does not contain any placeholders, Decision Designer completes the expression with an **... is a <boolean>** statement so that it has a placeholder.

Note: Placeholders can appear anywhere in the expression. However, a warning may appear if there is any ambiguity about the expected definition.
In an action column, you specify each action to take in each of the specified situations.

Procedure

To define condition and action columns:

1. Right-click the top cell of the column that you want to define and click Define column.
2. Construct your statement in the editor using the available elements from the data model.
   Note: Press Ctrl+Spacebar to open the completion menu when constructing your condition or action statement.
3. When you have constructed your condition or action statement, click Update.

Results

Your columns are now defined. If you need to redefine a column at any time, double-click the column header and change the required details.

When you have defined at least one condition column and an action column, you can begin to specify values for the placeholders in the condition and action definitions. See Specifying values for details.

Sorting and filtering columns

You can sort and filter columns.

About this task

You can sort only condition columns. You can sort an entire column or a group of related cells. Sorting a condition column impacts the order in which rules in the corresponding rows are executed.

You can filter both condition and action columns.

Procedure

1. Use the decision table to sort or filter the required columns, as summarized in the following table:

To sort a condition column:	
a. Click the arrow ▼ in the column that you want to sort. b. Select Ascending or Descending.	a. Click the arrow ▼ in the column that you want to filter. b. Type the value that you want to filter in the filter field.

2. Click Apply.

Rows

You can change the rows that form the rules in decision tables.
The rows contain the values that complete the rule statement that is defined by the condition and action columns in a table. Each row constitutes a complete rule.

To use the row menu, right-click the row heading in the row that you want to change.

Table 1. Table of row menu commands

Command	Description
Insert row	Inserts a new row above or below the selected row. Alternatively, hover your mouse over the border between the selected row and the row above or below, and click +.
Delete	Deletes the selected row.
Clear	Deletes the contents of the cells in the selected row.

You can also resize rows to set them to a specific height.

- **Adding a partially completed row**
  You add partially completed rows to a decision table.
- **Grouping and splitting rows**
  You can group and split rows in a decision table.

Adding a partially completed row

You add partially completed rows to a decision table.

Procedure

To add a partially completed row:

1. In the decision table editor, right-click a condition cell immediately next to the cell containing the value or values that you want to copy to the new row.
2. Click Insert row, and then click either Above or Below.
   A new row appears either above or below the row containing the condition cell that you selected. The values next to the cell that you right-clicked are automatically copied into the new row.

	State	Age of customer		Rental rejected
		min	max	
1	New York	18	21	true
2	New York	21	25	false
3	New York			
4	Rhode Island	16	21	true
5	Rhode Island	21	25	true

Grouping and splitting rows

You can group and split rows in a decision table.

About this task

You can add and remove rows to grouped rows. If you change the value of a cell in a group, you update all the other cells in a group.

You split a group to work with its rows individually. Any change that you make to a split row does not update other rows.

Procedure

- To group rows:

1. Select the cells of the rows that contain the shared condition value.
2. Right-click the cells and click Group.

- To split rows:
  1. Select the cells of the rows that contain the shared condition value.
  2. Right-click the cells, and click Split.

Cells

You can update table cells by entering changes directly or by using a rule editor.
The cells in the rows that form the rules in a decision table contain the values that complete the conditions and actions. The columns define the conditions and actions, which determine what you can enter in the cells.

To open the cell menu, right-click the cell that you want to edit. The cell menu provides the following commands, which vary between the condition and action cells.

Table 1. Table of cell editing commands

Command	Description
Cut	Copies and deletes the contents of the cell. The contents can then be pasted to another cell.
Copy	Copies the contents of a cell.
Paste	Pastes the copied contents of one cell to another cell.
Insert copied cells	Inserts copied cells and their values.
Insert row	Inserts a row above or below the row of the selected cell.
Clear	Deletes the contents of the cell.
Change operator	Inserts an operator to define the range of values. Condition columns only.
Disable/Enable	Disables or enables an action cell. A rule ignores the action of a disabled cell. This is equivalent to clearing the contents of the cell, except that you keep the information it contains so that it can be re-enabled later if required. Action columns only.
Edit custom value	Displays the contents of the cell in a rule editor. The rule editor includes a completion menu for building rule statements, and error checking that highlights errors in red and lists them at the bottom of the editing area.

- **Specifying values**
  You can enter values in the cell itself.
- **Changing the default operator for a cell**
  When you specify a default operator in a condition statement, you can override the operator in a particular row. You can also modify the value of a cell by adding an expression.

Specifying values

You can enter values in the cell itself.

About this task

To specify values, you can either enter simple values in the decision table editor or write expressions in the custom value editor.

Procedure

- To specify a value using the decision table editor:
  1. In the decision table editor, double-click the cell that you want to edit.
     A cursor or a list of allowed values displays, depending on the type of value you specify.
     Note: The cell you are editing sticks to the top when you are scrolling down the decision table.
  2. Type or select the required value, and press Enter.
- To specify a value using the custom value editor:
  1. In the decision table editor, right-click the cell that you want to edit and click Edit custom value.
  2. Write a statement that defines the value of the cell.
     Note: Press CTRL+Space bar to open the completion menu when editing the value of the cell.
  3. Click OK.

Changing the default operator for a cell

When you specify a default operator in a condition statement, you can override the operator in a particular row. You can also modify the value of a cell by adding an expression.

About this task

When you define a condition column statement, you set the default operator for all the cells in that column.

For example, if you define the following condition column statement, you set the default operator to **`is more than (>)`**:

```
'age of the customer' is more than <a number>
```

The operators you can choose from match the value type for the cell. You can change an operator later by editing the cell. You can edit a cell in the decision table editor, or use the custom value editor.

Procedure

- In the decision table editor, change the default operator for a cell using one of the following options:
  1. In the decision table editor, right-click the cell.
  2. Click Change operator, and then select an operator.
- To change the operator using the custom value editor:
  1. In the decision table editor, right-click the cell and click Edit custom value.
  2. In the custom value editor, modify the operator part of the statement.
  3. Click OK.
     The operator displays as a symbol next to the cell value.

Defining preconditions

You define preconditions to create variable definitions that are available throughout the decision table and conditions that must be satisfied before the decision table is executed.

About this task

You can define preconditions for a decision table. Preconditions can contain the following definitions:

- Variables that are available throughout the decision table
- Conditions that must be satisfied before the decision table is executed

Procedure

To define a precondition:

1. In the decision table editor, click Edit preconditions.
2. In the preconditions editor, press Ctrl+Spacebar.
   The completion menu opens. You define variables and conditions in the Preconditions section in the same way that you would create variables and conditions for a business rule in the Rule editor. See Using the rule editor for details.
3. Click Define.

Results

The variables that you defined are now available for creating condition definitions in the decision table, which means that the rows of the decision table cannot be evaluated before the conditions that are defined in the precondition are satisfied.

Using decision table locking facilities

You can apply a variety of locks to protect your decision tables against unintentional changes.

About this task

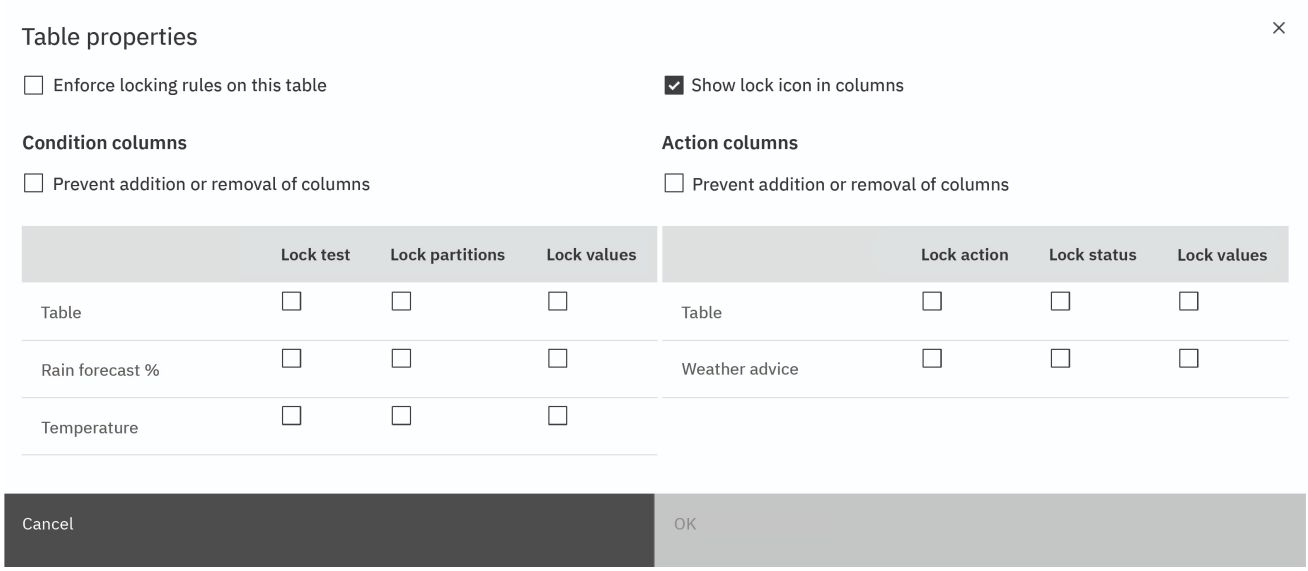
Decision tables have locking facilities that allow you to protect your decision tables against unintentional changes. For both condition columns and action columns, you can lock the structure, data, or both.

Locks can be scoped to specific columns, or to a whole decision table.

Procedure

1. Right-click any column header in your decision table, and then click Table Properties.

The Table Properties editor opens.



2. Choose which locks you want to apply:

   Enforce locking rules on this table
   : The default is off to allow you to edit your decision table and lock rules. This option must be set to on if you want the locks you have defined to apply.
   Note: If on, locks cannot be edited anymore.

   Show lock icons in columns
   : If on, the lock icon is displayed on any locked condition and action columns.

   Prevent addition and removal of columns
   : Prevents condition or action columns from being added or deleted.

3. In the Condition columns section, select any of the following locks:

   Lock test
   : The test expression associated with the condition column cannot be changed. Note that cell content, including cell-specific test operators, can still be changed.

   Lock partitions
   : Partition items cannot be added to or removed from the selected condition column. The values of existing partition items can still be changed, but no rows can be added to or removed from this column.

     Locking partitions locks partitions in all preceding columns because adding a new partition item to a previous column can create a new row in the locked column.

     For more information about partitioning, see Rows and cells.

   Lock values
   : The values of existing partition items or the cell operator cannot be changed.

4. In the Action columns section, select any of the following locks:

   Lock action
   : The expression associated with the column cannot be changed.

   Lock status
   : The status of the action cell cannot be changed. The action status specifies whether an action is enabled or disabled.

     For more information about enabling and disabling action cells, see Cells.

   Lock values
   : The values in the action cell of the selected column cannot be changed.

5. Click OK to apply your selections.

Decision tables errors and warnings

Decision Designer checks for overlaps and gaps in decision table conditions, and indicates problems using visual cues.
As you submit values to the cells of your decision table, Decision Designer checks the column automatically for errors. You can disable gaps and overlaps check from the column menu.

Overlap checks

Decision Designer can verify that the values you enter for a given condition throughout the different rows do not overlap or are not identical.

For example, consider a column for the age of the customer:

- Row 1: age is between 17 and 30
- Row 2: age is between 29 and 40

In this example, customers that are 29 years old satisfy the condition of both rows. Decision Designer reports this as an overlap warning.

Decision Designer checks for overlaps for all the entries in a column, and within partitioned groups of cells.

Gap checks

Decision Designer can verify that the cells in a condition column consider all possible cases, which helps you make sure there are no gaps in your tables.

For example, consider a column for the age of the customer:

- Row 1: age is between 17 and 30
- Row 2: age is between 32 and 40

In this example, customers that are 31 years old are never be taken into account. Decision Designer reports this as a gap warning.

Decision Designer evaluates gaps for all the entries in a column, and within partitioned groups of cells.

Declaring and managing dependencies

Declaring dependencies lets you reuse decision artifacts within a decision service.
Automation Decision Services allows you to reuse decision artifacts throughout a decision service. For example, you can use the output of a decision model in another decision model, or use the vocabulary defined in a data model in multiple other decision artifacts.

Declaring dependencies

Some dependencies are automatically declared when reusing decision artifacts:

- When adding function nodes in decision models. For more information, see Calling other models.
- When adding function task nodes in task models. For more information, see Adding a function task node.
- When selecting a data model at the creation of a decision model, task model, or predictive model. For more information, see Using data models in other decision artifacts.

Other dependencies need to be manually declared before you can reuse a decision artifact:

- When reusing data types from a data model in another data model. For more information, see Using data models in other decision artifacts.
- When using data types from an external library in a decision artifact. For more information, see Using an external library in a decision service.

Ordering dependencies

The order in which the dependencies of a decision artifact is displayed in the Dependencies tab determines how ambiguities between data types are resolved.

For example, if a decision model has a dependency on two data models that contain data types with the same fully qualified name, data types from the data model that is listed higher will be used in priority.

To reorder dependencies, use the up and down arrows located next to each dependency.

Removing dependencies

Important: When deleting a function node in a decision model or a function task node in a task model, the dependency to the reused model is not automatically removed. It needs to be removed manually, as described below.
You can remove a dependency between two decision artifacts from the Dependencies tab:

1. Open the source decision artifact, that is the decision artifact that reuses another one.
2. Go to the Dependencies tab.
3. Browse to the dependency that you want to remove and click the Remove  icon.

Creating decision operations

In a decision operation, you define an entry-point to a model. A decision service must contain at least one decision operation to be deployed.

About this task

A decision operation is defined by specifying the following information:

- A reference to the source model.
- A model function that contains all the parameters that are needed to run the model. For decision models and predictive models, a function is created automatically. For task models, the function needs to be created manually. For more information, see Creating functions.

Procedure

1. Open the Decision operations tab from your decision service.
2. Click Create.

3. Select the model for which you want to create an entry-point in the Source model drop-down list.
4. For task models, select a function in the Model function drop-down list.
   The list of parameters for the decision operation is displayed in read-only mode under Parameters.
5. (Optional) The decision operation name field is populated automatically with the function name. The decision operation name is used to identify your decision service when calling it. You can edit this name.
6. Click Create.

Results

The decision operation that you created is now available in the Decision operations tab. You can reopen the decision operation to edit its details.

What to do next

You can share your changes with your collaborators, and then create a version of your decision automation. When you create a version of your automation, it can be later used to build and deploy a decision service.

Enabling automatic refactoring

When automatic refactoring is enabled, common updates are automatically propagated to other decision artifacts. This feature is available as a technology preview.

About this task

With automatic refactoring on:

- Decision diagrams, business rules, and decision tables are refactored when:
  - A data type is renamed.
  - A composite type attribute is renamed.
  - An enumeration value is renamed.
  - The verbalization of a composite type or an attribute is updated.
- Business rules and decision tables are refactored when:
  - The gender of an attribute is updated.
  - The verbalization of a variable is updated.
- Data sets in the Run tab are refactored when changes are made to the data model, such as renaming an existing data type.
- JSON names are refactored when you set an attribute as a list.
- Group IDs and package names are refactored when the group ID or name of a decision service is updated.

Automatic refactoring needs to be manually enabled for each decision service.

Procedure

To enable automatic refactoring in a decision service:

1. Open the Decision services tab on your decision automation page in Decision Designer.
2. Click the overflow menu ⋮ for your decision service, and select Edit details.
3. Select Enable automatic refactoring (experimental) at the bottom of the window.
4. Click Save.

Testing decision services

Automation Decision Services lets you test the models you build directly in Decision Designer or at build time.

- **Running models**
  As you build decision models, task models, and predictive models, you can verify that they produce the results you expect by running them with test data.
- **Unit testing**
  Automation Decision Services allows you to define unit test scenarios that you can run at build time.

Running models

As you build decision models, task models, and predictive models, you can verify that they produce the results you expect by running them with test data.

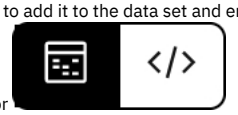
About this task

The run feature allows you to apply data to the decision logic that is defined in your model to make sure that it works as expected.

When you test a model, you define and run test data to validate specific scenarios. You can provide the data in a friendly form or in its underlying JSON format.

Warning: You cannot run models that contain errors.

Procedure

1. Open a model, and click the Run tab.
2. Click Add test data set to create a data set.
   An empty form with a list of data items that you can add to the data set is generated.
   Note: You can easily rename and delete data sets using the Rename test data set and Delete test data set buttons in the top bar.
3. Click any data item to add it to the data set and enter a value for the item. If you prefer to edit the content of the data set in JSON, use the content switcher to open the rich JSON editor 
   For more information about supported data formats, click the **i** icon at the top of the data set.
4. Click Run to run the model with the data set.

Results

The run report is displayed in the results pane. In addition to the result of the decision, this pane shows data about the execution, such as the execution and compilation times, and which rules executed and their order of execution.

Click Show JSON output to view all input, output, and execution data in JSON format.

Example

Open the Approval decision model available in the Loan Approval decision service in the Banking sample, and click the Run tab. You can see data sets that correspond to different test scenarios. When you run the Mr Doe loan data set, the loan is approved and you get the following result:

```
{
    "approved" : true,
    "message" : "Congratulations! Your loan has been approved"
}
```

In this data set, look for the Yearly income field and change its value from 100000 to 80000. When you run the data set with this new value, the loan is not approved and you get the following result:

```
{
    "approved" : false,
    "message" : "Too big Debt/Income ratio: 0.45597806940658686"
}
```

This result validates that the yearly income of the borrower influences the approval of the loan. The number of months during which the borrower repays the loan is also supposed to influence this approval. If you change the value of the number of monthly payments from 72 to 120 and run this data, the loan is now approved again:

```
{
    "approved" : true,
    "message" : "Congratulations! Your loan has been approved"
}
```

This result validates that the number of monthly payments influences the approval of the loan, and that it can compensate for a yearly income that might be too low.

Related information

- Decision returns null
- A rule is not executed

Unit testing

Automation Decision Services allows you to define unit test scenarios that you can run at build time.
Unit tests are made of two sets of files: a Java test file and a set of scenarios written in JSON. Unit tests are created in a separate source folder to keep the test code separate from the main code.

You can either run tests locally in a CI/CD stack, or directly in Decision Designer.

For a complete example of how to create and run unit tests, see the Testing a decision service at build time sample available on GitHub.

Restriction: You cannot run unit tests on predictive models or other models that call predictive models.

- **Creating a Java test file**
  You create a Java file to allow your tests to be automatically discovered and executed. You must create one Java file for each model that you want to test.
- **Adding JSON scenarios**
  You define unit test scenarios in JSON files. You create one JSON file per scenario.

Creating a Java test file

You create a Java file to allow your tests to be automatically discovered and executed. You must create one Java file for each model that you want to test.

Procedure

1. Open the decision library containing the model that you want to test and create a `src/test/java/<packageName>` folder.
   The *<packageName>* can be any name you want.
2. Create a `.java` file in the directory you just created, for example `MyTest.java`.
3. Add the following content in the Java file:

```
package <packageName>;

import com.ibm.decision.run.test.junit5.DecisionTest;
import com.ibm.decision.run.test.junit5.JSONTestDirectoryFactory;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.DynamicNode;
import org.junit.jupiter.api.TestFactory;

import java.util.stream.Stream;

@DecisionTest(decisionFunction = "<functionID"> /* OR  decisionOperation = "<operationID>" */)
@DisplayName("<DisplayName>")
public class <TestClassName> {
    @TestFactory
    public Stream<DynamicNode> decisionTests() {
       JSONTestDirectoryFactory.verbose = true;
        return JSONTestDirectoryFactory.createTests(this.getClass());
    }
}
```

   Where:
   - *<functionID>* corresponds to the model name and the function name separated by an hyphen, for example `LoanValidation-loan-validation-decision-model`.
     Tip: You can use one of the following options to easily find the `functionID` of the model that you want to test:
     - Build your decision service and open the <model>/target/classes/META-INF/decisions folder. Then, open the JSON file it contains and copy the value of the `name` property.
     - Run the unit test with an erroneous `functionID` value. This throws an exception with a list of all the function names that are available.
   - *<operationID>* corresponds to the operation name of the model.
     Tip: You can use one of the following options to easily find the `functionID` of the model that you want to test:
     - Build your decision service and open the <model>/target/classes/META-INF/decisions folder. Then, open the JSON file it contains and copy the value of the `operationName` property.
     - Run the unit test with an erroneous `operationID` value. This throws an exception with a list of all the operation names that are available.
   - *<DisplayName>* is the custom name for the test class. This annotation is optional.
   - *<TestClassName>* begins with `Test`, for example `TestLoanValidation`.

   The `JSONTestDirectoryFactory.verbose` parameter is optional. When set to `true`, this parameter enables a higher level of verbosity. Running unit tests in verbose mode can be useful when setting up your tests. It returns additional information such as the JSON schema of the input of the decisions.

   The reference documentation for the `DecisionTest` annotation can be found in the Reference section.

What to do next

You can now define test scenarios in JSON and add them to the `src/test/resources` folder.

Adding JSON scenarios

You define unit test scenarios in JSON files. You create one JSON file per scenario.

About this task

A JSON file consists of three parts: a name, input data, and a list of assertions.

You can retrieve the input data content from Decision Designer by copying a JSON payload from a data set in the Run tab. For more information about the Run feature, see Testing models in Decision Designer. Alternatively, you can find your data sets in the <model>/resources/validate_dataset folder of your decision library.

You can define one or more assertions in a test. Each assertion consists of five items:

`label`
: Defines the name of the assertion.

`pointer`
: Represents the path to the tested values in the output.

`operator`
: Defines the type of comparison you want to perform. The list of supported operators is available below.

`ignoringOrder` (optional)
: Makes an order agnostic comparison of collections if set to `true`. It is set to `false` by default.

`expected`
: Contains all expected values.

Table 1. List of supported operators in assertions

Operator	Definition	Example	Details

Operator	Definition	Example	Details
⊃	Contains	<pre>{ "label": "Assertion 1", "pointer": "/customer/address", "operator": "⊃", "expected": "Paris" }</pre>	These operators can be used on any type of data. They support the **ignoringOrder** option for collections.
⊅	Does not contain	<pre>{ "label": "Assertion 1", "pointer": "/customer/address", "operator": "⊅", "expected": "London" }</pre>	
=	Equals	<pre>{ "label": "Assertion 1", "pointer": "/approval/approved", "operator": "=", "expected": false }</pre>	These operators can be used on any type of data. They support the **ignoringOrder** option for collections.
!=	Does not equal	<pre>{ "label": "Assertion 1", "pointer": "/approval/approved", "operator": "!=", "expected": false }</pre>	
>	Is greater than	<pre>{ "label": "Assertion 1", "pointer": "/customer/age", "operator": ">", "expected": 18 }</pre>	These operators can be used on numbers.
≥	Is greater than or equal to	<pre>{ "label": "Assertion 1", "pointer": "/customer/age", "operator": "≥", "expected": 18 }</pre>	
<	Is less than	<pre>{ "label": "Assertion 1", "pointer": "/customer/age", "operator": "<", "expected": 18 }</pre>	
≤	Is less than or equal to	<pre>{ "label": "Assertion 1", "pointer": "/customer/age", "operator": "≤", "expected": 18 }</pre>	
∈	Is within a range	<pre>{ "label": "Assertion 1", "pointer": "/customer/age", "operator": "∈", "expected": { "min": 10, "max":15, "minExclusive": true, "maxExclusive": true } }</pre>	These operators can be used on numbers.
∉	Is not within a range	<pre>{ "label": "Assertion 1", "pointer": "/customer/age", "operator": "∉", "expected": { "min": 10, "max":15, } }</pre>	

Procedure

1. Create a **resources/*<packagename>*/*<operationID>*** folder under the **src/test/** folder you created in Creating a Java test file.
   The *<packagename>* must be the same as the one you used when creating the Java test file.
2. Create a **.json** file in the directory you just created, for example **lowcreditscore.json**.
3. Define your scenario using the details provided in the About this task section.
   The following example shows a complete test file named **too big debt income ratio** that contains two assertions:

```
{
  "name": "too big debt income ratio",
  "input": {
    "loan": {
      "numberOfMonthlyPayments": 72,
      "startDate": "2020-06-01T00:00:00Z",
      "amount": 185000,
      "loanToValue": 0.7
    },
    "borrower": {
      "firstName": "John",
```

```
      "lastName": "Doe",
      "SSN": {
        "areaNumber": "123",
        "groupCode": "45",
        "serialNumber": "6789"
      },
      "latestBankruptcy": {
        "date": "2019-03-10T00:00:00Z",
        "chapter": 7,
        "reason": "Unemployment"
      },
      "yearlyIncome": 100000,
      "zipCode": "91320",
      "creditScore": 500,
      "birthDate": "1968-05-12T00:00:00Z"
    },
    "currentTime": "2020-02-04T08:41:21.242Z"
  },
  "assertions": [
    {
      "label": "Approved",
      "pointer": "/approval/approved",
      "operator": "=",
      "expected": false
    },
    {
      "label": "Message, sorry",
      "pointer": "/approval/message",
      "operator": "ↄ",
      "expected": "sorry"
    }
  ]
}
```

What to do next

You are now ready to run your unit tests. You can either run tests locally using Maven or directly in Decision Designer using the embedded build service.

To run unit tests locally, follow the instructions described in Building and deploying from a CI/CD stack.

To run unit tests in Decision Designer, you must first push all the content of the **src/test** folder to your Git repository and load it to Decision Designer. When you are ready to run your unit tests, follow the instructions described in Building and deploying from Decision Designer. Click the View logs icon next to the deployment status and browse to the Tests section to see the test results.

Important: Unit tests are executed every time you build and deploy your decision service. You must update unit tests whenever you update the model you are testing to avoid deployment failures.

Deploying decision services

The next step after designing decision services in Decision Designer is to build and deploy them. To do so, you can use a CI/CD stack or the embedded build service available in Automation Decision Services.

- **Building and deploying from Decision Designer**
  Automation Decision Services provides the embedded build service, which allows you to trigger building and deploying a decision service in a selected version of your decision automation from Decision Designer.
- **Building and deploying from a CI/CD stack**
  You might want to share your decision services on an environment where you can develop, test, deploy, and execute them in a continuous way.
- **Deploying to decision runtime with REST API**
  You can deploy decision service archives by using the runtime REST API.
- **Promoting deployed decision services to another decision runtime**
  You can promote a decision service archive from the development environment to the production environment.

Building and deploying from Decision Designer

Automation Decision Services provides the embedded build service, which allows you to trigger building and deploying a decision service in a selected version of your decision automation from Decision Designer.

Before you begin

The decision runtime must be installed.

For more information about the installation, see Preparing to install Automation Decision Services.

About this task

When decision services are deployed successfully, they can be used by the decision runtime for executing decisions. Client applications can call decision services by using the decision runtime REST API.

- Decision service archives are generated for decision services, and deployed to the decision runtime.

Decision service archives that are built from Decision Designer are deployed to the decision runtime in the deployment space ID (deploymentSpaceId) named **`embedded`**.

Restriction: The embedded build service cannot be customized. You might need to use a CI/CD stack for your organization if you want to customize the building and deploying part.

Procedure

1. Open the Deploy tab from your decision automation page.
2. Expand the version of the decision automation that you selected, and then click Deploy next to the decision service that you want to build and deploy.
3. Click Deploy in the confirmation window.
   It might take a while to finish deploying a decision service.

   Tip: Check errors in the log and test report if building or deploying a decision service fails. Click the icon  in the deployment status. Contact your administrator or IT specialist to resolve issues if necessary.
   You can undeploy the decision service by clicking the Undeploy icon.

Results

The value for the decisionId parameter is displayed in the Decision ID column. This value is used for executing the decision service later. For more information, see Executing decision services.

You can test the deployed decision service by using the decision runtime REST API in the Swagger UI facility:

1. Click the Test icon {...} for the decision service in the Deploy tab.
2. Swagger UI opens and you can start testing the deployed decision service.

For more information about the runtime REST API, see the Decision runtime REST API reference in Reference.

What to do next

The decision service is now ready to be used by the decision runtime to execute decisions. For more information, see Executing decision services.

Building and deploying from a CI/CD stack

You might want to share your decision services on an environment where you can develop, test, deploy, and execute them in a continuous way.

About this task

You can use a continuous integration and delivery (CI/CD) stack to manage your builds and deployments.

For that, you need the following minimum requirements:

- A Git server where you deploy the decision service source files.
- A Git client to fetch the decision service source files and build them.
- A settings.xml file (~/.m2/settings.xml) whose sections repositories and pluginRepositories point to the artifact repository.
- An artifact repository to host the decision build Maven plug-in and the artifacts that are generated for external libraries.

Note: If you do not have your own artifact repository in place, you can find a sample implementation on GitHub, https://github.com/icp4a/automation-decision-services-ci-cd-stack, for you to install and use.

Procedure

1. Install the decision build Maven plug-in.
   Follow the instructions given in Setting up your build environment.

2. Build a decision service with Maven.
   a. Run the following command:

      `git clone <URL git repo>`

      Where *<URL_git_ repo>* is the remote repository where you deployed the decision service from Decision Designer.

   b. Run the following command:

      `cd <repo_name>/<project_name> ; mvn clean install`

      Where *<repo_name>* is the name of the repository that is created by the **`git clone`** command.

      Where *<project_name>* is the name of the folder that contains the decision service to be built.

      The Maven command **`mvn`** inspects the pom.xml files of the project. It fetches all the dependencies, including the decision service build Maven plug-in that you installed in step 1. It compiles the decision service. Finally, it installs the resulting decision service archive (.jar file) into your local Maven repository (typically $HOME/.m2/repository). This jar file is installed under *<repo_name>*/*<project_name>*/.decisionservice/target.
3. The decision service archive can be deployed by using the decision runtime REST API.
   For more information, see Deploying to decision runtime with REST API.

Results

The decision service archive is now ready to be used by the decision runtime to return a decision. For more information, see Executing decision services.

Deploying to decision runtime with REST API

You can deploy decision service archives by using the runtime REST API.

Before you begin

For more information about the runtime REST API methods that are mentioned in the procedure, see the Decision runtime REST API reference in Reference.

