API Gateway Analytics

API Gateway Analytics

April 2016

Copyright

Trademarks

Akana, Inc.

Akana, Inc.
12100 Wilshire Blvd, Suite 1800
Los Angeles, CA 90025
(866) SOA-9876
www.akana.com
info@kana.com

Disclaimer

The information provided in this document is provided "AS IS" WITHOUT ANY WARRANTIES OF ANY KIND INCLUDING WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF INTELLECTUAL PROPERTY. Akana may make changes to this document at any time without notice. All comparisons, functionalities and measures as related to similar products and services offered by other vendors are based on Akan's internal assessment and/or publicly available information of Akana and other vendor product features, unless otherwise specifically stated. Reliance by you on these assessments / comparative assessments is to be made solely on your own discretion and at your own risk. The content of this document may be out of date, and Akana makes no commitment to update this content. This document may refer to products, programs or services that are not available in your country. Consult your local Akana business contact for information regarding the products, programs and services that may be available to you. Applicable law may not allow the exclusion of implied warranties, so the above exclusion may not apply to you.

Contents

Chapter 1 | Overview

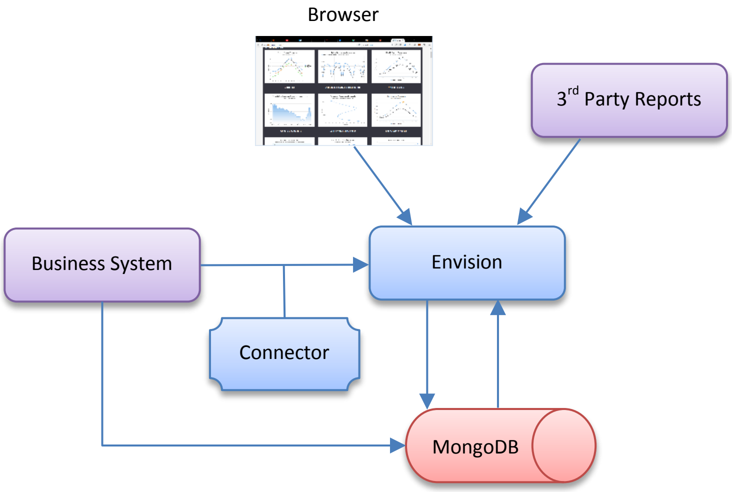
The API Gateway can collect a wide range of data that can be analyzed with the help of the Envision analytics platform. This collaboration is enabled through a set of plugins for each products that provide cross-over functionality that enhance each product's experience. The list of capabilities provided by the plugins are as follows.

- Configuration of an operational metrics data set, charts, and dashboards in Envision.
- Population of the operational data set in Envision from messages processed by the API Gateway.
- API Gateway controlled security (authorization) for operational data in Envision.
- Population of user created data sets in Envision from messages processed by the API Gateway.
- SLA enforcement for user created data set metrics.

There are three plugins that together provide these capabilities, one for the Network Director, the Policy Manager Console, and Envision respectively. This document will describe the plugins including their architecture, installation, and configuration.

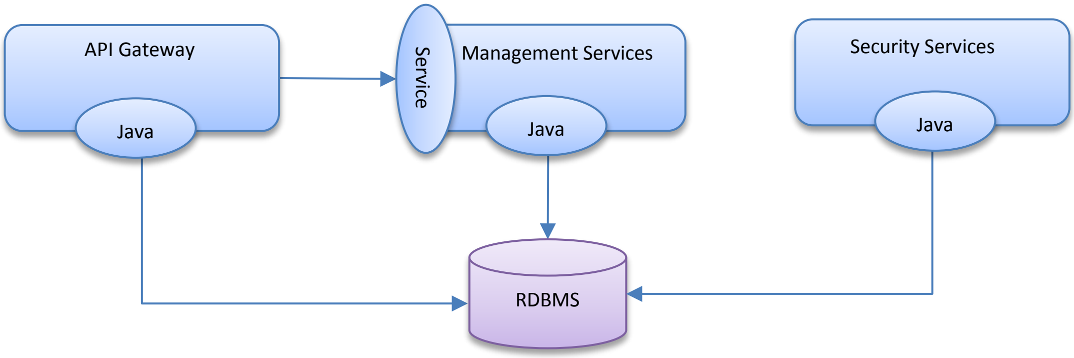
Chapter 2 | Plugin Architecture

The Envision analytics platform analyzes data received from data feeds generated by analytics connectors. An analytics connector will extract data from a business system and feed the data to Envision using an Envision API. The following is a picture of how Envision interacts with users and other systems. Note the use of a Connector to collect information from a business system and feed it to the Envision platform.

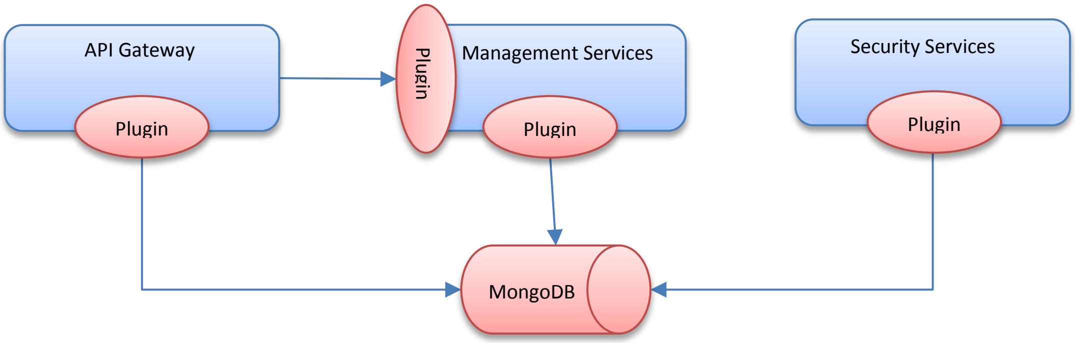


The API Gateway product provides an analytics connector that will feed metrics collected from API's and services it manages. In the picture above the API Gateway is a Business System feeding Envision.

The API Gateway provides metrics collection and aggregation capabilities out-of-the-box. It only does so for a limited set of metrics with predefined dimensions. There are two ways these metrics are collected and persisted in the API Gateway. The first is via a metrics collection service included in the Akana Management Services feature. This provides a remote API that the Network Director can deliver metrics through. The second is via a metrics reporting Java API which all Policy Manager provided services and API's used to report their metrics and the Network Director can use as an alternative to using the remote API.

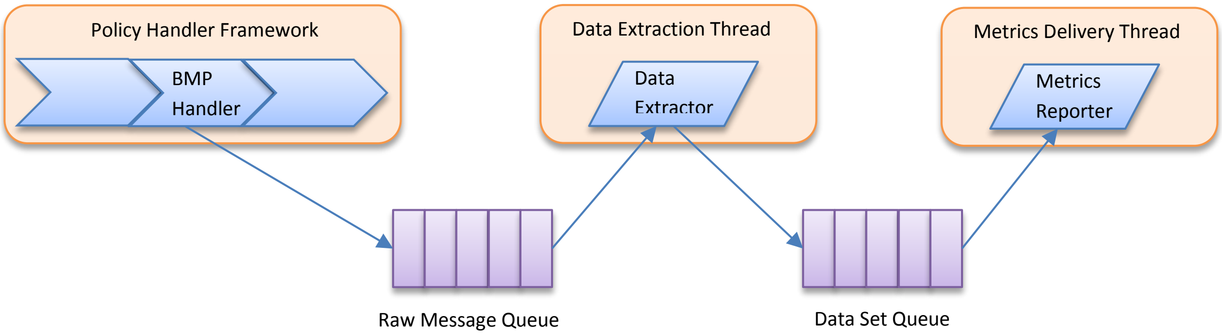


When the Envision plugin is installed with the Policy Manager features the Java API used for reporting metrics is overridden with one that writes those metrics to the MongoDB NoSQL database Envision uses. This Java API override also replaces the remote API's implementation so any metrics delivered remotely from the Network Director are also written to the NoSQL database. When the analytics plugin is installed with the Network Director the Network Director can then also write directly to the NoSQL database as an option.



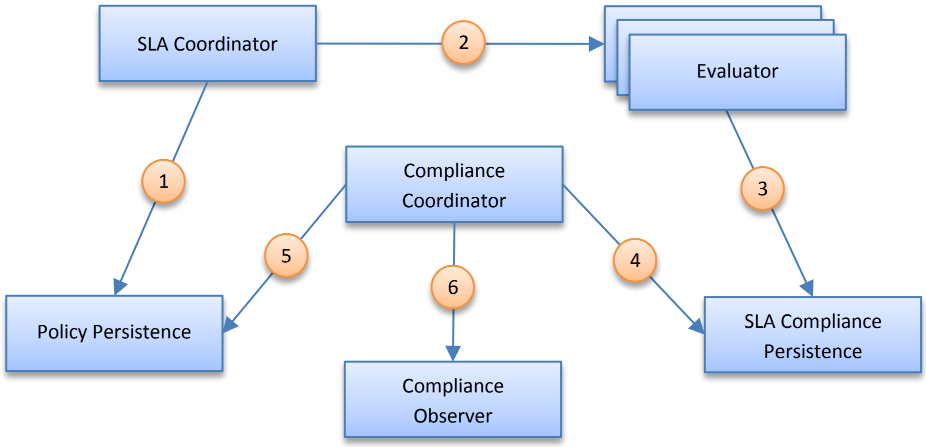
Envision supports analyzing data from any source, any domain, only relying on user defined metadata describing a set of data (data set) and how the data should be processed. The Envision plugin introduces a new policy, the business metrics policy, that will instruct the Network Director how to extract data from managed API/service messages that cant hen be fed into Envision to fill user created data sets.

The business metrics policy handler supports a variety of data extraction mechanisms including XPath, JsonPath, RegEx, and script based expressions. Based on the policy configuration a multitude of these mechanisms may have to be used to extract data from each message processed by the Network Director. To avoid adding latency to the delivery of each message the policy handler will extract data from a message in a separate thread.

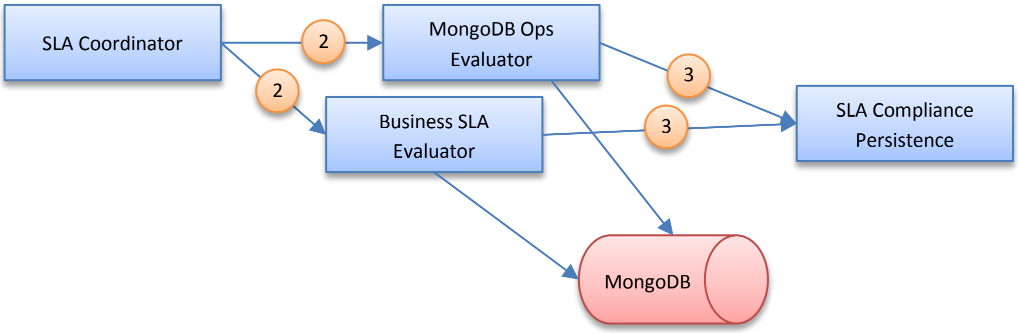


Based on the needs specified in the Business Metrics Policy the Business Metrics Policy handler will copy portions of the request/response/fault message (headers, body) to the Raw Message Queue along with instructions on how data should be extracted. The Data Extractor executes in the Data Extraction Thread pulling the records from the Raw Message Queue, extracting the data, and placing the extracted data in the Data Set Queue. The Metrics Reporter executes in the Metrics Delivery Thread calling the API for persisting the extracted data.

The API Gateway supports using MongoDB as a metrics store using the MongoDB plugin. The Envision plugins will automatically install the MongoDB plugin so all metrics, including Business Metrics Policy collected metrics, will be stored in the MongoDB NoSQL database. That means all administration (viewing, exporting, archiving) of metrics will be performed against the MongoDB database. This also mean that all SLA calculations will be performed using the MongoDB database. This is achieved by replacing the Evaluator module of the SLA Engine. The SLA Engine modules are illustrated in the diagram below.

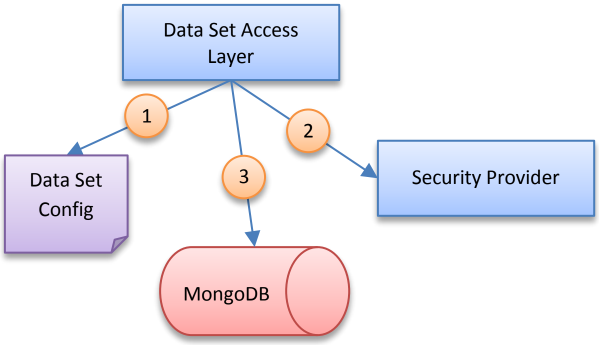


When using the MongoDB database the Evaluator module, which evaluates whether SLA's are violated, is swapped out with a module that not only can query for information from MongoDB but will actually use the power of MongoDB to perform calculations using the MongoDB processor itself. This frees up computing resources for the Policy Manager process. In addition, an evaluator for SLA's based on business metrics is introduced. This evaluator will also use MongoDB. The following diagram illustrates the addition of the two MongoDB based evaluators.



Not all the plugins used in the integration of the API Gateway with Envision change the behavior of the API Gateway. An API Gateway plugin is also installed on the Envision analytics platform which provides integrated security (authorization) around the access to records in data sets. By default Envision secures data in data sets based on the owner's shared settings on the data set. Optionally data sets created by different business systems such as the API Gateway can be protected through the use of a security plugin.

Each data set in Envision can optionally reference a security provider. That provider is consulted when the data queries are performed in Envision.



The Data Set Access Layer in Envision first consults the Data Set configuration to identify a Security Provider (if any). It then consults the Security Provider providing the provider contextual information about the data access request. The security provider then providers additional filter criteria that when added to the query of MongoDB will filter out records that should not be accessible. By translating security permissions into a NoSQL query security can be enforced in a highly efficient way by the NoSQL server instead of filtering out records in memory.

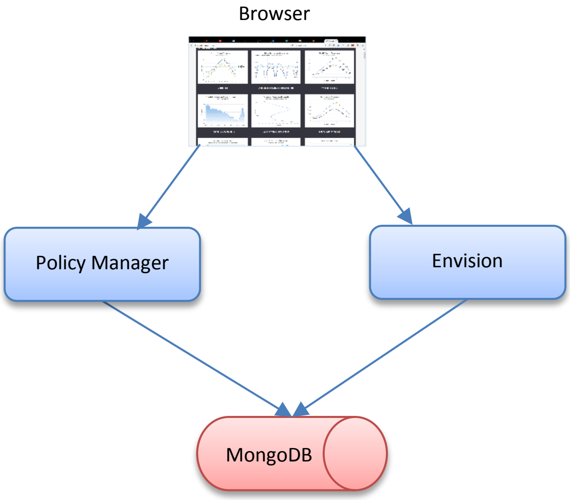
The API Gateway plugin for Envision will create a Security Provider for the operational metrics data set it installs in Envision. The filter created is based on the user's access to the API's/services the operational metrics are collected for. A user who does not have access to an API in the Policy Manager console for example will not have access to data in the operational metrics data set for that API regardless of how a chart is configured.

Chapter 3 | Deployment Architecture

The API Gateway and Envision are two different products, but they both are deployed using the Akana Platform. Therefore they can be deployed separately or together. The following section will describe a few options for deploying the two products together. In all deployment options the MongoDB NoSQL database is depicted as a single object however it can be clustered, sharded, and scaled in multiple ways which is out of the scope of this document.

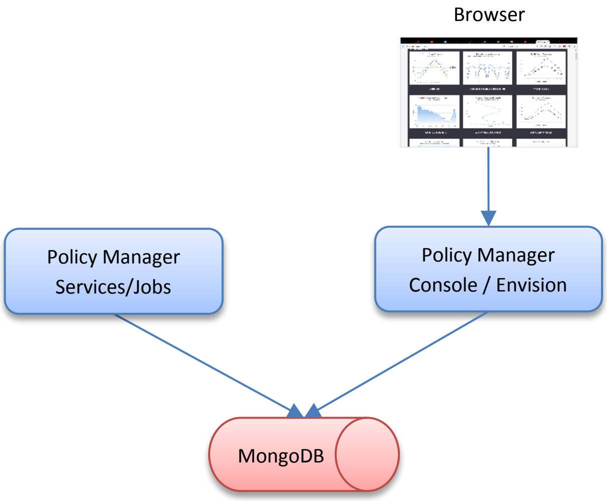
Separate Containers

In this deployment Policy Manager and Envision are installed on two completely separate containers. They each will have their own HTTP host and/or port when accessing the two consoles from a browser. Each container will access the same MongoDB NoSQL database. Each product can be scaled independently.



