Salesforce Industries provides two rules frameworks to accomplish your business objectives: **Context Rules** and **Advanced Rules**.

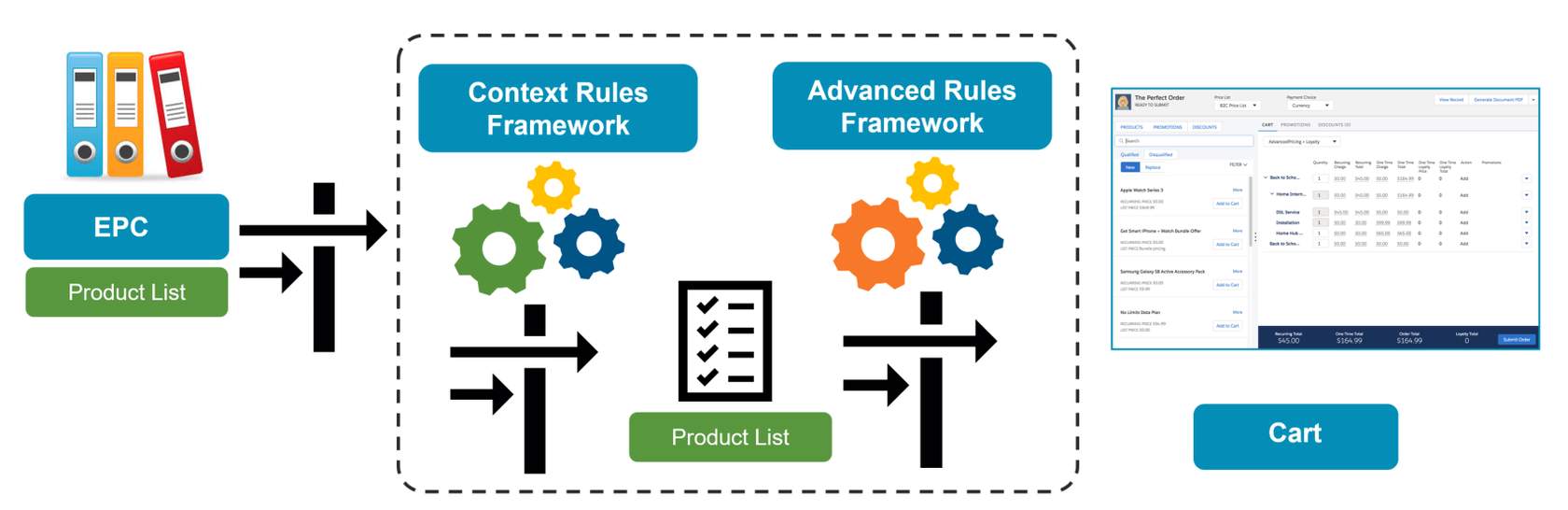
**Dual Frameworks, Working Together**

The Context Rules and Advanced Rules frameworks work in tandem to run business rules in your Industries CPQ environment. These business rules tailor the Shared Catalog (EPC) data into product and service offers that are relevant and available to customers.

**Context Rules** qualify products, promotions, price lists, price list entries and pricing adjustments in the Cart. This framework can also be used for Salesforce Industries API Caching, which is used for digital commerce.

**Advanced Rules** is Salesforce Industries' original rules framework, and it is used primarily to create rules for product compatibility or configuration

Both rules frameworks operate in tandem to provide comprehensive rules to govern all Industries CPQ interactions.