Procedure

1. Deploy a decision service archive by using the following runtime REST API method:

   ```
   POST https://<server>:<port>/ads/runtime/api/v1/deploymentSpaces/{deploymentSpaceId}/decisions/{decisionId}/archive
   ```

   - *<server>* is the hostname of the decision runtime, which is specified in the decision_runtime.ingress.hostname parameter.
   - *<port>* is the port of the decision runtime.

   For example, you can use a cURL command:

   ```
   curl -i -X 'POST' \
   'https://<server>:<port>/ads/runtime/api/v1/deploymentSpaces/{deploymentSpaceId}/decisions/{decisionId}/archive' \
   -H 'accept: */*' \
   -H 'Content-Type: application/octet-stream' \
   -H 'Authorization: ZenApiKey <base64(zen_user_id:zen_api_key)>' \
   --data-binary "@/path/to/archive.jar"
   ```

2. Update the existing custom metadata, or set additional metadata (for example, metadata for machine learning providers) by using the following runtime REST API method:

   ```
   PUT https://<server>:<port>/ads/runtime/api/v1/deploymentSpaces/{deploymentSpaceId}/decisions/{decisionId}/metadata
   ```

   For example, you can use a cURL command:

   ```
   curl -X 'PUT' \
   'https://<server>:<port>/ads/runtime/api/v1/deploymentSpaces/{deploymentSpaceId}/decisions/{decisionId}/metadata' \
   -H 'accept: */*' \
   -H 'Content-Type: application/json' \
   -H 'Authorization: ZenApiKey <base64(zen_user_id:zen_api_key)>' \
   -d '@/path/to/metadata.json'
   ```

   Example of /path/to/metadata.json:

   ```
   {
     "map": {
         "metadata_1": {
         "kind": "{PLAIN|ENCRYPTED}",
         "readOnly": true|false,
         "value": "string_1"
       },
       ...,
       "metadata_N": {
         "kind": "{PLAIN|ENCRYPTED}",
         "readOnly": true|false,
         "value": "string_N"
       }
     }
   }
   ```

   You can also update the existing custom metadata or additional metadata by updating the created metadata file /path/to/metadata.json. The custom metadata is metadata that does not exist in the decision service archive. The decision service metadata is not mutable.

   For more information about the decision service metadata, see Decision service metadata.

3. Publish the new set of metadata by using the following runtime REST API method:

   ```
   curl -X 'PUT' \
   'https://<server>:<port>/ads/runtime/api/v1/deploymentSpaces/{deploymentSpaceId}/decisions/{decisionId}/metadata' \
   -H 'accept: */*' \
   -H 'Content-Type: application/json' \
   -H 'Authorization: ZenApiKey <base64(zen_user_id:zen_api_key)>' \
   -d '@/path/to/metadata.json'
   ```

Promoting deployed decision services to another decision runtime

You can promote a decision service archive from the development environment to the production environment.

About this task

For security reasons, the production environment where decisions are executed might need to contain only the decision runtime. In this case, the decision runtime can be isolated from the development environment where decisions are authored and tested.

As a consequence, you might need to promote a deployed decision service archive from the development environment to the production environment. You can promote decision service archives from the decision runtime on one installation to the decision runtime on another installation.

For more information about the decision runtime REST API methods that are used in the procedure, see the Decision runtime REST API reference in Reference.

Procedure

1. Get the decision service archive from the development environment by using the following decision runtime REST API method:

   ```
   GET https://<server>:<port>/deploymentSpaces/{deploymentSpaceId}/decisions/{decisionId}/archive
   ```

   - *<server>* is the hostname of the decision runtime, which is specified in the decision_runtime.ingress.hostname parameter.
   - *<port>* is the port of the decision runtime.

   For example, you can use a cURL command:

   ```
   curl -k -X 'GET' \
   'https://<server>:<port>/deploymentSpaces/{deploymentSpaceId}/decisions/{decisionId}/archive' \
   -H 'accept: application/octet-stream' \
   -H 'Authorization: ZenApiKey <base64(zen_user_id:zen_api_key)>'
   ```

2. Get the metadata from the development environment by using the following decision runtime REST API method:

   ```
   GET https://<server>:<port>/deploymentSpaces/{deploymentSpaceId}/decisions/{decisionId}/metadata > /path/to/metadata.json
   ```

   For example, you can use a cURL command:

   ```
   curl -X 'GET' \
   'https://<server>:<port>/deploymentSpaces/{deploymentSpaceId}/decisions/{decisionId}/metadata' \
   -H 'accept: application/json' \
   -H 'Authorization: ZenApiKey <base64(zen_user_id:zen_api_key)>'\
   > /path/to/metadata.json
   ```

3. Deploy them to the decision runtime in the production environment.
   For more information, see Deploying to decision runtime with REST API.

Executing decision services

Decision services are executed by using a Kubernetes service.
The decision runtime is a Kubernetes service for remotely executing decisions and managing decision service resources that are logically organized in deployment spaces.

You can also execute decision services by using the execution Java API.

A deployment space is a set of decision service archives and their associated metadata, and it is a logical way to group decisions. For example, a deployment space can be dedicated to a development phase (development, test, or production) or a group of users.

A set of metadata is associated to each decision service, which contains the following information:

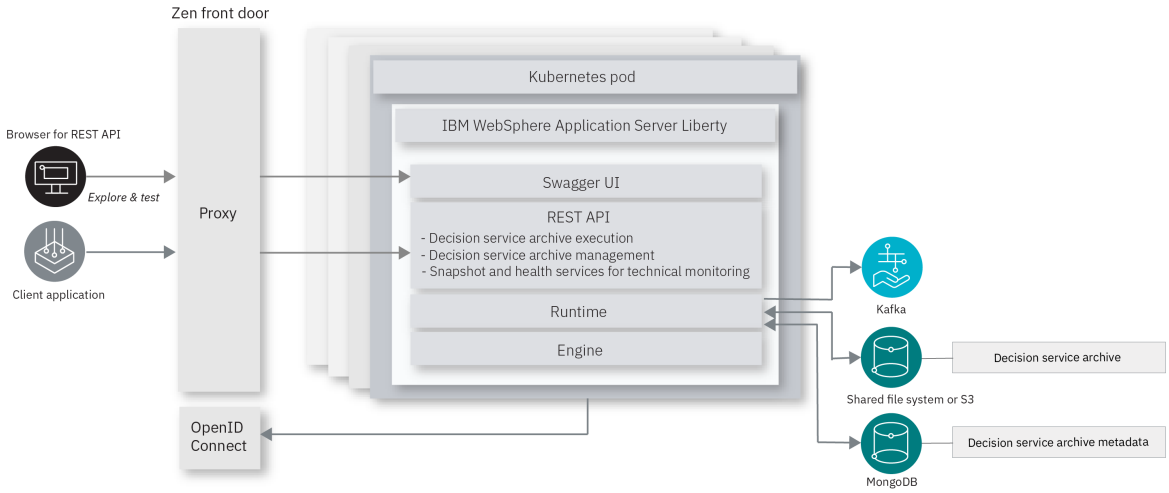
- Information about the decision service; for example, date of build, decision ID (decisionId), commit ID (commitId).
- Parameters that are used by the execution; for example, machine learning credentials
- User custom parameters

For more information about the metadata, see Decision service metadata.
The same decision service archive can be added to different deployment spaces. However, a separate set of metadata exists for each decision service archive in each deployment space. For example, it can be used to enable different credentials for the machine learning service.

The decision runtime comes with a set of REST API for the following purposes:

- Executing decisions
- Managing deployment spaces, decision service archives, and their associated metadata
- Retrieving a snapshot of the runtime state and configuration for diagnostic purposes
- Retrieving execution traces

The following figure shows the decision runtime architecture:



A decision runtime instance is deployed as a Kubernetes pod that is based on WebSphere® Application Server Liberty.

Each decision runtime instance is able to execute multiple distinct decisions. The runtime caches decision service archives and metadata to lower the cost of loading the decision service archives.

A deployed decision service archive cannot be modified. When a modifiable metadata is changed, the runtime automatically uses the new value after a configurable delay.

You can deploy any number of decision runtime replicas to scale and resist infrastructure failures. It can be done because they communicate only with the storage and do not need to communicate with each other.

The REST API and Swagger UI web application for the decision runtime are accessed externally through the IBM Cloud Pak Platform UI (Zen) front door. When single sign-on (SSO) is used, the Zen front door handles authentication and adds authorization information to the request that is sent to the decision runtime. These applications can use the Zen API key for authentication. For more information, see User permissions and authentication modes.

Kafka is an event framework to connect data systems and transmit large amount of data reliably and efficiently. The decision runtime can send an event that contains technical and business information for each decision to a Kafka topic. For more information, see Emitting decision execution events.

Samples and tutorials

Several samples and tutorials show you how to execute a decision service. These samples and tutorials are available in GitHub. Click Automation Decision Services samples to be redirected to the GitHub repository.

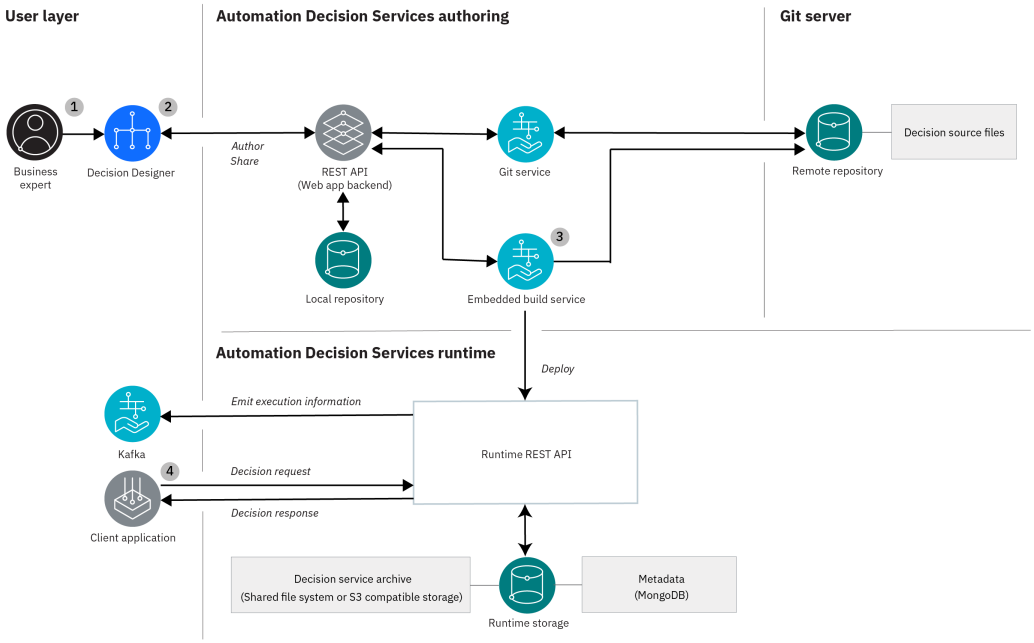
1. You can use the embedded build service in Decision Designer, and execute decision services by using the generated OpenAPI methods in the Swagger UI facility:
   - Task 6 in Getting started – You can use a basic decision service.
   - Tutorial: Using an external library – You can use a decision service that uses an external library.
   - Tutorial: Using machine learning to make better decisions – You can use a decision service that uses a predictive model.
   - Tutorial: Testing a decision service at build time – You can add unit tests to your decision service that are run at build time.
   - Sample: Exploring the execution trace – You can build an HTML page and explore execution traces.
2. You can use a CI/CD stack:
   - Sample: Loan validation application – This sample illustrates how to build a web application that calls a decision service. This decision service is deployed to a custom storage.
   - Archives: You can use the delivered decision service archives ready to be executed in the runtime.
3. You can use the execution Java API:
   - Sample: Executing a decision service in Java – You can execute decision services by using the execution Java API, and display execution traces.
   - Sample: Creating a web microservice for executing a decision service – You can create a web microservice to execute a decision service. You can use a decision service with or without a predictive model.

- **Decision services from the embedded build service**
  Decision services can be built and deployed in the embedded build service when you trigger it from Decision Designer. These decision services are deployed to the decision runtime for execution.
- **Decision services from a CI/CD stack**
  Decision services can be built and deployed in a CI/CD stack. These decision services are deployed to the decision runtime for execution.
- **Decision service metadata**
  Every decision service comes with a set of metadata. Use the decision runtime REST API to view or modify the decision service metadata that are not coming from the decision service archive.
- **Executing decision services with decision runtime**
  The decision runtime is a Kubernetes service for remotely executing decisions and managing decision service resources.
- **Executing decision services with execution Java API**
  You can execute decisions in a Java™ application that is running in a Kubernetes container by using the execution Java API for Automation Decision Services.

Decision services from the embedded build service

Decision services can be built and deployed in the embedded build service when you trigger it from Decision Designer. These decision services are deployed to the decision runtime for execution.
The build, deployment, and execution processes involving the embedded build service are composed of a series of tasks.

The following diagram shows a flow for execution:

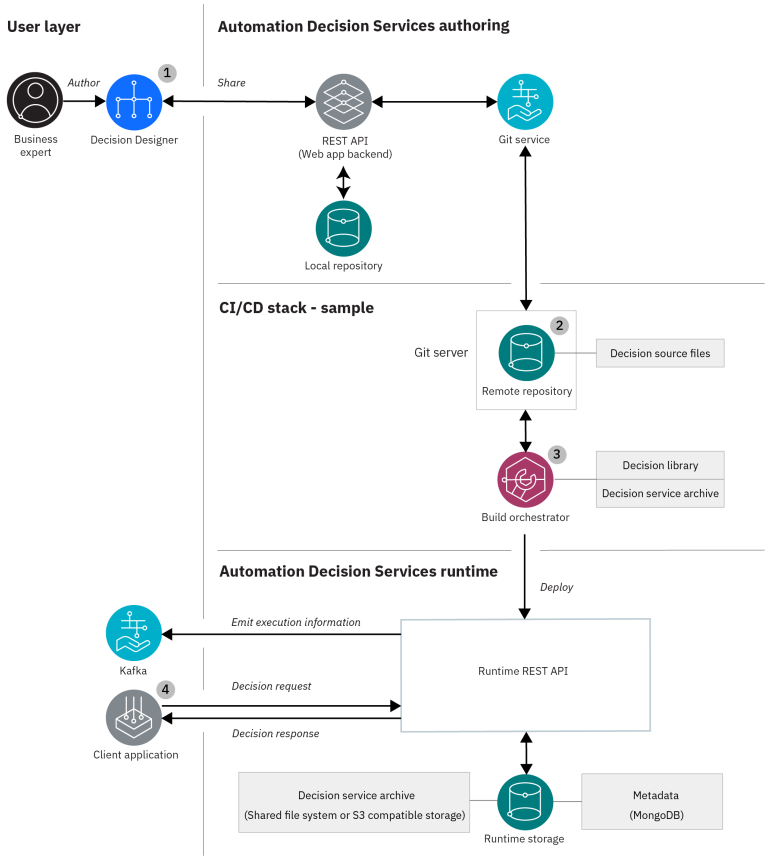


1. A business expert authors decision artifacts in Decision Designer and shares the decision artifacts.
2. Building and deploying a decision service is triggered from Decision Designer.
3. The embedded build service builds a decision service and deploys a decision service archive with metadata to the runtime service through its REST API. Decision service archives are deployed to the decision runtime in the deployment space ID (deploymentSpaceId) named `embedded`. The decision service is ready for execution.
4. The client application calls the decision service with one of the following methods:
   - The client application connects to the decision runtime and calls the decision service by using the decision runtime REST API. For more information, see Calling decision services.
   - You download the decision service archive from the decision runtime. The client application calls the decision service archive by using the execution Java API. For more information, see Executing decision services with execution Java API.

Decision services from a CI/CD stack

Decision services can be built and deployed in a CI/CD stack. These decision services are deployed to the decision runtime for execution. The build, deployment, and execution processes involving a CI/CD stack are composed of a series of tasks.

The following diagram shows a flow for execution:



1. A business expert authors decision artifacts in Decision Designer and shares the decision artifacts.
2. The shared artifacts are pushed to a remote repository where the decision source files are stored.
3. The build orchestrator compiles the decision source files and links them into a decision library. The compiled artifacts are contained in the decision library and bundled into a decision service archive.
   The decision service archive can be deployed by using the decision runtime REST API. For more information, see Deploying to decision runtime with REST API.
4. The client application calls the decision service with one of the following methods:
   - The client application connects to the decision runtime and calls the decision service by using the decision runtime REST API. For more information, see Calling decision services.
   - The client application calls the decision service archive by using the execution Java API. For more information, see Executing decision services with execution Java API.

Decision service metadata

Every decision service comes with a set of metadata. Use the decision runtime REST API to view or modify the decision service metadata that are not coming from the decision service archive.
For more information, see the endpoints ending with **`/metadata`** in the decision runtime REST API in Reference.

Metadata in the decision service archive

Table 1 shows the metadata that exists in the decision service archive. This metadata is automatically available through the decision service management REST API, enabling its full identification and traceability across its entire lifecycle.

Note: You cannot edit or delete the decision service metadata.
Each metadata has a simple value of type: String.

Table 1. Decision service metadata

Name	Description	Example value
`contentVersion`	The version of this file.	`"1.0.0"`
`decisionId`	A globally unique identifier for the decision service archive.	`samples.LoanValidation:LoanValidationDecisionService:4.6.2.2`
`decisionServiceDescription`	The business description of the decision service.	This decision service evaluates loan requests and renders approval or denial outcomes.
`commitId`	The Git commit identifier for the source of the deployed version of the decision service archive.	`"39ff2ddde9ac2ee21c7d67b664beaf03524e0ed6"`
`buildTime`	The time at which the decision service archive was built in ISO 8601 representation.	`"2021-04-19T10:36:42Z"`
`deploymentSpaceId`	The identifier of the deployment space.	When you use the embedded build service to deploy the decision service archive, it is `"embedded"`.

Name	Description	Example value
`deploymentTime`	The time of the deployment of the decision service archive in the runtime.	`2021-11-03T11:36:46Z`
`mavenGroupId`	The Maven group ID of this archive.	`"samples.LoanValidation"`
`mavenArtifactId`	The Maven artifact ID of this archive.	`"LoanValidation"`
`mavenVersion`	The version of this archive given at build time.	`"4.6.2.2"`
`decisionServiceName`	The name of the decision service.	`"samples.LoanValidation"`
`decisionServiceId`	The identifier of the decision service.	`"LoanValidation"`
`decisionServiceVersion`	The version of the decision service.	`"4.6.2.2"`
`engineRuntimeVersion`	The version of the decision engine embedded in the decision service archive.	`"2.8.1"`
`engineAPIVersion`	The version of the decision engine API used when building this decision service archive.	`"2.8.0"`
`mlIntegrationRuntimeVersion`	The version of the machine learning runtime embedded in the decision service archive.	`"6.0.12"`
`deploymentSpaceHashCodeId`	A hexadecimal representation of the hashcode (SHA256) of `deploymentSpaceId`	`9289140B1AC28DBDA1437B283E6CA608E33186654E7D3A995DA268C35906CD4C`
`decisionHashCodeId`	A hexadecimal representation of the hashcode (SHA256) of `decisionId`	`E9C7618B3B7EF1FB99A93510C920227A297DE45F0EE2471BC58439CF922056E`

Mandatory metadata for invoking machine learning services

When a decision service needs to access to a machine learning provider, it must specify the required information by using the related metadata REST API. For more information about this REST API, see PUT /deploymentSpaces/{deploymentSpaceId}/decisions/{decisionId}/metadata method in the Decision runtime REST API reference in Reference.

The corresponding metadata must follow this pattern: `ml_provider_name` and `ml_provider_info_id`. For `ml_provider_name`, there are 3 cases: Open Prediction Service (OPS) provider, Watson® Machine Learning (WML) provider, and Amazon SageMaker.

OPS provider case:

```
"<ml_provider_name>": {
   "name": "<ml_provider_name>",
   "kind": "ENCRYPTED",
   "readOnly": false,
   "value": {
     "name": "<ml_provider_name>",
     "type": "OPS",
     "description": "A description",
     "version" : "V1",
     "authInfo": {
       "authType" : "NONE"
     },
     "providerAdditionalInfo": {
       "providerType" : "OPS",
       "mlUr" : "<your-OPS-url>"
     }
    }
  }
```

WML provider case:

```
"<ml_provider_name>": {
   "name": "<ml_provider_name>",
   "kind": "ENCRYPTED",
   "readOnly": false,
   "value": {
     "name":"<ml_provider_name>",
     "type":"WML",
     "description": "A description",
     "version" : "V1",
     "authInfo":{
       "authType":"IAM",
       "apiKey":"<your_apikey>",
       "authUrl":"https://iam.bluemix.net/identity/token"
     },
     "providerAdditionalInfo":{
       "providerType":"WML",
       "mlUrl":"<your_WML_url>",
       "spaceId":"<your-WML--spaceId>"
     }
   }
 }
```

AWS SageMaker case:

```
"<ml_provider_name>": {
   "name": "<ml_provider_name>",
   "kind": "ENCRYPTED",
   "readOnly": false,
   "value": {
     "name":"<ml_provider_name>",
     "type":"AWS",
     "description": "A description",
     "version" : "V1",
     "authInfo":{
```

```
      "authType":"AWS",
      "accessKeyId":"<your_access_key_id>",
      "accessKeyValue":"<your_access_key_value>"
    },
    "providerAdditionalInfo":{
      "providerType":"AWS",
      "region":"<your-aws-region>"
    }
  }
}
```

The following `ml_provider_info_id` pattern can be used for all provider cases:

```
"<ml_provider_info_id>": {
  "name": "<ml_provider_info_id>",
  "kind": "PLAIN",
  "readOnly": false,
  "value": {
     "id": "<ml_provider_info_id>",
     "providerId" : "<ml_provider_name>",
     "deploymentId": "<ml_deployment_id>"
  }
}
```

One such entry must exist for each machine learning provider that is required by the decision service. The two mandatory metadata are coupled, therefore the value for **<ml_provider_name>** must be the same.

You can find the values for the metadata (for example, **name**, **type**, or **mlUrl**) in the following locations:

- For **<ml_provider_name>**:
  <predictiveModelName>/resources/extannotations/decisions/*<decisionAutomationName>*/*<decisionServiceName>*/*<predictiveModelName>*/provider.json
- For **<ml_provider_info_id>**:
  <predictiveModelName>/resources/extannotations/decisions/*<decisionAutomationName>*/*<decisionServiceName>*/*<predictiveModelName>*/providerInfo.json

These JSON files are available in the remote Git repository that the decision service is connected to.
Note: The values for the secrets (for example, **apiKey**) are not available in the JSON file.
The values are the same only when you use the same machine learning provider as the one that is used in the authoring environment.

If a different machine learning provider is targeted, the **mlUrl**, **authUrl**, **instanceId**, and **apiKey** must be changed.

When you use a different machine learning provider, the same model must be created on this provider. This model has a different **deploymentId** as the one that is used in the authoring environment. Therefore, **deploymentId** in the **<ml_provider_info_id>** metadata must be set to the **deploymentId** that is effectively targeted.

Table 2. Machine learning provider metadata

Name	Description
name	The name of the machine learning provider. You can find the value for **name** in the JSON file that is described earlier.
type	Supported types: • **WML** for Watson Machine Learning • **OPS** for Open Prediction Service • **AWS** for AWS SageMaker
mlUrl	The URL of the provider that is extracted from IBM Watson™ Studio or IBM Open Prediction Service.
authUrl	If **mlUrl** points to a production IBM Cloud® (for example, **https://us-south.ml.cloud.ibm.com**), **authUrl** is **https://iam.bluemix.net/identity/token**. Currently, only applies to Watson Machine Learning.
spaceId	The **spaceId** that is extracted from the machine learning provider. Currently, only applies to Watson Machine Learning.
apiKey	The **apiKey** is obtained from the **https://cloud.ibm.com/iam/apikeys** service. It applies to Watson Machine Learning only.
id	The ID of the machine learning provider information. You can find the value for the ID in the JSON file that is described earlier.
providerId	The name of the machine learning provider. You can find the value for the name in the JSON file that is described earlier.
deploymentId	The ID of the deployment that is targeted for prediction on the machine learning provider.
region	The AWS region for your account. Applies to AWS SageMaker only.
accessKeyId	Your AWS access key ID. Applies to AWS SageMaker only.
accessKeyValue	Your AWS access key value. Applies to AWS SageMaker only.

Executing decision services with decision runtime

The decision runtime is a Kubernetes service for remotely executing decisions and managing decision service resources.
Important:
Consumption entitlement mode is used for reporting and metering decision service execution. For more information, see Metering and tracking usage for decision runtime.

- **Metering and tracking usage for decision runtime**
  IBM® License Service (ILS) reports information about decision services for metering, such as the number of successfully executed decision services. The metering mode for the consumption entitlement is configured by default.
- **User permissions and authentication modes**
  The decision runtime uses user permission and authentication modes to control access to REST API endpoints and manage decision service archives and their metadata.
- **Selecting decision services for execution**
  A client application can choose to execute a decision service with the most recently deployed version, the latest semantic version, the last sorted version, or a specific version. Your client application does not need to be changed or aware of the decision ID of the decision service to be executed.
- **Calling decision services**
  Your client application calls an endpoint through the decision runtime REST API to execute a decision service. The client calls the decision service by using the standard HTTP GET and POST commands that the web browsers use when they interact with remote servers.
- **Execution trace for decision runtime REST API**
  An execution trace shows an actual graph or tree of information that represents execution of a decision service. You can display an execution trace by specifying the executionTraceFilters parameter in the request body when your application calls a decision service with the decision runtime REST API.
- **Runtime snapshots**
  The decision runtime snapshot feature provides a detailed state of the decision runtime at the time the snapshot is taken.
- **Swagger UI for decision runtime API**
  The decision runtime REST API is accessible from the Swagger UI facility.
- **Emitting decision execution events**
  Kafka is an event framework to connect data systems and transmit large amount of data in an efficient and reliable way.

Metering and tracking usage for decision runtime

IBM® License Service (ILS) reports information about decision services for metering, such as the number of successfully executed decision services. The metering mode for the consumption entitlement is configured by default.

License Service

License Service discovers the software that is installed in your infrastructure, and generates reports:

1. You deploy Automation Decision Services to your Kubernetes cluster.
2. You deploy License Service to the same cluster.
3. You can start retrieving a report for the license usage.

For more information about License Service, see the License Service.

Consumption entitlement

The **`MONTHLY_API_CALL`** metric that is reported by the decision runtime represents the daily number of executed decisions. For example, if an application executes decision services 10,000 times per hour in one day, the metric shows 240,000 calls for that day.

The **`MONTHLY_API_CALL`** metric for Automation Decision Services includes only successful decision service executions that occurs within a day, which means any failed execution is excluded. Decision services that are executed after midnight are counted in the report for the following day.

The metric is refreshed every 10 minutes by default. You can modify the refresh interval by using the spec.decision_runtime.ils.flush_interval parameter.

Note: About Establishment:

- Establishment is not a metric that is tracked by License Service.
- An Establishment is a single physical site, including the surrounding campus and satellite offices located within 50 kilometers, of Licensee's site address.

  Licensee must obtain an entitlement for each Establishment at or for which the Program will be used. Licensee is permitted to deploy an unlimited number of copies of the Program within the Establishment. An entitlement for an Establishment is unique to that Establishment and may not be shared, nor may it be reassigned other than for the permanent closing of the Establishment.
- Establishment, and Monthly API Call entitlements are not alternative means for licensing the Program. Licensee must obtain the appropriate number of Establishment entitlements as well as the appropriate number of Monthly API Call.

Parameters for License Service

The spec.decision_runtime.ils.flush_interval parameter is used for License Service. For more information, see Decision runtime parameters.

Retrieving license usage

To view the license usage, retrieve the License Service token, and access the **`http://$LICENSING_URL/status?token=$TOKEN`** URL:

```
export TOKEN=$(kubectl get secret ibm-licensing-token -o jsonpath={.data.token} -n ibm-common-services | base64 -d)
```

You can also retrieve the license report in a **`ZIP`** file by running the following command:

```
curl -k https://$LICENSING_URL/snapshot?token=$TOKEN --output report.zip
```

For more information, see APIs.

User permissions and authentication modes

The decision runtime uses user permission and authentication modes to control access to REST API endpoints and manage decision service archives and their metadata.

User permissions

The following distinct permissions exist to control access to the various REST API endpoints.

Table 1. Permissions

User permissions	Description
`decision service user`	Users with this permission can execute decisions and invoke related endpoints. Examples of what related endpoints can do: • List the operations of a decision service. • Generate an OpenAPI specification for a decision service. • Retrieve an example payload for a decision service. • Generate the schemas of the input and output for a decision service. For more information, see the Decision runtime section for the Decision runtime REST API in Reference.
`decision service manager`	Users with this permission can manage the decision service archives and associated metadata by using the create, retrieve, update, and delete operations on their respective storage service. For more information, see the Decision storage management section for the Decision runtime REST API in Reference. Important: If you are using the embedded build service, you must have this permission type to build and deploy a decision service archive to the decision runtime.
`decision runtime monitor`	This role allows users to take a snapshot of the state of the decision runtime on demand. For more information, see the GET /health and GET /snapshot methods in the Decision runtime section for the Decision runtime REST API in Reference.

For more information about how to configure these permissions, see Configuring decision runtime.

Two endpoints are unprotected, that is, they do not require any permission to be used:

- `/about` endpoint - For more information, see the GET /about method in the Decision runtime section for the Decision runtime REST API in Reference.
- `/health` endpoint - For more information, see the GET /health method in the Decision runtime section for the Decision runtime REST API in Reference.

Predefined user roles

Predefined user roles are available for you to use in the IBM Cloud Pak Platform UI (Zen) console. Each predefined role corresponds to a permission that is defined in the Zen console.

Check Managing user permissions to find more information about predefined user roles and associated permissions, and how you can configure them in the Zen console.

For more general information about the Zen console, see Managing users in the IBM Cloud Paks documentation.