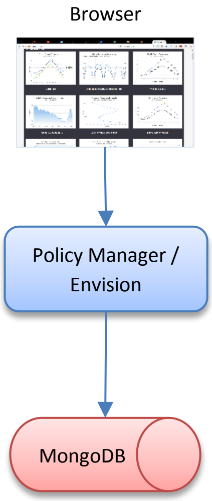
Shared Consoles

In this deployment the Policy Manager services and jobs will be installed on separate containers from the Policy Manager Console and Envision. Since both the Policy Manager Console and Envision features are driven primarily by end-user interaction they may share more common scaling demands then those of the Policy Manager jobs and services which are driven by a scheduler and Network Director interaction. Scaling can then be accommodated differently for web traffic than back-end traffic. In this deployment both the Policy Manager Console and Envision can share the same host and port. Both containers would still access the same MongoDB NoSQL database.


Browser
Policy Manager Services/Jobs
Policy Manager Console / Envision
MongoDB

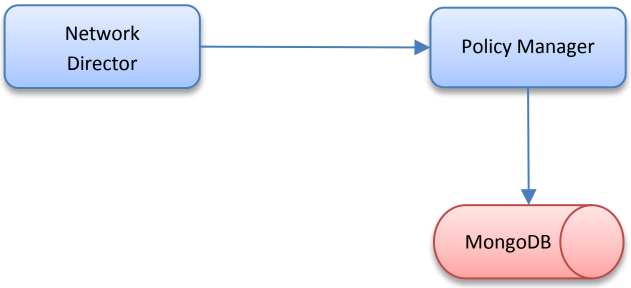
All-in-One

In this deployment the Policy Manager Console, Services, Jobs, and Envision are all installed on the same container. This deployment offers the least flexibility in scaling options but is the simplest deployment for small scale environments.



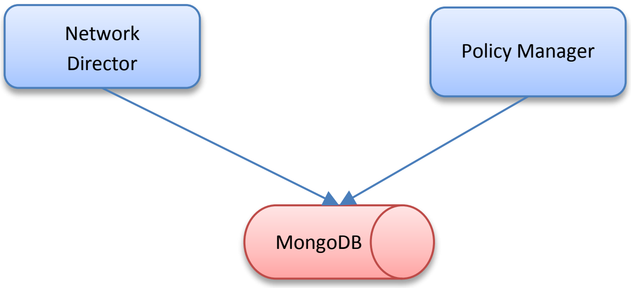
Remote Metrics Collection

In this deployment the Network Director will report the metrics it collects to the Policy Manager through a remote API. The Policy Manager will access the MongoDB NoSQL database. The Network Director will not require a route to the MongoDB database.



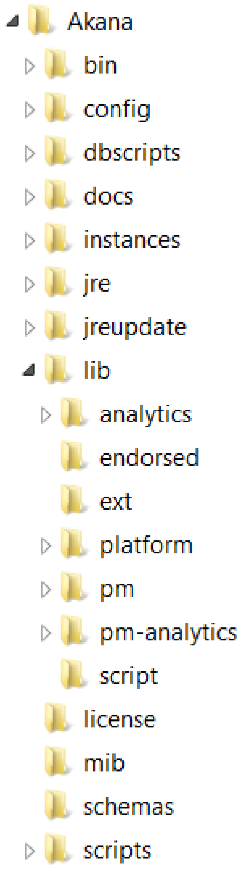
Direct Metrics Collection

In this deployment the Network Director reports the metrics it collects directly to the same MongoDB NoSQL database that the Policy Manager (and Envision) accesses.



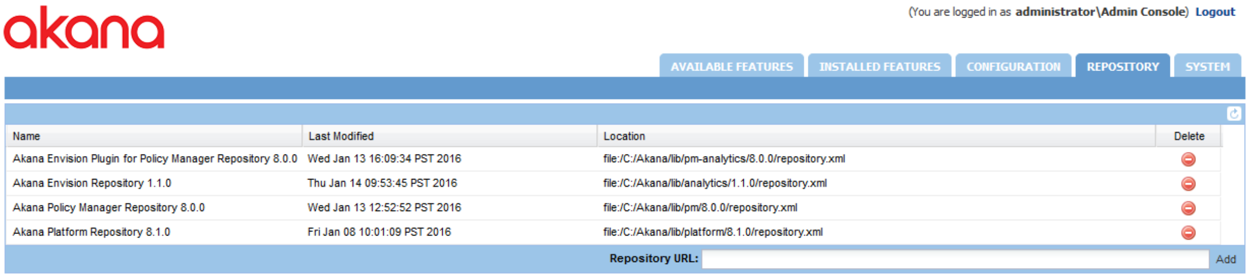
Chapter 4 | Installation

A set of plugins are provided that provide the analytics connector and security plugin capabilities. These plugins act as a bridge between the API Gateway (Policy Manager) and Envision products. In order for them to function properly the Policy Manager, Envision, and PM Analytics Plugin repositories need to be installed in the same root Akana directory. To do this you must download and unzip the Envision, Policy Manager, and PM Analytics Plugin zip files in the same location. The result directory structure should resemble the following:



- Akana
  - bin
  - config
  - dbscripts
  - docs
  - instances
  - jre
  - jreupdate
  - lib
    - analytics
    - endorsed
    - ext
    - platform
    - pm
    - pm-analytics
    - script
  - license
  - mib
  - schemas
  - scripts

Note the “pm”, “analytics”, and “pm-analytics” directories in the “lib” directory. When viewing the repositories in the admin console of an Akana container (process) you should see both repositories as seen below.



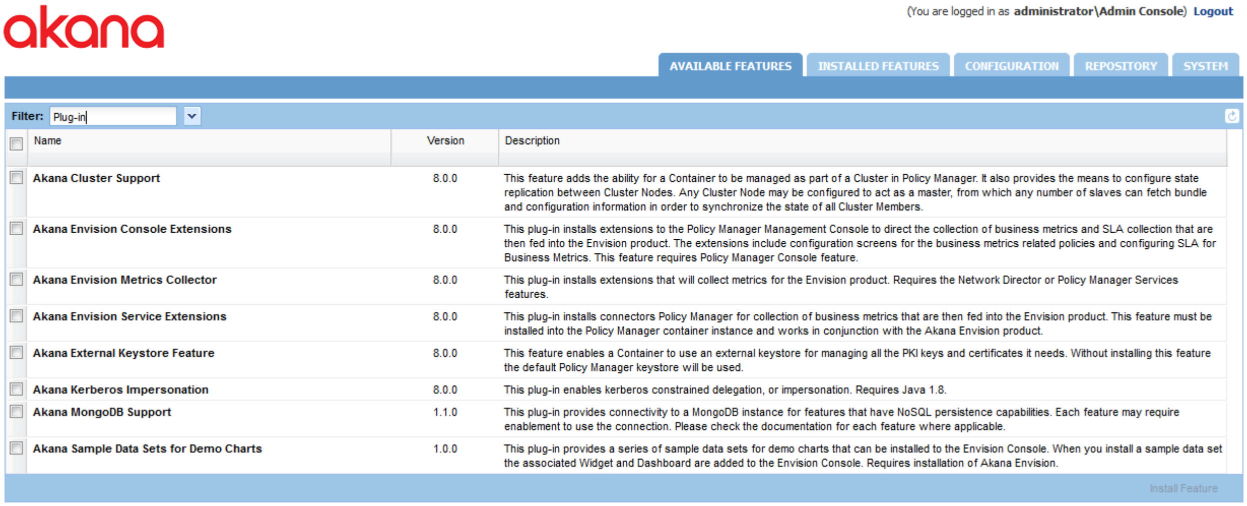
Name	Last Modified	Location	Delete
Akana Envision Plugin for Policy Manager Repository 8.0.0	Wed Jan 13 16:09:34 PST 2016	file:/C:/Akana/lib/pm-analytics/8.0.0/repository.xml	⊖
Akana Envision Repository 1.1.0	Thu Jan 14 09:53:45 PST 2016	file:/C:/Akana/lib/analytics/1.1.0/repository.xml	⊖
Akana Policy Manager Repository 8.0.0	Wed Jan 13 12:52:52 PST 2016	file:/C:/Akana/lib/pm/8.0.0/repository.xml	⊖
Akana Platform Repository 8.1.0	Fri Jan 08 10:01:09 PST 2016	file:/C:/Akana/lib/platform/8.1.0/repository.xml	⊖

Repository URL:

The plugins needed for product integration are as follows:

- Akana Envision Metrics Collector – This plugin is installed on a container with the Network Director feature. It adds the handler for the Business Metrics Policy and the supporting extraction and reporting capabilities.
- Akana Envision Policy Manager Console Extensions – This plugin is installed on a container with the Akana Policy Manager Console feature. It adds the user interface for the Business Metrics Policy and the Business QoS (SLA) Policy.
- Akana Envision Policy Manager Services Extensions – This plugin is installed on a container with the Akana Policy Manager Services (Management and Security) and/or Policy Manager Jobs features. It adds the metrics API replacement that supports business metrics and storage in the MongoDB NoSQL database as well as the SLA Engine extension for Business Qos Policies.
- Akana MongoDB Support – This plugin is installed automatically when installing the Akana Envision Policy Manager Console Extensions or the Akana Envision Policy Manager Services Extensions plugins. It is not installed automatically by the Akana Envision Metrics Collector plugin which uses the remote metrics reporting API by default. It can however be installed on a container with the Network Director feature to enable direct storage to a MongoDB NoSQL database instead of using the remote API.
- Akana Policy Manager Security Provider for Envision – This plugin is installed on a container with the Akana Policy Manager Services and Envision feature. It installs the security plugin for protecting API Gateway collected operational metrics.

They plugins should all be available in the Available Plug-Ins page.



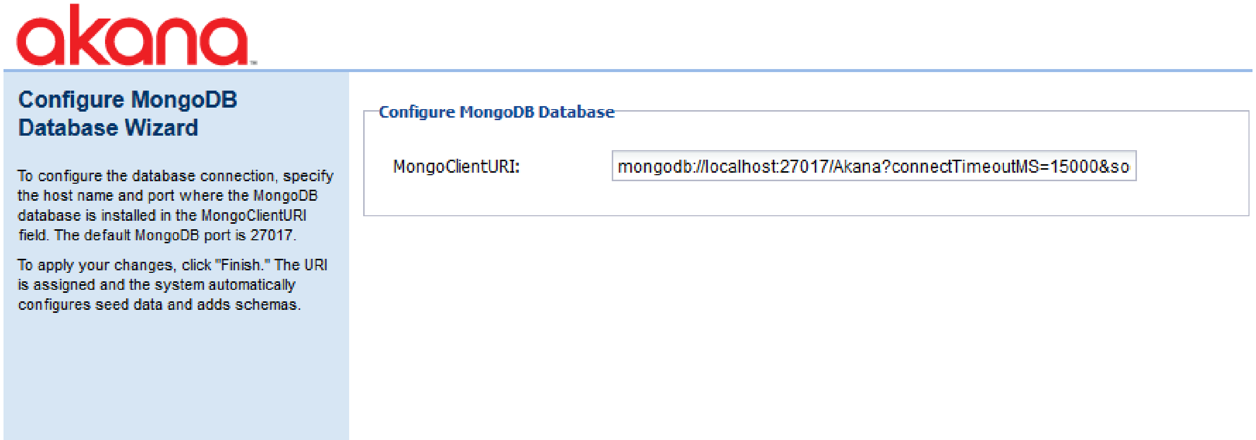
Filter: Plug-in

□	Name	Version	Description
□	**Akana Cluster Support**	8.0.0	This feature adds the ability for a Container to be managed as part of a Cluster in Policy Manager. It also provides the means to configure state replication between Cluster Nodes. Any Cluster Node may be configured to act as a master, from which any number of slaves can fetch bundle and configuration information in order to synchronize the state of all Cluster Members.
□	**Akana Envision Console Extensions**	8.0.0	This plug-in installs extensions to the Policy Manager Management Console to direct the collection of business metrics and SLA collection that are then fed into the Envision product. The extensions include configuration screens for the business metrics related policies and configuring SLA for Business Metrics. This feature requires Policy Manager Console feature.
□	**Akana Envision Metrics Collector**	8.0.0	This plug-in installs extensions that will collect metrics for the Envision product. Requires the Network Director or Policy Manager Services features.
□	**Akana Envision Service Extensions**	8.0.0	This plug-in installs connectors Policy Manager for collection of business metrics that are then fed into the Envision product. This feature must be installed into the Policy Manager container instance and works in conjunction with the Akana Envision product.
□	**Akana External Keystore Feature**	8.0.0	This feature enables a Container to use an external keystore for managing all the PKI keys and certificates it needs. Without installing this feature the default Policy Manager keystore will be used.
□	**Akana Kerberos Impersonation**	8.0.0	This plug-in enables kerberos constrained delegation, or impersonation. Requires Java 1.8.
□	**Akana MongoDB Support**	1.1.0	This plug-in provides connectivity to a MongoDB instance for features that have NoSQL persistence capabilities. Each feature may require enablement to use the connection. Please check the documentation for each feature where applicable.
□	**Akana Sample Data Sets for Demo Charts**	1.0.0	This plug-in provides a series of sample data sets for demo charts that can be installed to the Envision Console. When you install a sample data set the associated Widget and Dashboard are added to the Envision Console. Requires installation of Akana Envision.

Determining which container to install a plugin on depends on which deployment architecture you are following (see Chapter 3). The key is that each plugin supports certain features as noted above and that the plugins need to be installed on the same container as the supported feature. The configuration of each of the plugins is described below.

Akana MongoDB Support

This plugin has only one configuration setting, the URI of the MongoDB database. This URI includes the host and port of the database server as well as the database name in the path. By default the name of the database installed for all Akana products is “SOA”.



Configure MongoDB Database Wizard

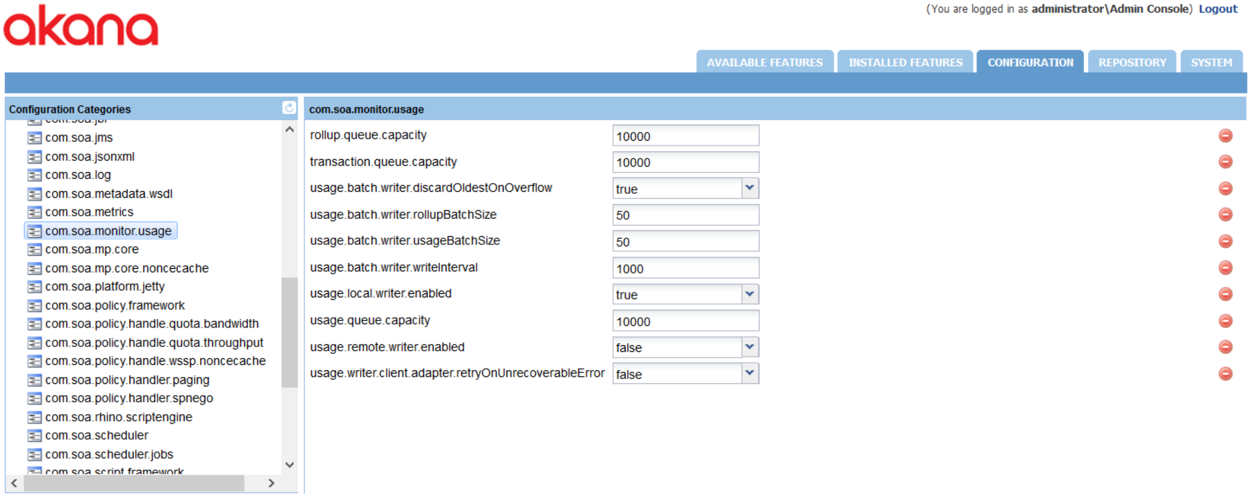
To configure the database connection, specify the host name and port where the MongoDB database is installed in the MongoClientURI field. The default MongoDB port is 27017.