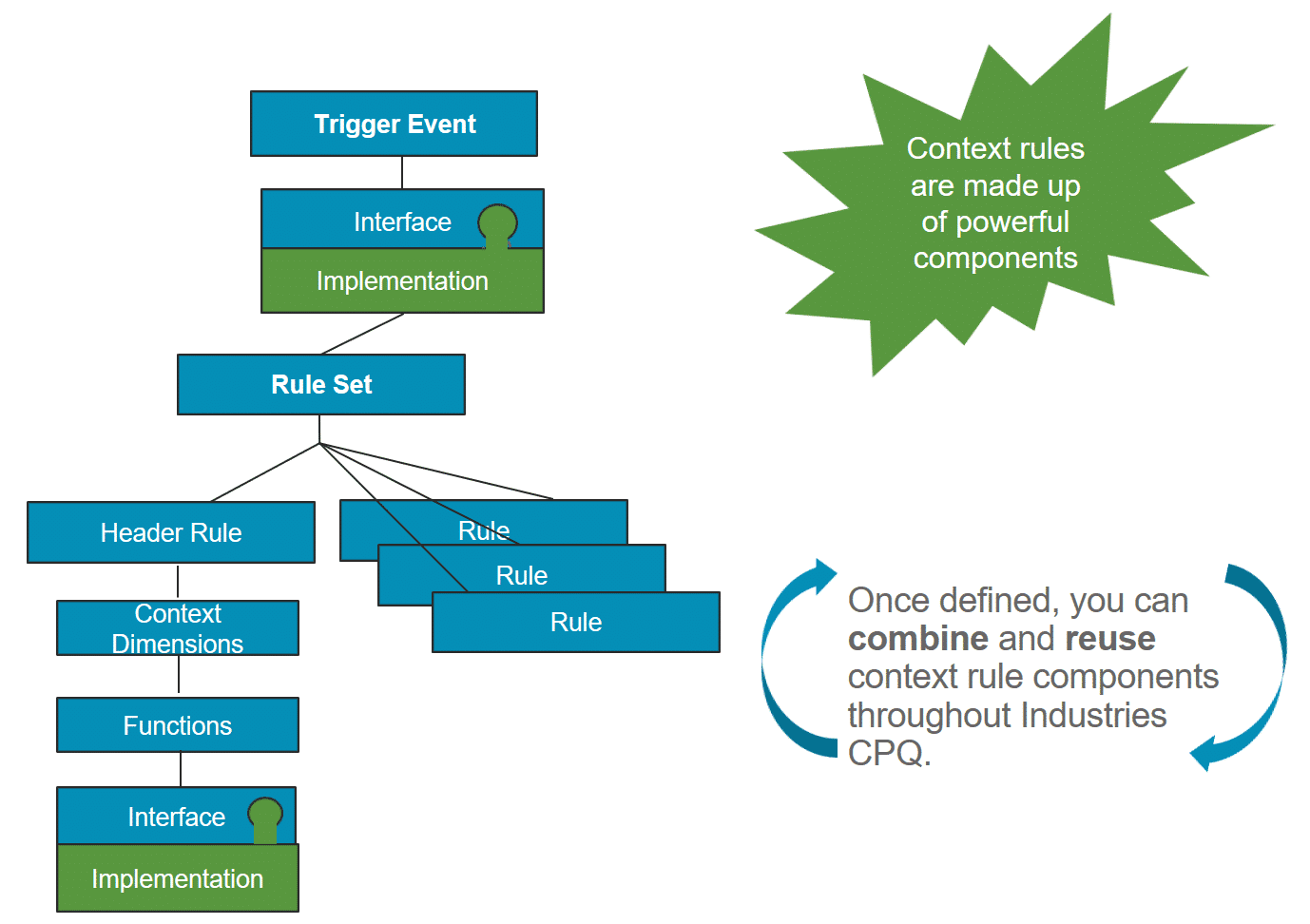


The diagram shows how the two rules frameworks work together in the Cart. The process begins by gathering all active products in EPC, moves through the Context Rules Framework to filter the product list, then to the Advanced Rules Framework to further refine the product list, and finally presents available and eligible products and promotions in the Cart.