Table 2. Predefined user roles

Predefined role	Description	Associated permission
Decision User	Users with this role can perform actions that are allowed with the Execute decision services permission.	Execute decision services
Deployed Decision Manager	Users with this role can perform actions that are allowed with the Manage deployed decision services permission.	Manage deployed decision services
Decision Runtime Monitor	Users with this role can perform actions that are allowed with the Monitor decision runtime permission.	Monitor decision runtime
Decision Runtime Deployment Spaces Manager	Users with this role can perform actions that are allowed with the Manage deployment spaces permission.	Manage deployment spaces

Authentication modes

Two authentication modes are available for the decision runtime. They are specified with the ads_configuration.decision_runtime.authentication_mode parameter:

- `basic`
- `zen`

Table 3. Authentication modes

Authentication mode	Description
`basic`	Users who are authenticated through the basic authentication mode are granted permissions as they are configured.

Authentication mode	Description
`zen`	When **`zen`** mode is used, the decision runtime is accessible through the IBM Cloud Platform proxy (Zen) gateway, and single sign-on (SSO) is managed by Identity Access Management (IAM). Basic authentication can be used as well. Users and associated permissions can be managed in the IBM Cloud Pak Platform UI (Zen). For more information, see Managing user permissions. Client applications must use API keys. For more information about generating the API keys, see Generating API keys for authentication. For more information about invoking a decision service with an API key, see Authorizing HTTP requests by using the Zen API key.

Selecting decision services for execution

A client application can choose to execute a decision service with the most recently deployed version, the latest semantic version, the last sorted version, or a specific version. Your client application does not need to be changed or aware of the decision ID of the decision service to be executed.

Executing the most recently deployed, latest semantic, or last sorted version

When a new version of a decision service is deployed, it is automatically selected for execution. Your client application does not need to be changed or aware of the decision ID of the decision service to be executed.

The resolution of the selected decision is cached, and the cache update it subject to a configurable delay. You can configure this delay in the update_interval parameter. As a consequence, if a decision service is deployed just after the execution request of the previously deployed decision service, the previously deployed decision service might be executed instead of the last deployed decision service.

For more information, see ads_configuration.decision_runtime_service.decision_selection parameters in Configuration parameters.

Executing a specific version

Your client application does not need to be aware of the decision ID of the decision service to be executed. However, it needs to be aware of the deployed decision service versions, and specify which version to get selected.

This strategy can be used when a new deployment of a decision service is not necessarily used immediately. For example, you can use it when a new version applies discounts for certain items, and these discounts can be used on a specified date.

- **Executing the last deployed version**
  When the last deployed version of a decision service is requested to be executed, the decision service with a specified decision service identifier and the most recent deployment time is selected.
- **Executing the last semantic version**
  When the last semantic version of a decision service is requested to be executed, the decision service with a specified decision service identifier and the latest deployment version in semantic versioning order is selected.
- **Executing the last sorted version**
  When the last sorted version of a decision service is requested to be executed, the decision service with a specified decision service identifier and the latest deployment version in alphanumeric order is selected.
- **Executing a specific version**
  When a specific version of a decision service is requested to be executed, the decision service with a specified decision service identifier and a specified decision version is selected.

Executing the last deployed version

When the last deployed version of a decision service is requested to be executed, the decision service with a specified decision service identifier and the most recent deployment time is selected.
The most recent deployment time is determined based on the value in the metadata **`deploymentTime`**.

For more information about the endpoint to execute the last deployed version of a decision service, see the Last deployed version execution section in the Decision runtime REST API reference in Reference.

Important: For more information, see also the **Executing the most recently deployed, latest semantic, or last sorted version** section in Selecting decision services for execution
.

Executing the last semantic version

When the last semantic version of a decision service is requested to be executed, the decision service with a specified decision service identifier and the latest deployment version in semantic versioning order is selected.

Semantic versioning for decision services

Semantic versioning is a versioning format that is used in a number of open source projects. This format is a formal convention to determine the version number of a new software release and the severity of changes in each new release. For more information about the semantic versioning, see https://semver.org/.

You can use the semantic versioning format to version decision services for the decision runtime. You specify versions for your decision services before they are deployed. For more information, see Creating versions.

See the semantic versioning format:

`[major].[minor].[patch]-[pre-release]+[build metadata]`

The decision runtime supports only a subset of the semantic versioning format, and does not support the build metadata:

Table 1. Supported semantic versioning format in the decision runtime

Version	Supported in decision runtime	Mandatory or optional	Supported type	Description
Major	Yes	Mandatory	Number	It indicates that you make incompatible API changes.
Minor	Yes	Mandatory	Number	It indicates that you make changes that are compatible with an earlier version.
Patch	Yes	Mandatory	Number	It indicates that you make bug fixes compatible with an earlier version.
Pre-release	Yes	Optional	String If you try to deploy a decision service with a per-release value that is numerical and not a string, you have an `HTTP Bad Request` return code, and the decision service cannot be deployed.	It indicates the state of a release. For example, you can set a string **`rc.2`** as your second release candidate. The decision runtime supports only the string format for the pre-release version. A semantic version with a release value is sorted after a version that does not have a pre-release version. For example, version **`2.0.1`** is greater than version **`2.0.1-rc.2`**.
Build metadata	No	-	-	If your decision service has a version with build metadata, the information about the build metadata is not taken into account. The decision service is deployed without any error or warning.

The semantic versioning allows you to keep track of changes and evolutions of your decision services; therefore, you can create several releases of a decision services.

When you try to deploy a decision service that does not have a version in semantic versioning format, the decision service is deployed but a warning is returned after its deployment.

When you use the decision runtime REST API for executing a decision service that contains any version in nonsemantic versioning format, a warning is returned for the first nonsemantic version of this decision service.

Using the semantic versioning for execution

The latest deployment version is determined based on the value in the metadata **`decisionServiceVersion`**.

For more information about the endpoint to execute the last semantic version of a decision service, see the Last semantic version execution section in the Decision runtime REST API reference in Reference.

The semantic versioning selector selects the latest semantic version of a specific deployment service for the REST API requests. The major, minor, and patch numbers are compared numerically.

For example:

- Version **`2.2.1`** is greater than version **`2.2.0`**
- Version **`2.3.0`** is greater than version **`2.2.1`**
- Version **`3.0.0`** is greater than version **`2.3.0`**

You can also indicate the major and minor numbers to select a specific version of a decision service in the REST API operations.

For example, if you have the same decision service that is deployed with five different versions: **`4.0.0`**, **`3.0.0`**, **`2.2.1`**, **`2.2.0`**, and **`2.1.0`**:

- If you set the major number to **`2`** and the minor number to **`2`**, then the semantic versioning selector selects version **`2.2.1`**.
- If you set only the major number to **`3`**, version **`3.0.0`** is selected for the decision service.
- If you set neither the major nor the minor numbers, the last semantic version is selected, which is version **`4.0.0`**.

Important: For more information, see also the **Executing the most recently deployed, latest semantic, or last sorted version** section in Selecting decision services for execution
.

Executing the last sorted version

When the last sorted version of a decision service is requested to be executed, the decision service with a specified decision service identifier and the latest deployment version in alphanumeric order is selected.
The latest deployment version is determined based on the value in the metadata **`decisionServiceVersion`**.

For more information about the endpoint to execute the last sorted version of a decision service, see the Last sorted version execution section in the Decision runtime REST API reference in Reference.

Note:
If you plan to use numerical versions, consider using the last semantic version execution.

For example:

- The last sorted version execution: Version `9.0.0` is considered greater than version `10.0.0`.
- The last semantic version execution: Version `10.0.0` is considered greater than version `9.0.0`, which is usually the expected behavior.

For more information about the semantic version execution, see Executing the last semantic version.
Important: For more information, see also the **Executing the most recently deployed, latest semantic, or last sorted version** section in Selecting decision services for execution
.

Executing a specific version

When a specific version of a decision service is requested to be executed, the decision service with a specified decision service identifier and a specified decision version is selected.
The decision service version selection is based on the value in the **`decisionServiceVersion`** metadata. For example, it can be "**`1.0`**", "**`2.5.3-SNAPSHOT`**", or "**`RELEASE`**". The decision runtime looks up the exact match of what you specify in the metadata.

For more information about the endpoint to execute a specific version of a decision service, see Decision service metadata and the Specific version execution section for the Decision runtime REST API in Reference.

Calling decision services

Your client application calls an endpoint through the decision runtime REST API to execute a decision service. The client calls the decision service by using the standard HTTP GET and POST commands that the web browsers use when they interact with remote servers.
A client application calls a decision service by creating a secure connection with a server, and then calling the decision service through the REST API.

For further information:

- about parameters, see Configuration parameters.
- about the exact URL patterns for calling decision services, see the Decision runtime REST API reference in Reference.
- about metadata, see Decision service metadata.

Sections in this topic:

Endpoint to execute with a decision ID (decisionId)

The endpoint URI for a decision service uses the following format:

```
POST https://<zen_hostname>:
<zen_port>/ads/runtime/api/v1/deploymentSpaces/{deploymentSpaceId}/decisions/{decisionId}/operations/{operation}/extendedExecut
e
```

- *<zen_hostname>* is the hostname of the decision runtime.
- *<zen_port>* is the port of the decision runtime.

Table 1. Parameters that are used in the endpoint URI

Parameter	Mandatory or Optional	Description
deploymentSpaceId	Mandatory	URL encoded identifier that uniquely identifies the place where a decision was deployed. For more information, see Decision service metadata.
decisionId	Mandatory	URL encoded identifier that uniquely identifies a decision to execute. For more information, see Decision service metadata. If you are using the embedded build service, you can view the value of decisionId in the Deploy tab in Decision Designer. For more information, see Building and deploying from Decision Designer.
operation	Mandatory	Name of the operation. It is used to identify which decision service to execute within the decision service archive. The operation name is decided at authoring time.

Endpoint to execute the last deployed version of a decision service

The endpoint URI that selects a decision service based on their last deployment time uses the following format:

```
POST https://<zen_hostname>:
<zen_port>/ads/runtime/api/v1/selectors/lastDeployedDecisionService/deploymentSpaces/{deploymentSpaceId}/operations/{operation}
/extendedExecute?decisionServiceId={decisionServiceId}
```

The deployment time of a decision service refers to the **`deploymentTime`** metadata of the decision service.

- *<zen_hostname>* is the hostname of the decision runtime.
- *<zen_port>* is the port of the decision runtime.

Table 2. Parameters that are used in the endpoint URI

Parameter	Mandatory or Optional	Description
deploymentSpaceId	Mandatory	URL encoded identifier that uniquely identifies the place where a decision was deployed.
operation	Mandatory	Name of the operation. It is used to identify which decision service to execute within the decision service archive. The operation name is decided at authoring time.
decisionServiceId	Mandatory	Query parameter that identifies the decision service from which the decision service archive is built. The decision service ID is decided at authoring time. Its value is that of the **decisionServiceId** metadata of the decision service. For more information about how to view decisionServiceId, see Decision service metadata.

Endpoint to execute the last semantic version of a decision service

The endpoint URI that selects a decision service based on their last semantic version uses the following format.

Extended execution:

```
POST https://<zen_hostname>:
<zen_port>/ads/runtime/api/v1/selectors/lastSemanticDecisionServiceVersion/deploymentSpaces/{deploymentSpaceId}/operations/{operation}/extendedExecute
```

Simple execution with simplified input and output payloads:

```
POST https://<zen_hostname>:
<zen_port>/ads/runtime/api/v1/selectors/lastSemanticDecisionServiceVersion/deploymentSpaces/{deploymentSpaceId}/operations/{operation}/execute
```

- *<zen_hostname>* is the hostname of the decision runtime.
- *<zen_port>* is the port of the decision runtime.

Table 3. Parameters that are used in the endpoint URI

Parameter	Mandatory or Optional	Description
deploymentSpaceId	Mandatory	URL encoded identifier that uniquely identifies the place where a decision was deployed.
operation	Mandatory	Name of the operation. It is used to identify which decision service to execute within the decision service archive. The operation name is decided at authoring time.
decisionServiceId	Mandatory	Query parameter that identifies the decision service from which the decision service archive is built. The decision service ID is decided at authoring time. Its value is that of the **decisionServiceId** metadata of the decision service.
majorVersion	Optional	Query parameter that identifies the major version number of the decision service. For more information about semantic versions, see Executing the last semantic version.
minorVersion	Optional	Query parameter that identifies the minor version number of the decision service.

Endpoint to execute the last sorted version of a decision service

The endpoint URI that selects a decision service based on their sorted version uses the following format:

```
POST https://<zen_hostname>:
<zen_port>/ads/runtime/api/v1/selectors/lastSortedDecisionServiceVersion/deploymentSpaces/{deploymentSpaceId}/operations/{operation}/extendedExecute?decisionServiceId={decisionServiceId}
```

The version of a decision service refers to the **decisionServiceVersion** metadata of the decision service.

- *<zen_hostname>* is the hostname of the decision runtime.
- *<zen_port>* is the port of the decision runtime.

Table 4. Parameters that are used in the endpoint URI

Parameter	Mandatory or Optional	Description
deploymentSpaceId	Mandatory	URL encoded identifier that uniquely identifies the place where a decision was deployed.
operation	Mandatory	Name of the operation. It is used to identify which decision service to execute within the decision service archive. The operation name is decided at authoring time.
decisionServiceId	Mandatory	Query parameter that identifies the decision service from which the decision service archive is built. The decision service ID is decided at authoring time. Its value is that of the **decisionServiceId** metadata of the decision service.

Endpoint to execute a specific version of a decision service

The endpoint URI that selects a specific version of a decision service uses the following format:

```
POST https://<zen_hostname>:
<zen_port>/ads/runtime/api/v1/selectors/decisionServiceVersion/deploymentSpaces/{deploymentSpaceId}/operations/{operation}/exte
ndedExecute?decisionServiceId={decisionServiceId}&decisionServiceVersion={decisionServiceVersion}
```

- *<zen_hostname>* is the hostname of the decision runtime.
- *<zen_port>* is the port of the decision runtime.

Table 5. Parameters that are used in the endpoint URI

Parameter	Mandatory or Optional	Description
deploymentSpaceId	Mandatory	URL encoded identifier that uniquely identifies the place where a decision was deployed.
operation	Mandatory	Name of the operation. It is used to identify which decision service to execute within the decision service archive. The operation name is decided at authoring time.
decisionServiceId	Mandatory	Query parameter that identifies the decision service from which the decision service archive is built. The decision service ID is decided at authoring time. Its value is that of the **decisionServiceId** metadata of the decision service.
decisionServiceVersion	Mandatory	Query parameter that identifies the version of the decision service. Its value is that of the **decisionServiceVersion** metadata of the decision service.

Request body

You can use the decision runtime REST API to call decisions by using the JSON payload.

The request body has the following format:

```
{
  "executionId": "<value_1>",
  "input": "<value_2>",
  "executionTraceFilters": { ... }
}
```

Table 6. Parameters that are used in the request body

Parameter	Mandatory or Optional	Description
executionId	Optional	Optional identifier for the execution of the decision service. If this parameter is not set, the decision runtime creates a decision execution identifier that is globally unique.
input	Mandatory	JSON object that represents input data of the decision service. The JSON object can be obtained from the Run part in Decision Designer by editing the JSON data set.
executionTraceFilters	Optional	JSON object that specifies which traces are activated in which format. If this parameter is not set, the execution trace is **null** as a result. For more information about getting execution traces, see Execution trace for decision runtime REST API.

An example of the request body:

```
{
  "executionId": "executionId_0001",
  "input": {
    "customer": {
      "name": "Mary",
      "age": 20
    },
    "ageLimit": 21
  },
  "executionTraceFilters": {
    "executionDuration": true,
    "printedMessages": true,
    "decisionModel": {
      "inputParameters": "Object",
      "outputParameters": "Object",
      "inputNode": "Object",
      "outputNode": "Object"
    },
    "rules": {
      "boundObjectsAtStart": "Object",
      "boundObjectsAtEnd": "Object",
      "allRules": true,
      "executedRules": true,
      "nonExecutedRules": true
    },
    "ruleflow": {
      "allTasks": false,
      "executedTasks": false,
      "notExecutedTasks": false,
      "selectedRules": false
    }
  }
}
```

HTTP return codes and execution results

When your client application calls a decision service, you get an HTTP return code. The response body is formatted in JSON payload.

Table 7. HTTP return codes

HTTP return code	Description
200	The decision operation was successfully executed. The output of the decision is returned in the response body.
400	The decision operation could not be executed due to a problem in the input payload.
401	The decision operation could not be performed due to an authentication problem.
403	No sufficient privileges to execute the decision. For more information, see User permissions and authentication modes.
404	The decision service or decision operation was not found.
500	The decision operation could not be performed due to an internal error.

The JSON body of the response has the following format:

```
{
  "decisionId": "<value_5>",
  "decisionOperation": "<value_6>",
  "executionId": "<value_7>",
  "executionTrace": "<value_8>",
  "output": "<value_9>",
  "incident": {
    "incidentID": "<incident_ID>",
    "incidentCategory": "<error_message>",
    "stackTrace": "<xyzException>:<stack_message>"
  }
}
```

- executionTrace might be null if no execution trace filter is set.
- output might be null if the execution is not completed due to an error.
- incident is returned only when errors occur.

For more information about the execution trace and the incident, see the Schema section in the Decision runtime REST API reference in Reference.
The following example shows a response body with the HTTP return code 200 (the execution was successful):

```
{
  "decisionId": "decisions/My-Getting-Started/mySalutationDecisionService/1.0.0-SNAPSHOT/mySalutationDecisionService-1.0.0-
20200120.153649-11.jar",
  "decisionOperation": "myModel",
  "executionId": "24543a29-ce83-4e20-afbc-93fccf7949d5",
  "executionTrace": null,
  "output": "Good evening Prof. Jones!"
}
```

The following example shows a response body with the HTTP return code 500 (the execution failed):

```
{
  "decisionId": "decisions/My-Getting-Started/mySalutationDecisionService/1.0.0-SNAPSHOT/mySalutationDecisionService-1.0.0-
20200120.153649-11.jar",
  "decisionOperation": "myModel",
  "executionId": "1f10653a-c21f-4653-854e-c6e56da4bdaf",
  "executionTrace": null,
  "output": null,
  "incident": {
    "incidentId": "edee977a-5d2e-47ca-b31e-4e866649016c",
    "incidentCategory": "Decision error",
    "stackTrace": "com.fasterxml.jackson.databind.exc.UnrecognizedPropertyException: Unrecognized field \"nme\" (class
newtype),  ...  }
}
```

Execution trace for decision runtime REST API

An execution trace shows an actual graph or tree of information that represents execution of a decision service. You can display an execution trace by specifying the executionTraceFilters parameter in the request body when your application calls a decision service with the decision runtime REST API.
The executionTraceFilters parameter is a JSON object that specifies which traces are activated in which format.

If this parameter is not set, the execution trace is `null` as a result.

The format for the request body in JSON payload:

```
{
  "executionId": "<value_1>",
  "input": "<value_2>",
  "executionTraceFilters": { ... }
}
```

For more information about the parameters, see the Schemas section for a corresponding REST API method in the Decision runtime REST API reference in Reference.

Note: If you want to include an exception stack trace in the response payload when an incident occurs, you must set the decisionRuntimeService.stackTraceEnabled parameter to `true` at the time of installing the product. By default, this parameter is set to `false`.

Sample

The sample `Exploring the execution trace` is available in GitHub. With this sample, you can build an HTML page and explore execution traces. It also contains a sample request body for displaying execution traces. Click Automation Decision Services samples to be redirected to the GitHub repository.

Runtime snapshots

The decision runtime snapshot feature provides a detailed state of the decision runtime at the time the snapshot is taken.
The snapshot feature can be used for many different purposes. For example:

- To diagnose problems in the decision runtime by providing detailed information on its state.
- To collect information related to an issue that users are facing.
- To obtain basic metrics on the decision runtime and its subcomponents.
- To identify performance issues, bottlenecks, and system resources sinks.

For more information about the REST API method for snapshots, see the GET /snapshot method in the Decision runtime REST API reference in Reference.
Remember: Using the snapshot feature can have an impact on performance. You must not call the GET /snapshot method frequently.
The runtime snapshot feature has two main parts: static and dynamic properties.

Static properties

The static properties concern the configuration properties of the decision runtime and its environment. These properties are fixed during the execution of the decision runtime, and not going to be changed.

The runtime properties are listed as follows:

- Authentication and SSL parameters
- MongoDB database and decision archive storage parameters
- Emitter parameters
- Logging parameters

The runtime environment properties are listed as follows:

- Operating system and environment properties
- JVM properties
- Different class paths (**`bootClasspaths`**, **`runtimeClasspaths`**, and **`libraryPaths`**)

Dynamic properties

The dynamic properties concern the properties of the runtime during its execution:

- Memory usage
- Archives and metadata cache states
- Garbage collectors states
- Statistics concerning the emitter events
- A dump of the runtime threads that are running in the decision runtime.
  The stack traces of the threads are shown only if the **`ads_configuration.decision_runtime_service.stack_trace_enabled`** parameter is set to **`true`**.
  For more information about the parameter, see Configuration parameters.

Swagger UI for decision runtime API

The decision runtime REST API is accessible from the Swagger UI facility.
You can access the Swagger UI facility with the following URL:

`https://<server>:<port>/ads/runtime/api/swagger-ui/`

The resulting web page provides a user interface to execute decision service archives intuitively. For each execution request, the Swagger interface displays the resulting HTTP return code, along with the body of the response in the JSON format. The Swagger interface also generates and displays an equivalent CURL command for each execution request.

It also provides a user interface to explore the data model of the decision runtime REST API. For more information about the decision runtime REST API endpoints, see the Decision runtime REST API in Reference (available in the Swagger UI view).

Emitting decision execution events

Kafka is an event framework to connect data systems and transmit large amount of data in an efficient and reliable way.

Because Kafka is an open source event framework, it can consume events by using various technologies for different purposes, such as:

- Real time technical or business monitoring
- Alerts
- Logging and auditing
- Data analytics

You can configure the decision runtime to send an event that contains technical and business information for each decision to a Kafka topic.

Examples of technical information:

- Decision identification
- Execution date
- Execution duration

Examples of business information:

- Input of the decision
- Output of the decision

For more information about the configuration, see Configuring the event emitter for an external Kafka instance.

Event data as JSON objects

The decision execution evens are JSON objects with attributes:

Table 1. Events

Field	Description	Type
`data.inputParameters`	Input parameters of the decision.	JSONObject
`data.outputParameters`	Output parameters of the decision.	JSONObject
`decisionId`	Identifier for the decision.	String
`decisionOperation`	Name of the decision operation.	String
`decisionServiceId`	Business identifier of the decision service.	String
`decisionServiceName`	Business name of the decision service.	String
`decisionServiceVersion`	Version of the decision service.	String
`deploymentSpaceId`	Identifier of the deployment space of the decision.	String
`engineApiVersion`	Version of the decision engine API that is used by the decision runtime.	String
`executionId`	Unique identifier for this execution of the decision.	String
`executionMillis`	Duration of the decision execution in milliseconds,	Long
`executedRules`	List of rules that were executed.	Array of strings
`executionSuccess`	Specifies whether the execution of the decision was successful.	Boolean
`incidentCategory`	Type of incident that occurred during the execution of the decision	String
`incidentStackTrace`	Stack trace for the incident (if available).	String
`nonExecutedRules`	List of rules that were not executed.	Array of strings
`runtimeNodeName`	Hostname of the decision runtime.	String
`timestamp`	Start time of the decision execution request.	Long

Example of an emitted event

See the following example of a business event for a decision execution:

```
{
   "decisionId":"/samples/LoanValidation/LoanValidationDecisionService/3.1.2/LoanValidationDecisionService-3.1.2.jar",
   "decisionOperation":"LoanValidation",
   "decisionServiceId":"samples.LoanValidation",
   "decisionServiceName":"LoanValidation",
   "decisionServiceVersion":"3.1.2",
   "deploymentSpaceId":"test_environment",
   "engineApiVersion":"2.5.0",
   "data":{
      "inputParameters":{
         "customer":{
            "age":1,
            "name":"<name>"
         },
         "ageLimit":1
      },
      "outputParameters":"PASSED(mySimple02CustomerDecision)"
   },
   "executedRules":[
      "/LoanValidation/Duration/duration",
      "/LoanValidation/Bankruptcy Score/bankruptcy score",
      "/LoanValidation/Salary Score/salary score",
      "/LoanValidation/Yearly interest rate/yearly interest rate",
      "/LoanValidation/Corporate Score/corporate score",
      "/LoanValidation/Repayment/repayment",
      "/LoanValidation/Grade/grade",
      "/LoanValidation/Insurance/insurance"
   ],
   "executionId":"b4707973-ee80-45cc-bff4-86e3f2b79432",
   "executionMillis":3,
   "executionSuccess":true,
   "incidentCategory":null,
   "incidentId":null,
   "incidentStackTrace":null,
   "nonExecutedRules":[
      "/LoanValidation/Approval/bad score",
      "/LoanValidation/Approval/risky grade",
      "/LoanValidation/Approval/too big Debt to Income ratio",
      "/LoanValidation/Approval/age not valid",
      "/LoanValidation/Approval/no name",
```

```
      "/LoanValidation/Approval/wrong SSN",
      "/LoanValidation/Approval/wrong SSN format",
      "/LoanValidation/Approval/wrong zip format",
      "/LoanValidation/Approval/max amount exceeded"
   ],
   "runtimeNodeName":"worker0.mycluster.mycompany.com",
   "timestamp":1605602384767
}
```

Metadata

Some decision service metadata control the emission of business events:

Table 2. Metadata

Name	Description	Default Value
`businessEventInputEnabled`	Controls the emission of input parameters in the business data.	true
`businessEventOutputEnabled`	Controls the emission of output parameters in the business data.	true
`businessMonitoringEnabled`	Controls the emission of business monitoring events.	true

Executing decision services with execution Java API

You can execute decisions in a Java™ application that is running in a Kubernetes container by using the execution Java API for Automation Decision Services.
The execution Java API does not use or depend on any installed services in Automation Decision Services. It does not use the decision runtime REST API or decision service storage. Instead, decision service archives are directly used in the execution API. These decision service archives can be packaged in your Java application or accessed with a URL.

For more information, see the Execution Java API reference in Reference.

Important:
You must use IBM License Metric Tool to monitor the number of execution. For more information, see Monitoring executions with IBM License Metric Tool.

Environment

The execution API must be used in an environment in Java 17. The Java libraries that are used by the external libraries for your decision services must be compatible with the Java version (Java 17) for the execution.

Restriction: The execution Java API is compatible with decision service archives that are generated in Automation Decision Services V22.0.2 or later.

Samples

Samples that use the execution Java API are available in GitHub. Click Automation Decision Services samples to be redirected to the GitHub repository.

- Sample: Executing a decision service in Java (available in the ExecutionApiSample subfolder) – You can execute decision services by using the execution Java API, and display execution traces.
- Sample: Creating a web microservice for executing a decision service (available in the WebMicroServiceSample subfolder) – You can create a web microservice to execute a decision service. You can use a decision service with or without a predictive model.

- **Monitoring executions with IBM License Metric Tool**
  When you use the execution Java™ API, you must use IBM License Metric Tool (ILMT) to monitor decision service execution.
- **Application packaging**
  You need to add certain JAR files and other items in the class path in your applications that use the execution Java™ API.
- **Executing a decision operation**
  You execute a decision operation that is packaged in a decision service archive by using the execution Java™ API.
- **Input and output format**
  You can use JSON for input and output for executed decision operations.
- **Machine learning services**
  When a decision service needs to access machine learning services, it must specify certain parameters for each service in the value of the key `decisionMedatata` in the run context of the `DecisionRunner` instance.
- **Execution trace for execution Java API**
  You might want to retrieve execution traces of an engine. You define what information is recorded in the trace by using trace configuration.

Monitoring executions with IBM License Metric Tool

When you use the execution Java™ API, you must use IBM License Metric Tool (ILMT) to monitor decision service execution.

For more information about IBM License Metric Tool, see IBM License Metric Tool information.

The number of fully executed decision services must be measured by IBM License Metric Tool to check whether the number exceeds the authorized maximum number of executions.

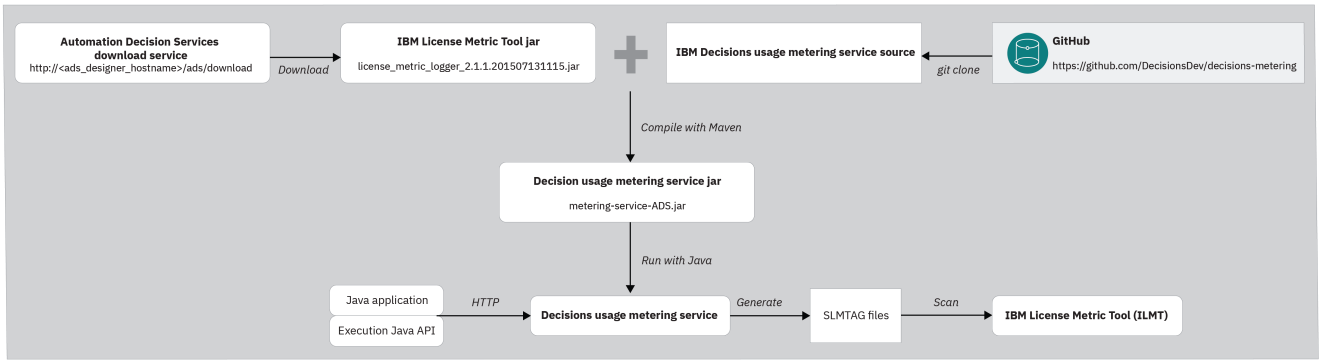
The architecture of the IBM License Metric Tool monitoring is composed of three main components:

- A client inside the execution Java API that connects to a decision usage metering service. You must configure the client to enable monitoring by IBM License Metric Tool, and to connect the client to the decision usage metering service.
- A decision usage metering service that registers a subscription from each node on which the execution Java API is running. Then, the bridge receives the count for executed decision services regularly, and buffers it in memory and database. After a certain period, these metrics are stored in an XML file that is readable by the

IBM License Metric Tool scanners.

- IBM License Metric Tool reads the generated XML files on each node, and exposes it as a report on its administration console. The metric name to check is `MONTHLY_API_CALL`.