To apply your changes, click "Finish." The URI is assigned and the system automatically configures seed data and adds schemas.

Configure MongoDB Database

MongoClientURI: mongodb://localhost:27017/Akana?connectTimeoutMS=15000&so

If the plugin is installed with the Network Director feature writing directly to the MongoDB database will require the usage.local.writer.enabled configuration option in the com.soa.monitor.usage configuration category to be set to “true”.



com.soa.jms
com.soa.jsonxml
com.soa.log
com.soa.metadata.wsdl
com.soa.metrics
com.soa.monitor.usage
com.soa.mp.core
com.soa.mp.core.noncecache
com.soa.platform.jetty
com.soa.policy.framework
com.soa.policy.handle.quota.bandwidth
com.soa.policy.handle.quota.throughput
com.soa.policy.handle.wssp.noncecache
com.soa.policy.handler.paging
com.soa.policy.handler.spnego
com.soa.rhino.scriptengine
com.soa.scheduler
com.soa.scheduler.jobs
com.soa.script.framework

com.soa.monitor.usage	
rollup.queue.capacity	10000
transaction.queue.capacity	10000
usage.batch.writer.discardOldestOnOverflow	true
usage.batch.writer.rollupBatchSize	50
usage.batch.writer.usageBatchSize	50
usage.batch.writer.writeInterval	1000
usage.local.writer.enabled	true
usage.queue.capacity	10000
usage.remote.writer.enabled	false
usage.writer.client.adapter.retryOnUnrecoverableError	false

Akana Envision Metrics Collector

No configuration required.

Akana Envision Policy Manager Console Extensions

This plugin has only one configuration setting which is the same as the MongoDB Support plugin (see above).

Akana Envision Policy Manager Services Extensions

This plugin has only one configuration setting which is the same as the MongoDB Support plugin (see above).

Akana Policy Manager Security Provider for Envision

This plugin has no configuration. It is only applicable when both the Policy Manager Management Services and Envision features are installed in the same container.

Chapter 5 | Usage

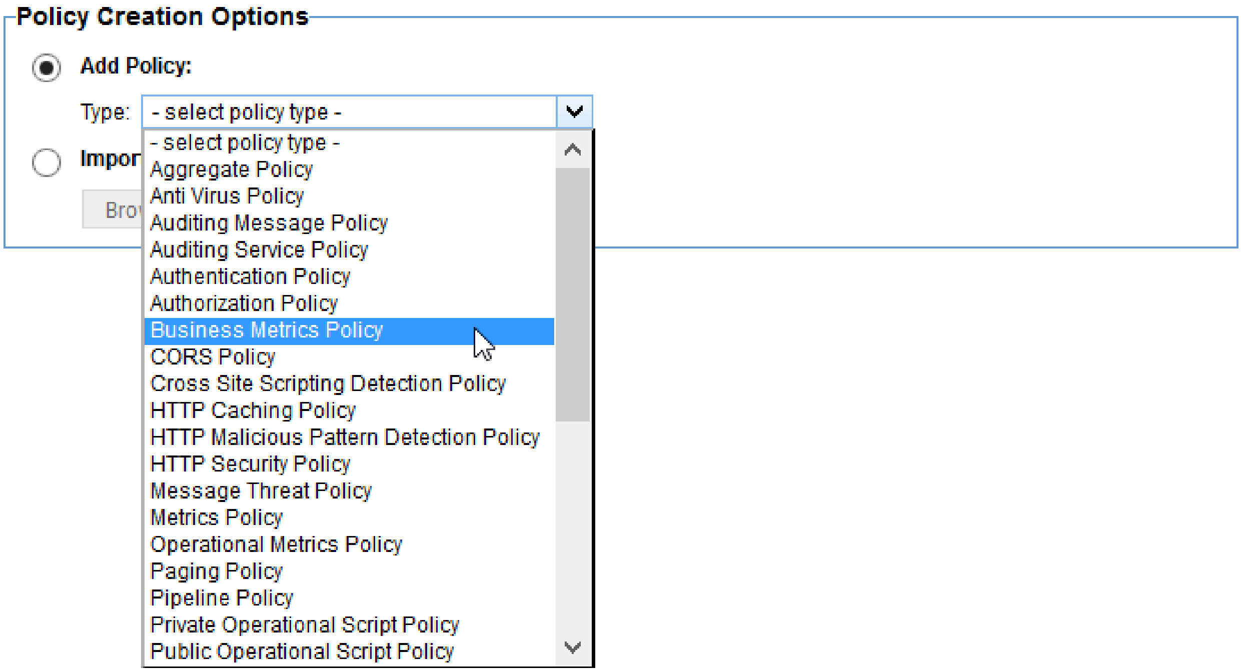
As described earlier there are plugins that are enhance both the API Gateway as wells as Envision. The following sections describe the enhanced user experience from the perspective of each product.

API Gateway Enhancements

With the installation of the Envision plugins in the API Gateway business metrics collection and SLA enforcement can now be controlled through the use of two policies, the Business Metrics Operational Policy and the Business Service Level QoS Policy. This section will describe how these policies are used with detailed examples.

Business Metrics Policy

The Business Metrics Policy provides instructions for extracting data from processed messages and mapping that data to fields in an Envision data set. To create a Business Metrics Policy select the Business Metrics Policy type in the first page of the Create Operational Policy wizard.



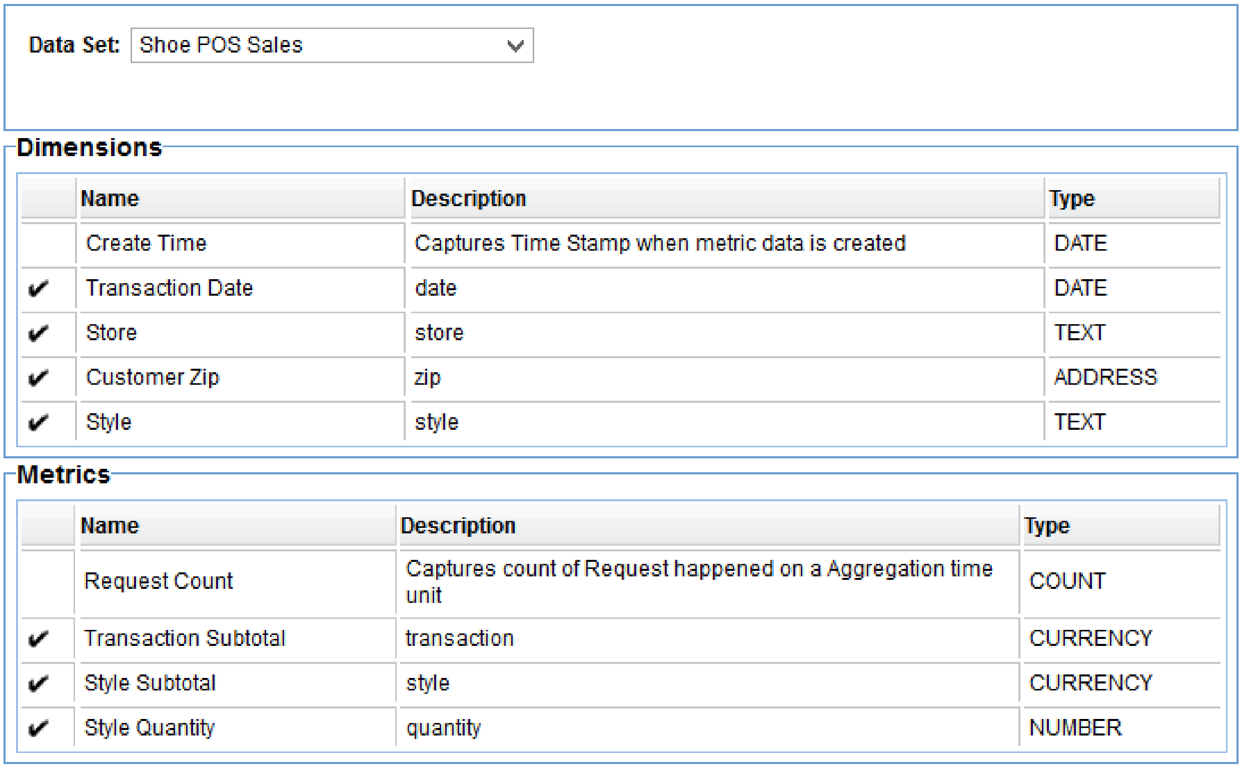
Policy Creation Options

● Add Policy:

Type: - select policy type -

○ Impor

The Business Metrics Policy editor is a multi-page wizard. On the first page you select the data set you want to fill with the policy.



Data Set: Shoe POS Sales

Dimensions

	Name	Description	Type
	Create Time	Captures Time Stamp when metric data is created	DATE
✓	Transaction Date	date	DATE
✓	Store	store	TEXT
✓	Customer Zip	zip	ADDRESS
✓	Style	style	TEXT

Metrics

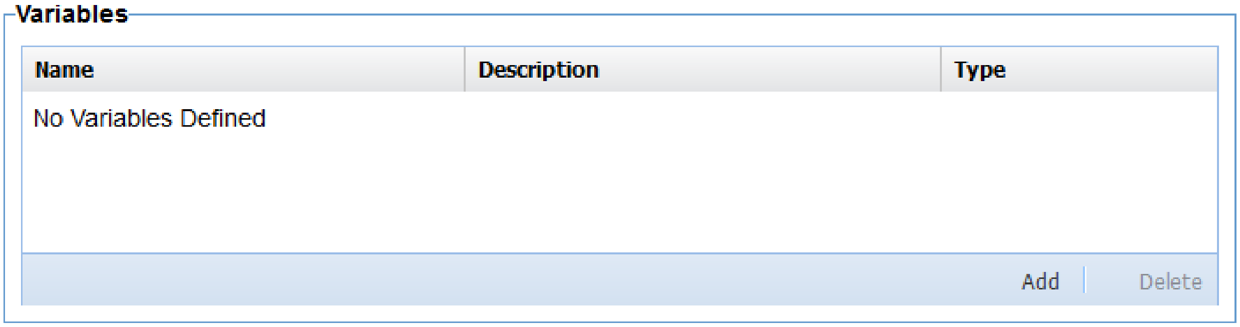
	Name	Description	Type
	Request Count	Captures count of Request happened on a Aggregation time unit	COUNT
✓	Transaction Subtotal	transaction	CURRENCY
✓	Style Subtotal	style	CURRENCY
✓	Style Quantity	quantity	NUMBER

A pull-down at the top will display the public data sets created in Envision as well as the logged in user's private data sets in Envision (if any). The dimensions and metrics for the selected data set are displayed in read-only tables to ensure the correct data set is selected.

The next four pages provide instructions for the creation of structures, variables, dimensions, and metrics. Dimensions and metrics have been discussed already. A variable is a value that doesn't correspond one-to-one with a dimension or metric in the target data set, but it can be used in an expression that will result in the creation of a dimension or metric. It can be used in multiple expressions in fact so it provides an opportunity for reuse and optimization by instructing the engine to perform a function once instead of repeatedly.

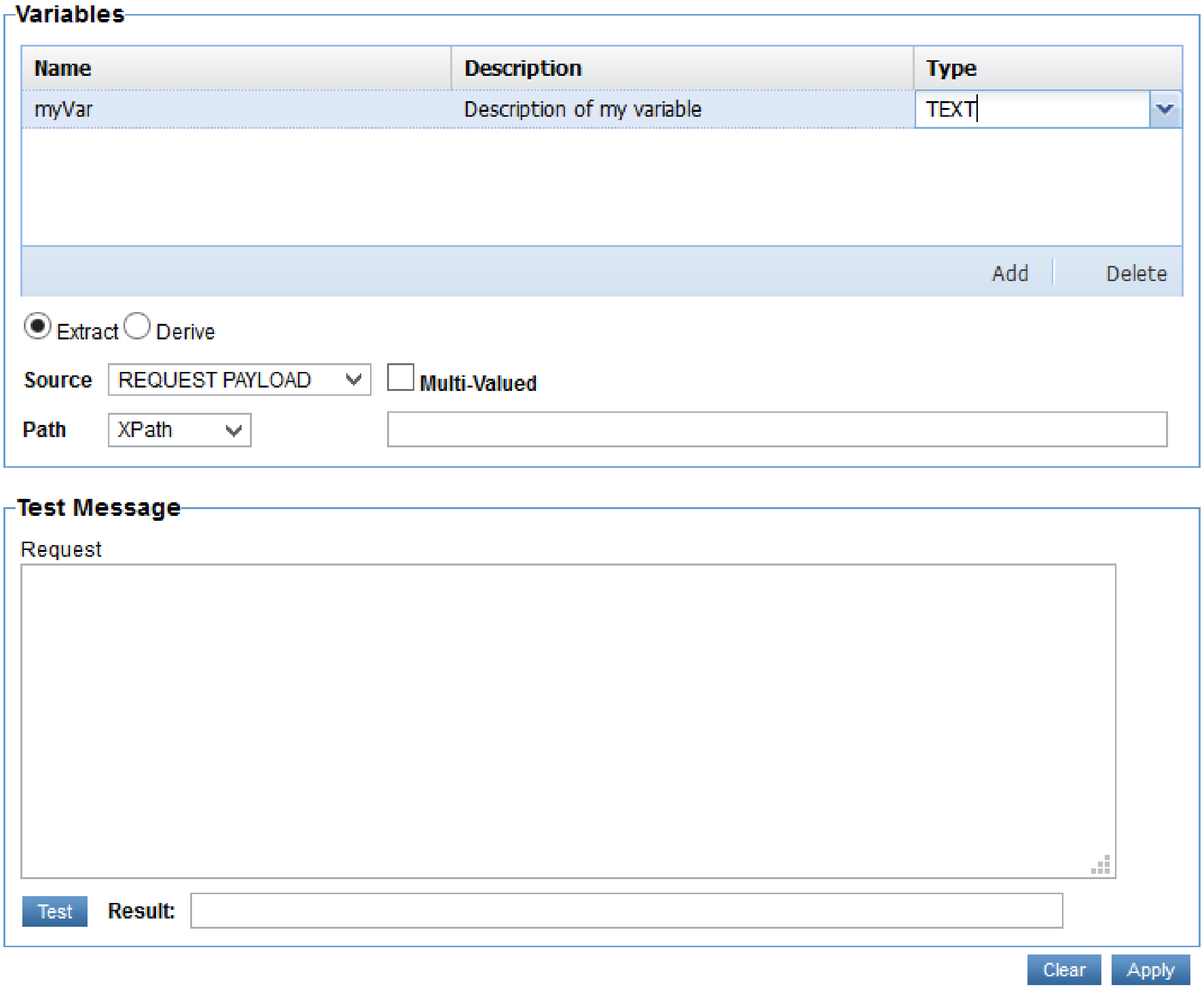
A structure models a sub-component of request or response message that may appear multiple times. For example if there are several items with their own properties that appear in a purchase transaction each item would be a structure. It is necessary to identify structures if dimensions or metrics are found in these sub-components and they need to be grouped together for aggregation. For example if each item in a purchase has a type and a price, in order for the aggregation engine to sum prices by type price cannot be collected without associating it to, or grouping it with, the type.

All four pages are organized in a similar way and provide similar capabilities. A table at the top of the page identifies the items being populated, structures, variables, dimensions, or metrics. For dimensions and metrics those tables are static as the rows are populated from the data set definition in Envision. Structures and variables on the other hand can be added and removed.



Name	Description	Type
No Variables Defined		

To create a variable select **Add** from the *Variables* table.