**The Context Rules Framework**

The Context Rules Framework allows you to build context rules to apply qualification and penalty rules. The framework includes the following elements, as shown below:

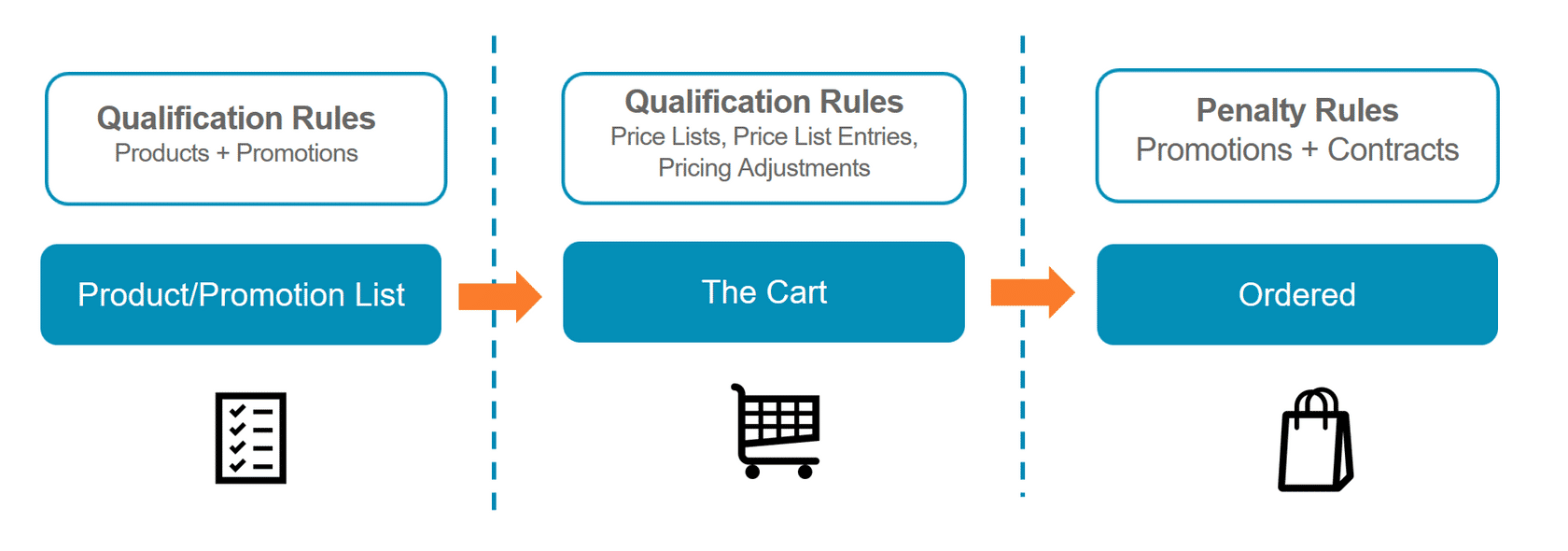
* Trigger events (such as adding a product to the Cart) that run an Industries CPQ interface, which then calls the active interface implementation
* Rule sets that run one or more context rules
* At least one context rule, which includes a context dimension, context scope and context mappings. We cover these components in more detail in the next few lessons.



**Context Rule Types**

In the Vlocity Product Console, you can create the following types of context rules:

* **Qualification:**This rule type determines eligible products and promotions for a customer before you add them to the Cart, and determine eligible price lists, price list entries, and pricing adjustments for products in the Cart.
* **Penalty:** This rule type determines penalties for a customer who cancels contracts or promotions they've already ordered.