Consumption entitlement for executing decision services in pure Java environment



Parameters

The following parameters are used to configure the decision usage metering service client:

Table 1. Decision usage metering service client configuration parameters

Parameter	Description	Default value
Decision usage metering service URI	The base URI to connect to the decision usage metering service.	Mandatory value
Instance identifier	The instance identifier of the execution Java API that is installed on a specific node of a cluster. This identifier must start with the '/' character.	Mandatory value
Flush interval	The period between the metrics that are sent to the bridge. The value is in number of seconds.	300 seconds
SSL context	The SSL context provided by the user to connect securely the execution Java API to the decision usage metering service.	The default SSL context of the execution Java API.
Connection timeout	The client connection timeout to connect to the decision usage metering service.	By default, the timeout is infinite.

Sample

Check the following sample that executes a decision service and enables the monitoring with IBM License Metric Tool:

```
import myModel.Borrower;
import myModel.Loan;
import samples.LoanValidation.LoanValidationOutput;
import samples.LoanValidation.loanValidation.Input;

private static long flushInterval = 60L;

[..]

ILMTConfiguration configuration = new ILMTConfiguration.Builder()
            .withConnectionTimeout(10)
            .withFlushInterval(flushInterval)
            .withILMTBridgeURI(new URI("http://localhost:8888"))
            .withInstanceIdentifier("/myInstanceIdentifier")
            .build();

DecisionRunnerProvider provider = new ClassLoaderDecisionRunnerProvider.Builder().enableMetering(configuration).build();

DecisionRunner runner = provider.getDecisionRunner("LoanValidation");

Input in = new Input();

Borrower borrower = new Borrower();
borrower.setZipCode("12345");
borrower.setCreditScore(700);
borrower.setFirstName("John");
borrower.setLastName("Doe");
borrower.setYearlyIncome(100000);
in.borrower = borrower;

Loan loan = new Loan();
loan.setAmount(185000);
loan.setNumberOfMonthlyPayments(72);
in.loan = l;

LoanValidationOutput out = (LoanValidationOutput) runner.execute(in);
```

Get the swidtag file (ibm.com_IBM_Automation_Decision_Services-23.0.2.swidtag) from *<DECISION_DESIGNER_SERVER_URL>*/download/, and rename it to ibm.com_IBM_Automation_Decision_Services.swidtag. Finally, add it to the application classloader.

Configuring the usage metering service

The usage metering service produces license files that are compliant with IBM License Metric Tool.

Information about the usage metering service for the execution Java API is available on the following GitHub repository: https://github.com/DecisionsDev/decisions-metering.

Application packaging

You need to add certain JAR files and other items in the class path in your applications that use the execution Java™ API.
The following items must be added in the class path:

- execution-api.jar
- engine-de-api.jar
- jackson-databind version 2.15.3 and its dependencies
- jackson-datatype-jsr310 version 2.15.3

Use the <DECISION_DESIGNER_SERVER_URL>/download/index.json file to determine which version of each Automation Decision Services JAR files to be used.

The execution-api.jar and engine-de-api.jar files are not deployed to the Maven central repository by default.

You can configure Maven to connect to Decision Designer and use it as a Maven repository. You can then get execution-api.jar and its dependencies from the Maven repository.

For more information about using Decision Designer as a Maven repository and downloading the JAR files, see Setting up your build environment.

Executing a decision operation

You execute a decision operation that is packaged in a decision service archive by using the execution Java™ API.
For more information about the API, see the Execution Java API reference manual in Reference.

1. Getting a DecisionRunnerProvider instance
   Get a **`DecisionRunnerProvider`** instance that corresponds to the way the decision service archives are accessed.
2. Getting a DecisionRunner instance
   Get a **`DecisionRunner`** instance that corresponds to the decision operation from the **`DecisionRunnerProvider`** instance.
3. Executing a decision operation by using Java objects
   Execute the decision operation by using **`DecisionRunner`**.

Getting a `DecisionRunnerProvider` instance

Your Java application must have access to decision service archives for execution.

If the decision service archives are accessible at give URLs, you can use **`URLDecisionRunnerProvider`**:

```
URL url = ...;
DecisionRunnerProvider provider = new URLDecisionRunnerProvider
    .Builder()
    .addURL(urlToDSA1)
    .addURL(urlToDSA2)
    .build();
```

If the decision service archives are added to the class path of the application, you can use **`ClassLoaderRunnerProvider`**:

```
DecisionRunnerProvider provider = new ClassLoaderDecisionRunnerProvider
    .Builder()
    .build();
```

The lookup strategy for the decision service archive is to first search in the thread context class loader, then in the class loader that loads the classes of the API.
You can also specify the class loader that contains the decision service archive:

```
ClassLoader cl = ..; // classloader containig the DSA
DecisionRunnerProvider provider = new ClassLoaderDecisionRunnerProvider
    .Builder()
    .withClassLoader(cl)
    .build();
```

For an instance of **`DecisionRunnerProvider`**, the engine runtime version for all decision service archives must be the same.

Getting a `DecisionRunner` instance

A **`DecisionRunner`** instance is dedicated to a particular decision operation. It is the main class for executing a decision operation.

You use the **`DecisionRunnerProvider`** instance to get the **`DecisionRunner`** instance that corresponds to a given operation from the accessible decision service archives.

To obtain a **`DecisionRunner`** instance for the decision operation **`operationName`**:

```
String operationName = ..; // the name of an operation
DecisionRunner runner = provider.getDecisionRunner(operationName);
```

Executing a decision operation by using Java objects

To execute a decision operation named **LoanValidation** by using Java objects as input and output:

```
DecisionRunnerProvider provider = ...;
DecisionRunner runner = provider.getDecisionRunner("LoanValidation");
Object in = ...;
Object out = runner.execute(in);
```

Here is a complete example for executing a decision operation of the decision service `Loan Approval` in the Banking sample decision automation. Add the decision service archive loanApproval-*<version>*.jar to the application class path. This decision service archive is in the archives folder in the GitHub repository for samples and tutorials. For more information, see Samples and tutorials in GitHub.

```
import com.ibm.decision.run.DecisionRunner;
import com.ibm.decision.run.provider.ClassLoaderDecisionRunnerProvider;
import com.ibm.decision.run.provider.DecisionRunnerProvider;
import decisions.techpreview_samples.loan_validation.loanvalidationdata.Borrower;
import decisions.techpreview_samples.loan_validation.loanvalidationdata.Loan;
import decisions.techpreview_samples.loan_validation.loanvalidationdecisionmodel.Approval_decision_modelOutput;
import decisions.techpreview_samples.loan_validation.loanvalidationdecisionmodel.approval_decision_model.Input;

[..]

DecisionRunnerProvider provider = new ClassLoaderDecisionRunnerProvider.Builder().build();

DecisionRunner runner = provider.getDecisionRunner("approval");

Input in = new Input();

Borrower borrower = new Borrower();
borrower.setZipCode("12345");
borrower.setCreditScore(700L);
borrower.setFirstName("John");
borrower.setLastName("Doe");
borrower.setYearlyIncome(100000L);
in.borrower = borrower;

Loan loan = new Loan();
loan.setAmount(185000L);
loan.setNumberOfMonthlyPayments(72L);
in.loan = loan;

Approval_decision_modelOutput out = (Approval_decision_modelOutput)runner.execute(in);
```

Alternatively, the input can be created from a map of objects instead of a specific input Java class:

```
import com.ibm.decision.run.DecisionRunner;
import com.ibm.decision.run.provider.ClassLoaderDecisionRunnerProvider;
import com.ibm.decision.run.provider.DecisionRunnerProvider;
import decisions.techpreview_samples.loan_validation.loanvalidationdata.Borrower;
import decisions.techpreview_samples.loan_validation.loanvalidationdata.Loan;
import decisions.techpreview_samples.loan_validation.loanvalidationdecisionmodel.Approval_decision_modelOutput;

[..]

DecisionRunnerProvider provider = new ClassLoaderDecisionRunnerProvider.Builder().build();

DecisionRunner runner = provider.getDecisionRunner("approval");

Map<String, Object> inParams = new HashMap<String, Object>();

Borrower borrower = new Borrower();
borrower.setZipCode("12345");
borrower.setCreditScore(700L);
borrower.setFirstName("John");
borrower.setLastName("Doe");
borrower.setYearlyIncome(100000L);
inParams.put("borrower", borrower);

Loan loan = new Loan();
loan.setAmount(185000L);
loan.setNumberOfMonthlyPayments(72L);
inParams.put("loan", loan);

Approval_decision_modelOutput out = (Approval_decision_modelOutput) runner.execute(runner.createInput(inParams));
```

Sample

The sample `Executing a decision service in Java` is available in GitHub. Click Automation Decision Services samples to be redirected to the GitHub repository.

Input and output format

You can use JSON for input and output for executed decision operations.
A specific decision runner called **JSONDecisionRunner** provides facilities to use JSON for input and output of the executed decision operation:

```
String opName = "LoanValidation";

// Get the specific DecisionRunner for using JSON as input/output
JSONDecisionRunner runner = provider.getDecisionRunner(opName, JSONDecisionRunner.class);

// create the input object from a JSON string
```

```
String inAsJSON = ...;
Object in = runner.getInputReader().readValue(inAsJSON);

Object out = runner.execute(in);

// create the JSON string representation of the output object
String outASjson = runner.getOutputWriter().writeValueAsString(out);
```

This example shows a complete code snippet for executing a decision operation named `LoanValidation` in a decision service that is packaged in the application class loader. Input and output are JSON strings:

```
// Create a decision runner provider for decision service archive in the application class loader
DecisionRunnerProvider provider = new ClassLoaderDecisionRunnerProvider
    .Builder()
    .build();

// Get the decision operation runner
JSONDecisionRunner runner = provider.getDecisionRunner("LoanValidation", JSONDecisionRunner.class);

// Get an example of input
Object inExample = runner.createInputExample();
// Convert the example to a JSON string
String inExampleAsJSON = runner.getInputWriter().writeValueAsString(inExample);

// Convert the example JSON string to a Java object
Object in = runner.getInputReader().readValue(inExampleAsJSON);

// Execute the decision operation
Object out = runner.execute(in);
// Convert the output Java object to a JSON string
String outAsJSON = runner.getOutputWriter().writeValueAsString(out);
```

Machine learning services

When a decision service needs to access machine learning services, it must specify certain parameters for each service in the value of the key `decisionMedatata` in the run context of the `DecisionRunner` instance.
The value corresponding to the `decisionMedata` is an `ObjectNode`. Each of this field corresponds to a machine learning service.

The following example passes parameters for the machine learning service called `myMLService`:

```
DecisionRunner runner = ..;

String executionId = ..; // the unique identifier of the execution

String providerInfoId = "rfc-info";
String providerId = "lml"; // the name of the ML provider
String providerInfo = "{\"id\": \"+ providerInfoId + \", \"providerId\": \" + providerId + \", \"deploymentId\": \"v0\"}";
String provider = "{ \"mlUrl\": \"http://0.0.0.0:8080\", \"type\": \"OPS\", \"name\": \" + providerId + \", \"instanceId\":
\"v0\"}"

ObjectMapper mapper = (ObjectMapper) context.get(ObjectMapper.class.getName());
Object inData = ..;

ObjectNode decisionMetadata = mapper.createObjectNode();
decisionMetadata.put(providerInfoId, providerInfo);
decisionMetadata.put(providerId, provider);

RunContext context = runner.createRunContext(executionId);
context.put("decisionMetadata", decisionMetadata);

runner.execute(inData, context);
```

`providerInfoId` must be the same as the one in the providerInfo.json file in the resources/extannotations/<packageName>/ folder for the predictive model.

- If the execution uses the same deployment of the same machine learning provider as the one that is used in the authoring environment, the provider information can be found in the provider.json file in the resources/extannotations/<packageName>/ folder for the predictive model, except for `apiKey` if it is of the type `WML`. This machine learning deployment must exist on the same machine learning provider. It might be checked either in the Automation Decision Services authoring environment or in the machine learning provider interface.
- When you use another deployment, `deploymentId` and `mlUrl` must be changed accordingly. If the provider type is `WML`, then `authUrl`, `apiKey`, and `instanceId` must be filled as well. The `instanceId` is the `spaceId` for Watson® Machine Learning, where the machine learning model is deployed.

For more information about parameters for machine learning services, see the Mandatory metadata for invoking machine learning services section in Decision service metadata.

Optionally, you can specify `SSLContext` that is used to communicate with the machine learning services:

```
import javax.net.ssl.SSLContext;
[..]
context.put(SSLContext.class.getName(), sslContext);
```

Execution trace for execution Java API

You might want to retrieve execution traces of an engine. You define what information is recorded in the trace by using trace configuration.

You need to set the trace configuration **com.ibm.decision.run.trace.TraceConfiguration** for the run context before the execution, as shown in the following example:

```
RunContext runContext = runner.createRunContext(executionId);
TraceConfiguration tc = new TraceConfiguration();
tc.rules.allRules = true;
tc.rules.executedRules = true;
tc.rules.nonExecutedRules = true;
runContext.setTraceConfiguration(tc);
runner.execute(input, runContext);
Trace trace = runContext.getTrace();
```

The information in the trace depends on the trace configuration and the type of execution unit. For example, if you have the trace of a ruleflow engine, and the trace configuration is set correctly, you can get a list of non-executed rules by writing:

```
trace.rootRecord.properties.get("nonExecutedRules")
```

For more information, see the classes **com.ibm.decision.run.trace.Trace**, **com.ibm.decision.run.TraceRecord**, and **com.ibm.decision.run.trace.TraceConfiguration** in the Execution Java API reference in Reference.

Sample

The sample `Executing a decision service in Java` is available in GitHub. It contains a sample code that displays execution traces. Click Automation Decision Services samples to be redirected to the GitHub repository.

Reference

Use the reference topics to help you develop, test, and execute decision services.

- **Configuration parameters**
  Configuration parameters are used to install Automation Decision Services on Kubernetes.
- **Languages reference**
  Automation Decision Services provides two languages: the rule language and the Advanced Rule Language (ARL).
- **Annotations Java API reference**
  Use the annotations Java API for adding annotations to your external libraries.
- **Unit testing Java API reference**
  Use the unit testing Java API for creating and annotating unit tests.
- **Execution Java API reference**
  Use the execution Java API for executing decision services.
- **Decision runtime REST API reference**
  Use the decision runtime REST API for executing and managing decision services. This link opens Swagger UI.

Configuration parameters

Configuration parameters are used to install Automation Decision Services on Kubernetes.

If a parameter is absent or has no value, then it means that the operator and Automation Decision Services pattern refer to the default value. You can overwrite the default value by entering a new value in your custom resource.

The following topics contain the parameters for configuring Automation Decision Services.

- **Decision Designer parameters**
- **Decision runtime parameters**
- **Shared parameters by Decision Designer and the decision runtime**

Decision Designer parameters

If a parameter is absent or has no value, then it means that the operator and Automation Decision Services pattern refer to the default value. You can overwrite the default value by entering a new value in your custom resource.

Table 1. **spec.decision_designer** parameters

Parameters	Description	Default/Example values
enabled	A flag to control whether Decision Designer is deployed or not.	The default value is **false**. When the enabled parameter for Decision Designer is set to **true**, then the enabled parameter for the decision runtime is set to **true** by default. If the enabled parameter for the decision runtime is **false** and the enabled parameter for Decision Designer is true, an error message is displayed.
admin_secret_name	The name of the secret that contains all the Decision Designer sensitive data.	ibm-dba-ads-designer-secret

Parameters	Description	Default/Example values
deployment_profile_size	A flag to control the default CPU/memory resources and replica count to be applied to Decision Designer services and pods. Possible values are: small, medium, large, and extra-large. Note: The horizontal pod autoscaler is configured with an extra-large profile size.	small
git_servers_certs	The name of a ConfigMap that contains the Git server public certificates that Automation Decision Services must trust. Each certificate is composed of a key name that ends with .crt and a value that is the PEM-encoded text of the certificate.	`null` (optional)
ml_providers_certs	The name of a ConfigMap that contains the machine learning providers public certificates that Automation Decision Services must trust. Each certificate is composed of a key name that ends with .crt and a value that is the PEM-encoded text of the certificate.	`null` (optional)
mount_pvc_for_logstore	If set to false, the logs of the Decision Designer (including embedded runtime) are not stored in the PVC and only sent to `stdout`.	`false`
other_trusted_certs	The name of a ConfigMap that contains external public certificates that Automation Decision Services must trust. Each certificate is composed of a key name that ends with .crt and a value that is the PEM-encoded text of the certificate.	`null` (optional)

Table 2. Other `spec.decision_designer` parameters

Parameters	Description	Default/Example values
credentials_service.image.repository	The registry and namespace where the credentials container image is located.	cp.icr.io/cp/cp4a/ads/ads-credentials
credentials_service.replica_count	The number of desired replica of the credentials container.	2
credentials_service.resources.limits.memory	The memory limit for the credentials container.	2048
credentials_service.resources.limits.cpu	The CPU limit for the credentials container.	2000
credentials_service.resources.limits.ephemeral_storage	The ephemeral storage limit for the credentials container.	For details depending on profile size, see System requirements
credentials_service.resources.requests.memory	The memory requested for the credentials container.	512
credentials_service.resources.requests.cpu	The CPU requested for the credentials container.	500
credentials_service.resources.requests.ephemeral_storage	The ephemeral storage requested for the credentials container.	For details depending on profile size, see System requirements
embedded_build_service.executor_count	The number of Jenkins executors on the built-in node.	5
embedded_build_service.image.repository	The registry and namespace where the embedded build container image is located.	cp.icr.io/cp/cp4a/ads/ads-build
embedded_build_service.replica_count	The number of desired replica of the embedded build container.	2
embedded_build_service.resources.limits.memory	The memory limit for the embedded build container.	2Gi
embedded_build_service.resources.limits.cpu	The CPU limit for the embedded build container.	2000
embedded_build_service.resources.limits.ephemeral_storage	The ephemeral storage limit for the embedded build container.	For details depending on profile size, see System requirements
embedded_build_service.resources.requests.memory	The memory requested for the embedded build container.	512
embedded_build_service.resources.requests.ephemeral_storage	The ephemeral storage requested for the embedded build container.	For details depending on profile size, see System requirements
embedded_build_service.resources.requests.cpu	The CPU requested for the embedded build container.	500
git_service.image.repository	The registry and namespace where the git container image is located.	cp.icr.io/cp/cp4a/ads/ads-gitservice
git_service.replica_count	The number of desired replica of the git container.	2
git_service.resources.limits.memory	The memory limit for the git container.	2048
git_service.resources.limits.cpu	The CPU limit for the git container.	2000
git_service.resources.limits.ephemeral_storage	The ephemeral storage limit for the git container.	For details depending on profile size, see System requirements
git_service.resources.requests.memory	The memory requested for the git container.	512
git_service.resources.requests.cpu	The CPU requested for the git container.	500
git_service.resources.requests.ephemeral_storage	The ephemeral storage requested for the git container.	For details depending on profile size, see System requirements
parsing_service.autoscaling.enabled	Specifies whether horizontal pod autoscaling is enabled or not.	Default value is false, except in the case of an extra-large deployment profile size.
parsing_service.autoscaling.max_replicas	Maximum number of replicas.	5
parsing_service.autoscaling.min_replicas	Minimum number of replicas.	2
parsing_service.autoscaling.target_cpu_average_utilization	Determines when a new pod is created, based on the value of `resources.requests.cpu`.	Default value is 160.
parsing_service.image.repository	The registry and namespace where the parsing container image is located.	cp.icr.io/cp/cp4a/ads/ads-parsing
parsing_service.replica_count	The number of desired replica of the parsing container.	2
parsing_service.resources.limits.memory	The memory limit for the parsing container.	2048
parsing_service.resources.limits.cpu	The CPU limit for the parsing container.	2000

Parameters	Description	Default/Example values
parsing_service.resources.limits.ephemeral_storage	The ephemeral storage limit for the parsing container.	For details depending on profile size, see System requirements
parsing_service.resources.requests.memory	The memory requested for the parsing container.	512
parsing_service.resources.requests.cpu	The CPU requested for the parsing container.	500
parsing_service.resources.requests.ephemeral_storage	The ephemeral storage requested for the parsing container.	For details depending on profile size, see System requirements
rest_api.http_client_timeout	Internal HTTP client timeout in milliseconds. (optional)	Default value is 60000.
rest_api.image.repository	The registry and namespace where the rest api container image is located.	cp.icr.io/cp/cp4a/ads/ads-restapi
rest_api.replica_count	The number of desired replica of the rest api container.	2
rest_api.resources.limits.memory	The memory limit for the rest api container.	2048
rest_api.resources.limits.cpu	The CPU limit for the rest api container.	2000
rest_api.resources.limits.ephemeral_storage	The ephemeral storage limit for the rest api container.	For details depending on profile size, see System requirements
rest_api.resources.requests.memory	The memory requested for the rest api container.	512
rest_api.resources.requests.cpu	The CPU requested for the rest api container.	500
rest_api.resources.requests.ephemeral_storage	The ephemeral storage requested for the rest api container.	For details depending on profile size, see System requirements
run_service.autoscaling.enabled	Specifies whether horizontal pod autoscaling is enabled or not.	Default value is false, except in the case of an extra-large deployment profile size.
run_service.autoscaling.max_replicas	Maximum number of replicas.	5
run_service.autoscaling.min_replicas	Minimum number of replicas.	2
run_service.autoscaling.target_cpu_average_utilization	Determines when a new pod is created, based on the value of `resources.requests.cpu`.	Default value is 160.
run_service.image.repository	The registry and namespace where the run container image is located.	cp.icr.io/cp/cp4a/ads/ads-run
run_service.replica_count	The number of desired replica of the run container.	2
run_service.resources.limits.memory	The memory limit for the run container.	2048
run_service.resources.limits.cpu	The CPU requested for the run container.	2000
run_service.resources.limits.ephemeral_storage	The ephemeral storage limit for the run container.	For details depending on profile size, see System requirements
run_service.resources.requests.memory	The memory requested for the run container.	512
run_service.resources.requests.cpu	The CPU requested for the run container.	500
run_service.resources.requests.ephemeral_storage	The ephemeral storage requested for the run container.	For details depending on profile size, see System requirements

Decision runtime parameters

If a parameter is absent or has no value, then it means that the operator and Automation Decision Services pattern refer to the default value. You can overwrite the default value by entering a new value in your custom resource.

Table 1. `spec.decision_runtime` parameters

Parameters	Description	Default/Example values
enabled	A flag to control whether the decision runtime must be deployed or not.	The default value is `false`. When the enabled parameter for Decision Designer is set to `true`, then the enabled parameter for the decision runtime is set to `true` by default. If the enabled parameter for the decision runtime is `false` and the enabled parameter for Decision Designer is true, an error message is displayed.
admin_secret_name	The name of the secret that encapsulates all the sensitive data of the decision runtime.	`ibm-dba-ads-runtime-secret`
archive_storage_type	The type of decision archive storage. Possible values are: • `fs` sets the decision archive to be stored on the file system. • `s3` sets the decision archive to be stored on an S3 (Simple Storage Service) object storage system. Note: This value cannot be changed after decision archives have been deployed to the runtime.	fs
authentication_mode	The authentication mode to access the runtime. Either `basic` or `zen`.	`zen`
decision_runtime_service.fs_group	Sets the `securityContext.fsGroup` field of pods instantiated for the corresponding services. For more information, see the Kubernetes documentation.	This parameter is optional. Type `int` (positive or null).

Parameters	Description	Default/Example values
deployment_profile_size	A flag to control the default CPU/memory resources and replica count to be applied to decision runtime services and pods. Possible values are: small, medium, large, and extra-large. This parameter allows you to override the Cloud Pak **`shared_configuration.sc_deployment_profile_size`** parameter. Note: The horizontal pod autoscaler is configured with an extra-large profile size.	small
event_emitter.allow_missing_events	Determines the behavior of the event emitter when its internal queue is full. • If true, then events are dropped when the queue is full. • If false, then event emission is synchronized with the execution of the decision, slowing it down as a consequence. In all cases, failure to emit an event does not prevent a decision from running.	true
event_emitter.enabled	Enables the event emitter in the decision runtime to send events to a Kafka cluster.	false
event_emitter.dequeuer_threads	Specifies how many threads are polling the events queue to send the events to the Kafka topic. If this parameter is not set, the number of threads is computed at the decision runtime start based on the number of CPU allocated to the runtime.	null
event_emitter.kafka_bootstrap_servers	A comma-separated list of host/port pairs that is used for establishing the initial connection to the Kafka cluster. For example: **`host1:port1,`** and **`host2:port2`**. For more information, see the Kafka documentation.	Mandatory if the event emitter is enabled.
event_emitter.kafka_connection_secret_name	Name of a Kubernetes secret that contains the user credentials for authentication on the Kafka cluster. When the **`kafka_sasl_mechanism`** parameter is set to **`PLAIN`** or **`SCRAM-SHA-512`**, the secret must contain keys **`kafka-username`** and **`kafka-password`**. Optionally, the secret can contain a key **`kafka-server-certificate`** associated to a PEM-encoded TLS certificate that allows to establish TLS trust when connecting to the Kafka cluster. In all cases, The TLS trust certificates configured with other parameters such as **`spec.trusted_certificate_list`** and **`spec.decision_runtime.decision_runtime_service.tls.certs_config_map_name`** in the decision runtime are also used when connecting to the Kafka cluster.	Mandatory when the **`kafka_sasl_mechanism`** parameter is set to **`PLAIN`** or **`SCRAM-SHA-512`**.
event_emitter.kafka_properties_config_map_name	Some custom Kafka client parameters can be added the same way as in https://kafka.apache.org/documentation/#producerconfigs.	null
event_emitter.kafka_sasl_mechanism	The SASL mechanism that is used for user authentication on the Kafka cluster when the parameter **`kafka_security_protocol`** is set to **`SASL_SSL`**. Valid values: • **`PLAIN`** • **`SCRAM-SHA-512`**	**`PLAIN`**
event_emitter.kafka_security_protocol	Protocol that is used to communicate with the Kafka cluster. Valid values: • **`PLAINTEXT`** for communication with no TLS encryption and no user authentication • **`SASL_SSL`** for communication with TLS encryption and a SASL-based user authentication.	**`SASL_SSL`**
event_emitter.kafka_topic	Name of the Kafka topic that is used to send events.	Mandatory if the event emitter is enabled.
event_emitter.queue_capacity	The decision runtime queues its events before it sends them asynchronously to the kafka topic. This parameter controls how many events can be contained in the queue.	50000
resources.limits.ephemeral_storage	The ephemeral storage limit for the decision runtime container.	For details depending on profile size, see System requirements
resources.requests.ephemeral_storage	The ephemeral storage requested for the decision runtime container.	For details depending on profile size, see System requirements
s3.bucket_name	The name of the s3 bucket / container.	N/A
s3.region	The connection region depends on the storage platform and type (local, regional, or multi-regional) and on the connection type (public, private, or direct).	For example, for a public regional connection in Europe on IBM Cloud, you can choose **`eu-de`**
s3.storage_region	On IBM Cloud, the storage region depends on the type of connection region and of bucket storage (standard, long-term, ...).	For example, for a **`standard`** bucket storage type and **`eu-de`** connection region, the storage region is set to **`eu-de-standard`**. This parameter is optional with a MinIO server.
s3.server_url	The URL of the S3 storage system. The connection URL depends on the platform, the type of region (multi-region, unique region, or local region), and the type of endpoint (public, private, direct).	For example, on IBM Cloud: **`https://s3.eu-de.cloud-object-storage.appdomain.cloud`**
s3.secret_name	The name of the secret that contains the credentials to connect to the S3 storage system.	If the secret is not set, an anonymous connection is attempted.

Parameters	Description	Default/Example values
s3.connection_timeout	The S3 connection timeout, in milliseconds. A value of `0` means an infinite timeout and is not recommended.	This setting is handled by the Amazon S3 SDK. For more information, see https://aws.amazon.com/sdk-for-java/. The default value is 10000.
s3.request_timeout	A zero value or negative value disables this feature.	This setting is handled by the Amazon S3 SDK. For more information, see https://aws.amazon.com/sdk-for-java/. The default value is 0.
storage_decision_check_status_interval	The interval, in milliseconds, between checks and updates of the cached decisions. Type: integer	The default value is 30000.