Name	Description	Type
myVar	Description of my variable	TEXT

● Extract ○ Derive

Source: REQUEST PAYLOAD □ Multi-Valued

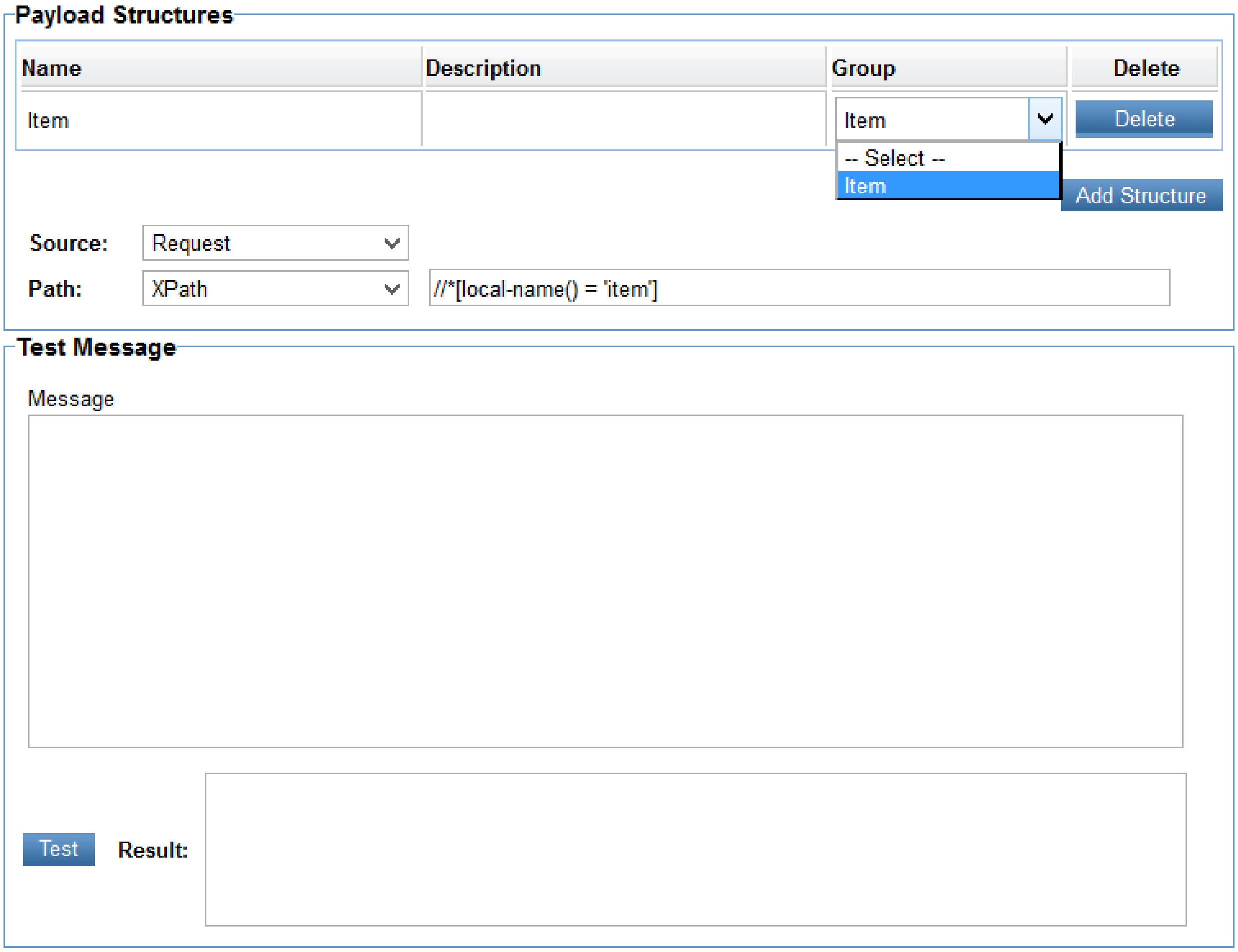
Path: XPath

Request

Result:

A new row will be added in the table. Each cell of the row is editable. The Type cell is a pull-down of the variable types supported which correspond to the types supported for metrics and dimensions by Envision.

Structures do not have a type. Instead they are mapped to Groups defined in a Data Set.



Payload Structures

Name	Description	Group	Delete
Item		Item	Delete

Source: Request

Path: XPath //*[local-name() = 'item']

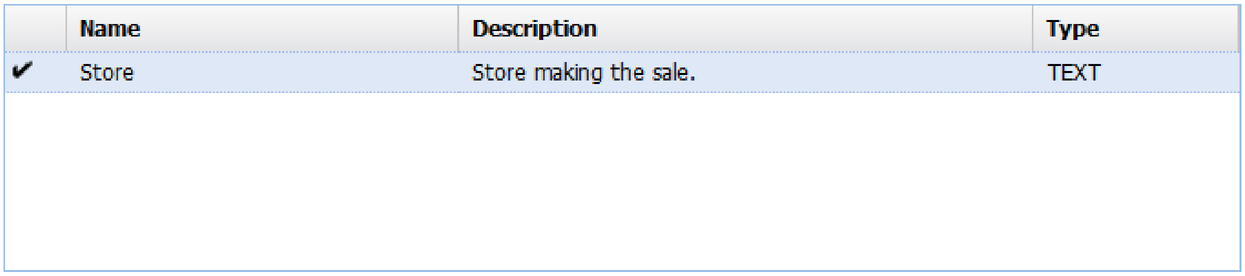
Test Message

Message

Result:

The *Group* column in the *Payload Structures* table holds a pull-down populated with all the Groups identified in a Data Set. The user selects a Group in the pull-down to associate the structure with the Group.

When any row of the target tables (structures, variables, dimensions, metrics) are selected the page is expanded to include options for how to extract or derive the value for the item identified by the selected row. Those options will described in more detail below. When finished configuring those options select **Apply** on the bottom right of the page. A checkmark will be displayed to the left of the item's row indicating its value has been defined.



	Name	Description	Type
✔	Store	Store making the sale.	TEXT

To start over again with those options select **Clear**. To delete a variable select the row identifying the variable to delete and select **Remove**.

For any selected item in a target table (variable, dimension, metric) its value can be generated by either extraction or derivation as identified by the *Extract* and *Derive* radio buttons.

Value Extraction

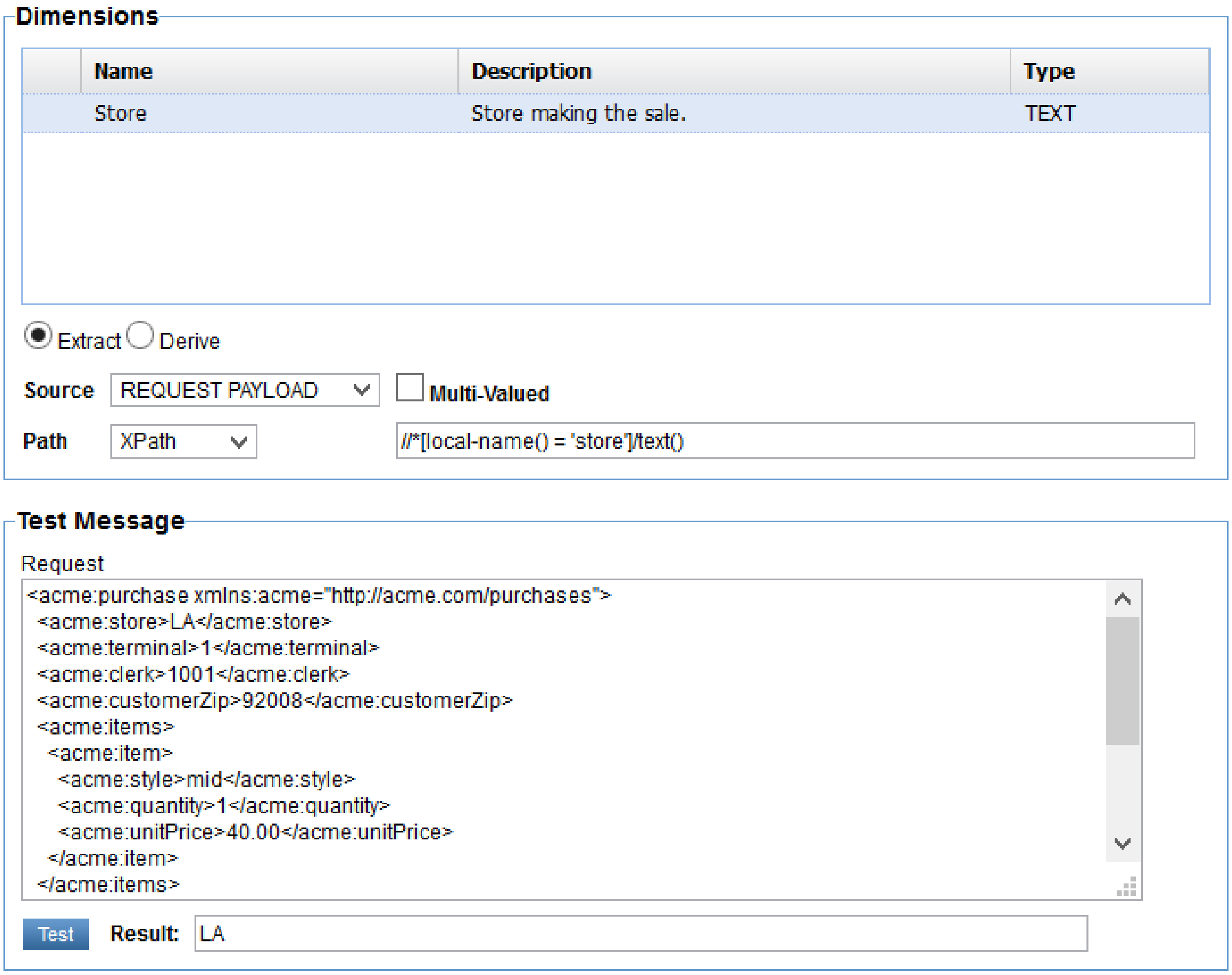
Value Extraction is the act of filling the value of a targeted item (structure, variable, dimension, variable) with data found in the exchange of messages between a client and an API/service. To extract data from the message exchange we must first identify the source of the extraction. This is identified using the *Source* pull-down. The options are:

- **Request Payload** – Body of the message sent by the client.
- **Request Header** – Header of the message sent by the client
- **Response Payload** – Body of the message returned by the service.
- **Response Header** – Header of the message returned by the service.
- **Fault Payload** – Body of the fault message returned by the service.
- **Fault Header** – Header of the fault message returned by the service.
- **Form Parameter** – Form parameter in the request message if form encoded.
- **Path Parameter** – Path parameter in the requested URL as defined in the service’s metadata.
- **Query Parameter** – Query parameter in the requested URL.
- **<Structure>** – Each structure defined on the *Structures* page will appear as an option in the *Source* pull-down. When selecting the structure as the source any expressions used to dictate how to extract the variable, dimension, or variable will be performed within the context of the structure. If multiple structures are found while executing the policy the expressions for extracting variables, dimensions, or variables will be executed for each structure found producing multiple results.

Only *Request Payload*, *Response Payload*, and *Fault Payload* are available when defining a structure.

When extracting data from a payload, request, response, or fault, you specify a path to, or location of, the value you want to extract from the payload content. This path is described using either XPath for XML payloads, JSONPath for JSON payloads, or RegEx for any payload as selected from the *Path* pull-down.

You then enter the path using the selected path language in the text box to the right. As a convenience a text area where you can paste a sample message payload is provided below and a Test button so that you can verify the path you enter will return the correct information.



Dimensions

	Name	Description	Type
	Store	Store making the sale.	TEXT

● Extract ○ Derive

Source REQUEST PAYLOAD □ **Multi-Valued**

Path XPath //*[local-name() = 'store']/text()

Test Message

Request

```
<acme:purchase xmlns:acme="http://acme.com/purchases">
 <acme:store>LA</acme:store>
 <acme:terminal>1</acme:terminal>
 <acme:clerk>1001</acme:clerk>
 <acme:customerZip>92008</acme:customerZip>
 <acme:items>
  <acme:item>
   <acme:style>mid</acme:style>
   <acme:quantity>1</acme:quantity>
   <acme:unitPrice>40.00</acme:unitPrice>
  </acme:item>
 </acme:items>
```

Result: LA

In the illustration above we have chosen to fill the Store dimension with the text from the “store” element in the Request Payload using XPath.

In some situations there are multiple values that the path expression will match. By default the values are concatenated in a comma delimited string. If you would like to do something with the individual values select the *Multi-Valued* checkbox. Once selected a pull-down of additional options will be displayed to its right. The options displayed are specific to the type of item you are trying to provide a value to.

For a Number or Currency the options are:

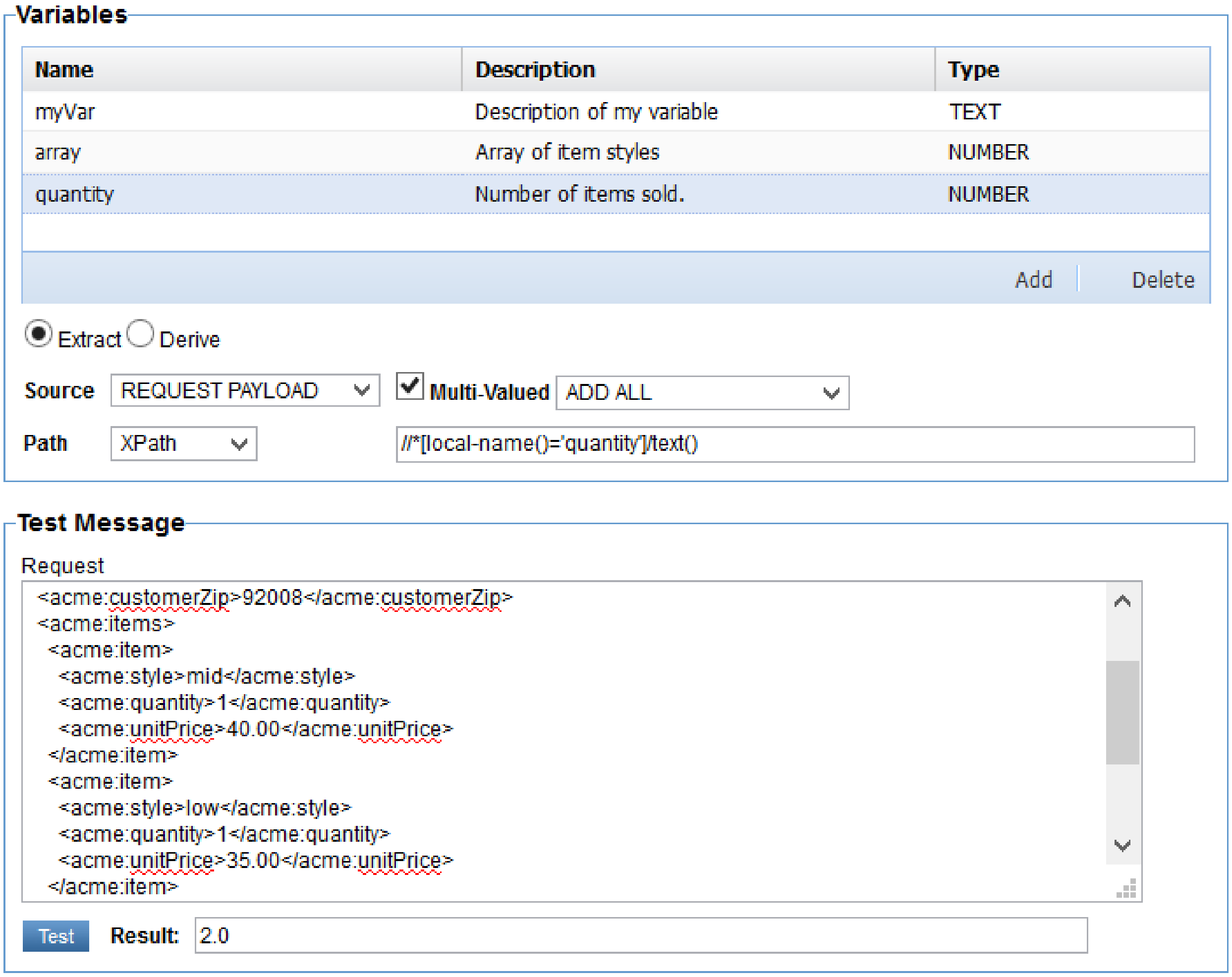
- **Add All** – Sums the values.
- **Get Maximum** – Selects the largest value.
- **Get Minimum** – Selects the smallest value.
- **Get First** – Selects the first value.
- **Get Last** – Selects the last value.
- **Number of Values** – Returns a count of the matching values.