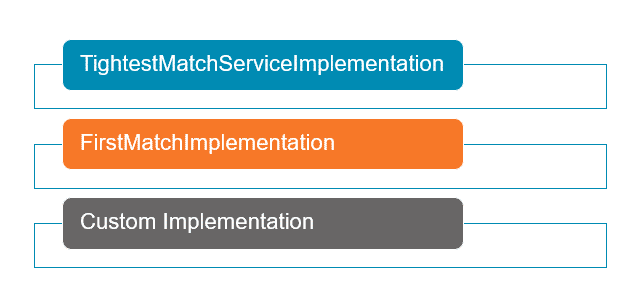


**Qualification Rules for Price List Entries: TightestMatchInterface**

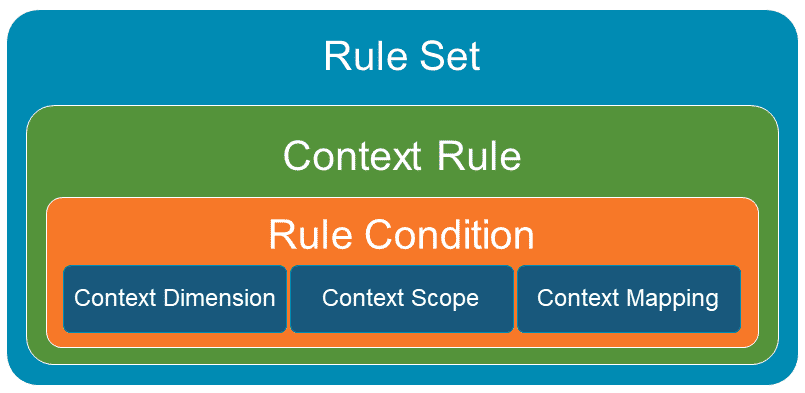
The **TightestMatchInterface** is called when the application must price a product in the cart.

If the **TightestMatchServiceImplementation** is active and default, this interface will use weights to determine which price list entry to select.

If **FirstMatchImplementation** is active and default, this interface ignores those weights and selects the first match that it finds.



**What's a Rule Set?**



A rule set is a collection of one or more context rules applied to a product, promotion, price list, or price list entries. These rules are evaluated as a whole when performing a check against a product or promotion.

* To express complex conditions, you must group rules into a rule set.
* You apply rule sets to products or promotions, rather than individual context rules.
* Rule sets are stored as Vlocity Rule objects.

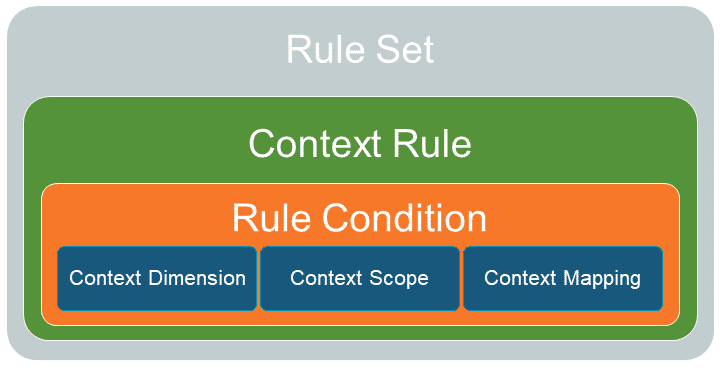
**Expression Modes**

Rule evaluation uses an expression mode (e.g. And, Or, Custom) to compile conditions into a logical expression. Rule evaluation determines the qualification or disqualification of products or promotions.

A rule set expression mode is based on one of the following conditions.

* **And:**Requires that all of the rules in the rule set pass for the rule set to evaluate to Pass.  This is the default and is used only with qualification rules.
* **Or:** Requires that at least one of the rules in the rule set passes for the rule set to evaluate to Pass.
* **Custom:** Requires a custom expression to evaluate the rule set. For example, to set the conditions as ((RuleA AND RuleB) OR (RuleC AND RuleD)).
* **If Else If:** Tests the first rule in the rule set, and tests the subsequent rule only if the first rule does not pass.
* **If:** Tests the first rule in the rule set.