Table 2. `spec.decision_runtime.decision_runtime_service` parameters

Parameters	Description	Default/Example values
autoscaling.enabled	Specifies whether horizontal pod autoscaling is enabled or not.	Default value is false, except in the case of an extra-large deployment profile size.
autoscaling.max_replicas	Maximum number of replicas.	5
autoscaling.min_replicas	Minimum number of replicas.	2
autoscaling.target_cpu_average_utilization	Determines when a new pod is created, based on the value of `resources.requests.cpu`.	Default value is 160, which means that a new pod is created when the CPU usage is more than 1.6 the value of `resources.requests.cpu`.
cache.config.expiry	An optional XML fragment to configure how decision service archives expire in the cache. Three expiry types are supported: • time to live (`ttl`): the maximum time an entry can remain in the cache • time to idle (`tti`): the maximum time an entry can remain untouched • no expiry (`none`): entries do not expire (the default) For example, to configure a time-to-live expiry of 15 minutes: `<expiry>` `  <ttl unit="minutes">15</ttl>` `</expiry>` The value of the unit attribute is one of "nanos", "micros", "millis", "seconds", "minutes", "hours", or "days".	none
cache.config.resources	Specifies the maximum size of the decision service archives cache. The cache size can be expressed as a number of entries, or as a memory size. When the maximum value is reached, one or more entries are evicted from the cache. For example, to configure a cache with at most 100 entries: `<heap unit="entries">100</heap>` To configure a cache that cannot occupy more than 250 MB of memory: `<heap unit="MB">250</heap>` The value of the unit attribute is one of "entries", "B", "kB", "MB", "GB", "TB", "PB".	`<heap unit="entries">100</heap>`
decision_selection.cache.config.expiry	Same as `cache.config.expiry` parameter.	Same as `cache.config.expiry` parameter.
decision_selection.cache.config.resources	Same as `cache.config.resources` parameter.	Same as `cache.config.resources` parameter.
decision_selection.threads	The maximum number of threads that perform updates of the decision selection results cache concurrently.	1
decision_selection.update_interval	The interval, in milliseconds, between decision selection results cache updates.	120000 (2 minutes)
decision_selection.query_interval	The interval, in milliseconds, between the execution of decision selection storage queries in the same thread. This delay is intended to reduce the possible bursts of MongoDB operations when the selection results cache has a large number of entries.	1000 (1 second)
image.digest	The image digest name of the container, if you prefer to use the digest instead of the tag. Digest has priority over tag.	
image.repository	The registry and namespace where the container images are located.	cp.icr.io/cp/cp4a/ads/ads-runtime
metadata.cache.config.check_interval	Integer. The interval, in milliseconds, between two metadata cache updates.	30000
metadata.cache.config.expiry	An optional XML fragment to configure how metadata expire in the cache. The syntax is identical to `cache.config.expiry`.	none
metadata.cache.config.manager.threads	Integer. The number of threads that perform metadata cache updates.	1
metadata.cache.config.resources	Specifies the maximum size of the metadata cache. The syntax is identical to `cache.config.resources`.	For more information about profile size values, see System requirements

Parameters	Description	Default/Example values
persistence.resources.requests.storage	The size of the Persistent Volume (PV) requested through a Persistent Value Claim (PVC) for the runtime instance. A string that represents a Kubernetes byte quantity. The PVC refers to the storage class that is defined by **`shared_configuration.storage_configuration.sc_fast_file_storage_classname`**, if applicable. The required minimum performance for the volume is 300 IOPS, as defined on IBM Cloud. Applicable only when S3 compatible storage is configured.	
persistence.storage_class_name	If **`decision_runtime_service.persistence.use_dynamic_provisioning`** is true, define the name of the storage class that is requested for the persistent volume of the runtime. Applicable only when S3 compatible storage is configured.	
persistence.use_dynamic_provisioning	Boolean. Controls the type of persistent storage claim that is created for the runtime. Applicable only when S3 compatible storage is configured.	true
replica_count	The number of desired replicas of the container. Ignored if **`autoscaling.enabled`** is set to true.	Default value is 1 in starter mode, 2 in any other installation mode.
resources.limits.cpu	Specifies the CPU limit for the container.	2
resources.limits.memory	Specifies the memory limit for the container.	2Gi
resources.requests.cpu	Specifies the CPU request for the container.	1
resources.requests.memory	Specifies the memory request for the container.	512Mi
stack_trace_enabled	A flag that controls whether the decision runtime includes an exception stack trace in the response payload when an incident occurs.	**`false`**
tls.allow_self_signed	A flag that controls whether self-signed certificates are allowed or not to connect to decision storage or ML providers.	**`false`**
tls.certs_config_map_name	The name of the ConfigMap that holds the TLS certificates.	**`null`** (optional)
tls.enabled	A flag that controls whether TLS is required for decision storage or for ML providers connections (HTTPS protocol).	Not present. TLS is supported in all cases when connecting to ML providers.
tls.verify_hostname	A flag that controls whether hostnames of TLS certificates are verified to connect to decision storage or ML providers.	**`false`**

Table 3. License Service parameters for decision runtime

Parameters	Description	Default/Example values
spec.decision_runtime.ils.flush_interval	The number of seconds between each attempt to send a metric to License Service.	Default value is **`600`**.

Shared parameters by Decision Designer and the decision runtime

Some parameters can be shared by Decision Designer and the decision runtime.

If a parameter is absent or has no value, then it means that the operator and Automation Decision Services pattern refer to the default value. You can overwrite the default value by entering a new value in your custom resource.

Table 1. **`spec`** parameters

Parameters	Description	Default/Example values
deployment_profile_size	A flag to control the default CPU/memory resources and replica count to be applied to Automation Decision Services services and pods. Possible values are: • **`small`** • **`medium`** • **`large`** • **`extra-large`** This setting applies to the entire Automation Decision Services installation if nothing is specified in Designer or Runtime sections. You can override this setting for Decision Designer and/or the decision runtime component. Note: The horizontal pod autoscalers are configured with an **`extra-large`** profile size. IE: same as designer or runtime + extra comment / maybe add links to Designer & Runtime topics nearby	**`small`**
image_pull_policy	Pull policy for all containers.	IfNotPresent
image_pull_secrets	Shared image pull secrets.	[]

Parameters	Description	Default/Example values
run_as_user	Specify the user to run the security context of the pod. The value is usually a number that corresponds to a user ID. • For Cloud Native Computing Foundation (CNCF) platforms such as Amazon Web Services (AWS) and Google Kubernetes Engine (GKE), a value is required for this parameter. • On OpenShift Container Platform (OCP) and Red Hat OpenShift Kubernetes Service (ROKS), this parameter is not required.	None
storage_classname	Default storage class name used for Zen and Automation Decision Services. Can be overridden in specific sections, such as **`spec.zen.block_storage_classname`**.	Example: **`ocs-storagecluster-cephfs`**
trusted_certificate_list	List of secrets containing TLS CA certificates that must be trusted by Automation Decision Services in addition to the system Root Certificate Authorities. Each listed secret must contain a **`tls.crt`** key with a PEM encoded certificate.	[]
version	Main application version, it must not be updated for iFixes	Example value: 23.0.2

Table 2. **`spec.mongo`** parameters

Parameters	Description	Default/Example values
admin_secret_name	The name of the MongoDB secret.	**`ibm-dba-ads-mongo-secret`**
db	The database that is used by the REST API and the Run Service with embedded MongoDB. If none is provided, a MongoDB secret is generated automatically.	**`ads-db`**
db_history	The database that is used by the REST API for temporary data with embedded MongoDB. If none is provided, a MongoDB secret is generated automatically.	**`ads-history`**
fs_group	Sets the **`securityContext.fsGroup`** field of pods instantiated for the corresponding services. For more information, see the Kubernetes documentation.	This parameter is optional. Type **`int`** (positive or null).
git_db	The database that is used by the Git Service with embedded MongoDB. If none is provided, a MongoDB secret is generated automatically.	**`ads-db`**
image.repository	The registry and namespace where the mongo container image is located.	cp.icr.io/cp/cp4a/ads/mongo
image.tag	The image tag name of the mongo container.	5.0
persistence.resources.requests.storage	The size of the persistent volume (PV) requested through a Persistent Value Claim (PVC) for the embedded MongoDB instance. A string that represents a Kubernetes byte quantity. The PVC refers to the storage class that is defined in **`shared_configuration.storage_configuration.sc_fast_file_storage_classname`**, if defined. The required minimum performance for the volume is 300 IOPS, as defined on IBM Cloud®.	30Gi
persistence.storage_class_name	If **`mongo.persistence.use_dynamic_provisioning`** is true, define the name of the storage class that is required for the persistent volume of the embedded MongoDB instance. Setting the **`storage_class_name`** to null means using the default storage class. This parameter cannot be changed after installation.	
persistence.use_dynamic_provisioning	Boolean. Controls the type of persistent storage claim that is created for the embedded MongoDB instance.	true
resources.limits.memory	The memory limit for the mongo container.	1024
resources.limits.cpu	The CPU limit for the mongo container.	1000
resources.requests.memory	The memory requested for the mongo container.	256
resources.requests.cpu	The CPU requested for the mongo container.	500
tls	Indicates whether to use secure connections with the embedded MongoDB. If **`false`**, a MongoDB secret is generated automatically.	**`true`**
use_embedded	Specify this parameter if you decide to use the embedded MongoDB (only for evaluation purposes). Not recommended.	**`false`** **`true`** in starter mode only

Table 3. **`spec.zen`** parameters

Parameters	Description	Default/Example values
block_storage_classname	Zen block storage class name.	**`spec.storage_classname`** is used as default (if set). Otherwise, example values are **`aregp2`** or **`ocs-storagecluster-ceph-rbd`**.
file_storage_classname	Zen file storage class name.	**`spec.storage_classname`** is used as default (if set). Otherwise, example values are **`efs-sc`** or **`ocs-storagecluster-cephfs`** .
scale_config	Common services scale configuration.	The default value is the same as the profile size that is set in Automation Decision Services. Otherwise the default value is **`starterset`**. It can be **`small`**, **`medium`**, **`large`**, or **`xlarge`** (the latter is only on x86_64).

Languages reference

Automation Decision Services provides two languages: the rule language and the Advanced Rule Language (ARL).

- **Rule language**
  Business experts write the business rules that drive the decision logic using the rule language. The syntax of the rule language is close to that of the natural language.
- **Advanced Rule Language (ARL)**
  As an alternative to the rule language, advanced users can write rules using the Advanced Rule Language in task models.

Rule language

Business experts write the business rules that drive the decision logic using the rule language. The syntax of the rule language is close to that of the natural language. A basic rule uses an if-then statement to associate a condition (if) with an action (then). The rule states what decision to make when a condition is true. For example, here is a simple rule that describes the action to take for gold customers, which is to apply a 10% discount:

```
if
   the category of the customer is gold
then
   print "apply a 10% discount to the shopping cart of this customer";
```

The language is comprised of a series of statements, or building blocks:

- Constructs to build the different parts of your rules – the condition part, which is optional, and the action part.
- Operators and functions to do arithmetic operations, associate or negate conditions, and compare expressions.
- Literals to declare values as being of a particular type.
- Punctuation to structure rules, avoid ambiguity, and provide clarity.

Note: Some of the topics in this section use metasymbols to describe the syntax allowed in business rules. The table below lists the usage of each metasymbol:

Usage	Metasymbol
Optional	`?`
Optional group	`[ ... ]`
Repetition (one or more)	`+`
Repetition (zero or more)	`*`
Grouping	`( ... )`
Ordered choice	`|`

- **Rule structure**
  A rule is composed of two parts: the condition part, which is optional, and the action part. You use constructs to build your rules.
- **Data types and functions**
  The rule language supports number, Boolean, string, date, and time data types. You use data-specific functions to perform tasks such as associating or negating conditions, comparing expressions, and performing arithmetic operations.
- **Punctuation in rules**
  The rule language uses punctuation to identify specific language artifacts and to avoid ambiguous syntax. When you edit business rules, mandatory punctuation is usually predicted in the completion menu.
- **String interpolation**
  String interpolation is an alternative to building strings via concatenation.

Rule structure

A rule is composed of two parts: the condition part, which is optional, and the action part. You use constructs to build your rules.

- **Understanding the structure of business rules**
  The structure of the rule depends on the type of rule that you want to write. There are two ways to build rules in the rule language.
- **Condition part**
  The condition part defines a set of conditions that must be satisfied for the rule's action to be triggered. This part starts with the **`if`** keyword and can be preceded by preconditions.
- **Action part**
  The action part describes the actions that the rule completes when the conditions are met. This part might start with the **`then`** keyword and can contain an **`else`** construct.

Understanding the structure of business rules

The structure of the rule depends on the type of rule that you want to write. There are two ways to build rules in the rule language.

- **`if/then/else`** statements
- Action statements

`if/then/else` statements

The parts must be defined in the following order:

1. for each definitions part
2. definitions part
3. if (conditions) part

4. then (actions) part
5. else (optional actions) part

The syntax of **if/then/else** statements should be as follows:

```
[<for-each definitions>]
[<definitions>]
[if
    <condition>]
[then]
    (<action>;)+
[else
    (<action>;)+]
```

for-each definitions
This construct is used at the very beginning of the rule, and before the **definitions** part, to apply the rest of the rule to every object in a collection.

definitions
This construct introduces the part of the rule where you define variables.

Variables allow you to write more concise rules by using a short, convenient identifier to represent a value or the result of an expression.

Variables defined in the definitions part of a rule are local to the rule in which they are defined. A local variable can be used anywhere within this same rule, but is not available in other rules.

if
This construct introduces the part of the rule in which you define conditions.

If the conditions in the **if** part are met, the actions in the **then** part of the rule are executed. The **if** part of a rule is optional. If there is no **if** part, all actions in the **then** part of the rule are performed each time the rule is executed.

If the conditions in the **if** part of the rule are not met, the actions in the **else** part of the rule are executed. If the conditions are not met and the rule does not have an **else** part, the rule does nothing.

then
This construct introduces the part of the rule that defines the actions that are executed if the condition part of a rule is satisfied.

Every rule must have an action part. If a rule has no definition part and no condition part, the then keyword is optional and the actions in this part of the rule are always executed whenever the rule is executed.

The actions in the action part of a rule are executed in the order they appear. Start each action statement on a new line and end each action statement with a semi-colon (;).

else
This construct introduces actions that are executed if the conditions of a rule are not satisfied.

The else part of a rule is optional and allows you to specify one or more actions to perform if the conditions in the condition part of the rule are not met.

The actions in the else part of a rule are executed in the order they appear. Start each action statement on a new line and end each action statement with a semi-colon (;).

If one or more variables with preconditions are defined in the definition part of a rule, the rule is only processed if these preconditions are satisfied and all local variables are correctly initialized. This means that if these preconditions are not satisfied, the rule is not processed at all, and the actions in the else part of the rule are never executed, whether or not the conditions in the condition part of the rule are met.

Action statements

The parts must be defined in the following order:

1. for each definitions part
2. actions part

The syntax of action statements should be as follows:

```
[<for-each definitions>]
    (<action>;)+
```

for-each definitions
This construct is used to apply one or more actions to all objects in a collection.

It should be immediately followed by a colon (**:**) and a list of one or more actions, each on a separate line, each preceded by a dash.

Condition part

The condition part defines a set of conditions that must be satisfied for the rule's action to be triggered. This part starts with the **if** keyword and can be preceded by preconditions.

- **Preconditions**
  Use preconditions to define variables or to apply a rule to the elements of a collection.
- **Conditions**
  Use condition constructs to define the conditions to be evaluated.

Preconditions

Use preconditions to define variables or to apply a rule to the elements of a collection.

- **for each (object) called (variable), in (list)**
  This construct specifies that the rest of the rule must be applied to each element of a collection.
- **definitions**
  This construct introduces the part of the rule where you define variables.

for each (object) called (variable), in (list)

This construct specifies that the rest of the rule must be applied to each element of a collection.

Purpose

You use this construct at the very beginning of the rule, and before the **definitions** part, to apply the rest of the rule to every object in a collection.

This construct is the equivalent of the construct **set** ***<variable>*** **to** ***<definition>*** **in** ***<list>*** that can be used in the **definitions** part of the rule.

Syntax

```
for each <object> [called <variable>], in <list>
```

Description

When using a **for each** statement to perform an action on a set of items, a variable is created automatically to represent each item in the collection. You use the **called** ***<variable>*** construct to refer to this variable with a custom name. This custom name can also be used to specifically refer to this variable in a nested **for each** construct.

Note: The **called** ***<variable>*** part is mandatory in the preconditions of decision tables.

Example

The following rule shows how to add heats to a list of matching heats by looking at each heat, checking whether it meets the conditions, and if so, adding it to the decision.

```
for each heat called heat, in heats
if
   the area of swimmer is the area of heat
   and the category of heat is 'swimmer heat'
then
   add "Here is a heat that we recommend for you: " + the name of heat to decision;
```

definitions

This construct introduces the part of the rule where you define variables.

Purpose

You use this construct to define local variables that you can use elsewhere in the same rule. Use local variables to produce more concise and readable rules.

Syntax

```
definitions
    (set <variable> to <definition> in <list> | from <object> [where <test>]
;)*
```

Description

Variables allow you to write more concise rules by using a short, convenient identifier to represent a value or the result of an expression.

Variables defined in the definitions part of a rule are local to the rule in which they are defined. A local variable can be used anywhere within this same rule, but is not available in other rules. You define local variables using the set (variable) to (definition) construct.

Example

The following example declares the local variable **'preferred customer'** in the **definitions** part of the rule and uses it in the **if** and **then** parts of the rule.

```
definitions
   set 'preferred customer' to a customer from customer;
if
   the age of 'preferred customer' is less than 18
then
   print "apply a 10% discount to the shopping cart of this customer";
```

- **set (variable) to (definition)**
  This construct defines a variable as an object, a collection, or a value.
- **from (object)**
  This construct defines a local variable in the rule that you can use to access data from objects that are related to known objects.
- **in (list)**
  This construct defines a local variable in the rule that you can use to access objects from a collection.
- **(attribute) of (list)**
  You use this construct to access the list of values of a given attribute for a list of objects.
- **where (test)**
  This construct matches objects against tests.

set (variable) to (definition)

This construct defines a variable as an object, a collection, or a value.

Purpose

You use this construct in the **definitions** part of a rule to declare a local variable.

Syntax

- To define a variable as an object of a specific type:

  ```
  set <variable> to <type> [in <list> | from <object>] [where <test>] ;
  ```

- To define a variable as multiple objects of a specific type:

  ```
  set <variable> to <type> [in <list>] [where <test>] ;
  ```

- To define a variable as a constant:

  ```
  set <variable> to <object> [where <test>] ;
  ```

- To define a variable as multiple constants:

  ```
  set <variable> to <list> [where <test>] ;
  ```

Description

Variables produce more concise rules by replacing a value or the result of an expression with a short, convenient identifier. A local variable can be used anywhere within the rule in which it is defined, but is not available in other rules.

Enclose the name of each variable in single quotes. This is compulsory for variable names that contain spaces. While it is not essential to enclose one-word variable names in single quotes, it makes them easier to identify and reduces the risk of confusion.

Example

The following examples show how to define a variable as an object.

```
definitions
   set i to an item;
   set house to a house
      where the price of this house is more than 1000;
```

The following examples show how to define a variable as a literal, string, or collection.

```
definitions
   set category to Gold ;
   set s to "a string";
   set 'expensive items' to all items
      in the items of the shopping cart of customer
      where the price of each item is more than 200;
```

The following example shows how to define a variable as the result of an expression.

```
definitions
   set expr to the price of the car of customer + 100;
   set h2 to the house of customer
      where the price of this house is more than 1000;
```

The following example shows how to define a variable as another variable.

```
definitions
   set expr2 to expr;
```

from (object)

This construct defines a local variable in the rule that you can use to access data from objects that are related to known objects.

Purpose

You use this construct to expand the set of data that can be used by a rule.

Syntax

```
from <object>
```

Description

The **from** construct is used to retrieve one distinct object from another one, and creates a variable that you can use in the rule to refer to this data. You cannot use it to retrieve a collection of objects. To access a collection of objects, use the in (list) construct.

Example

The following rule declares a variable based on the relationship between a customer and the customer's preferred item.

```
definitions
    set 'preferred CD' to a CD
        from the preferred item of customer;
if
     the price of 'preferred CD' is more than 25
then...
```

Here the decision node does not have access to "CD" objects from its input data. In this case, to write a rule that uses a CD object, you must first declare a variable of type CD that pulls in data from the preferred item object of the customer. This is possible only if the customer object (and therefore the customer's preferred item) is already known to the decision node.

in (list)

This construct defines a local variable in the rule that you can use to access objects from a collection.

Purpose

You use this construct in the definition or condition part of a rule to access objects from collections.

Syntax

```
in <list>
```

Description

The **in** construct is used to retrieve data from a collection of objects, and creates a variable that you can use in the rule to refer to this data. The *<list>* parameter of the **in** construct must be an expression that denotes a collection.

The **in** construct can be used in the **definitions** part of a rule and in count conditions specified in the **if** part of a rule.

Example

The following definition declares a variable as an item in the collection of shopping cart items.

```
definitions
    set item to a CD
        in the items of the shopping cart of customer ;
if
    there is an item
then...
```

The following condition tests that there is an item in the collection of shopping cart items.

```
if
    there is an item
        in the items of the shopping cart of customer ;
then...
```

(attribute) of (list)

You use this construct to access the list of values of a given attribute for a list of objects.

Purpose

You use this construct in any part of the rule to access a list of attribute values.

Syntax

`<attribute> of <list>`

Description

You use this construct with other constructs that work with lists. For example, you can use it with the **`in <list>`** construct in a condition, or with the **`set`** construct in an action. The list can be an expression or an embedded list.

Example

The following action sets the value of the variable to the list of the names of the banks of all the customers.

```
set decision to the bank accounts of customers;
```

The following definition creates a variable that represents the list of all the cities from the addresses of all customers.

```
definitions
   set 'registered city' to the cities of the addresses of customers;
```

In the following rule, the action executes only if there is one or more customers who live in France.

```
if
   the countries of the addresses of customers contain "France"
then
   set decision to "include EU countries";
```

where (test)

This construct matches objects against tests.

Purpose

Matches objects against tests.

Syntax

`where <test>`

Description

In the **`definitions`** part of a rule, you can use one or more **`where`** clauses with the **`set`** construct to test that an object meets one or more conditions in order to be a valid value for the variable being declared. In the **`definitions`** part of the rule, the **`where`** statement does not end with a comma. Instead, each **`set`** construct should end with a semi-colon (;).

In the **`if`** part of a rule, you can use one or more **`where`** clauses in count conditions (**`there is more than`**, **`there is less than`**, and so on) to test that an object meets one or more conditions before being counted.

In a **`where`** construct, an implicit variable that refers to the current instance of the object being tested is automatically declared. In the test, you can thus refer to potential instances of the variable as **`this`** ***`<object>`*** (or **`each`** ***`<object>`*** if they are part of a collection.)

Example

The following example shows the **`definitions`** part of a rule that defines the local variable **`'expensive items'`** as a collection of items whose price is greater than 100. The implicit variable **`each item`** is used in the **`where`** clause to refer to each **`item`** object in the collection.

```
definitions
    set 'expensive items' to all items
        where the price of each item is more than 100;
```

Conditions

Use condition constructs to define the conditions to be evaluated.

- **all (type) in (list)**
  The **`all <type>`** construct introduces a collection of objects of a given type.
- **all of the following conditions are true**
  This construct groups together a series of conditions such that the combined statement evaluates to true only if all the listed conditions evaluate to true.
- **any of the following conditions is true**
  This construct groups conditions together such that the combined statement evaluates to true if one or more of the listed conditions evaluate to true.
- **in (list)**
  This construct defines a local variable in the rule that you can use to access objects from a collection.

- **it is not true that (a condition)**
  This construct negates a condition statement.
- **(attribute) of (list)**
  You use this construct to access the list of values of a given attribute for a list of objects.
- **none of the following conditions are true**
  This construct negates a group of conditions.
- **the number of (type)**
  This construct returns the number of objects in the current data set or in the specified collection.
- **there are (number) (type)**
  This construct tests whether there is a specific number of objects of a given type in the specified collection.
- **there are at least (number) (type)**
  This construct tests whether there is at least a number of objects of a given type in the specified collection.
- **there are at most (number) (type)**
  This construct tests whether there is no more than a specified number of objects of a given type in a collection.
- **there are less than (number) (type)**
  This construct tests whether there are fewer than a specified number of objects of a given type.
- **there are more than (number) (type)**
  This construct tests whether there are more than a specified number of objects of a given type.
- **there is at least one (type)**
  This construct tests whether there is at least one object of a given type in a collection.
- **there is at most one (type)**
  This construct tests whether there is at most one object of a given type in a collection.
- **there is no (type)**
  This construct tests whether a data set contains no objects of the given type.
- **there is one (type)**
  This construct tests whether a collection contains one and only one object of a given type.

all (type) in (list)

The **all <type>** construct introduces a collection of objects of a given type.

Purpose

You use this construct to filter a collection of objects of a specific type.

Syntax

```
all <type> [in <list>] [,where <test>[,]]
```

Description

You use this construct to collect all object instances in a collection.

The **in** construct is used to retrieve data from a collection of objects related to an existing object. The **<list>** parameter of the **in** construct must be an expression that denotes a collection.

You can add a **where** clause to test one or more conditions on an object. When you use the **where** clause with collections of objects, the comma before the **where** is required, whereas the comma at the end of the condition is optional.

Example

The following rule checks whether the total amount of all the coupons is at least 150. If this condition is true, a message is displayed to grant the customer a 10% discount on the next trip.

```
if
    the total amount of all coupons in the coupons of customer is at least 150
then
    print "10% discount on the next trip";
```

The following rule checks whether the average price of all the tickets of this trip is between 100 and 200.

```
if
    the average price of all tickets in the tickets of trip is between 100 and 200
then
    print "The average price of all the tickets of this trip is between 100 and 200";
```

The following rule checks whether the number of purchases is more than 100. If this condition is true, a message is displayed. Notice that when **the number of** operator is applied to a collection that is defined by the **all** construct, the article **all** is removed to improve readability.

```
if
    the number of purchases is more than 100
then
    print "send 25% discount voucher";
```

all of the following conditions are true

This construct groups together a series of conditions such that the combined statement evaluates to true only if all the listed conditions evaluate to true.

Purpose

The construct **all of the following conditions are true** provides a more compact and convenient alternative to the logical operator **and** when you want to combine multiple conditions. The resulting statement is considered true only if each of the listed conditions is true.

Syntax

```
all of the following conditions are true:
    (- <condition>)+ [,]
```

Description

This is equivalent to writing **if** *<condition 1>* **and** *<condition 2>* **and ...** *<condition 2n>*. However, if you have a large number of conditions, using the **all of the following conditions are true** construct can make the rule more readable.

The construct must be followed by a colon and one or more condition statements, each one on a separate line and preceded by a dash. Note that the dash that precedes each condition must be a 'short dash'.

Optionally, you can add a comma after the last condition statement in the list to signify the end of the list of grouped conditions. This separates those conditions that form part of the construct from any additional condition statements that might follow.

Example

The following group of conditions tests that a customer is Gold junior member.

```
if
   all the following conditions are true:
      - the category of customer is Gold
      - the age of customer is at most 15
then...
```

any of the following conditions is true

This construct groups conditions together such that the combined statement evaluates to true if one or more of the listed conditions evaluate to true.

Purpose

The construct **any of the following conditions are true** provides a more compact and convenient alternative to the logical operator **or** when you want to combine multiple conditions. The resulting statement is considered true if at least one of the listed conditions is true.

Syntax

```
any of the following conditions is true:
    (- <condition> )+ [,]
```

Description

This is equivalent to writing **if** *<condition 1>* **or** *<condition 2>* **or ...** *<condition n>*. However, if you have a large number of conditions, using the **any of the following conditions is true** construct can make the rule more readable.

The construct must be followed by a colon and one or more condition statements, each one on a separate line and preceded by a dash. Note that the dash that precedes each condition must be a 'short dash'.

Optionally, you can add a comma after the last condition statement in the list to signify the end of the list of grouped. This will separate those conditions that form part of the construct from any additional condition statements that might follow.

Note:
Indentation in rules is for readability only; it does not influence the way the rule is processed.

Example

This example shows how to test if a customer is a Gold member or a senior customer.

```
if
   any of the following conditions is true:
       - the category of customer is Gold
       - the age of customer is more than 60
then...
```

The following more complex example shows the use of a comma to mark the end of a group of conditions. In this example, the order must be over $50, the customer must be either a Gold customer or a senior customer, and the order must be shipping to NJ.

```
if
   all of the following conditions are true:
     - 'order total' is greater than $50
     - any of the following conditions is true:
        - the category of customer is Gold
        - the age of customer is more than 60,
      - 'shipping address' is in NJ
then...
```

Without the comma to mark the end of the **any of the following conditions are true** block, the final condition (**'shipping address' is in NJ**) would be considered part of the previous group, and the rule would be interpreted to mean that the order must be over $50 and (either the customer is a Gold customer or a senior customer or the shipping address is in NJ).

in (list)

This construct defines a local variable in the rule that you can use to access objects from a collection.

Purpose

You use this construct in the definition or condition part of a rule to access objects from collections.

Syntax

in *<list>*

Description

The **in** construct is used to retrieve data from a collection of objects, and creates a variable that you can use in the rule to refer to this data. The *<list>* parameter of the **in** construct must be an expression that denotes a collection.

The **in** construct can be used in the **definitions** part of a rule and in count conditions specified in the **if** part of a rule.

Example

The following definition declares a variable as an item in the collection of shopping cart items.

```
definitions
    set item to a CD
        in the items of the shopping cart of customer ;
if
    there is an item
then...
```

The following condition tests that there is an item in the collection of shopping cart items.

```
if
    there is an item
        in the items of the shopping cart of customer ;
then...
```

it is not true that (a condition)

This construct negates a condition statement.

Purpose

You use this construct when there is no operator to express the opposite of a condition.

Syntax

it is not true that *<condition>*

Description

When there is no operator to express the opposite of a condition, you can insert the **it is not true that** construct in front of an existing condition to negate it.

This can be useful in cases where a boolean (true/false) attribute defined in your model is verbalized as an expression such as 'customer is a loyalty member', but where you want to test for the opposite of this expression ('customer is not a loyalty member'). You can write 'it is not true that customer is a loyalty member' to achieve the required result.

In general, the logic of a negative expression is more difficult to interpret, so it is advisable to avoid them where possible.

Example

The following example uses the **it is not true that** construct to check the age of the customer, assuming that the boolean property 'is a minor' has been verbalized in the vocabulary, but that the opposite of the expression 'is a major' has not been defined.

```
if
   it is not true that customer is a minor
```

The following example shows how to negate a set of conditions that have been grouped using the **all of the following conditions are true** construct. The resulting expression will return True except if the customer is both a gold member and over 60.

```
if
   it is not true that all of the following conditions are true:
          - the category of customer is gold
          - the age of customer is more than 60
```

(attribute) of (list)

You use this construct to access the list of values of a given attribute for a list of objects.

Purpose

You use this construct in any part of the rule to access a list of attribute values.

Syntax

```
<attribute> of <list>
```

Description

You use this construct with other constructs that work with lists. For example, you can use it with the **in *<list>*** construct in a condition, or with the **set** construct in an action. The list can be an expression or an embedded list.

Example

The following action sets the value of the variable to the list of the names of the banks of all the customers.

```
set decision to the bank accounts of customers;
```

The following definition creates a variable that represents the list of all the cities from the addresses of all customers.

```
definitions
   set 'registered city' to the cities of the addresses of customers;
```

In the following rule, the action executes only if there is one or more customers who live in France.

```
if
   the countries of the addresses of customers contain "France"
then
   set decision to "include EU countries";
```

none of the following conditions are true

This construct negates a group of conditions.

Purpose

You use this construct to group together a series of condition statements that you want to negate.

Syntax

```
none of the following conditions are true:
    (- <condition> )+ [,]
```

Description

The group evaluates to true only if none of the conditions listed are true.

The construct must be followed by a colon and one or more condition statements, each one on a separate line and preceded by a dash. Note that the dash that precedes each condition must be a ‘short dash’.