For Text the options are:

- **Get First** – Selects the first value.

- **Get Last** – Selects the last value.
- **Number of Values** – Returns a count of the matching values.

The Test button can be used to test each of these functions. In the illustration below we are assigning the sum of the quantity field for each line item of a purchase to the quantity variable.



Variables

Name	Description	Type
myVar	Description of my variable	TEXT
array	Array of item styles	NUMBER
quantity	Number of items sold.	NUMBER

● Extract ○ Derive

Source REQUEST PAYLOAD ☑ **Multi-Valued** ADD ALL

Path XPath //*[local-name()='quantity']/text()

Test Message

Request

```
<acme:customerZip>92008</acme:customerZip>
<acme:items>
 <acme:item>
   <acme:style>mid</acme:style>
   <acme:quantity>1</acme:quantity>
   <acme:unitPrice>40.00</acme:unitPrice>
 </acme:item>
 <acme:item>
   <acme:style>low</acme:style>
   <acme:quantity>1</acme:quantity>
   <acme:unitPrice>35.00</acme:unitPrice>
 </acme:item>
```

Result: 2.0

Value Derivation

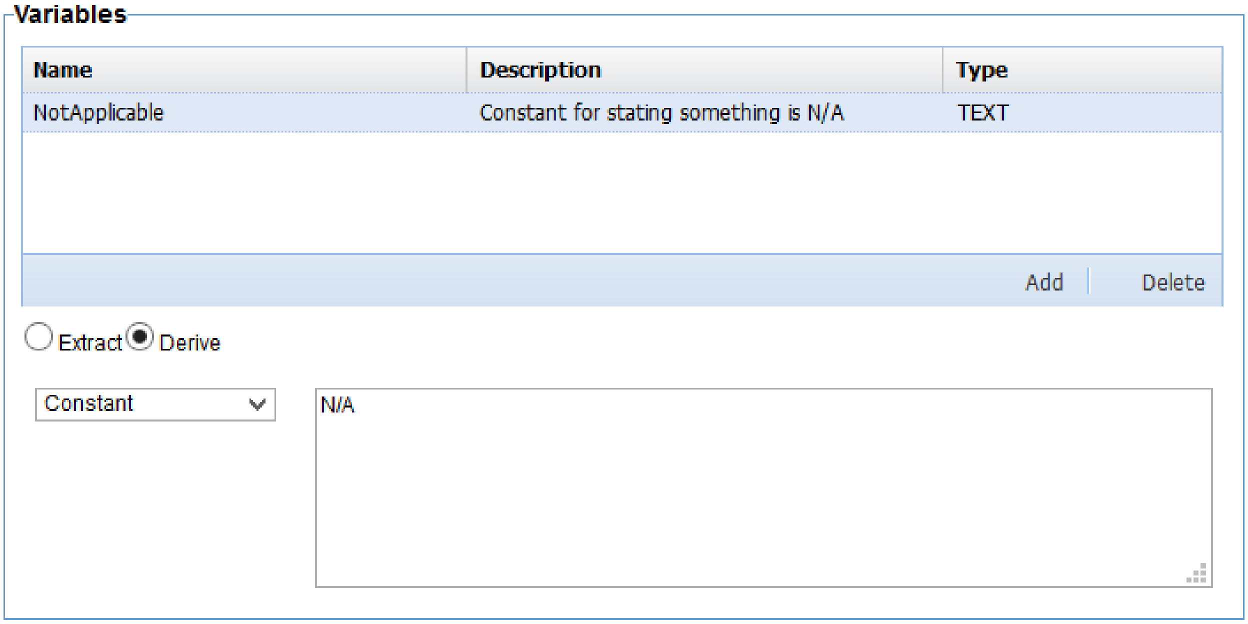
Value Derivation is the act of filling the value of a targeted item (variable, dimension, variable) from constants or expressions. The expressions can reference variables whose values are filled through data extraction so derivation can indirectly utilize extraction.

When the *Derive* radio button is selected a pull-down of the following derivation options is displayed.

Constant

The value of the target item is assigned the literal value entered in the text box to the right. The literal value should match the type of the item. For example entering “foo” in the text box for a NUMBER variable would be an error.

The following illustration demonstrates the creation of a variable that can be used to identify something that is not applicable with the value of “N/A”.



Variables

Name	Description	Type
NotApplicable	Constant for stating something is N/A	TEXT

○ Extract ● Derive

Constant

N/A

Simple Expression

The value of the target item is filled by the result of an expression written using the Unified Expression Language (UEL) as described as part of JSR-245. Other variables can be referenced in the expression simply by using their names. For example

Spring Expression

The value of the target item is filled by the result of an expression written using the Spring Expression Language (SpEL).

Operational Metric

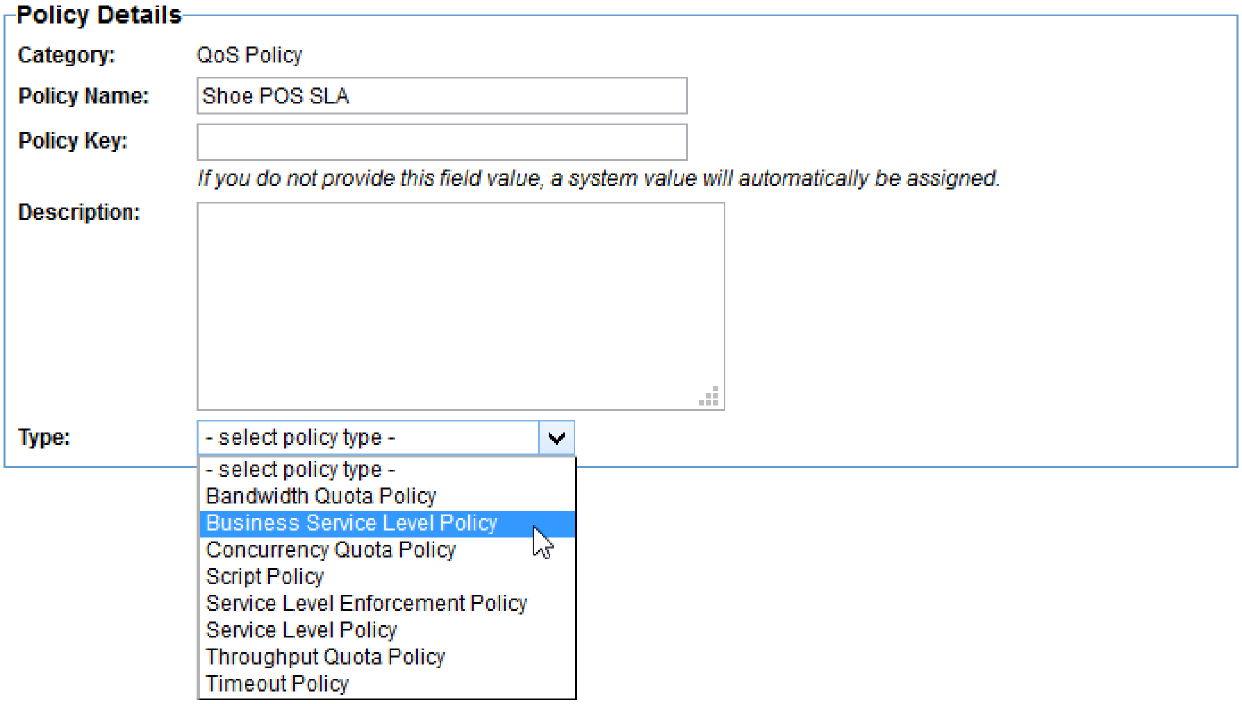
The value of the target item is filled with a value of one of the standard operational dimensions or metrics. When this option is selected a pull-down with the acceptable values will be displayed to the right.

Addresses

An Address dimension type is actually stored as a structure with a zip code, city, state, latitude, and longitude. The job of the business metrics policy is to extract or derive a single field of this structure. The Akana Envision Policy Manager Services Extensions plugin will fill the rest of the fields in the structure. The policy does not require the identification of which field is being populated initially. The system will determine which field is being populated by querying the geolocation data in the database. First it checks to see if the value collected by the policy is a zip code, then a city, then state, and finally country. Once this has been determined the system will populate the latitude and longitude fields as well. These become useful when using maps in Envision.

Business Service Level Policy

The Business Service Level policy provides the ability to define a Service Level Agreement (SLA) based on data set defined in Envision instead of the standard operational metrics provided by Policy Manager. For example, an SLA can be created based on the number of items sold, or the number of calls made from a particular user-agent. To create a Business Service Level Policy select the Business Service Level Policy type in the first page of the Create QoS Policy wizard.



Policy Details

Category: QoS Policy

Policy Name: Shoe POS SLA

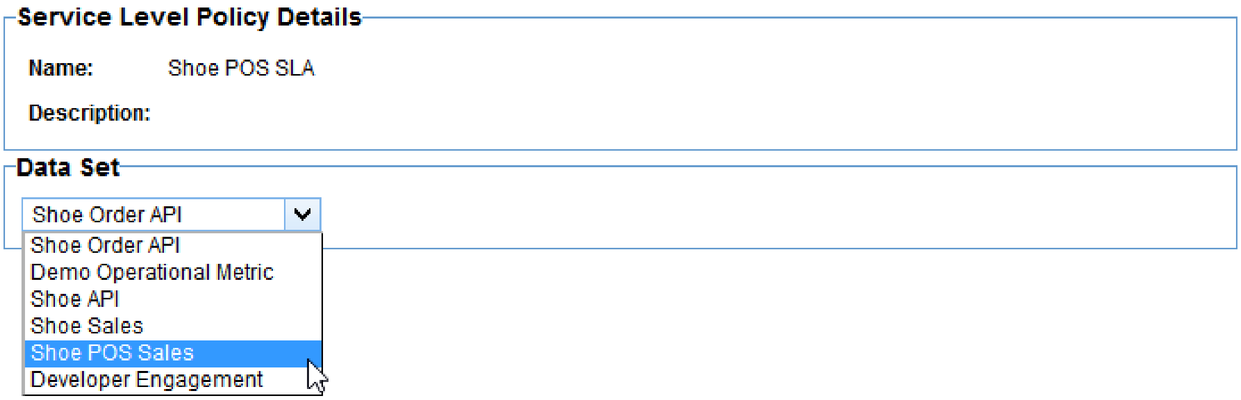
Policy Key:

If you do not provide this field value, a system value will automatically be assigned.

Description:

Type: - select policy type -

The Business Service Level Policy editor is a multi-page wizard. On the first page you select the data set you want to use as the basis of the policy.



Service Level Policy Details

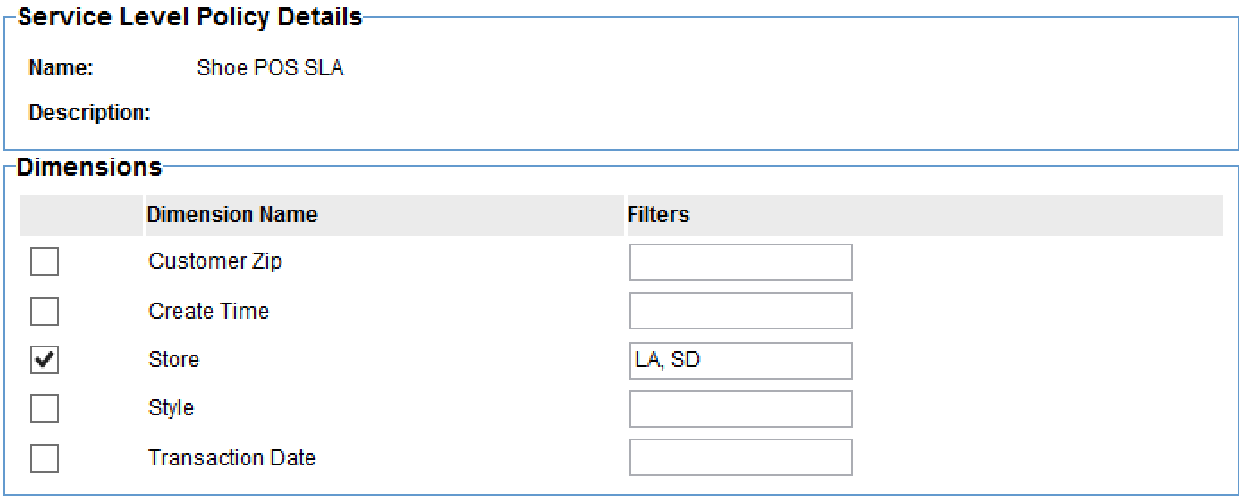
Name: Shoe POS SLA

Description:

Data Set

Shoe Order API

Next you select what records in the data set you want the metrics to measure the service level with. Essentially you are defining a query, or filter.



Service Level Policy Details

Name: Shoe POS SLA

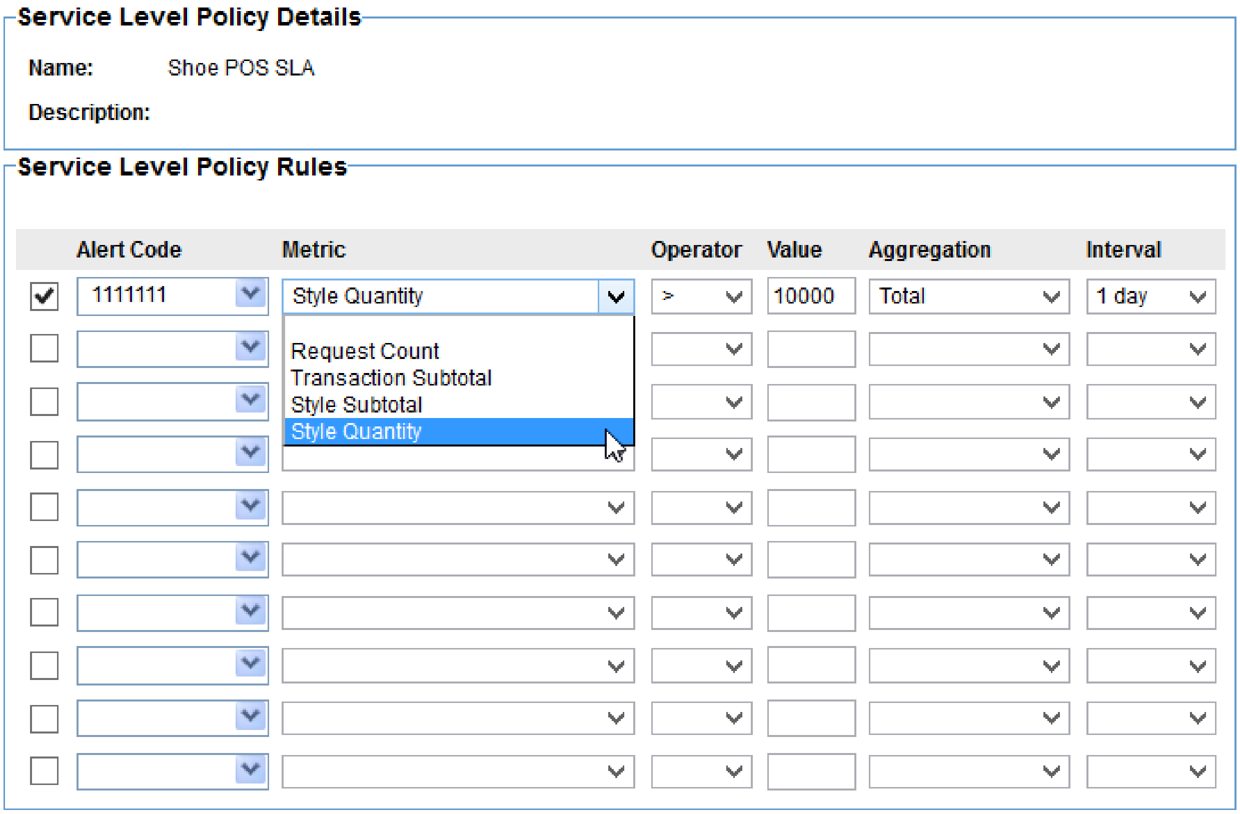
Description:

Dimensions

	Dimension Name	Filters
□	Customer Zip	
□	Create Time	
☑	Store	LA, SD
□	Style	
□	Transaction Date	

If you want the metrics to be aggregated for all records in the data set nothing need be selected on the screen. However if you want to filter the records by one or more dimensions, select those dimensions. In the text fields to the right of the selected dimensions enter the values of the dimensions you want to filter by. In the illustration above we are filtering the data set to only the transactions made at the LA and SD stores.