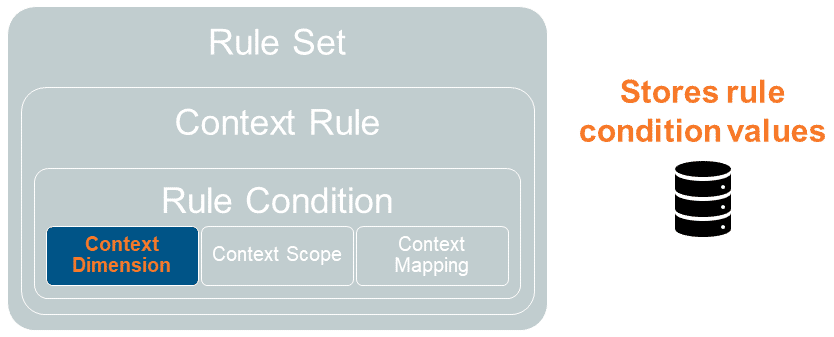
**What Does a Context Rule Contain?**



A context rule contains the information needed to determine when a rule set should run in the Cart.

* Rule condition
* Context dimension
* Context scope
* Context mapping

**What's a Context Dimension?**

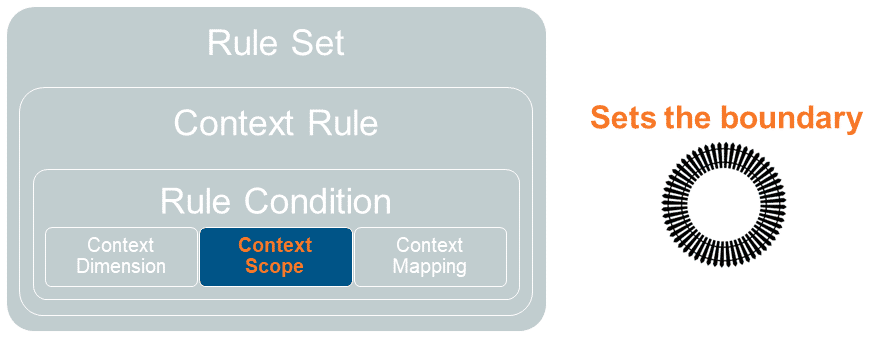


A context dimension is a variable that describes the possible values to use in a rule condition.

You can reuse context dimensions across multiple rule conditions.

The context rules service engine compares the context dimension against data; for example, from an sObject, a function, or a static value defined in the context mapping.

**What's a Context Scope?**



A context scope describes the relational path from a root sObject, such as an Order, to related sObjects, setting the boundaries for the rule function.

Context scopes represent an sObject from which the system retrieves data used by the evaluation logic of rules to determine the output.

**Supported Context Scopes**

Salesforce Industries supports the following root context scopes:

* Order
* Opportunity
* Quote
* Asset
* Any

Salesforce Industries also supports Account and Contract context scopes, which use the root context scopes to determine the entity path to the required data element.

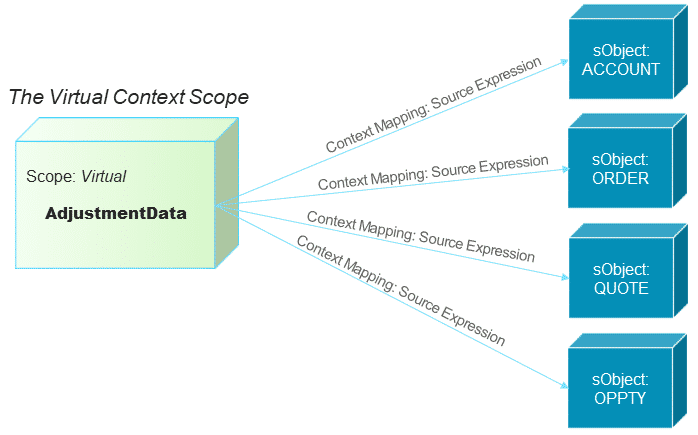
**Note:**The Asset context scope can only be used for rules invoked during repricing.