Optionally, you can add a comma after the last condition statement in the list to signify the end of the list of grouped conditions. This will separate those conditions that form part of the construct from any additional condition statements that might follow.

Example

The following group of conditions tests that a customer is not a Gold senior member and not under 60.

```
if
   none of the following conditions are true:
      - the category of customer is gold
      - the age of customer is least 60
then...
```

the number of (type)

This construct returns the number of objects in the current data set or in the specified collection.

Purpose

You use this construct to count the number of objects of the specified type and return the total as a number.

Syntax

```
the number of <type> [in <list>] [where <test>,]
```

Description

You can use the optional **in <list>** clause to count the objects in a specific list. You can use one or more optional **where** clauses to apply additional conditions to the objects that are included in the returned count.

This construct cannot be used in the **definitions** part of a rule.

Example

The following condition is met if there are at least 6 vehicles:

```
if
   the number of vehicles is more than 5
then...
```

there are (number) (type)

This construct tests whether there is a specific number of objects of a given type in the specified collection.

Purpose

You use this construct to determine whether the collection contains exactly the specified number of occurrences of a particular object.

Syntax

```
there are <number> <type> [in <list>] [where <test>,]
```

Description

Use the **in** clause to apply the test to a specific collection of objects. Use one or more optional **where** clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the **if** part of a rule.

Example

The following condition tests if the number of Gold customers is equal to 8.

```
if
   there are 8 customers in the customers of bank
      where the category of each customer is Gold
then...
```

there are at least (number) (type)

This construct tests whether there is at least a number of objects of a given type in the specified collection.

Purpose

You use this construct to determine whether the collection contains at least the specified number of occurrences of a particular object.

Syntax

```
there are at least <number> <type> [in <list>] [where <test>,]
```

Description

Use the **in** clause to apply the test to a specific collection of objects. Use one or more optional **where** clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the **if** part of a rule.

Note:
The *<number>* argument cannot be a BigDecimal or a BigInteger. Although the language parser does not generate an error, an error may be generated.

Example

The following condition tests if there are at least 10 Gold customers.

```
if
   there are at least 10 customers in the customers of bank
      where the category of each customer is Gold
then...
```

there are at most (number) (type)

This construct tests whether there is no more than a specified number of objects of a given type in a collection.

Purpose

You use this construct to determine whether the specified collection contains no more than the specified number of occurrences of a particular object.

Syntax

```
there are at most <number> <type> [in <list>] [where <test>,]
```

Description

Use the **in** clause to apply the test to a specific collection of objects. Use one or more optional **where** clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the **if** part of a rule.

Note:
The *<number>* argument cannot be a BigDecimal or a BigInteger. Although the language parser does not generate an error, an error may be generated.

Example

The following condition tests if the number of Gold customers is lower than or equal to 3.

```
if
   there are at most 3 customers in the customers of bank
      where the category of each customer is Gold
then...
```

there are less than (number) (type)

This construct tests whether there are fewer than a specified number of objects of a given type.

Purpose

You use this construct to determine whether the specified collection contains fewer than the specified number of occurrences of a particular object.

Syntax

```
there are less than <number> <type> [in <list>] [where <test>,]
```

Description

Use the **in** clause to apply the test to a specific collection of objects. Use one or more optional **where** clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the **if** part of a rule.

Note:
The *<number>* argument cannot be a BigDecimal or a BigInteger. Although the language parser does not generate an error, an error may be generated.

Example

The following condition tests if the number of Gold customers is lower than 3.

```
if
   there are less than 3 customers in the customers of bank
      where the category of each customer is Gold
then...
```

there are more than (number) (type)

This construct tests whether there are more than a specified number of objects of a given type.

Purpose

You use this construct to determine whether the specified collection contains more than the specified number of occurrences of a particular object.

Syntax

```
there are more than <number> <type> [in <list>] [where <test>,]
```

Description

Use the **in** clause to apply the test to a specific collection of objects. Use one or more optional **where** clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the **if** part of a rule.

Note:
The **<number>** argument cannot be a BigDecimal or a BigInteger. Although the language parser does not generate an error, an error may be generated.

Example

The following condition tests if there are more than 10 customers with the Gold category.

```
if
   there are more than 10 customers in the customers of bank
      where the category of each customer is Gold,
then...
```

there is at least one (type)

This construct tests whether there is at least one object of a given type in a collection.

Purpose

You use this construct to determine whether the specified collection contains at least one occurrence of a particular object.

Syntax

```
there is at least one <type> [in <list>] [where <test>,]
```

Description

Use the **in** clause to apply the test to a specific collection of objects. Use one or more optional **where** clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the **if** part of a rule.

Example

The following condition tests whether or not a **Customer** object exists.

```
if
    there is at least one customer
```

```
       where...
then...
```

The following condition tests if a senior Gold customer exists.

```
definitions
  set 'gold customers' to all customers in the customers of bank
      where the category of each customer is Gold;
if
   there is at least one customer in 'gold customers'
        where the age of this customer is at least 60,
then...
```

there is at most one (type)

This construct tests whether there is at most one object of a given type in a collection.

Purpose

You use this construct to determine whether the specified collection contains zero or one occurrence of a particular object.

Syntax

```
there is at most one <type> [in <list>] [where <test>,]
```

Description

Use the **in** clause to apply the test to a specific collection of objects. Use one or more optional **where** clauses to filter the objects to be counted with one or more conditions.

This construct can only be used in the **if** part of a rule.

Example

The following condition tests if there is at most one Gold customer in the list of customers.

```
if
   there is at most one Customers in the customers of 'bank'
      where the category of this customer is Gold,
then...
```

there is no (type)

This construct tests whether a data set contains no objects of the given type.

Purpose

You use this construct to determine whether the specified collection contains no occurrences of a particular object.

Syntax

```
there is no <type> [in <list>] [where <test>,]
```

Description

Use the **in** clause to apply the test to a specific collection of objects. Use one or more optional **where** clauses to filter the objects to be counted with one or more conditions.

This construct can be used only in the **if** part of a rule.

Example

The following condition tests that there is no object of type **Customer** with an age over 60 in the list of customers.

```
if
   there is no customer in the customers of bank where the age of this customer is at least 60
then...
```

there is one (type)

This construct tests whether a collection contains one and only one object of a given type.

Purpose

You use this construct to determine whether the specified collection contains one and only one occurrence of a particular object.

Syntax

```
there is one <type> [in <list>] [where <test>,]
```

Description

Use the **in** clause to apply the test to a specific collection of objects. Use one or more optional **where** clauses to filter the objects to be counted with one or more conditions.

This construct can be used only in the **if** part of a rule.

Example

The following condition tests if there is one Gold customer in the list of customers.

```
if
   there is one customer in the customers of bank
      where the category of this customer is Gold
then...
```

Action part

The action part describes the actions that the rule completes when the conditions are met. This part might start with the **then** keyword and can contain an **else** construct.

- **add (object) to (variable)**
  The **add** construct adds an item to a list of objects.
- **define (variable) as (value)**
  The **define** construct creates a variable that you can use to perform actions.
- **for each (object) in (list)**
  This construct specifies the actions that must be performed to every element of a collection.
- **(attribute) of (list)**
  You use this construct to access the list of values of a given attribute for a list of objects.
- **print (string)**
  This construct prints a string to the standard output.
- **remove (object) from (variable)**
  The **remove** construct removes an item from a list.
- **set (variable) to (value)**
  This construct specifies a value for a variable.

add (object) to (variable)

The **add** construct adds an item to a list of objects.

Purpose

You use this construct in the action part of a rule to add an object to a list of objects.

Syntax

- In decision models:

  ```
  add <object> to decision
  ```

- In task models:

  ```
  add <object> to <variable>
  ```

Description

The *<object>* specifies the object that you want to add. The list in which you want to include this object corresponds to the type of the decision node output, which must be a list.

Example

The following action adds a string with the name of a race to the decision, which is a list of strings.

```
add "New matching race: " + the name of race to decision;
```

define (variable) as (value)

The **define** construct creates a variable that you can use to perform actions.

Purpose

You use this construct in the action part of a rule to create a temporary local variable.

Syntax

```
define <variable> as (<object> | <list>)
```

Description

The **define** ***<variable>*** keyword defines the name of the new variable. You cannot use this construct to set a value for an existing variable. The **as** ***<variable value>*** keyword defines the object that the variable refers to. The variable stores the object locally. You can then use this new variable in the rule action.

Example

In the following example, you define a new variable called **new order**. This new order is created from the shipping address and the billing address of the customer.

```
define 'new order' as a new order where
        the shipping address is the shipping address of Customer,
        the billing address is the billing address of Customer ;
```

You can also use the **define <variable> as <variable value>** construct with the **a copy of <object> where <attribute> is <value>** construct to create a new variable. The following action creates a new variable called **the new address**.

```
define 'the new address' as a copy of the address of customer where the street is "Victor Hugo" ;
```

for each (object) in (list)

This construct specifies the actions that must be performed to every element of a collection.

Purpose

You use this construct to apply one or more actions to all objects in a collection.

Syntax

```
for each <object> [called <variable>,] in <list>:
    (- <action>+) [,]
```

Description

The construct should be immediately followed by a colon (:) and a list of one or more actions, each on a separate line, each preceded by a dash.

The implicit variable **this** ***<object>*** can be used in each action statement within the **for each** loop to refer to the current object. If you nest a **for each** statement inside another, you can use the **called** construct to define an explicit name for the object in the 'outer' **for each** loop in order to reference it clearly from within the nested **for each** loop.

Example

The following rule shows how to apply a discount to all the expensive items in a customer's shopping cart, if the customer is a Gold customer.

```
definitions
    set smith to a customer ;
    set 'expensive items' to all items
        in the items of the shopping cart of smith
            where the price of 'each item' is greater than 1000;
if
    the category of smith is Gold
then
    for each item in 'expensive items':
        - set decision to 5;
```

The following rule shows two nested **for each** statements. The **called** ***<variable>*** construct is used so that each customer can be explicitly referred to from within the nested **for each** loop.

```
if
   the category of customer is Gold
then
   for each customer called c, in 'senior customers':
```

```
- for each account in 'new accounts':
   - set decision to the mailing address of c;
```

(attribute) of (list)

You use this construct to access the list of values of a given attribute for a list of objects.

Purpose

You use this construct in any part of the rule to access a list of attribute values.

Syntax

<attribute> of _<list>_

Description

You use this construct with other constructs that work with lists. For example, you can use it with the **in _<list>_** construct in a condition, or with the **set** construct in an action. The list can be an expression or an embedded list.

Example

The following action sets the value of the variable to the list of the names of the banks of all the customers.

```
set decision to the bank accounts of customers;
```

The following definition creates a variable that represents the list of all the cities from the addresses of all customers.

```
definitions
   set 'registered city' to the cities of the addresses of customers;
```

In the following rule, the action executes only if there is one or more customers who live in France.

```
if
   the countries of the addresses of customers contain "France"
then
   set decision to "include EU countries";
```

print (string)

This construct prints a string to the standard output.

Purpose

You use this construct to define an action phrase that by default outputs a string to the standard output.

Syntax

print "_<string>_"

Example

The following action prints the string **"Hello World!"** to the standard output.

```
print "Hello World!" ;
```

remove (object) from (variable)

The **remove** construct removes an item from a list.

Purpose

You use this construct in the action part of a rule to remove an object from a list of objects.

Syntax

- In decision models:

  remove _<object>_ from decision

- In task models:

  ```
  remove <object> from <variable>
  ```

Description

The *<object>* specifies the object that you want to remove. The list from which you want to remove this object corresponds to the type of the decision node output, which must be a list.

Example

The following action removes a race from the list of races of the decision.

```
remove race from decision;
```

set (variable) to (value)

This construct specifies a value for a variable.

Purpose

You use this construct in the action part of a rule to set the value of a variable.

Syntax

- In decision models:

  ```
  set decision to <value
  ```

- In task models:

  ```
  set <variable> to <value
  ```

Description

The **decision** keyword refers to the decision that the node makes. You can also use the variable name of the output of this decision in its place. The value can be a literal value, an expression, or a list.

Example

Consider a decision logic for determining the current age of a customer. This decision logic might have a rule that sets the decision to the age of the customer. As a result, the current age is equal to the age of the customer.

```
set decision to the age of customer;
```

You can also use the **set decision to <value>** construct with the **a copy of <object> where <attribute> is <value>** construct to update the value of an existing variable. The following action creates a copy of the address of the customer and updates its value.

```
set 'new address' to a copy of the address of customer where the street is "Victor Hugo" ;
```

Data types and functions

The rule language supports number, Boolean, string, date, and time data types. You use data-specific functions to perform tasks such as associating or negating conditions, comparing expressions, and performing arithmetic operations.

- **Numbers**
  Number types are represented as a set of digits with a possible decimal point separator, a grouping symbol, a minus sign and an exponent. You can use operators and functions to perform arithmetic operations and compare numbers.
- **Boolean**
  Boolean types are represented by the logical values true or false. You can use logical operators to perform operations on Boolean values.
- **Dates and times**
  You can use the Date & Time, Date, and Universal Date & Time value types to express date values. You can use the Time value type to express time values. You can use date and time functions include time logic to the rules.
- **Strings**
  String types are represented by a set of characters enclosed in double quotation marks ("Text string").
- **Lists**
  Lists are collections of objects that are arranged in a sequence. You can use basic list functions, aggregation functions, and advanced stream functions to perform operations on lists.
- **Object functions**
  Object functions define conditions based on objects (as opposed to attributes of objects). A customer is an object; the age of the customer is an attribute of the customer object.

Numbers

Number types are represented as a set of digits with a possible decimal point separator, a grouping symbol, a minus sign and an exponent. You can use operators and functions to perform arithmetic operations and compare numbers.

Table 1. Supported number types

Type	Example
Integer	`10`
Decimal, with decimal point separator	`10.5`
Decimal, with '000 grouping separator and decimal point separator	`150,500.51`
Negative integer	`-10`
Exponent	`10E-5`

The lexical representation of numbers (the decimal point separator and the grouping separator) is locale-specific.

Note:
Big number literals are not supported in the rule language. All literals manipulated in a rule must be within the range of the Java™ Double class, otherwise they are rounded to the closest Double value.

- **Mathematical operators and functions**
  Mathematical operators and functions perform operations on numbers. You can use them in variable definitions, conditions, and actions.
- **Number comparison functions**
  Number comparison functions define conditions based on numbers.

Mathematical operators and functions

Mathematical operators and functions perform operations on numbers. You can use them in variable definitions, conditions, and actions.

Table 1. Mathematical operators

Syntax	Description	Example
`- <number>`	Makes a number negative.	`set decision to - the number of errors`
`<number> + <number>`	Adds a number to another number.	`set decision to the base rental cost of car + 100`
`<number> - <number>`	Subtracts a number from another number.	`if the number of 'items purchased' - the number of 'items returned' is more than 0`
`<number> / <number>`	Divides a number by another number. If both numbers are integers, the result is also an integer (the result of the division without the remainder).	`set decision to the number of 'items ordered' / the number of orders`
`<number> * <number>`	Multiplies a number by another number.	`if the price of item * 'sales tax' is more than 100`

Table 2. Mathematical functions

Syntax	Description	Example
`lg(<number>)`	Returns the base 10 logarithm of a number.	`set decision to lg('initial value')`
`ln(<number>)`	Returns the natural logarithm (base e) of a number.	`set decision to ln('initial value')`
`max(<number>, <number>)`	Returns the greater value.	`set decision to max(the age of customer, 20)`
`min(<number>, <number>)`	Returns the smaller value.	`set decision to min(the age of customer, 20)`
`pow(<number>, <number>)`	Returns the power of a number raised to the second argument.	`set decision to pow('initial value', 3)`
`round(<number>, <number>)`	Rounds the first number to the number of decimal places that are given as second argument.	`set decision to round('intermediate total', 2)`
`sqrt(<number>, <number>)`	Returns the positive square root of a number.	`set decision to sqrt(side)`

Number comparison functions

Number comparison functions define conditions based on numbers.

Table 1. Number comparison functions

Syntax	Description	Example
`<number> does not equal <number>` or ≠	Tests that a number is not equal to another number.	`if the number of rentals does not equal 3`
`<number> equals <number>` or =	Tests that a number is equal to another number. This is equivalent to the **`is <number>`** operator	`if the number of rentals equals 3`
`<number> is <number>`	Tests that a number is equal to another number. This is equivalent to the **`equals <number>`** operator	`if the number of rentals is 3`

Syntax	Description	Example
`<number> is at least <number>` or ≥	Tests that a number is greater than or equal to another number.	`if the number of rentals` **`is at least`** `3`
`<number> is at least <number> and less than <number>`	Tests that a number is included in a range in which the first value is included and the last value is excluded.	`if the age of customer` **`is at least`** `12` **`and less than`** `25`
`<number> is at most <number>` or ≤	Tests that a number is less than or equal to another number.	`if the number of rentals` **`is at most`** `3`
`<number> is between <number> and <number>`	Tests that a number is included in a range in which both values are included.	`if the age of customer` **`is between`** `12` **`and`** `25`
`<number> is less than <number>` or <	Tests that one number is less than another.	`if the number of rentals` **`is less than`** `3`
`<number> is more than <number>` or >	Tests that one number is greater than another.	`if the number of rentals` **`is more than`** `3`
`<number> is more than <number> and at most <number>`	Tests that a number is included in a range in which the first value is excluded and the last value is included.	`if the age of customer` **`is more than`** `12 and` **`at most`** `25`
`<number> is strictly between <number> and <number>`	Tests that a number is included in a range in which the first value is excluded and the last value is excluded.	`if the age of customer` **`is strictly between`** `12` **`and`** `25`

Boolean

Boolean types are represented by the logical values true or false. You can use logical operators to perform operations on Boolean values.

- **Logical operators**
  Logical operators associate conditions. They perform logical operations on the Boolean values returned by these conditions.

Logical operators

Logical operators associate conditions. They perform logical operations on the Boolean values returned by these conditions.

Table 1. Logical operators

Syntax	Description	Example
`<condition> and <condition>`	Associates two conditions such that the resulting expression is only true if both conditions are true.	`if the category of customer is Gold` **`and`** `the age of customer is at least 60`
`<condition> or <condition>`	Associates two conditions such that the resulting expression is true if either (or both) conditions are true.	`if the category of customer is Gold` **`or`** `the age of customer is at least 60`

Dates and times

You can use the Date & Time, Date, and Universal Date & Time value types to express date values. You can use the Time value type to express time values. You can use date and time functions include time logic to the rules.
Several types of date and time are available.

In the following table, `a` is the symbol for AM or PM, and `z` is the time zone information, for example `+0200`.

Table 1. Date and time format

Type	Format
Date & Time	Standard date format that includes time, and is written as: `m/d/y h:mm:ss a`
Universal Date & Time	Date and time format that includes the time zone, and is written as: `m/d/y h:mm:ss a z`

Type	Format
Date	Simple date format that does not include time, and is written as: `m/d/y`
Time	Time format that is written as either: `h:mm:ss a` `h:mm a`

Note:
The rule language supports only the Gregorian calendar format.

- **Time point functions**
  Use time point functions to define and compare time points.
- **Time period functions**
  Use time period functions to define, calculate, and compare time points.
- **Calendar duration functions**
  Use duration functions to define and compute calendar durations.

Time point functions

Use time point functions to define and compare time points.
Note: When comparing two time points, milliseconds are not taken into account.

Date & time

Table 1. Functions for comparing time points

Syntax	Description	
`<date & time> is at the same time as <date & time>`	Checks that the two time points are identical, regardless of their time zone.	`if the arrival date of customer **is at the** `
`<date & time> is after <date & time>`	Checks that the time point comes after another time point.	`if 'flight departure' **is after** 01/01/2018 `
`<date & time> is after or the same as <date & time>`	Checks that the time point comes after another time point, or is the same as that other time point.	`if 'flight departure' **is after or the same**`
`<date & time> is before <date & time>`	Checks that the time point comes before another time point.	`if 'flight departure' **is before** 01/01/2018`
`<date & time> is before or the same as <date & time>`	Checks that the time point comes before another time point, or is the same as another time point.	`if 'flight departure' **is before or the sam**`
`<date & time> is after <date & time> and before <date & time>`	Checks that the time point is in a range in which both time points are excluded.	`if 'flight departure' **is after** 01/01/2018 `
`<date & time> is after <date & time> and on or before <date & time>`	Checks that the time point is in a range in which the first time point is excluded, and the last time point is included.	`if 'flight departure' **is after** 01/01/2018 `

Syntax	Description	
<date & time> is on or after <date & time> and before <date & time>	Checks that the time point is on or after a specific time point and before another time point.	if 'flight departure' **is on or after** 01/01
<date & time> is on or after <date & time> and on or before <date & time>	Checks that the time point is on or after a specific time point and on or before another time point.	if 'flight departure' **is on or after** 01/01
<date & time> is in the same calendar week as <date & time>	Checks that the two time points are in the same week.	if the scheduled departure of flight **is in**
<date & time> is on the same calendar day as <date & time>	Checks that the two time points are on the same day.	if the scheduled departure of flight **is on**
<date & time> is in the same calendar month as <date & time>	Checks that the two time points are on the same month.	if the scheduled departure of flight **is in**
<date & time> is in the same calendar year as <date & time>	Checks that the two time points are on the same year.	if the scheduled departure of flight **is in**
<date & time> is in the same hour as <date & time>	Checks that the two time points are in the same hour.	if the scheduled departure of flight **is in**
<date & time> is in the same minute as <date & time>	Checks that the two time points are in the same minute.	if the scheduled departure of flight **is in**
<date & time> is in the same second as <date & time>	Checks that the two time points are in the same second.	if the scheduled departure of flight **is in**

Table 2. Functions for comparing a time point and a time period

Syntax	Description	Example
<date & time> is after <time period>	Checks that the time point is at the same instant as the end of a specific period or later.	if 'purchase event' **is after** 'flight period'
<date & time> is before <time period>	Checks that the time point is earlier or at the same instant as the start of a specific period.	if the departure time of flight **is before** the calen
<date & time> is during <time period>	Checks that the time point takes place within a specific period.	if 'purchase event' **is during** the period of 2 hours
<date & time> is within <calendar duration> before <date & time>	Checks that the time point takes place within a calendar duration expressed in time units, and before another date.	if 'ticket purchase' **is within** 2 hours **before** the s

Syntax	Description	Example
`<date & time> is within <calendar duration> after <date & time>`	Checks that the time point takes place within a calendar duration expressed in time units, and after another date.	`if 'return flight' is within 24 hours after the sch`

Table 3. Functions for testing that a component of the representation of a time point has a specific value

Syntax	Description	Example
`<date & time> is at <time>`	Checks that the time point is at the specified time.	`if the return date of 'rented car' is at 05:00:00 PM`
`<date & time> is on <date>`	Checks that the time point is on a specific day.	`if the arrival date of customer is on 01/01/2018`
`<date & time> is on <day of week>`	Checks that the time point is on a specific day of the week.	`if the return date of 'rented car' is on Thursday`
`<date & time> is on <day of week> at <time>`	Checks that the time point is on a specific day of the week, at a specific time.	`if the return date of 'rented car' is on Thursday at 05:00:00 PM`
`<date & time> is on day <a number>`	Checks that the time point is on a specific day, specified by a number ranging from 1 to 31.	`if the arrival date of customer is on day 1`
`<date & time> is on <date & time>`	Checks that the two time points have the same date.	`if the arrival date of customer is on 01/01/2018 12:OO:OO PM`
`<date & time> is in hour <a number>`	Checks that the time point is on a specific hour.	`if 'flight departure' is in hour 13`
`<date & time> is in minute <a number>`	Checks that the time point is on a specific minute.	`if 'flight departure' is in minute 45`
`<date & time> is in second <a number>`	Checks that the time point is on a specific second.	`if 'flight departure' is in second 30`
`<date & time> is in <month>`	Checks that the time point is on a specific month.	`if 'flight departure' is in January`
`<date & time> is in <month> <year>`	Checks that the time point is in the specified month of the specified year.	`if 'flight departure' is in January 2019`
`<date & time> is in <year>`	Checks that the time point is in a specific year.	`if 'flight departure' is in 2019`

Table 4. Accessor functions for time points

Syntax	Description	
`the number of <time unit> between <date & time> and <date & time>`	Returns the number of years, months, days, hours, minutes, or seconds between two time points.	`if the number of minutes between the connection`
`the duration in <time unit> between <date & time> and <date & time>`	Returns the number of time units between the two time points measured in the specified time unit, where the *time unit* is one of *years*, *months*, *days*, *hours*, *minutes*, or *seconds*.	`set 'training duration' to the duration in month`

Syntax	Description	
`the <time component> of <date & time>`	Returns a time component of the specified time point, where the *time component* is one of *year*, *month*, *day of month*, *day of week*, *hour of day*, *minute of hour*, *second of minute*, *date*, or *time of day*.	`the day of week of the end of 'promotional perio`

Table 5. Conversion functions for time points

Syntax	Description	Example
`the local time of <date & time> in <string>`	Converts the time point to a specific time zone, in the format defined by IANA	`set the local time of 'flight departure' in "Pacific/Mi`
`the local time of <date & time> in the zone of <date & time>`	Converts the first time point to the time zone of the second time point.	`set the local time of 'flight departure' in the zone of`

Date

Table 6. Functions for comparing time points

Syntax	Description	Example
`<date> is after <date>`	Checks that the date comes after another date.	`the beginning of 'promotional period' is after 01/01/2018`
`<date> is after or the same as <date>`	Checks that the date comes after another date, or is the same as that other date.	`the beginning of 'promotional period' is after or the same as 0`
`<date> is before <date>`	Checks that the date comes before another date.	`the beginning of 'promotional period' is before 01/01/2018`
`<date> is before or the same as <date>`	Checks that the date comes before another date, or is the same as another date.	`the beginning of 'promotional period' is before or the same as`
`<date> is after <date> and before <date>`	Checks that the date is in a range in which both dates are excluded.	`the beginning of 'promotional period' is after the beginning of`
`<date> is after <date> and on or before <date>`	Checks that the date is in a range in which the first date is excluded, and the last date is included.	`the beginning of 'promotional period' is after the beginning of`
`<date> is on or after <date> and before <date>`	Checks that the date is on or after a specific date and before another date.	`the beginning of 'promotional period' is on or after the beginn`
`<date> is on or after <date> and on or before <date>`	Checks that the date is on or after a specific date and on or before another date.	`the beginning of 'promotional period' is on or after the beginn`

Table 7. Functions for testing that a component of the representation of a time point has a specific value

Syntax	Description	Example
`<date> is on <day of week>`	Checks that the date in on a specific day of the week.	`the beginning of 'promotional period' is on Sunday`
`<date> is on day <a number>`	Checks that the date is on a specific day, specified by a number ranging from 1 to 31.	`if the arrival date of customer is on day 1`
`<date> is in <month>`	Checks that the date in on a specific month.	`the beginning of 'promotional period' is in January`
`<date> is in <year>`	Checks that the date in on a specific year.	`the beginning of 'promotional period' is in the year 2018`
`<date> is in <month> the year <year>`	Checks that the date is in the specified month of the specified year.	`if the return date of 'rented car' is in October the year 2007`

Table 8. Accessor functions for time points

Syntax	Description	Example
`the number of <time unit> between <date> and <date>`	Returns the number of years, months, or days between two dates.	`if the number of months between the subscription date of custome`
`the <time component> of <date>`	Returns a time component of the specified date where the *time component* is one of *year*, *month*, *day of month*, or *day of week*.	`the day of month of the beginning of 'promotional period'`

Time

Table 9. Functions for comparing time points

Syntax	Description	
`<time> is after <time>`	Checks that the time point comes after another time point.	`if 'flight departure' is after 12:00:00 PM`
`<time> is after or the same as <time>`	Checks that the time point comes after another time point, or is the same as that other time point.	`if 'flight departure' is after or the same as 12:00:00 PM`
`<time> is before <time>`	Checks that the time point comes before another time point.	`if 'flight departure' is before 12:00:00 PM`
`<time> is before or the same as <time>`	Checks that the time point comes before another time point, or is the same as another time point.	`if 'flight departure' is before or the same as 12:00:00 PM`
`<time> is after <time> and before <time>`	Checks that the time point is in a range in which both time points are excluded.	`if 'flight departure' is after 12:00:00 PM and before 05:00:00`
`<time> is after <time> and on or before <time>`	Checks that the time point is in a range in which the first time point is excluded, and the last time point is included.	`if 'flight departure' is after 12:00:00 PM and on or before 05:`

Syntax	Description	
<time> is on or after <time> and before <time>	Checks that the time point is on or after a specific time point and before another time point.	if 'flight departure' **is on or after** 12:00:00 PM **and before** 05:
<time> is on or after <time> and on or before <time>	Checks that the time point is on or after a specific time point and on or before another time point.	if 'flight departure' **is on or after** 12:00:00 PM **and on or befo**

Day of week

Table 10. Functions for comparing time points

Syntax	Description	
<day of week> is after <day of week>	Checks that the day of the week comes after another day of the week. Note: Sunday is considered to be the first day of the week.	if the day of the return date of 'rented ca
<day of week> is after or the same as <day of week>	Checks that the day comes after another day of the week, or is the same as that other day.	if the day of the return date of 'rented ca
<day of week> is before <day of week>	Checks that the day of the week comes before another day of the week. Note: Sunday is considered to be the first day of the week.	if the day of the return date of 'rented ca
<day of week> is before or the same as <day of week>	Checks that the day comes before another day of the week, or is the same as another day of the week.	if the day of the return date of 'rented ca
<day of week> is after <day of week> and before <day of week>	Checks that the day is in a range in which both days are excluded.	if the day of the return date of 'rented ca
<day of week> is after <day of week> and on or before <day of week>	Checks that the day is in a range in which the first day is excluded, and the last day is included.	if the day of the return date of 'rented ca
<day of week> is on or after <day of week> and before <day of week>	Checks that the day is on or after a specific day of the week and before another day.	if the day of the return date of 'rented ca
<day of week> is on or after <day of week> and on or before <day of week>	Checks that the day is on or after a specific day of the week and on or before another day.	if the day of the return date of 'rented ca

Month of year

Table 11. Functions for comparing time points

Syntax	Description	
<month of year> is after <month of year>	Checks that the month comes after another month of the year.	if 'promotional period' **is after** Jun
<month of year> is after or the same as <month of year>	Checks that the month comes after another month of the year, or is the same as that other month.	if 'promotional period' **is after or**
<month of year> is before <month of year>	Checks that the month comes before another month of the year.	if 'promotional period' **is before** Ma
<month of year> is before or the same as <month of year>	Checks that the month comes before another month of the year, or is the same as another month.	if 'promotional period' **is before or**
<month of year> is after <month of year> and before <month of year>	Checks that the month is in a range in which both months of the year are excluded.	if 'promotional period' **is after** May
<month of year> is after <month of year> and on or before <month of year>	Checks that the month is in a range in which the first month is excluded, and the last month is included.	if 'promotional period' **is after** May
<month of year> is on or after <month of year> and before <month of year>	Checks that the month is on or after a specific month of the year and before another month.	if 'promotional period' **is on or aft**
<month of year> is on or after <month of year> and on or before <month of year>	Checks that the month is on or after a specific month of the year and on or before another month.	if 'promotional period' **is on or aft**

Year

Table 12. Functions for comparing time points

Syntax	Description	
<year> is after <year>	Checks that the year comes after another year.	if 'flight departure' **is after** 2018
<year> is after or the same as <year>	Checks that the year comes after another year, or is the same as that other year.	if 'flight departure' **is after or the same as** 2018
<year> is before <year>	Checks that the year comes before another year.	if 'flight departure' **is before** 2018
<year> is before or the same as <year>	Checks that the year comes before another year, or is the same as another year.	if 'flight departure' **is before or the same as** 2018

Syntax	Description	
`<year> is after <year> and before <year>`	Checks that the year is in a range in which both year are excluded.	`if 'flight departure' is after 2018 and before 2020`
`<year> is after <year> and on or before <year>`	Checks that the year is in a range in which the first year is excluded, and the last year is included.	`if 'flight departure' is after 2018 and on or before 2020`
`<year> is on or after <year> and before <year>`	Checks that the year is on or after a specific year and before another year.	`if 'flight departure' is on or after 2018 and before 2019`
`<year> is on or after <year> and on or before <year>`	Checks that the year is on or after a specific year and on or before another year.	`if 'flight departure' is on or after 2018 and on or before 2019`

Time period functions

Use time period functions to define, calculate, and compare time points.