Next define the rules for the policy. This page is identical to the standard Policy Manager Service Level Policy. You select an alert code to generate when an expression is met in a specified interval.



Service Level Policy Details

Name: Shoe POS SLA

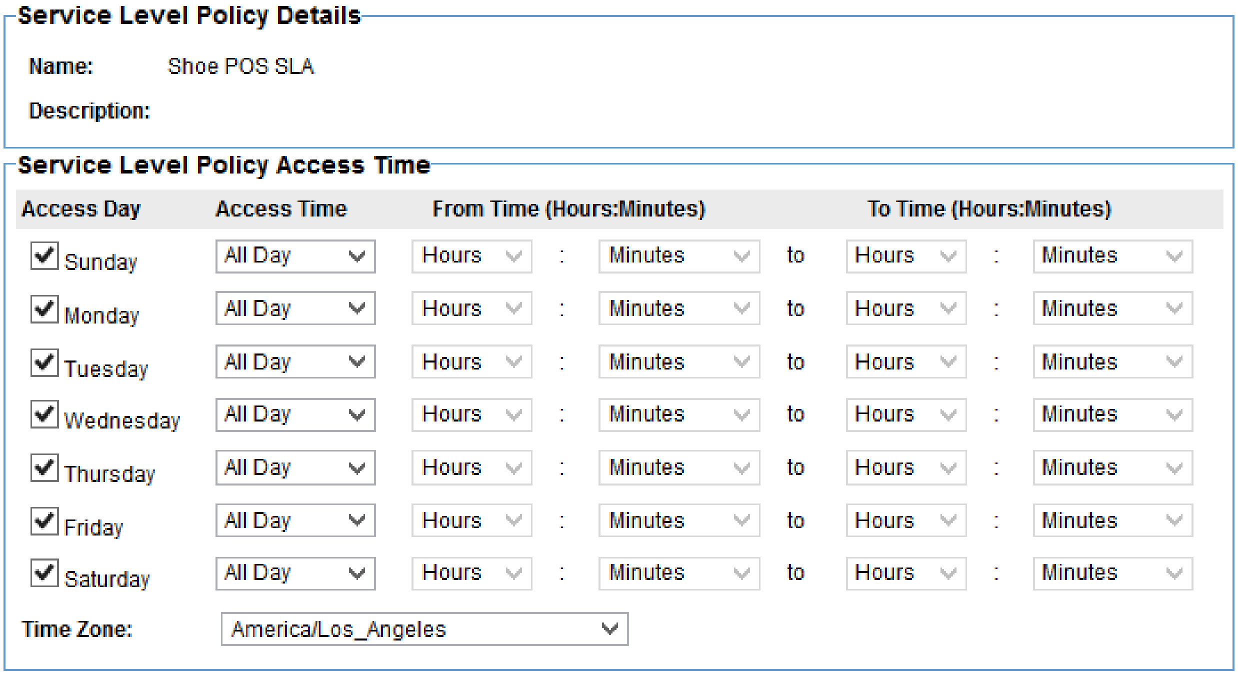
Description:

Service Level Policy Rules

	Alert Code	Metric	Operator	Value	Aggregation	Interval
☑	1111111	Style Quantity	>	10000	Total	1 day
□						
□						
□						
□						
□						
□						
□						
□						
□						

What makes this different than the standard service level policy is the metrics to choose from are those collected in the selected data set. As seen above an alert will be generated with over 10000 shoes of a certain style are sold in a day.

The final page is for defining the times that the policy should govern. This is also the same as the standard Policy Manager Service Level Policy.



Service Level Policy Details

Name: Shoe POS SLA

Description:

Service Level Policy Access Time

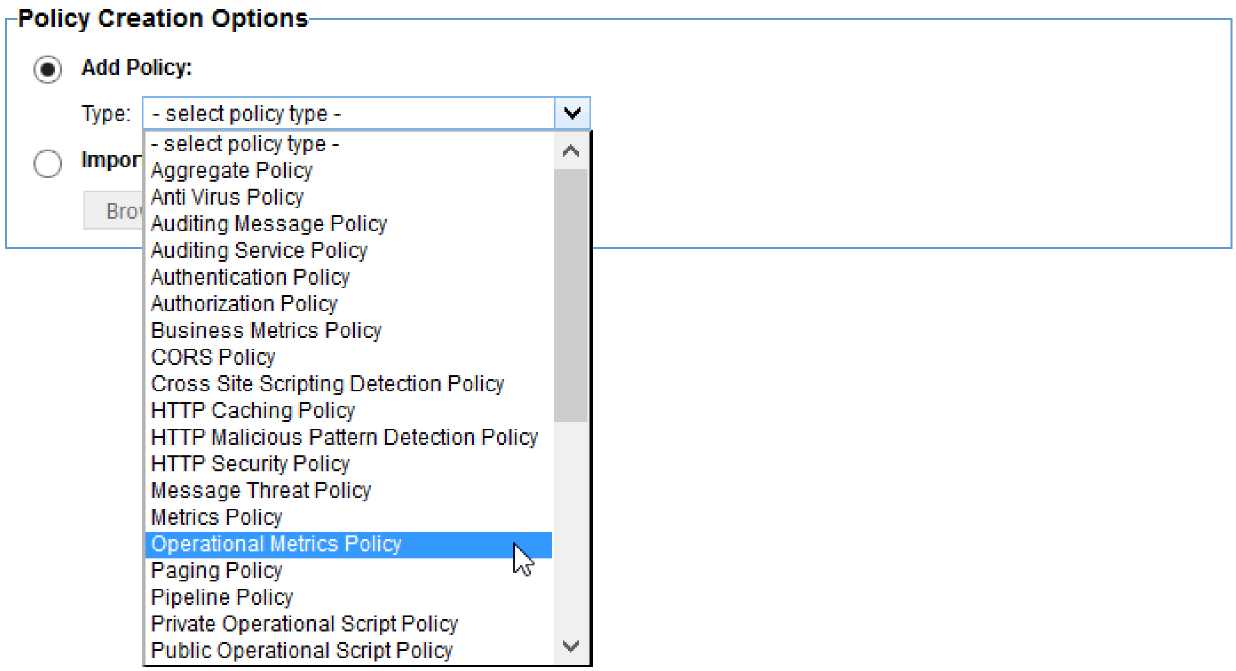
Access Day	Access Time	From Time (Hours:Minutes)				To Time (Hours:Minutes)		
☑ Sunday	All Day	Hours	:	Minutes	to	Hours	:	Minutes
☑ Monday	All Day	Hours	:	Minutes	to	Hours	:	Minutes
☑ Tuesday	All Day	Hours	:	Minutes	to	Hours	:	Minutes
☑ Wednesday	All Day	Hours	:	Minutes	to	Hours	:	Minutes
☑ Thursday	All Day	Hours	:	Minutes	to	Hours	:	Minutes
☑ Friday	All Day	Hours	:	Minutes	to	Hours	:	Minutes
☑ Saturday	All Day	Hours	:	Minutes	to	Hours	:	Minutes

Time Zone: America/Los_Angeles

The policy acts in a similar way to the standard service level policy. Alerts are generated when the expressions are met. As long as the expressions continue to be met on subsequent intervals nothing more is done. Once the expression is no longer met a “clear” alert is generated.

Operational Metrics Policy

The Operational Metrics Policy is a more flexible option to the standard Policy Manager Metrics Policy. It allows you to specify what operational metrics to collect for an API in a more specific way. To create an Operational Metrics Policy select the Operational Metrics Policy type in the first page of the Create Operational Policy wizard.



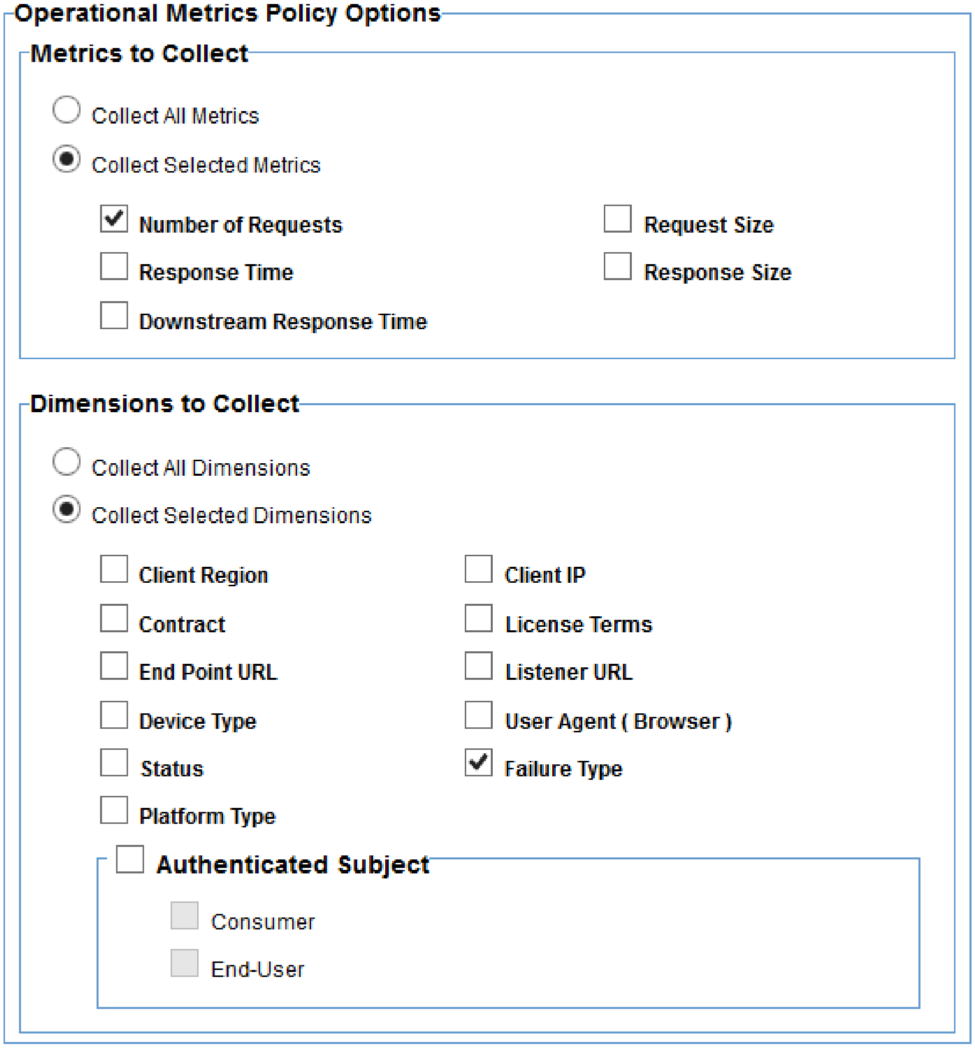
Policy Creation Options

● Add Policy:

Type: - select policy type -

○ Impor

The Operational Metrics Policy editor is a single page. On this page you select the metrics you want to collect and the dimensions you want to group the data by so that the data can be filtered in Envision when creating charts.



Operational Metrics Policy Options

Metrics to Collect

○ Collect All Metrics

● Collect Selected Metrics

☑ Number of Requests
□ Request Size
□ Response Time
□ Response Size
□ Downstream Response Time

Dimensions to Collect

○ Collect All Dimensions

● Collect Selected Dimensions

□ Client Region
□ Client IP
□ Contract
□ License Terms
□ End Point URL
□ Listener URL
□ Device Type
□ User Agent (Browser)
□ Status
☑ Failure Type
□ Platform Type

□ Authenticated Subject

□ Consumer
□ End-User

The metrics that can be collected are the same metrics collected by Policy Manager. The difference here is not all of them MUST be collected if they are not useful.

Many of the dimensions that can be used to group the aggregated metrics are also the same as those used by Policy Manager, Contract, License Terms, Failure Type (did the call result in an error or not). In addition the following dimensions can be used.

- **Client Region** – The location of the client making the call. It is the latitude and longitude of the client IP address.
- **Downstream URL** – URL of the downstream service invoked (if any).
- **Device Type** – Type of device making the call. This is derived from the User-Agent header.
- **Status** – Status code returned to the client making the call.
- **Platform Type** – Type of platform, or operating system, making the call. This is derived from the User-Agent header.
- **Client IP** – IP Address of the client making the call.

- **User Agent** – Value of the User-Agent header (if any).
- **Authenticated Subject** – The authenticated caller (if any).

Use Case – Point of Sale Metrics for Shoe Retailer

To help provide a better understanding of the API Gateway enhancements described earlier a use case will be explored in depth. The use case will center on a fictional shoe retailer ACME whose point of sale terminals communicate with a central purchasing SOAP based web service. Each transaction will be processed by the API Gateway. The retailer will use a business metrics policy to collect information about the transactions and feed them to Envision for aggregation and charting.

ACME has three brick and mortar stores in Los Angeles, San Francisco, and San Diego. The company sells three styles of shoes, high tops, mid tops, and low tops. ACME would like to track the sales of their shoes in a few different ways:

- Total company sales (minus sales tax)
- Total sales (minus sales tax) of each store
- Quantity sold by each store
- Quantity sold of each style
- Quantity sold of each style by each store
- Total sales (minus sales tax) by customer zip code
- Quantity sold of each style of shoe by customer zip code

The API

As mentioned already ACME uses a SOAP web service to connect its point of sale terminals to its backend purchasing system. A sample of the SOAP payload for a transaction is as follows:

```
<soapenv:Envelope xmlns:soapenv="…">
   <soapenv:Header/>
   <soapenv:Body>
      <sho:purchase xmlns:sho="http://acme.com/api/shoepos">
         <sho:store>LA</sho:store>
         <sho:terminal>1</sho:terminal>
         <sho:clerk>1001</sho:clerk>
         <sho:items>
             <sho:item>
                 <sho:style>mid</sho:style>
                 <sho:quantity>1</sho:quantity>
                 <sho:unitPrice>40.00</sho:unitPrice>
             </sho:item>
             <sho:item>
                 <sho:style>low</sho:style>
                 <sho:quantity>2</sho:quantity>
                 <sho:unitPrice>35.00</sho:unitPrice>
             </sho:item>
         </sho:items>
         <sho:subtotal>110.00</sho:subtotal>
         <sho:tax>8.80</sho:tax>
         <sho:total>118.80</sho:total>
         <sho:customerZip>90001</sho:customerZip>
```

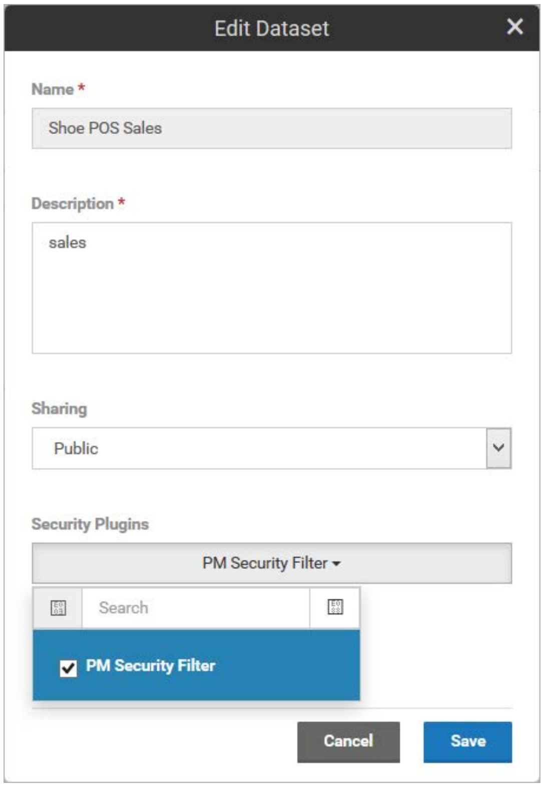
```
24)          <sho:transactionTime>2002-05-30T09:30:00Z</sho:transactionTime>
25)       </sho:purchase>
26)    </soapenv:Body>
27) </soapenv:Envelope>
```

The purchase starts on line 04. The store where the purchase was made is on line 05, the possible values are LA, SF, and SD. The terminal used to make the purchase is on line 06. The sales clerk who performed the transaction is on line 07. A customer can purchase multiple shoes in a single transaction. They are listed within the “items” element on line 08. There is a single “item” representing 0 or more shoes of a single style being purchased in the transaction. In this example a single mid top shoe is sold on lines 09 – 13. The style is identified on line 10. The quantity is on line 11. The price for a mid top shoe at the time of the purchase was $40 as stated on line 12. Lines 14 – 18 state that the customer also purchased two low top shoes at a price of $35 each. The subtotal (without tax) of the total purchase is on line 20. The sales tax for the purchase is on line 21. The total transaction amount (including tax) is on line 22. When a purchase is made the customer’s zip code is collected and recorded on line 23. The time of the transaction is on line 24. Due to system latency it may not be the same time as the gateway will process or record the data about the transaction.