**The Any Context Scope**

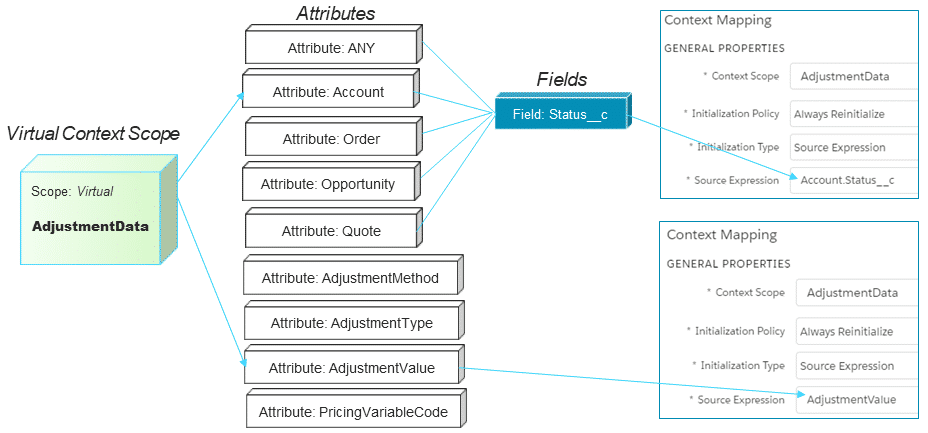
The Any context scope:

* Is a wildcard context scope that represents all root context scopes.
* Allows you to create Account and Contract scopes that are agnostic with regard to the root context scope. For example, instead of creating separate Order.Account and Quote.Account scopes, you can create a single Any.Account scope that locates the associated account for all root entities, including orders and quotes.
* Provides a flexible and efficient way to define scopes.

**Virtual Context Scopes**



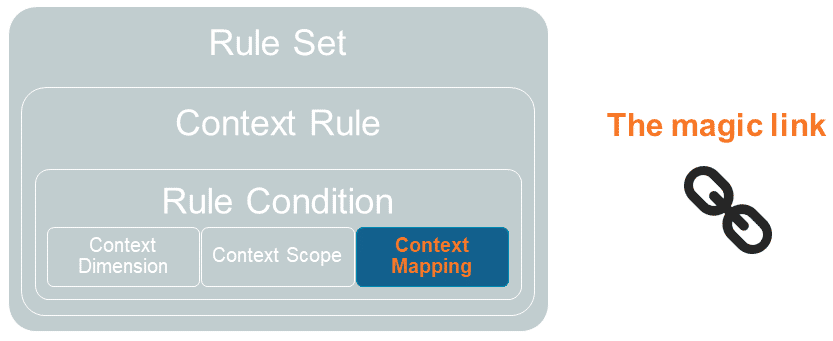
Virtual context scopes enable you to create context mappings to virtual objects. Virtual context scopes and virtual objects enable you to evaluate data entered by the user at run time that is stored only in memory, such as pricing adjustment data.



When you create context rules for pricing adjustments, you can create context mappings to virtual object data using the source expression property.

* Map to attributes on the virtual object.
* ANY, Account, Order, Quote and Opportunity and their fields are the only sObjects currently supported. (But you can’t access data from related sObject like standard context mappings.)
* Use the attribute names when writing source expressions.
* For the ANY, Account, Order, Quote and Opportunity attributes, you can refer to fields on those sObjects using the format of [attribute name].[API field name].