Table 1. Functions for computing a time period

Syntax	Description	
`the period between <date & time> and <date & time>`	Returns the time period between two time points.	`the period between the new scheduled arrival date of 'fi`
`the period of <calendar duration> before <date & time>`	Returns the time period expressed in time units that precedes a specific date.	`the period of 7 days before 'purchase event'`
`the period of <calendar duration> after <date & time>`	Returns the time period expressed in time units that follows a specific date.	`the period of 1 month after 'purchase event'`
`the period of <calendar duration> ending at <date & time>`	Returns the time period expressed in time units that ends at a specific date.	`the period of 7 days ending at 'purchase event'`
`the period of <calendar duration> starting at <date & time>`	Returns the time period expressed in time units that starts a specific date.	`the period of 7 days starting at 'purchase event'`
`the period of <calendar duration> before <time period>`	Returns the time period expressed in time units that precedes another time period.	`the period of 7 days before 'trip period'`
`the period of <calendar duration> after <time period>`	Returns the time period expressed in time units that follows another time period.	`the period of 1 month after 'trip period'`

Table 2. Functions for computing a time point

Syntax	Description	

Syntax	Description	
<calendar duration> before <time period>	Returns a new time point that precedes the specified time point by the specified duration.	3 hours **before** 'flight period'
<calendar duration> after <time period>	Returns a new time point that follows the specified time point by the specified duration.	2 months 3 days **after** 'flight period'

Table 3. Functions for comparing a time period with another time period or a time point

Syntax	Description	
<time period> is during the same time as <time period>	Checks that the two time periods are identical and on the same time zone.	if the calendar week of meeting **is during the same time as** th
<time period> overlaps with <time period>	Checks that the time period overlaps another time period.	if 'flight period' **overlaps with** 'trip period'
<time period> is before <time period>	Checks that the time period comes before another time period.	if 'flight period' **is before** 'trip period'
<time period> is after <time period>	Checks that the time period comes after another time period.	if 'trip period' **is after** 'flight period'
<time period> is before <date & time>	Checks that the time period comes before a specific time point	if 'trip period' **is before** 3/11/2019 5:32:38 PM
<time period> is after <date & time>	Checks that the time period comes after a specific time point.	if 'trip period' **is after** 3/7/2019 5:32:38 PM
<time period> includes <date & time>	Checks that the time period includes a specific time point.	if 'trip period' **includes** the time of 'the purchase event'
<time period> starts at <date & time>	Checks that a time period starts at a specific time point.	if 'trip period' **starts at** 3/7/2019 6:00:00 PM
<time period> ends at <date & time>	Checks that a time period ends at a specific time point.	if 'trip period' **ends at** 3/11/2019 6:00:00 PM

Table 4. Functions for comparing durations

Syntax	Description	
<time period> is longer than <time period>	Checks that the length of a time period is longer than another one.	if 'trip period' **is longer than** 'flight period'
<time period> is longer than or equal to <time period>	Checks that the length of a time period is equal to or longer than another one.	if 'trip period' **is longer than or equal to** 'flight p
<time period> is shorter than <time period>	Checks that the length of a time period is shorter than another one.	if 'trip period' **is shorter than** 'flight period'
<time period> is shorter than or equal to <time period>	Checks that the length of a time period is equal to or shorter than another one.	if 'trip period' **is shorter than or equal to** 'flight

Syntax	Description	
`<time period> is longer than <calendar duration>`	Checks that the length of a time period is longer than a calendar duration.	`if 'trip period' is longer than 6 hours`
`<time period> is longer than or equal to <calendar duration>`	Checks that the length of a time period is equal to or longer than a calendar duration.	`if 'trip period' is longer than or equal to 2 hours 3`
`<time period> is shorter than <calendar duration>`	Checks that the length of a time period is shorter than a calendar duration.	`if 'trip period' is shorter than 2 hours`
`<time period> is shorter than or equal to <calendar duration>`	Checks that the length of a time period is equal to or shorter than a calendar duration.	`if 'trip period' is shorter than or equal to 2 hours`

Table 5. Accessor functions for time periods

Syntax	Description	
`the start of <time period>`	Defines the start point of a time period.	`the start of meeting is before 2019/12/01 12:00:00 AM`
`the end of <time period>`	Defines the end point of a time period.	`the end of meeting is before 2019/12/01 12:00:00 AM`

Calendar duration functions

Use duration functions to define and compute calendar durations.

Table 1. Functions for calendar durations

Syntax	Description	Example
`<number> <time unit>`	Returns a calendar duration in years, months, days, hours, minutes, or seconds.	`7 months`
`<date & time> - <calendar duration>`	Returns a new time point that precedes the specified time point by the specified duration.	`the beginning of training - 1 month`
`<date & time> + <calendar duration>`	Returns a new time point that follows the specified time point by the specified duration.	`the beginning of training + 1 month`
`<calendar duration> + <calendar duration>`	Returns a calendar duration that corresponds to the sum of the two durations.	`the duration of training + the duration of certification`

Strings

String types are represented by a set of characters enclosed in double quotation marks ("Text string").
To include certain special characters within a String, you must prefix them with a backslash (\). For example, to include a double quotation mark within a String, you would write `\"`. The backslash is required here to indicate that the String has not yet ended.

The following special character sequences are accepted within text strings:

- **`\n`** (new line character)
- **`\t`** (tab character)
- **`\b`**
- **`\r`** (carriage return character)
- **`\f`** (line feed character)
- **`\'`** (single quotation mark)
- **`\"`** (double quotation mark)
- **`\\`** (backslash)
- **String functions**
  Text operators define conditions based on strings.

String functions

Text operators define conditions based on strings.

Table 1. String functions

Syntax	Description	Example
<text> contains <text>	Tests that a string contains another string.	if the name of customer **contains** "Smith"
<text> does not contain <text>	Tests that a string does not contain another string.	if the name of customer **does not contain** "Smith"
<text> does not end with <text>	Tests that a string does not end with another string.	if the rental agreement code of customer **does not end with** "XG5"
<text> does not start with <text>	Tests that a string does not start with another string.	if the rental agreement code of customer **does not start with** "XG5"
<text> ends with <text>	Tests that a string ends with another string.	if the rental agreement code of customer **ends with** "XG5"
<text> is empty	Tests that a string is empty. An empty string contains no characters and is equivalent to "". A string is not empty if it contains a space (" ")	if the rental agreement code of customer **is empty**
<text> is not empty	Tests that a string is not empty. A string is not empty if it contains a space (" ")	if the rental agreement code of the customer **is not empty**
print <text>	Prints a string to standard out.	**print** the message "Hello World"
<text> starts with <text>	Tests that a string starts with another string.	if the rental agreement code of the customer **starts with** "XG5"
the length of <text>	Returns the number of characters in a string.	if **the length of** 'customer name' is more than 3
<text> + <text>	Concatenates two strings.	set decision to 'first name' **+** ' ' **+** 'last name'
<number> + <text>	Concatenates a number and a string. The result is treated as a string.	print "Spend just $" **+** ('discount threshold' - order total) **+** " more to receive a
<text> + <number>	Concatenates a string and a number. The result is treated as a string.	print "You received a 10% discount because your order was over $" **+** 'discount thres

Lists

Lists are collections of objects that are arranged in a sequence. You can use basic list functions, aggregation functions, and advanced stream functions to perform operations on lists.

- **List functions**
  List functions access values in a collection of objects and calculate the number of objects in a specific collection.
- **Aggregation functions**
  Aggregation functions calculate the average, total, minimum, or maximum value of the numeric attributes in a collection of objects, or the number of objects in a collection.

- **Stream operators**
  Stream operators let you process data in a collection of objects. They can be chained and are applied from left to right.

List functions

List functions access values in a collection of objects and calculate the number of objects in a specific collection.

Table 1. List functions

Syntax	Description	Example
`<object> is one of <list>`	Tests that an object is part of a list.	`if the item of order is one of 'discounted items'`
`<object> is not one of <list>`	Tests that an object is not part of a list.	`if 'customer category' is not one of 'all categories'`
`<list> contain <object>`	Tests that a list contains an object. This function is equivalent to <object> is one of <list>.	`if 'customer categories' contain Platinum`
`<list> do not contain <object>`	Tests that a list does not contain an object. This function is equivalent to <object> is not one of <list>.	`if 'customer categories' do not contain Normal`
`<attribute> of <list>`	Accesses a list of attribute values.	`if the countries of the addresses of customers contain "France"`

Aggregation functions

Aggregation functions calculate the average, total, minimum, or maximum value of the numeric attributes in a collection of objects, or the number of objects in a collection.
Aggregation functions compute a value from a collection of values.

You cannot use aggregation functions in the action part of the rule.

Table 1. Aggregation functions

Syntax	Description	Example
the minimum of <list of numbers> [defaulting to <numerical expression>]	Returns the smallest value in a list. If the list is empty or null and no default value is provided, the returned value is `null`.	**the minimum of** the ages of 'the customers', where each number is at least 18 **defaulting to** 100
the maximum of <list of numbers> [defaulting to <numerical expression>]	Returns the biggest value in a list. If the list is empty or null and no default value is provided, the returned value is `null`.	**the maximum of** { 'the good value', 'the bad value', 'the ugly value' }
the sum of <list of numbers>	Sums up the numbers in a list. If the list is empty or null, the returned value is `0`.	**the sum of** the ages of 'the customers', where each number is at least 18
the average of <list of numbers> [defaulting to <numerical expression>]	Computes the average value of a list. If the list is empty or null and no default value is provided, the returned value is `0`.	**the average of** 'the amounts'
the number of elements in <list>	Returns the total number of elements in a specific list.	**the number of elements in** 'the customers'

Stream operators

Stream operators let you process data in a collection of objects. They can be chained and are applied from left to right.

Unary stream operators

Table 1. Unary stream operators

Syntax	Description	Example
<list>, made distinct	Eliminates duplicates and returns a unique list of elements. Elements in the resulting list appear in encounter order. If the list is empty or null, an empty list is returned.	'the customers'**, made distinct**
<list>, reversed	Reverses the sorting order of elements of a list. If the list is empty or null, an empty list is returned.	'the customers'**, reversed**
<list>, limited to the first <integer> elements	Returns the first n elements of a list, as specified by an integer. If the list is empty or null, an empty list is returned.	'the customers'**, limited to the first** 2 **elements**
<list>, limited to the number of first elements specified by <expression>	Returns the first n elements of a list, as specified by an expression. If the list is empty or null, an empty list is returned.	'the customers', **limited to the number of first elements specified by** 'nbCustomers'
<list>, limited to the last <integer> elements	Returns the last n elements of a list, as specified by an integer. If the list is empty or null, an empty list is returned.	'the customers'**, limited to the last** 2 **elements**
<list>, limited to the number of last elements specified by <expression>	Returns the last n elements of a list, as specified by an expression. If the list is empty or null, an empty list is returned.	'the customers', **limited to the number of last elements specified by** 'nbCustomers'
<list>, skipping the first <integer> elements	Discards the first n elements of a list, as specified by an integer. If the list is empty or null, an empty list is returned.	'the customers'**, skipping the first** 2 **elements**
<list>, skipping the number of first elements specified by <expression>	Discards the first n elements of a list, as specified by an expression. If the list is empty or null, an empty list is returned.	'the customers', **skipping the number of first elements specified by** 'nbCustomers'

Syntax	Description	Example
<list>, where [elements called <variable> satisfy that] <condition>	Returns the elements of a list that satisfy a specific condition. If the list is empty or null, an empty list is returned.	'the customers'**, where** the age of each customer is more than 18
<list>, limited to <type>	Returns the elements of a list that have a specific type. If the list is empty or null, an empty list is returned.	'the pets'**, limited to** dogs
<list>, sorted [in ascending order | in descending order]	Sorts the elements of a list in ascending or descending order. If no order is specified, the list is sorted in ascending order. If the list is empty or null, an empty list is returned.	the amounts of 'the customers'**, sorted in descending order**
<list>, sorted by <attribute> [in ascending order | in descending order] { then by <attribute> [in ascending order | in descending order] }	Returns the elements of a list in a given sort order. The elements can be returned in ascending or descending order. If no order is specified, the list is sorted in ascending order. If the list is empty or null, an empty list is returned.	'the customers'**, sorted by** name **then by** age **in descending order**

Binary stream operators

Table 2. Binary stream operators

Syntax	Description	Example
<list>, concatenated with <list>	Concatenates two lists. If both lists are empty or null, an empty list is returned.	'the existing customers'**, concatenated with** 'the new customers'
<list>, combined with <list>	Combines two lists. The resulting list does not contain any duplicate element. If both lists are empty or null, an empty list is returned. This operator is equivalent to **`<list>, concatenated with <list>, made distinct`**	'the existing customers'**, combined with** 'the new customers'
<list>, deducting <list>	Removes elements of a list from another list. The resulting list does not contain any duplicate element. If the first list is empty or null, an empty list is returned.	'the customers'**, deducting** 'the blocklisted customers'
<list>, belonging to <list>	Returns a list of elements belonging to both of the specified lists. The resulting list does not contain any duplicate element. If one of the lists is empty or null, an empty list is returned.	'the customers'**, belonging to** 'the blocklisted customers'

Element selectors

Table 3. Element selectors

Syntax	Description	Example
the first element in <list> [defaulting to <expression>]	Returns the first element in a list. If the list is empty or null and no default value is provided, an exception is thrown.	**the first element in** 'the customers'
the last element in <list> [defaulting to <expression>]	Returns the last element in a list. If the list is empty or null and no default value is provided, an exception is thrown.	**the last element in** 'the customers'
the element at position <integer> in <list> [defaulting to <expression>]	Returns the element that is at a specific position in a list, where the position is specified by an integer. If the list is null or the specified integer is negative or exceeds the number of elements in the list, an exception is thrown.	**the element at position** 2 **in** 'the customers'
the element at the position corresponding to <integer> in <list> [defaulting to <expression>]	Returns the element that is at a specific position in a list, where the position is specified by an expression. If the list is null or the specified integer is negative or exceeds the number of elements in the list and no default value is provided, an exception is thrown.	**the element at the position corresponding to** the index of the loan **in** 'the customers'

Stream constructors

These constructors are used to generate streams of integers. The following example shows a rule that adds ten items to a decision:

```
for each index, in [1 .. 10]
add a new item where the number is index to decision;
```

Table 4. Stream constructors

Syntax	Description	Example
the range from <integer> to <integer>	Returns a set of values contained between two integers.	**the range from** 0 **to** the number of 'the customers' - 1
[<integer> .. <integer>]	Returns a set of values contained between two integers.	[1 .. 10]
[<integer> .. <integer>) or [<integer> .. <integer> [	Returns a set of values contained between two integers, including the lower bound and excluding the upper bound.	[1 .. 10)
(<integer> .. <integer>] or] <integer> .. <integer>]	Returns a set of values contained between two integers, excluding the lower bound and including the upper bound.	(1 .. 10]
(<integer> .. <integer>) or] <integer> .. <integer> [	Returns a set of values contained between two integers, excluding the lower bound and the upper bound.	(1 .. 10)

Quantified conditions

Table 5. Quantified conditions

Syntax	Description	Example
all elements [called <variable>] in <list> satisfy that <condition>	Returns **true** if all the elements in a list satisfy the specified condition. If the list is empty or null, the returned value is **true**.	**all elements in** 'the visitors' **satisfy that** the age of each customer is more than 30 and the gender of each customer is male
any element [called <variable>] in <list> satisfies that <condition>	Returns **true** if any element in a list satisfies the specified condition. If the list is empty or null, the returned value is **false**.	**any element in** 'the visitors' **satisfies that** the age of this customer is more than 30 and the gender of this customer is male
no element [called <variable>] in <list> satisfies that <condition>	Returns **true** if none of the elements in a list satisfy the specified condition. If the list is empty or null, the returned value is **true**.	**no element in** 'the visitors' **satisfies that** the age of this customer is more than 30 and the gender of this customer is male

Object functions

Object functions define conditions based on objects (as opposed to attributes of objects). A customer is an object; the age of the customer is an attribute of the customer object.

Table 1. Object functions

Syntax	Description	Example
`<object> is <object>`	Tests that two objects are equivalent.	`if 'current customer' is the customer on 'rental agreement'`
`<object> is not <object>`	Tests that two objects are not equivalent.	`if 'current customer' is not the customer on 'rental agreement'`
`the number of <objects>`	Returns the total number of objects of the specified type.	`if the number of vehicles is more than 5`
`<attribute> of <object>`	Associates two business terms.	`if the address of customer contains "France"`
`new <object> where`	Creates a new instance of the specified object type. To define the values of this new object, you add the **where** keyword followed by one or more attributes.	`define Jane as a new customer where 'customer id' is "jane@example.com"`

Punctuation in rules

The rule language uses punctuation to identify specific language artifacts and to avoid ambiguous syntax. When you edit business rules, mandatory punctuation is usually predicted in the completion menu.

Backslash \

The backslash is used to prefix special characters in text strings.

Comma ,

The comma marks the end of **Where** statements.

Example

This example shows a **Where** statement that is closed by a comma.

```
if
  there is at least one car
    where
      the color of this car is blue,
then ...
```

Some language constructs may be terminated by an optional comma.

Example

This example shows a construct that is terminated by an optional comma. This may prevent the construct becoming ambiguous in a particular context.

```
all following conditions are true:
  - the color of car is blue
  - the price of car is more than 1000,
```

Double quotation mark "

Double quotation marks are used to enclose string literals; that is, text strings.

Example

This example shows an action phrase in which the message to be displayed is a text string enclosed in double quotation marks.

```
print "You received a discount!";
```

Parentheses ()

Parentheses can be used to group any expression. They can help clarify the default precedence of logical operators linking conditions to one another. You can also use parentheses to regroup condition statements and hence change the order in which they are interpreted.

Example

This example shows nested parentheses in an action phrase.

```
print "the customer: " + (the name of customer of ('shopping cart'));
```

Semicolon `;`

The semicolon marks the end of variable definitions and action phrases.

Single quotation mark `'`

Single quotation marks are used to enclose variables. Single quotation marks are optional for single-word variables and mandatory for variables that consist of several words.

Example

In this example, the variables **`customer`** and **`the cart`** are defined. The **`customer`** variable does not need to be enclosed in single quotation marks because it is a single word, whereas **`the cart`** needs to be delimited because it consists of two words.

```
definitions
    set customer to a customer;
    set 'the cart' to the shopping cart of customer;
```

String interpolation

String interpolation is an alternative to building strings via concatenation.
String interpolation provides a more readable and convenient syntax to create formatted strings:

- Build strings using a mix of literals, constants, and variables.
- Concatenate elements without using the concatenation operator **`+`**.
- Produce multiline strings.
- Format locale-sensitive information.

The following example shows two expressions that would produce the same output, using string concatenation and string interpolation:

```
// String concatenation:
set decision to Greeting + ", " + 'Courtesy Title' + " " + the last name of Person + "!" ;
// String interpolation:
set decision to `{ Greeting }, { 'Courtesy Title' }{ the last name of Person } !` ;
// Both would produce the same output, for example:
Good morning Mr. Jones!
```

Syntax

Interpolated strings are delimited with backtick characters (`). Their syntax is as follows:

```
`<text>? ( {<interpolationExpression> [:<format>] }  <text>?)*`
```

The **`<text>`** and **`<format>`** elements ares optional. The following table lists the details of each element:

Element	Description
`text`	Any type of literal text.
`interpolationExpression`	A rule language expression that produces a result to be formatted.
`format`	A string format used to format the resulting expression as a string. Note: Only the Java **`Format`** class and its subclasses are supported.

To include a brace, (**`{`** or **`}`**), in the text produced by an interpolated string, use escaped braces, (**`/{`** or **`/}`**).

Examples

The following example shows a multiline interpolated string that contains a list of values, including a formatted date:

```
print `Name: {the name of John},
famous: {John is famous},
birth date: {the birth date of John:date,short},
hair color: {the hair color of John},
category: {the category of John}.`;
```

The expected output would be similar to:

```
`Name: John,
famous: true,
birth date: Feb 4, 1977,
hair color: brown,
category: silver.`;
```

The following example shows a single line interpolated string that contains a formatted number:

```
print `Pi = {Value:number,#.##}`;
```

If `Value = 3.141592653589793`, the expected output would be:

```
Pi = 3.14
```

Advanced Rule Language (ARL)

As an alternative to the rule language, advanced users can write rules using the Advanced Rule Language in task models.
For more information about switching from the rule language to the ARL rule language in task models, see Choosing a language.

- **ARL syntax**
  The ARL rule language uses a syntax close to the syntax of the Java language.
- **ARL task model examples**
  This section provides examples of using the ARL rule language in task models.

ARL syntax

The ARL rule language uses a syntax close to the syntax of the Java language.

Curly braces

Curly braces, `{ }`, mark the beginning and the end of code blocks:

```
if (a.contains(b)) {
   note("a contains " + b);
}
```

Backquotes

Unlike in Java, you can use backquotes, ``, in the ARL rule language to reference variables and classes. This allows you to reference names that contain spaces and keywords:

```
String `the lazy dog` ="pluto";
int `=` =12;
```

Comments

You can add comments to help you structure and document rules.

The ARL rule language uses forward slashes, `//`, for single line comments:

```
// This is a comment
```

Block comments start with `/*` and run until a closing `*/`:

```
/* This is also a comment
but written over multiple lines */
```

- **Variables**
  Variables are containers that save data values.
- **Operators**
  Operators perform specific operations on one or more operands, and return a result.
- **Type conversion**
  The ARL rule language allows you to convert values from one data type to another.
- **Expressions**
  Expressions are made up of variables, operators, and method invocations.
- **Statements**
  Statements form complete units of execution. Statements end with a semicolon (`;`).
- **Keywords**
  Keywords are reserved words in the core ARL rule language.

Variables

Variables are containers that save data values.
Variables are assigned a data type. Their scope is limited to the code block where they are defined.

Variables can be declared with an initial value:

```
double discount = 1.0;
String name ="John Doe";
```

Or without an initial value:

```
int numberOfCustomers;
String name;
```

Variables can also be final, meaning that their value cannot be modified.

```
finalString name = "John Doe";
```

- **Data types**
  Variables in the ARL rule language must have an associated data type.

Data types

Variables in the ARL rule language must have an associated data type.
Data types are divided into two groups: primitive data types and non-primitive data types.

Primitive types

The ARL rule language supports eight primitive data types:

Table 1. List of supported primitive types

Data type	Description
`Boolean`	Stores `true` or `false` values
`byte`	Stores whole numbers from -128 to 127
`char`	Stores a single character Note: You can also use ASCII values to display certain characters.
`double`	Stores fractional numbers with a precision of 15 decimal points
`float`	Stores fractional numbers with a precision of 6 to 7 decimal digits
`int`	Stores whole numbers from -2,147,483,648 to 2,147,483,647
`long`	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
`short`	Stores whole numbers from -32,768 to 32,767

Non-primitive data type

The ARL rule language supports one non-primitive data type: **`string`**. The **`string`** data type stores a set of characters surrounded by double quotes.

For example:

```
"Hello world"
```

Operators

Operators perform specific operations on one or more operands, and return a result.

- Arithmetic operators
- Comparison operators
- Logical operators
- Bitwise operators
- Assignment operators
- **`instanceof`** operator
- **`?:`** operator

Arithmetic operators

Table 1. List of supported arithmetic operators

Operator	Description	Example	Result
`+`	Additive operator	`12+3`	`15`
`-`	Subtraction operator	`12-2`	`10`
`*`	Multiplication operator	`12*50`	`300`
`/`	Division operator	`12/2`	`6`
`%`	Remainder operator	`10%3`	`1`

The `+` operator can also be used to concatenate strings. For example:

```
String name ="John"+" "+"Doe"
```

Comparison operators

The following operators can be applied to all types:

Table 2. List of supported comparison operators

Operator	Description	Example	Result
`==`	Equal to	`12 == 5`	`false`
`!=`	Not equal to	`12 != 5`	`true`

The following operators can be applied to numeric types only:

Table 3. List of supported comparison operators

Operator	Description	Example	Result
`>`	Greater than	`12 > 5`	`true`
`>=`	Greater than or equal to	`12 >= 5`	`true`
`<`	Less than	`12 < 5`	`false`
`<=`	Less than or equal to	`12 <= 5`	`false`

Logical operators

Logical operators operate on Boolean values.

Table 4. List of supported logical operators

Operator	Description	Example	Result
`!`	Logical NOT	`!true`	`false`
`&`	Logical AND	`true & false`	`false`
`|`	Logical OR	`true | false`	`true`
`^`	Logical XOR (exclusive OR)	`true ^ false`	`true`
`&&`	Conditional AND	`true && false`	`false`
`||`	Conditional OR	`true || false`	`true`

With conditional operators, the second operand is evaluated only if necessary.

Bitwise operators

Bitwise operators operate on individual bits. They can be applied on integer values such as long, int, short, and byte.

Table 5. List of supported bitwise operators

Operator	Description	Example	Result
`~`	Binary complement	`~0`	`255`
`&`	Binary AND	`6 & 2`	`2`
`|`	Binary OR	`6 | 1`	`7`
`^`	Binary XOR (exclusive OR)	`6 ^ 2`	`4`
`<<`	Signed left shift	`10 << 1`	`20`
`>>`	Signed right shift operator	`10 >> 1`	`5`
`>>>`	Unsigned right shift	`-50 >>> 2`	`51`

Assignment operators

The assignment operator is used to assign a value to a variable. It is denoted by a single equal sign (**=**).

```
name = "John Doe"
```

The assignment operator can also be combined with arithmetic operators.

```
x += 5
```

In this example, **5** is added to the variable **x**. It is equivalent to **x = x + 5**.

`instanceof` operator

The **instanceof** operator is a comparison operator that can be used to test if an object is of a given type.

```
Person person = ...
Customer customer =null;
if (person instanceof Customer)
  customer = (Person)person;
```

`?:` operator

The **?:** operator is a ternary conditional operator. It is a condensed form of the **if-else** construct.

```
max = (x > y) ? x : y;
```

The first operand must be a Boolean expression.

Type conversion

The ARL rule language allows you to convert values from one data type to another.

Implicit type conversion

Primitive types support implicit conversions also known as widening conversions.

Widening conversions mean that you convert smaller types to larger types:

- `byte` to `short`, `int`, `long`, `float`, or `double`
- `short` to `int`, `long`, `float`, or `double`
- `int` to `long`, `float`, or `double`
- `long` to `float` or `double`
- `float` to `double`

Note: Converting `int` to `float`, or `long` to `float` or `double`, can result in a loss of precision.

Autoboxing and unboxing conversion

Autoboxing is the automatic conversion performed by the ARL compiler to convert primitive types to their corresponding object wrapper classes. For example, converting an `int` to an `Integer`, or a `double` to a `Double`.

If the conversion goes the other way, this is called unboxing.

Cast operator

Regular Java-like operators can be used to convert values from one type into another. For example:

```
int x = 20;
byte y = (byte)x;
```

As operator

Like in C#, the cast operator `as` can be used to perform conversions between reference types or nullable types. For example:

```
Customer customer = person as Customer;
```

Expressions

Expressions are made up of variables, operators, and method invocations.

- Intervals
- `switch`
- Instance creation
- Method invocation

Intervals

Intervals are specific to the ARL rule language.

Intervals can be used in `swicth` expressions, and with the `in` and `!in` operators.

Intervals can be finite or infinite, using `-oo` for minus infinity and `+oo` for plus infinity. Interval bounds can be included in intervals.

For example:

```
2 in [1,3]
x !in ]-oo,0]
```

`switch`

The `switch` keyword can be used in expressions. It is quite similar to the `switch` expression introduced in Java 14.

```
String label =switch (experience) {
      case1, 2:"Junior";
      case3, 4:"Senior";
      case5:"Expert";
      default:thrownewIllegalArgumentException("unexpected experience");
    };
```

Note: While `->` is used as a separator in Java 14, `:` is used as a separator in the ARL rule language.
For more information about the types that can be used with `switch`, see the Statements section.

Instance creation

Like in Java, the `new` keyword can be used to create an instance, with or without passing parameter values.

```
List mylist = newArrayList();
Point myPoint = newPoint(1,2);
```

Restriction: Unlike in Java, you can not create anonymous classes as expressions in the ARL rule language.