The Data Set

The first step in achieving the goals described in this use case is to define a Data Set for the purchases in Envision.

The Data Set is named “Shoe POS Sales”.



Edit Dataset

Name *

Shoe POS Sales

Description *

sales

Sharing

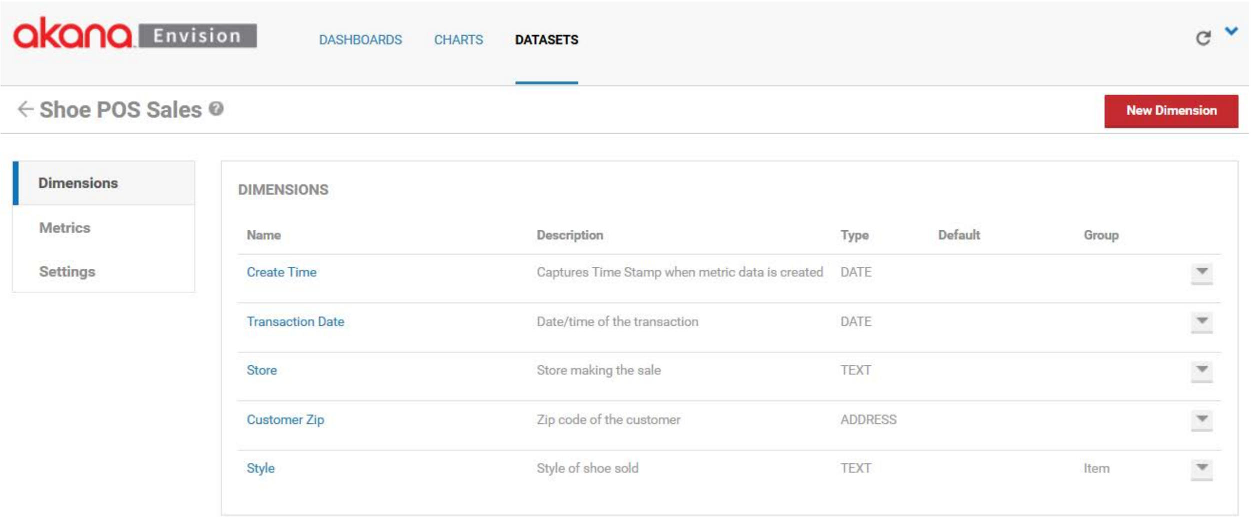
Public

Security Plugins

PM Security Filter

☑ PM Security Filter

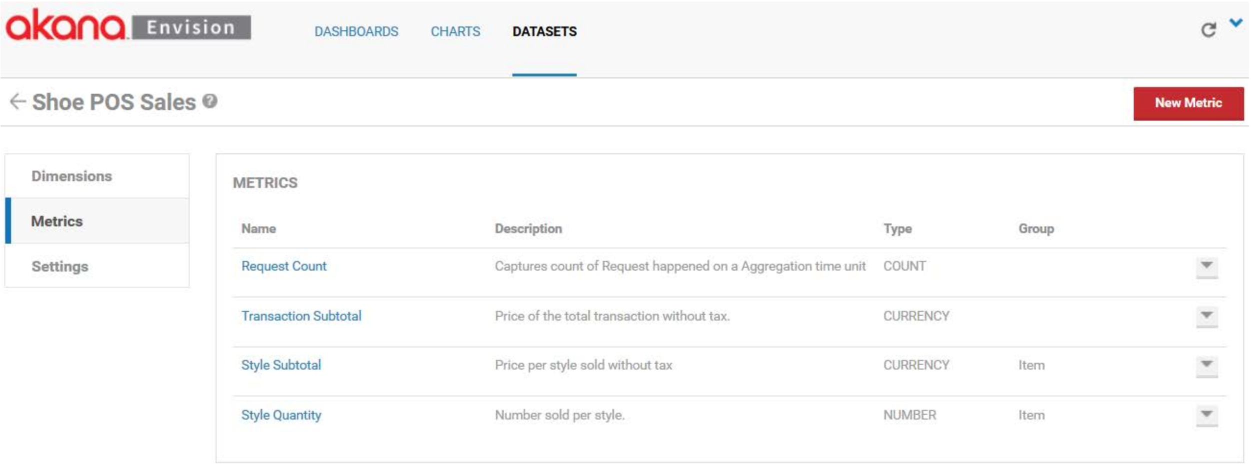
By selecting the PM Security Filter any data displayed in charts and dashboards will be secured by the Policy Manager security system. Only users with access to the APIs reporting the metrics will be able to view the information.



DIMENSIONS

Name	Description	Type	Default	Group
Create Time	Captures Time Stamp when metric data is created	DATE		
Transaction Date	Date/time of the transaction	DATE		
Store	Store making the sale	TEXT		
Customer Zip	Zip code of the customer	ADDRESS		
Style	Style of shoe sold	TEXT		Item

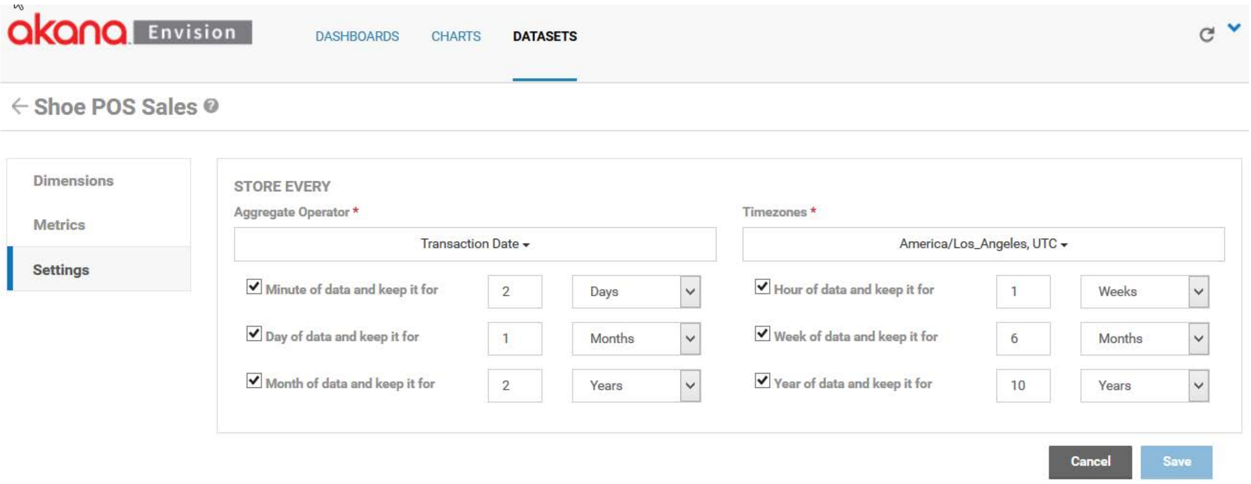
There are dimensions for each of the queries ACME wants to perform, Store, Customer Zip, and Style. The Customer Zip dimension is an ADDRESS instead of TEXT so that maps can be used to geographically view data. Note that because there may be multiple line items in a purchase there may be multiple shoe styles in a since purchase. Therefore the “Style” dimension is associated with a Group named “Item”. As we’ll see in the metrics, the “Item” Group will include all information about a purchase line item. Finally a Transaction Date dimension is added for querying based on time (what will become our X-Axis) as this information is provided in the POS API and can be different than the time the data is collected in the gateway (Create Time).



METRICS

Name	Description	Type	Group
Request Count	Captures count of Request happened on a Aggregation time unit	COUNT	
Transaction Subtotal	Price of the total transaction without tax.	CURRENCY	
Style Subtotal	Price per style sold without tax	CURRENCY	Item
Style Quantity	Number sold per style.	NUMBER	Item

There are Metrics for each of the measurements ACME wants to take, Transaction Subtotal, Style Subtotal, and Style Quantity. ACME’s measurements only require sum’s so only the SUM aggregation need be performed on each metric. Since there may be multiple styles sold in a single transaction the style specific metrics are placed in the “Item” group along with the “Style” dimension.



← Shoe POS Sales

STORE EVERY

Aggregate Operator * : Transaction Date

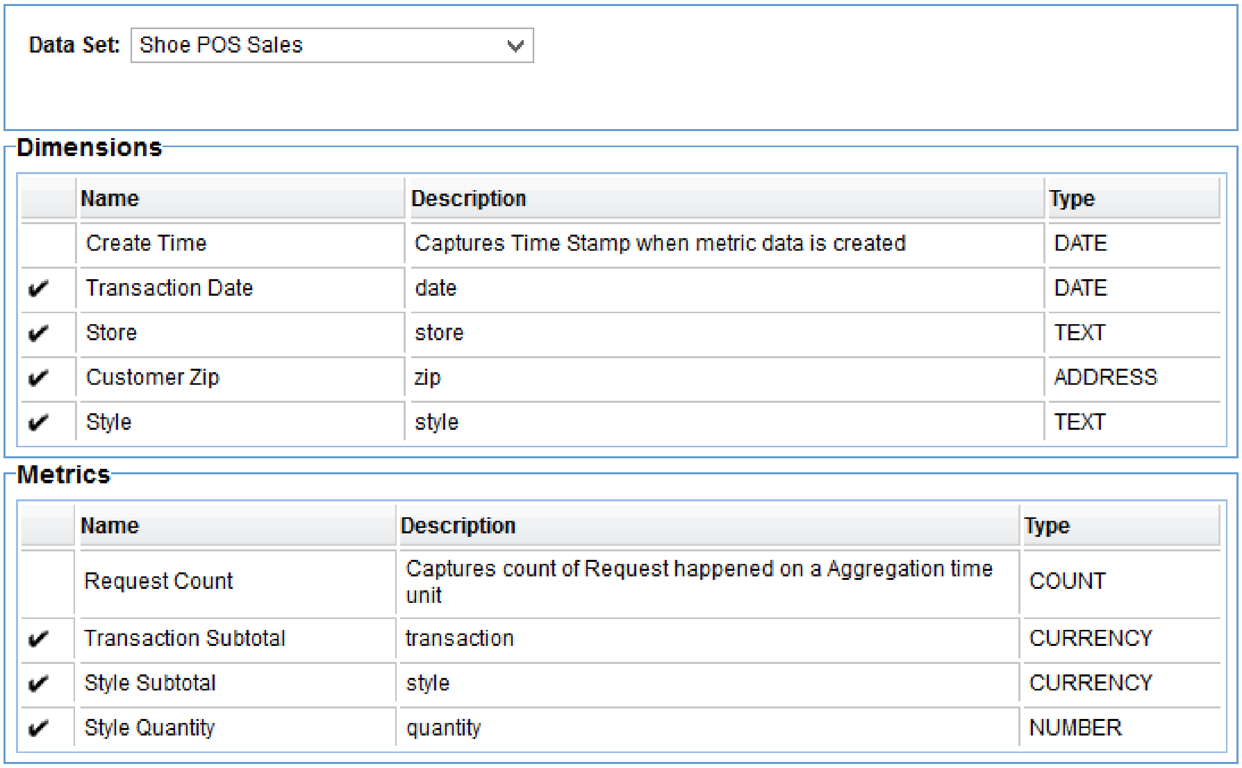
Timezones * : America/Los_Angeles, UTC

☑ Minute of data and keep it for	2	Days	☑ Hour of data and keep it for	1	Weeks
☑ Day of data and keep it for	1	Months	☑ Week of data and keep it for	6	Months
☑ Month of data and keep it for	2	Years	☑ Year of data and keep it for	10	Years

Because aggregation will be performed using the transaction date at the point of sale terminal the aggregate operator is set to the “Transaction Date” dimension. ACME is headquartered in Los Angeles so when determining the boundaries of daily metrics the America/Los_Angeles time zone is used.

The Policy

A Business Metrics Policy is created in Policy Manager named “Shoe POS Metrics Policy”.



Data Set: Shoe POS Sales

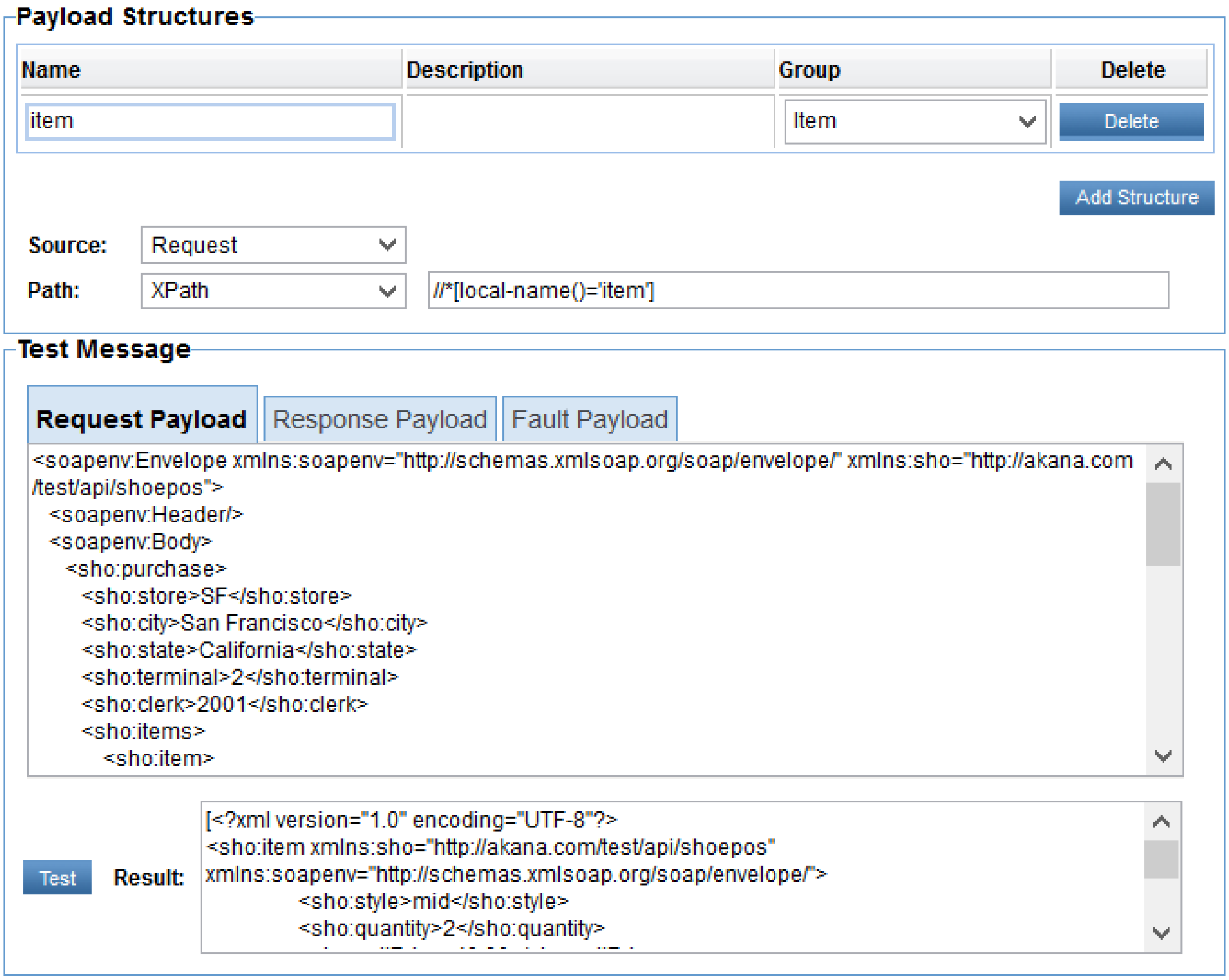
Dimensions

	Name	Description	Type
	Create Time	Captures Time Stamp when metric data is created	DATE
✔	Transaction Date	date	DATE
✔	Store	store	TEXT
✔	Customer Zip	zip	ADDRESS
✔	Style	style	TEXT

Metrics

	Name	Description	Type
	Request Count	Captures count of Request happened on a Aggregation time unit	COUNT
✔	Transaction Subtotal	transaction	CURRENCY
✔	Style Subtotal	style	CURRENCY
✔	Style Quantity	quantity	NUMBER

The policy fills the Shoe POS Sales Data Set. There is one structure defined named “Item” that corresponds to the “Item” group in the Data Set.