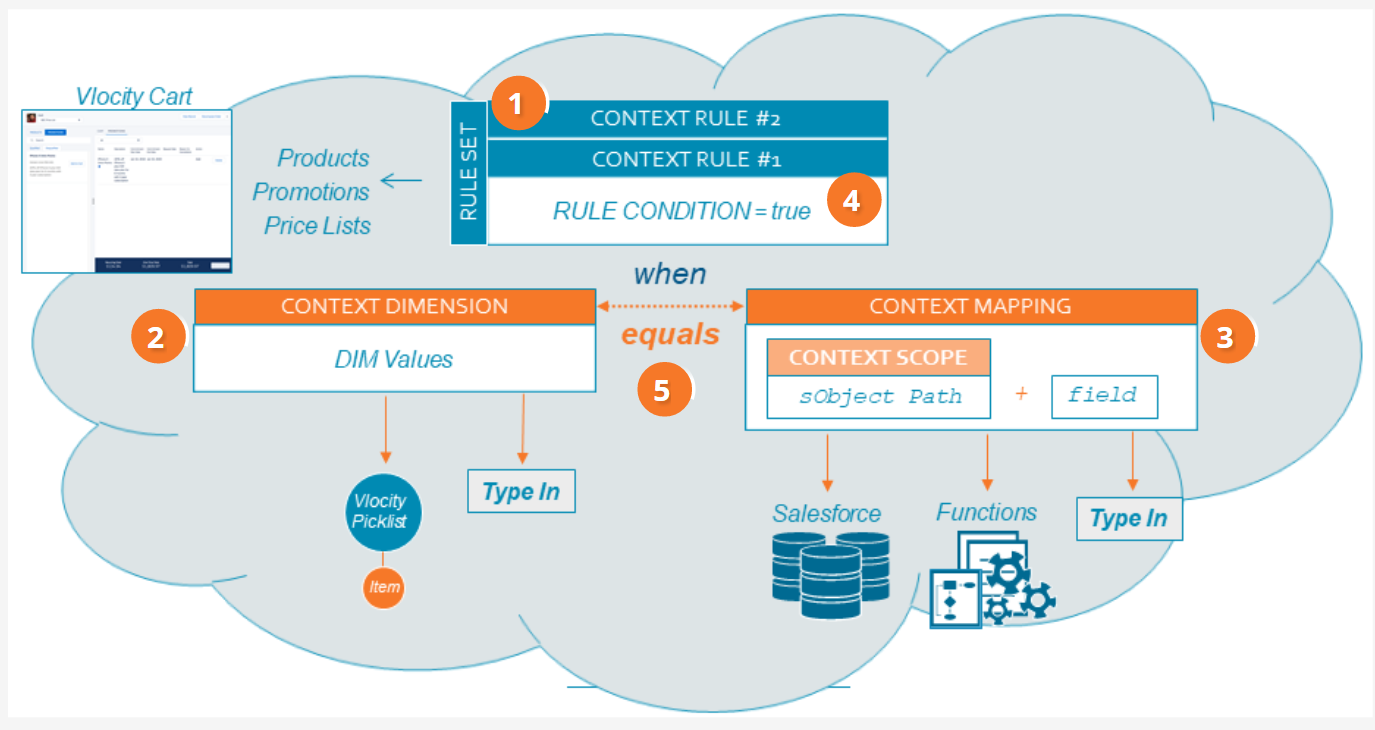
**What's a Context Mapping? (The Magic Link)**



A context mapping uses the context scope to identify the fields on sObjects or computed data to match against context dimension variables for evaluation by the Context rules service engine.

It allows you to map context scopes, such as sObjects, with context dimensions, such as fields or calculated formulas. This way, you create a path to the data for the context rules service engine, which forms a magic link between the data and the rules engine.

Context mappings are held in the org cache.



1. Rule set – a rule set contains one or more context rules. You can apply rule set to product promotion and price list
2. Context dimension – the context dimension includes possible values, either from Vlocity Picklist or Type In.
3. Context mappings – the context mapping includes context scope to pull data from salesforce, a function, or type in
4. When the context dimension and context mapping match – the rule condition evaluates to true, and the context rules engine runs the context rules in the rule set. Each context rule, Context Rule #1 and Context Rule #2, evaluates a condition
5. Resulting Logical Expression – the different places work together to form the following logical expression: Rule Condition = true WHEN {{Context Dimension}} == {{Context Mapping{Context Scope.Source Expression}}