Method invocation

Like in Java, static methods and instance methods can be invoked in the ARL rule language.

The following example shows a static method call:

```
BigInteger bi = ...
if (BigInteger.valueOf(12).equals(bi))
  // code to be executed
```

The following example shows an instance method call:

```
String s = ...
if (s.contains("A"))
  // code block
```

Some methods can take a variable number of arguments. This is also known as Variable Arguments, or Varargs.

The following example shows a method with a variable number of arguments:

```
public static int sum(int... args) {
  int sum = 0;
  for(int x: args){
    sum += x;
  }
  return sum;
```

It can be used as follows:

```
int result = sum(2,5,6);
```

Restriction: Methods with a `void` return type can only be used as statements.

Statements

Statements form complete units of execution. Statements end with a semicolon (`;`).
The ARL rule language supports basic statements and control statements.

Basic statements

The following types of expression can also be used as basic statements by ending the expression with a semicolon:

- Instance creation expressions
- Method invocations
- Assignment expressions

Control statements

- `if`
- `switch`
- `while`
- `do...while`
- `for`
- `for each`
- `break`
- `continue`
- `return`

`if`

The `if` statement contains a test and a statement. If the test returns `true`, the statement block is executed.

```
if (username == null)
  username = "John Doe";
```

The `if` statement can also contain an `else` clause. If the test returns `false`, the first statement block is skipped over and the statement block following `else` is executed.

```
if (username == null)
  usernames += ";John Doe";
else
  usernames += ";" + username;
```

`switch`

The `switch` statement is used to execute a block of code among several alternatives.

```
switch(n):
  case 1:
    // code to be executed
  case {2, 4, 6}:
    // code to be executed
  case [10,+oo[:
    // code to be executed
  default:
    // code to be executed
```

Note: Unlike in Java, it is not necessary to add a `break` statement at the end of each `case` clause.
The expression after the `switch` can be of type:

- Primitive

- `string`
- `enum`
- Comparable

The following example shows an expression of type comparable:

```
BigInteger x = ...
switch(x) {
  case java.math.BigInteger.valueOf(12L):
    // code to be executed
```

The `case` expression can be of type:

- Literal
- List of literals in curly braces
- Interval of literals

The following example:

```
switch(n):
  case 2
  case 4
  case 6:
    // code to be executed
  default:
    // code to be executed
```

Is equivalent to:

```
switch(n):
  case {2, 4, 6}:
    // code to be executed
  default:
    // code to be executed
```

while

The `while` statement contains a test and a statement. If the test evaluates to `true`, the statement block is executed. The `while` statement continues executing the statement block until the test evaluates to `false`.
An infinite loop can be implemented using the `while` statement as follows:

```
while (true)
  // do something
```

do...while

The `do...while` statement is close to the `while` statement, except that the test occurs at the bottom of the loop after the execution of the statement block.

```
do
  // do something
while (false)
```

for

The `for` statement creates a loop that consists of three expressions and a statement to be executed in the loop.

```
for (initialization;test;increment)
  statement
```

1. The `initialization` initializes a variable. This is executed only once.
2. The `test` is evaluated. If it evaluates to `true`, the statement block is executed.
3. The `increment` updates the value of `initialization`.
4. The `test` is evaluated again. The process continues until `test` evaluates to `false`.

It is equivalent to:

```
initialization
while (test) {
  statement
  increment
}
```

Variables in `for` loops are scoped.

```
for (int i = 0; i < 10; i += 1)
  // do something
```

In this example, the variable `i` does not exist outside of the `for` loop.

for each

The `for each` statement is used to iterate through items of a list or an array. It uses the `for` keyword with the following syntax:

```
for (Type varname : list-or-array-expression)
  statement
```

For example:

```
List<String> list = new ArrayList<java.lang.String>();

for (String str : list) {
  // code to be executed
}
```

break

The `break` keyword can be used in `while`, `do...while`, and `for` statements to break out of a loop.

```
boolean found = false;
for (int i = 0; i < data.length; i+=1) {
  if (data[i] == target) {
    found = true;
    break;
  }
}
```

Note: Labeled `break` statements are not supported in the ARL rule language.

`continue`

The `continue` keyword can be used in `while`, `do...while`, and `for` statements to skip the current execution of a loop and continue with the next iteration in the loop.

```
for (int i = 0; i < data.length; i+=1) {
  if (data[i] == target) {
    continue;
  }
  process(data[i]);
}
```

`return`

The `return` keyword is used to exit from a method, with or without a value.

```
return x * x;
```

This example implements a method that computes the square of a number.
Methods of type `void` can
For a method declared `void`, the `return` keyword can be used without a value.

Keywords

Keywords are reserved words in the core ARL rule language.
Note: Keywords can be used as package names, class names, and variable names if enclosed in backquotes (``).
The following list is the complete set of keywords in the ARL rule language:

Table 1. List of supported keywords

`abstract`	`aggregate`	`as`	`assert`	`break`	`case`
`catch`	`class`	`const`	`continue`	`default`	`do`
`else`	`enum`	`explicit`	`extends`	`false`	`final`
`finally`	`for`	`goto`	`if`	`implements`	`implicit`
`import`	`in`	`instanceof`	`interface`	`native`	`new`
`null`	`operator`	`out`	`package`	`private`	`property`
`protected`	`public`	`restricts`	`return`	`static`	`stipulation`
`stricfp`	`super`	`switch`	`synchronized`	`then`	`this`
`throw`	`throws`	`transient`	`true`	`try`	`typedef`
`void`	`volatile`	`when`	`while`	`xpath`	`xs`

ARL task model examples

This section provides examples of using the ARL rule language in task models.

Initial value

You can use the ARL rule language specify the initial value of a variable.

You can enter a literal value:

```
10.0
```

You can create a new object:

```
newShipment()
```

You can use the value of a variable to initialize the value of another variable:

```
12* monthlyIncome
```

In this example, the value of the `monthlyIncome` variable is used to compute the value of another variable.

Action

You can use the ARL rule language to modify variables in action nodes, and in the initial and final actions of task nodes. For example:

```
discount =10.0;
```

You can add a note:

```
note("the discount is now: "+discount);
```

You can also write arbitrary complex code:

```
for (Shipment shipment : shipments) {
  insert(shipment);
}
```

In this example, the shipments are copied from a variable of type list **shipments** and into the working memory.

The following methods can be used in actions:

note(<string>)
: Creates a note. Notes are displayed in the execution trace.

insert(<object>)
: Creates an object and insert it into the working memory.

retract(<object>)
: Removes an object from the working memory.

retractAll()
: Removes all objects from the working memory.

Transition conditions

You can use Boolean expressions in ARL to specify transition conditions.

The following example shows an object with a Boolean attribute:

```
loan.accepted
```

You can also write arbitrary complex code:

```
loan.accepted || (loan.amount <100&& customer.grade ==Grade.Gold)
```

Rule selection filter

You can specify a rule selection filter with a Boolean expression in the ARL rule language. The **rule** variable can be used to represent the rule to be selected or ignored.

The rule name can be used to select a rule:

```
return rule.name.startsWith("XYZ")
```

You can access the following attributes from the **rule** variable:

name
: The fully qualified name of rule, including its package name.

shortName
: The short name of the rule, without its package name.

packageName
: The name of the package containing the rule. It can be **null**.

description
: The description of the rule.

IBM Automation Decision Services - annotations

Packages

Package	Description
com.ibm.rules.engine.annotations	Annotations that have an effect on the rule compiler and execution.
ilog.rules.bom.annotations	Annotations to configure how a Java class is imported into a business object model and vocabularies.

Skip navigation links

- Overview
- Package
- Class
- Use
- Tree
- Deprecated
- Index

- Help

IBM Automation Decision Services - annotations

- All Classes

Skip navigation links

- Package
- Class
- Use
- Tree
- Deprecated
- Index
- Help

IBM Automation Decision Services - unit testing

- All Classes
- SEARCH: search reset

All Packages

Package Summary

Package	Description
com.ibm.decision.run.test.junit5	Classes for unit testing decisions using JUnit 5

Skip navigation links

- Package
- Class
- Use
- Tree
- Deprecated
- Index
- Help

IBM Automation Decision Services - unit testing

- All Classes

Skip navigation links
IBM Automation Decision Services - execution API

- Overview
- Package
- Class
- Use
- Tree
- Index
- Help

SEARCH: search reset

IBM Automation Decision Services - execution API

Packages
Package
Description
com.ibm.decision.run
Java API for executing decisions.
com.ibm.decision.run.provider
Classes supporting the discovery and usage of decision runners in order to execute decisions.
com.ibm.decision.run.trace
Contains classes related to the execution trace.
The main class of this package is `Trace`.
com.ibm.rules.engine.observer
Observer engine API.
com.ibm.rules.engine.util
Utility classes including properties and exceptions.

Troubleshooting

Find out here how to solve some issues you might encounter with your Automation Decision Services instance.

Automation Decision Services pods fail to start due to certificate issues

Symptoms

The following Automation Decision Services pods are not starting:

- Decision Designer (if Decision Designer is installed)
  - `<instance_name>-ads-restapi`
  - `<instance_name>-ads-credentials-service`
  - `<instance_name>-ads-run-service`
  - `<instance_name>-ads-git-service`
  - `<instance_name>-ads-embedded-build-service`
- Decision runtime pods (if the decision runtime is installed)
  - `<instance_name>-ads-runtime-service`

Causes

Decision Designer or the decision runtime is not available.

Diagnosing the problem

Check the logs for the pods. For example:

```
kubectl logs <pod_name> --since 1m
```

Check if there is any message regarding a certificate issue. For example:

```
2022-08-05T07:41:31.413+0000 W  NETWORK  [js] SSL peer certificate validation failed: unable to get local issuer certificate
2022-08-05T07:41:31.431+0000 E  QUERY    [js] Error: Authentication failed. :
2022-08-05T07:41:31.433+0000 E  -        [main] exiting with code 1
```

Resolving the problem

Configure the MongoDB certificate. For more information, see Step 4 in Configuring MongoDB storage.

MongoDB pod fails to start on `CreateContainerConfigError`

Symptoms

The Automation Decision Services mongo pod fails to start. The status of the pod is **`CreateContainerConfigError`**. The command **`kubectl get pod -l app.kubernetes.io/name=ads-mongo -o yaml`** displays the following error message:

```
container has runAsNonRoot and image will run as root
```

Resolving the problem

There are two possible solutions:

- On OpenShift, reconfigure the Security Context Constraint (SCC) of the cluster so that the pods start with the **`restricted`** SCC.
  1. Check the current SCC applied to the mongo pod by running the following command:

     ```
     kubectl get pod -l app.kubernetes.io/name=ads-mongo -o yaml | grep scc
     ```

  2. To inspect the users and service accounts assigned to an SCC, run the command:

     ```
     oc get scc <scc_name> -o yaml
     ```

For example:

```
oc get scc anyuid -o yaml
```

Tip: The *scc_name* is returned by the command you run in step 1.

3. To remove an SCC assigned to a user, run the command:

```
oc adm policy remove-scc-from-user <scc_name> <user_name>
```

- Force the mongo pod to start with a non-zero UID.
  Add the following parameter in the custom resource file:

```
ads_configuration:
  mongo:
     run_as_user: 1000
```

The change is applied after the operator reconciliation loop.

MongoDB pod fails to start on container exit code 14 or 100

Symptoms

The Automation Decision Services mongo pod fails to start. The status of the mongo container is **Error** with exit code 14 or 100. Logs of the container show errors like:

```
Unable to read the storage engine metadata file: FileNotOpen: Failed to read metadata from /data/db/storage.bson
```

```
code 100: exception in initAndListen: IllegalOperation: Attempted to create a lock file on a read-only directory: /data/db, terminating
```

Causes

There is probably a mismatch of permissions between the existing data on the persistent volume assigned to the pod and the UID/GID of the running mongoDB container. This can happen when applying the procedure described for MongoDB pod fails to start on CreateContainerConfigError, or when reusing a persistent volume from a previous configuration of Automation Decision Services.

Resolving the problem

Follow this procedure to fix the permissions.

1. Scale down the mongodb deployment to terminate all **ads-mongo** pods.

```
kubectl scale deployment ibm-ads-operator-controller-manager --replicas 0
```

```
kubectl scale deployment -l 'app.kubernetes.io/name=ads-mongo' --replicas 0
```

2. Determine the UID used to run the mongo container on your cluster. On OpenShift, the **restricted** Security Context Constraint (SCC) is applied by default to Automation Decision Services pods. This SCC assigns the lowest possible UID from a range defined on the namespace. To get this range, use the following command (replace **<icp4a namespace>** with the actual namespace where the Cloud Pak is installed).

```
kubectl get ns <ads namespace>  -o yaml | grep "openshift.io/sa.scc.uid-range"
```

The output looks something like `openshift.io/sa.scc.uid-range: 1000750000/10000`.

So in this example, the UID of the mongo container is **1000750000**.

Alternatively, you can explicitly set the UID of the mongo container as explained in MongoDB pod fails to start on CreateContainerConfigError.

3. Log in as a cluster administrator, and then start a temporary helper pod with access to the persistent volume. You might have to adapt the name of the persistent volume claim (PVC). Run the command **kubectl get pvc** to get the actual name of the PVC for your instance.

```
kubectl apply -f - <<EOF
apiVersion: v1
kind: Pod
metadata:
  name: mongo-pvc-helper
spec:
   securityContext:
      runAsUser: 0
   containers:
   - command:
       - sh
       - -c
       - while true ; do echo alive ; sleep 10 ; done
     image: busybox
     imagePullPolicy: Always
     name: mongo-pvc-helper
     resources: {}
     securityContext:
       capabilities:
         drop:
         - ALL
     volumeMounts:
     - mountPath: /mongodata
       name: mongodata
   volumes:
   - name: mongodata
```

```
    persistentVolumeClaim:
      claimName: ads-ads-mongo-pvc
EOF
```

4. Use this new pod to adjust the permissions of the MongoDB data files.

```
kubectl exec mongo-pvc-helper -it sh
$ chown -R 1000750000:0 /mongodata
$ rm /mongodata/mongod.lock
$ exit
```

5. Kill the temporary helper pod.

```
kubectl delete pod mongo-pvc-helper
```

6. Restart the **ads-mongo** instance.

```
kubectl scale deployment -l 'app.kubernetes.io/name=ads-mongo' --replicas 1

kubectl scale deployment ibm-ads-operator-controller-manager --replicas 1
```

Cannot create a new decision automation or access an existing one

Symptoms

The creation of a new decision automation fails. Opening an existing decision automation also fails.

Causes

Pods have started but they do not form an akka cluster. In some occasions, the internal Git Service component of Decision Designer needs to be restarted manually.

Resolving the problem

You must restart the Git Service.

1. Get the list of the Decision Designer **git service** pods.

```
$ kubectl get pods -l app.kubernetes.io/name=ads-git-service -o wide
NAME                                      READY   STATUS    RESTARTS   AGE     IP              NODE
NOMINATED NODE   READINESS GATES
ads-ads-git-service-dd5757549-29sg8   1/1     Running   0          7m23s   10.123.45.678   worker2.mycluster
<none>           <none>
ads-ads-git-service-dd5757549-5m9px   1/1     Running   7          3d23h   10.123.4.56     worker0.mycluster
<none>           <none>
```

2. Pick one pod in the list and apply the following command to it.

```
$ kubectl exec ads-ads-git-service-dd5757549-29sg8 -- curl 'https://localhost:8558/cluster/members' | jq
{
"members": [],
"oldestPerRole": {},
"selfNode": "akka://git-cluster@10.254.15.252:2551",
"unreachable": []
}
```

   If the command fails with an error `Connection refused`, restart the pod of the **git service** after checking that the **mongo service** is in Ready status.

   If the command succeeds, inspect the JSON output. Make sure that the **unreachable** list is empty and that the **members** structure contains one entry for each pod listed in step 2. The output displayed above shows no member. Probably another pod of the **git service** is unreachable. It needs to be restarted.

3. To restart a **git service** pod, run the following command.

```
$ kubectl delete pod ads-ads-git-service-dd5757549-5m9px
```

   After the new pod becomes ready, the command above should return the following output on each pod. Note that the **unreachable** list is empty and that the **members** list shows an entry for each pod.

```
$ kubectl exec ads-ads-git-service-5567986886-2gt96 -- curl 'http://localhost:8558/cluster/members'|jq .
{
  "leader": "akka://git-cluster@10.254.20.34:2551",
  "members": [
    {
      "node": "akka://git-cluster@10.254.20.34:2551",
      "nodeUid": "-7211840593743872363",
      "roles": [
        "gitShardRole",
        "dc-default"
      ],
      "status": "Up"
    },
    {
      "node": "akka://git-cluster@10.254.4.203:2551",
      "nodeUid": "-3866802142924369892",
      "roles": [
        "gitShardRole",
        "dc-default"
      ],
```

```
      "status": "Up"
    },
    {
      "node": "akka://git-cluster@10.254.8.109:2551",
      "nodeUid": "7848020673616832611",
      "roles": [
        "gitShardRole",
        "dc-default"
      ],
      "status": "Up"
    }
  ],
  "oldest": "akka://git-cluster@10.254.4.203:2551",
  "oldestPerRole": {
    "gitShardRole": "akka://git-cluster@10.254.4.203:2551",
    "dc-default": "akka://git-cluster@10.254.4.203:2551"
  },
  "selfNode": "akka://git-cluster@10.254.8.109:2551",
  "unreachable": []
}
```

Decision returns null

Symptoms

In Decision Designer, when you run a decision, the decision returns null.

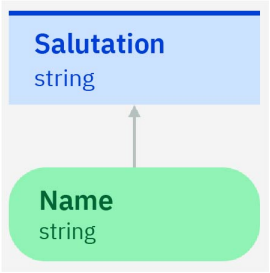
Causes

The first step when trying to understand why a decision returns null is to check your decision model and its associated decision logic to make sure they are well defined.

If there is no issue with the decision model and its decision logic, the decision might return null because of null input.

Implicit tests are triggered when you execute a rule. The rule engine checks if the input that is necessary to execute the rule is available. This input can be provided by an input data node or a decision node. If one of the input is null, the rule is not executed and returns null.

In the following example, the decision node **Salutation** has one direct input **Name**.

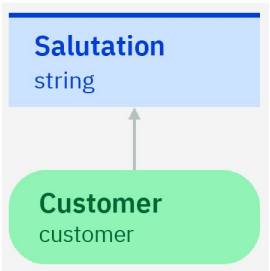


The decision logic of the **Salutation** node is defined as follows:

```
set decision to "Hello " + Name
```

If **Name** is null, the rule is not executed and returns null.

In the following example, the decision node **Salutation** has one direct input **Customer**. **Customer** has two attributes: name and address.



The decision logic of the **Salutation** node is defined as follows:

```
set decision to "Hello " + the name of
Customer
```

If **Customer** is null, the rule is not executed and returns null. If **Customer** is not null and the name of the customer is null, the rule is executed and returns **Hello null**.

Resolving the problem

You can rewrite the decision logic and define a default value to avoid the decision returning null.

If we take the second example described above, we can rewrite the decision logic of the **Salutation** node as follows:

```
if
  the name of Customer is defined
```

```
then
  set decision to "Hello " + the name of Customer ;
```

And add a default value:

```
set decision to "We don't have correct information to
greet you."
```

For more information about defining a default value, see Defining a default value.

A rule is not executed

Symptoms

In Decision Designer, when you run a decision model, a rule is not executed.

Important: This troubleshooting topic does not apply to task models.

Causes

There can be different reasons for a rule not to execute, including the definition of the decision logic or the interaction policy you selected.

Another potential reason might be that an exception occurred because of unknown values in the rule condition. The following types of semantic exceptions can result in an unknown value:

- `ArithmeticException`
- `NumberFormatException`
- `NullPointerException`
- `IndexOutOfBoundsException`

Expressions in conditions that result in exceptions during their evaluation are treated as unknown values and are handled by the rule engine. The rule engine evaluates the overall rule conditions by using the following three-valued logic:

NOT(A)

A	¬A
F	T
U	U
T	F

AND(A,B)

A ∧ B		B		
		F	U	T
A	F	F	F	F
	U	F	U	U
	T	F	U	T

OR(A,B)

A ∨ B		B		
		F	U	T
A	F	F	U	T
	U	U	U	T
	T	T	T	T

F = FALSE, U = UNKNOWN, T = TRUE

If the rule condition evaluates to true, despite unknown values in the condition expressions, the action of the **then** part is applied. If the rule condition evaluates to false, despite unknown values in the condition expressions, the action of the **else** part is applied. If the rule condition evaluates to unknown, because of unknown values in the condition expressions, the rule is not executed.

In the following example, if the status of the borrower is unknown, no rule action is executed.

```
if
   the status of Borrower starts with "duplicate"
then
   reject 'the loan' ;
   add "duplicate detected" to the messages of 'the loan' ;
else
   add "no duplicate found" to the messages of 'the loan' ;
```

In the following example, if the status of the borrower is unknown and the comment of the loan contains the word **duplicate**, then the loan is rejected and a `duplicate detected` message is sent.

```
if
   the status of Borrower starts with "duplicate" or the comment of 'the loan' contains "duplicate"
then
   reject 'the loan' ;
   add "duplicate detected" to the messages of 'the loan' ;
else
   add "no duplicate found" to the messages of 'the loan' ;
```

However, if the status of the borrower is unknown and the comment is unknown, then no rule action is taken. The loan is not rejected and the message `no duplicate found` is not added to the loan.

In the following example, if the status of the borrower is unknown and the comment in the loan contains the word **duplicate**, neither the **then** nor the **else** actions are taken. In other words, the loan is not rejected and the message **no duplicate found** is not added to the message of the loan.

```
if
   the status of Borrower starts with "duplicate" and the comment of 'the loan' contains "duplicate"
then
   reject 'the loan' ;
   add "duplicate detected" to the messages of 'the loan' ;
```

```
else
   add "no duplicate found" to the messages of 'the loan' ;
```

For the loan to be rejected, both expressions must be true.

An exception is thrown

Symptoms

In Decision Designer, when you run a decision model or a task model, an exception is thrown.

The following error message shows an example of a **NullPointerException**:

```
Run error
java.lang.NullPointerException

Exception java.lang.NullPointerException
    in action of rule 'corporate score"
        Bindings: [-200.0, decisions.techpreview_samples.loan_validation.loanvalidationdata.newtype1@d5a71066,200.01
    in node 'Corporate Score'
    in decision model 'LoanValidationDecisionModel-loanValidationDecisionModel'
    in decision operation with execution id "'
```

Other types of exception can be thrown, such as **ArithmeticException**.

Causes

An exception occurred in the rule action. This kind of exception can occur, for instance, when the attribute of an object cannot be retrieved.

If you take the error example shown in the Symptoms section above, it is thrown when you run the corporate score rule from the Loan Approval sample and the credit score cannot be found:

```
set decision to 'Bankruptcy Score' + 'Salary Score' + the credit score of Borrower ;
```

Depending on the type of model that you are running, exceptions can either be thrown in the action part of the rule or the condition part of the rule:

Table 1. Where can an exception be thrown?

Rule part	Decision model	Task model
Rule condition	✗	✓
Rule action	✓	✓

In decision models, exceptions in rule conditions are automatically processed through automatic exception handling. For more information, see A rule is not executed.

Resolving the problem

To avoid running into an exception, you can add a condition to your rule.

For example, if you take the corporate score rule, you can add a condition stating that the value of the decision is only set if the credit score is not null:

```
if the credit score of Borrower is not null
then set decision to 'Bankruptcy Score' + 'Salary Score' + the credit score of Borrower ;
```

An `ads-runtime` pod is evicted because the `EmptyDir` volume for `archive-cache-dir` exceeds the limit

Symptoms

You see a similar message when **ads-runtime** pods are evicted:

```
reason: Evicted
message: 'Usage of EmptyDir volume "archive-cache-dir" exceeds the limit "100Mi". '
```

Causes

The volume of **archive-cache-dir** is too small for your usage.

To optimize the resource usage, the default limit is configured in **EmptyDir**. It is used by all pods. For **archive-cache-dir**, the limit is set to 100 MB in the **small** profile size, and 1 GB in all other cases.

Resolving the problem

Limit the number of decisions in cache.

For more information about configuring the **ads-runtime** cache to limit the number of decisions, see **unit="entries"** in the **cache.config.resources** parameter in Decision runtime parameters.

Restriction: Because you cannot specify a maximum size on disk for decision service archives, you can limit only the count or size in memory. The size of decision service archives cannot be estimated as it depends on your code.

Glossary

This glossary provides terms and definitions for IBM Automation Decision Services

Bold terms in definitions indicate other terms defined in the glossary.

A B C D E F I P R S T V

A

action task node
: In a **ruleflow**, a task node that contains rule action statements. These action statements are executed each time the task node is called.

attribute
: A characteristic of a **composite type**. For example, the brand of a car is one of the car attributes. An attribute has a **data type** and a **verbalization**. It can have one or more values.

B

binary repository
: A system that stores and manages **decision libraries** and **decision service archives** as binary artifacts.

branch
: An independent line of development. It contains **decision services** and their change history for this branch.

build orchestrator
: The build orchestrator uses the **decision build Maven plugin** to compile **decision artifacts** via build plans that are defined by IT, and deploy **decision libraries** to the **binary repository**.

business rule
: A policy, constraint, or required operation that applies a specific set of business conditions or dependencies.

C

collaborator
: User who works on a project with other users.

composite type
: A **data type** that has one or more **attributes**.

D

data model
: A data model is a collection of **data types** and their associated **verbalization**.

data type
: The representation of a **business concept**. A data type has a name and a **verbalization**. The application provides predefined data types. Users can define custom data types, which can be **enumeration types** or **composite types**.

decision artifact
: **Data models**, **decision models**, **task models**, and **predictive models** are decision artifacts.

decision automation
: The shared space where collaborators create **decision services**. A decision automation is synchronized to a **branch** in a **remote repository**.

Decision Designer
: A component of the Automation Decision Services platform where users can develop **decision services**.

decision library
: Compiled versions of **data models**, **decision models**, **task models**, or **predictive models** that are stored in the **binary repository**.

decision logic
: A set of **business rules** and/or **decision tables** that is used to make a decision based on some input data.

decision model
: The implementation of a decision. It contains a **diagram** and a **decision logic** and can be used as a **function** in other decision models and **task models**.

decision node
: In a **decision model**, the graphic representation of an **end decision** or subdecision. This decision has a name and a **data type**. It contains a **decision logic** that uses inputs from other **decision nodes** or **input data nodes** to produce an output that is the result of the decision.

decision operation
: An entry-point to a model. It contains a reference to a model and a **function**, and is used to create **versions**.

decision runtime
: A server that exposes **decision services** through **decision service endpoints**.

decision service
: A set of related decisions that are managed and deployed together. A decision service contains **decision models**, **data models**, and **predictive models**.

decision service archive
: A collection of compiled decision source files. A **decision service archive** is compiled by the **build orchestrator** from decision source files. This archive is then pushed to a **deployment space**, from where it can be loaded by the **decision runtime**. At this point, the archive becomes part of a runtime decision service, which is the web service that can be called from a client application.

decision service endpoint
: The URL where the **decision service** can be accessed by a client application. This endpoint is exposed by the **decision runtime**.

decision table

A decision table represents several **business rules** that share the same logic but apply different conditions and actions. Each row in a decision table forms a complete **business rule**.

deployment space
: A set of decision service archives and their associated metadata. A deployment space can be dedicated to a development phase or a group of users.

diagram
: In a **decision model**, the visual representation of a decision.

E

end decision
: The final decision that is made in a **decision model**.

enumeration type
: A **data type** that contains a list of values.

F

folder
: A **task model** artifact that can be used to organize to group and organize other artifacts, such as **business rules** and **decision tables**.

function
: The list of input and output parameters needed to execute a **decision model**, a **predictive model**, or a specific **ruleflow** within a **task model**. It is used to create **decision operations**, and for model composition.

function node
: In a **decision model**, the graphical representation of a function. It provides values that are computed in another **decision model** within the same **decision service**.

function task node
: In a **task model**, the graphical representation of a function. It provides values that are computed in a **decision model**, a **predictive model**, or another **task model** within the same **decision service**.

I

input data node
: In a **decision model**, the graphic representation of data that is used to make decisions. This data has a name and a **data type**.

interaction policy
: Interaction policies define how **business rules** interact with each other by governing their order of execution. Interaction policies apply at the **decision node** level.

P

predictive model
: A model that implements a prediction based on a machine learning model. A predictive model returns a probability or score that can be used in **decision models**.

predictive node
: In a **decision model**, the graphical representation of a prediction. It provides predictions that are computed in **predictive models**.

R

remote repository
: A repository hosted on a Git server that is synchronized with a **project**. Decision source files are pushed to the remote repository before being built and deployed.

rule language
: A language for expressing rules with natural language terms and syntax. It uses user-defined vocabulary that expresses **business concepts**.

rule task node
: In a **ruleflow**, a task node that refers to rule artifacts and orders them.

ruleflow
: A method of controlling and ordering the execution of rule artifacts. A ruleflow is defined in terms of task nodes.

run
: The process of checking that a **decision model** works the way that is expected. Users define **test data** and run the **decision model** with it to see the decision results.

S

share
: Committing changes to a project. When a user shares changes, these changes become available to other users who work in the same project.

subflow task node
: in a **ruleflow**, a task node that refers another ruleflow. A subflow tasks node can reference a ruleflow in the current task model, or in another task model in the same **decision service**.

T

task model
: The implementation of a decision. It contains rule artifacts and a **ruleflow** to organize their execution.

test data
: The data that users enter to validate that the decision logic works as expected.

V

variable set
: A group of variables that can be used across a **task model**.

verbalization
: (v.) The process of associating terms and phrases to **data types** and **attributes** to use them in a decision.

  (n.) The phrase defined by the user to refer to a **data type** or an **attribute**. Each verbalization is defined for a specific locale, so a **data type** or an **attribute** can have more than one verbalization. E.g: the data type birthDate can be verbalized as "birth date" and "date de naissance".

version
: A pointer to a specific point in the change history. A version has a name and a description. A version is reflected in the **remote repository** as a Git tag.