Name	Description	Group	Delete
item		Item	Delete

Source: Request

Path: XPath //*[local-name()='item']

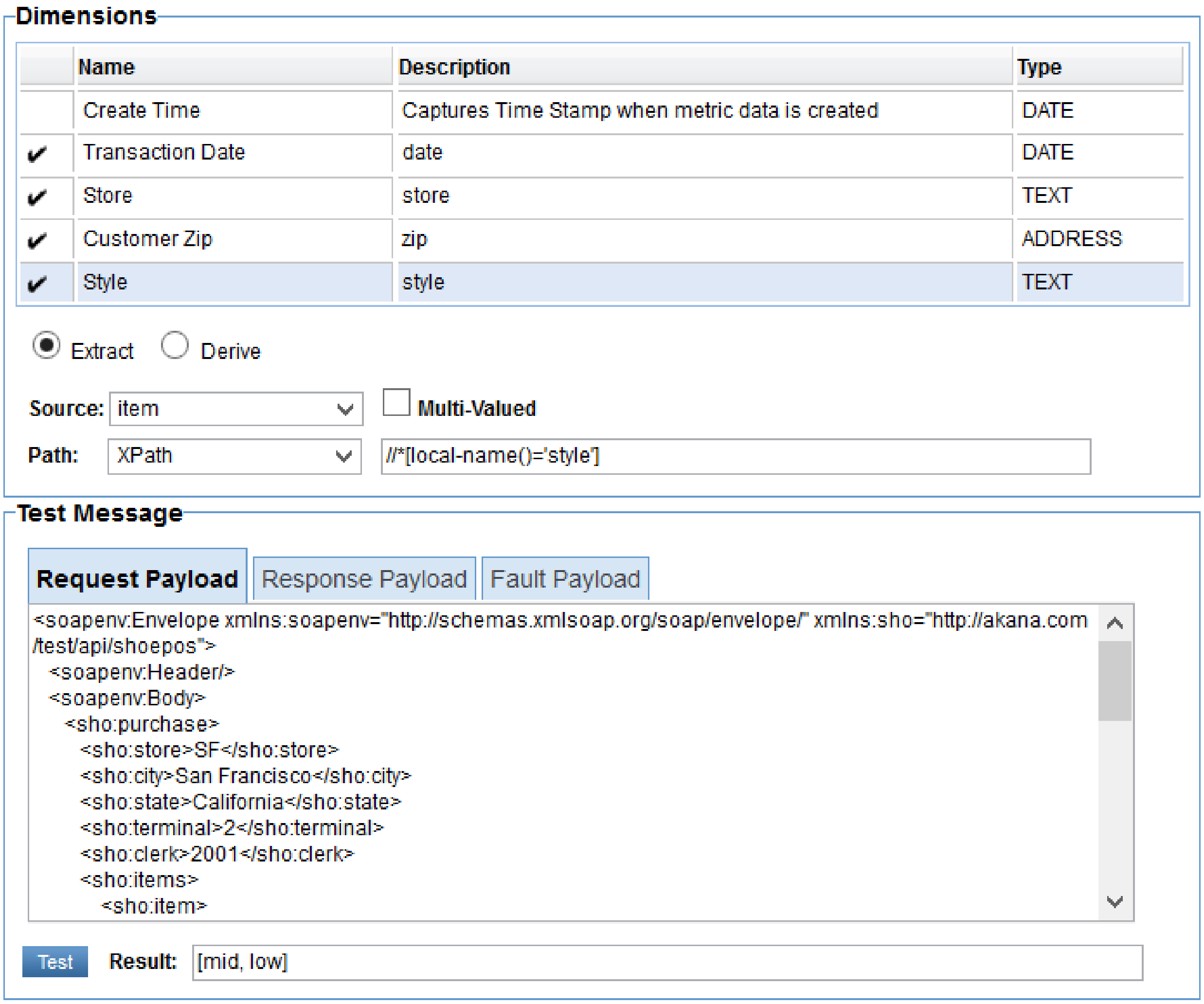
```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:sho="http://akana.com
/test/api/shoepos">
  <soapenv:Header/>
  <soapenv:Body>
    <sho:purchase>
      <sho:store>SF</sho:store>
      <sho:city>San Francisco</sho:city>
      <sho:state>California</sho:state>
      <sho:terminal>2</sho:terminal>
      <sho:clerk>2001</sho:clerk>
      <sho:items>
        <sho:item>
```

Result:

```
[<?xml version="1.0" encoding="UTF-8"?>
<sho:item xmlns:sho="http://akana.com/test/api/shoepos"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
            <sho:style>mid</sho:style>
            <sho:quantity>2</sho:quantity>
```

The "Item" structure is identified in the message using an XPath expression. When tested against the sample message described previously the XPath expression returns the shoe:item elements in the message. All subsequent XPath expressions for metrics and dimensions in the "Item" group will be performed relative to each of the shoe:item elements in the message.

There are no variables defined in the policy. Each of the dimensions are extracted from the message using XPath expressions.



Dimensions

	Name	Description	Type
	Create Time	Captures Time Stamp when metric data is created	DATE
✓	Transaction Date	date	DATE
✓	Store	store	TEXT
✓	Customer Zip	zip	ADDRESS
✓	Style	style	TEXT

● Extract ○ Derive

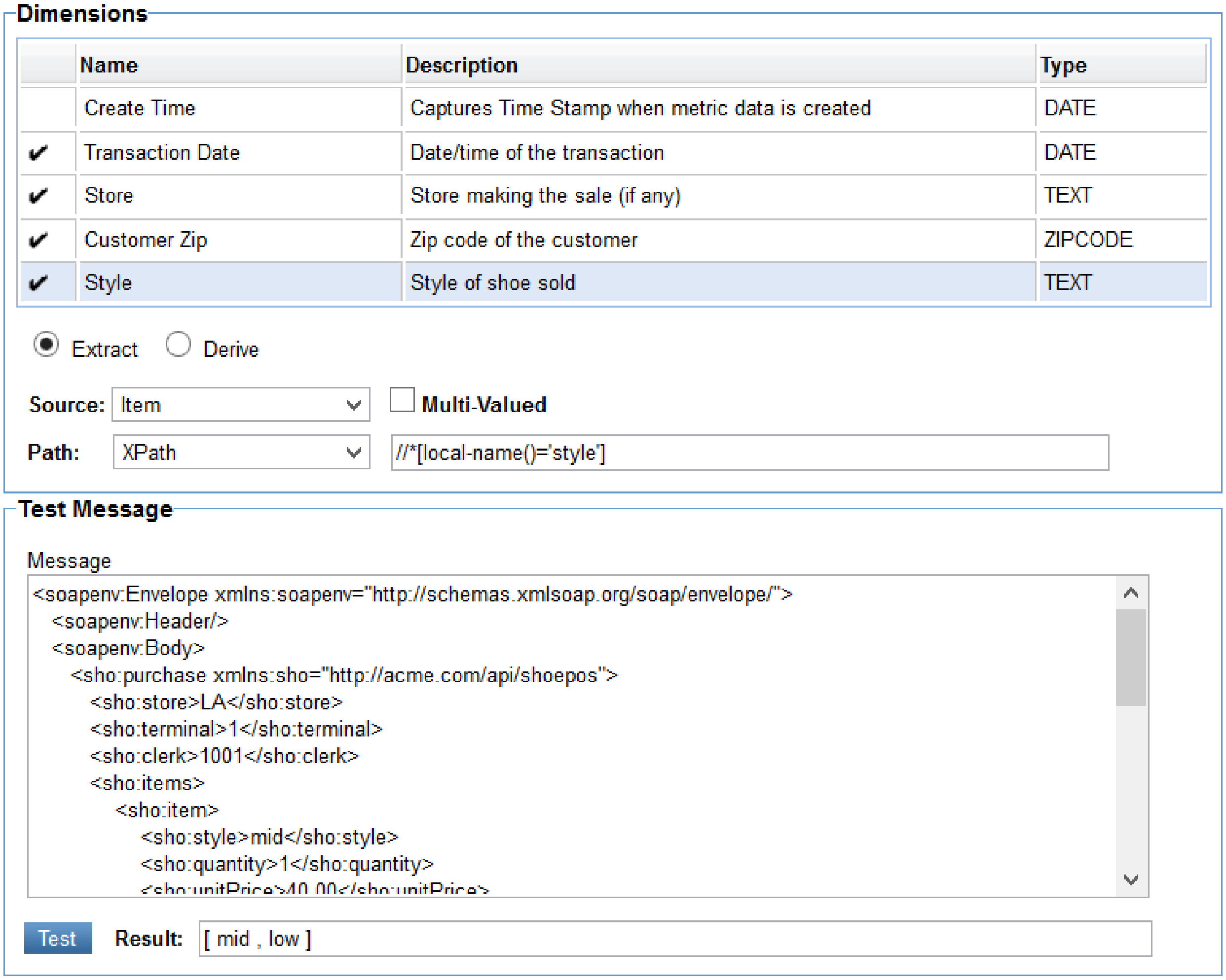
Source: item □ **Multi-Valued**

Path: XPath //*[local-name()='style']

Test Message

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:sho="http://akana.com
/test/api/shoepos">
  <soapenv:Header/>
  <soapenv:Body>
    <sho:purchase>
      <sho:store>SF</sho:store>
      <sho:city>San Francisco</sho:city>
      <sho:state>California</sho:state>
      <sho:terminal>2</sho:terminal>
      <sho:clerk>2001</sho:clerk>
      <sho:items>
        <sho:item>
```

Result: [mid, low]



Dimensions

	Name	Description	Type
	Create Time	Captures Time Stamp when metric data is created	DATE
✓	Transaction Date	Date/time of the transaction	DATE
✓	Store	Store making the sale (if any)	TEXT
✓	Customer Zip	Zip code of the customer	ZIPCODE
✓	Style	Style of shoe sold	TEXT

● Extract ○ Derive

Source: Item □ **Multi-Valued**

Path: XPath //*[local-name()='style']

Test Message

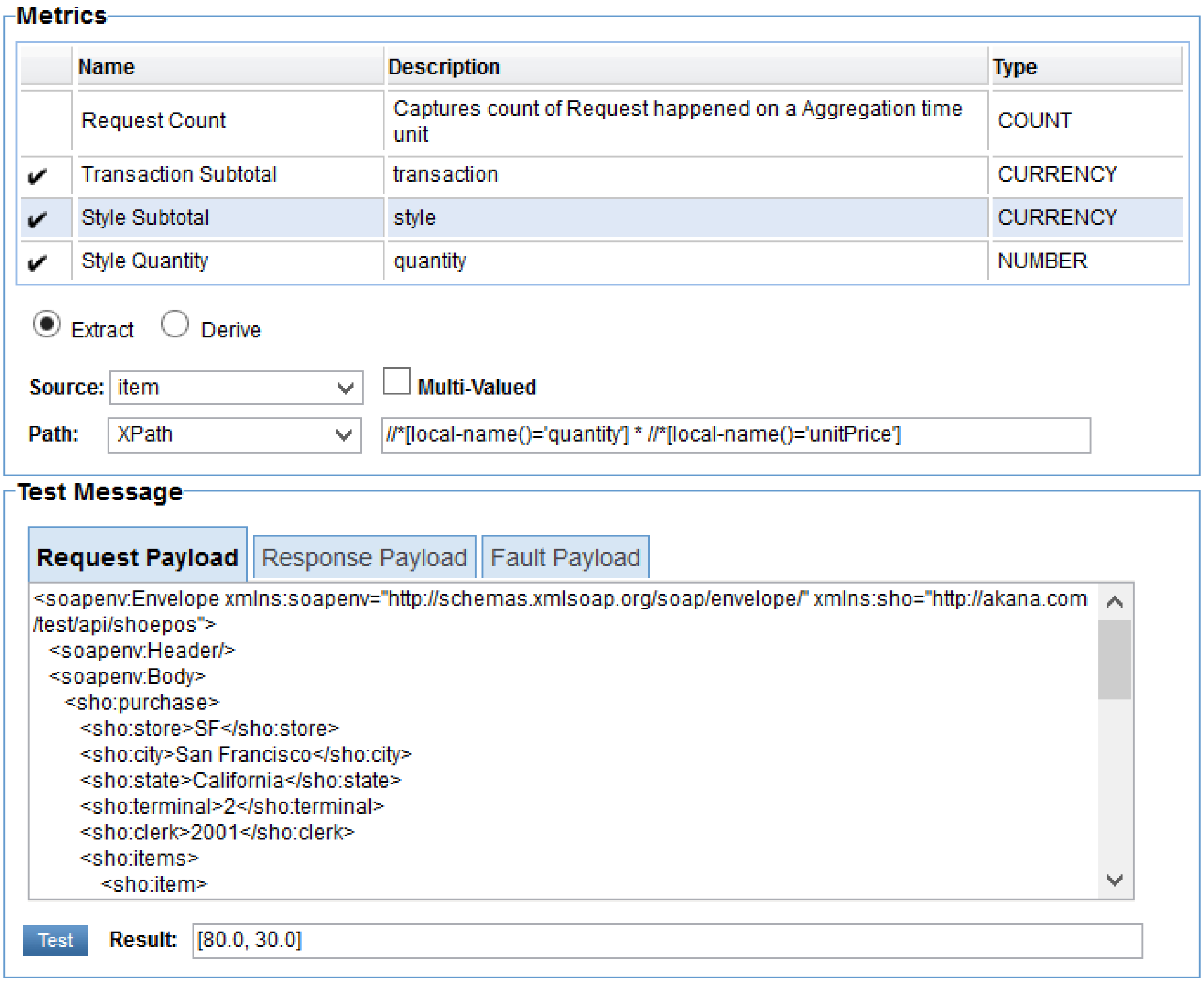
Message

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Header/>
  <soapenv:Body>
    <sho:purchase xmlns:sho="http://acme.com/api/shoepos">
      <sho:store>LA</sho:store>
      <sho:terminal>1</sho:terminal>
      <sho:clerk>1001</sho:clerk>
      <sho:items>
        <sho:item>
          <sho:style>mid</sho:style>
          <sho:quantity>1</sho:quantity>
          <sho:unitPrice>40.00</sho:unitPrice>
```

Result: [mid , low]

The "Style" dimension is in the Item group so it's XPath expression is evaluated relative to each of the shoe:item elements in the message. When tested against our sample two styles are returned.

The metrics are defined in a similar way to the dimensions.



Metrics

	Name	Description	Type
	Request Count	Captures count of Request happened on a Aggregation time unit	COUNT
✔	Transaction Subtotal	transaction	CURRENCY
✔	Style Subtotal	style	CURRENCY
✔	Style Quantity	quantity	NUMBER

● Extract ○ Derive

Source: item □ **Multi-Valued**

Path: XPath //*[local-name()='quantity'] * //*[local-name()='unitPrice']

Test Message

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:sho="http://akana.com
/test/api/shoepos">
  <soapenv:Header/>
  <soapenv:Body>
    <sho:purchase>
      <sho:store>SF</sho:store>
      <sho:city>San Francisco</sho:city>
      <sho:state>California</sho:state>
      <sho:terminal>2</sho:terminal>
      <sho:clerk>2001</sho:clerk>
      <sho:items>
         <sho:item>
```

Result: [80.0, 30.0]

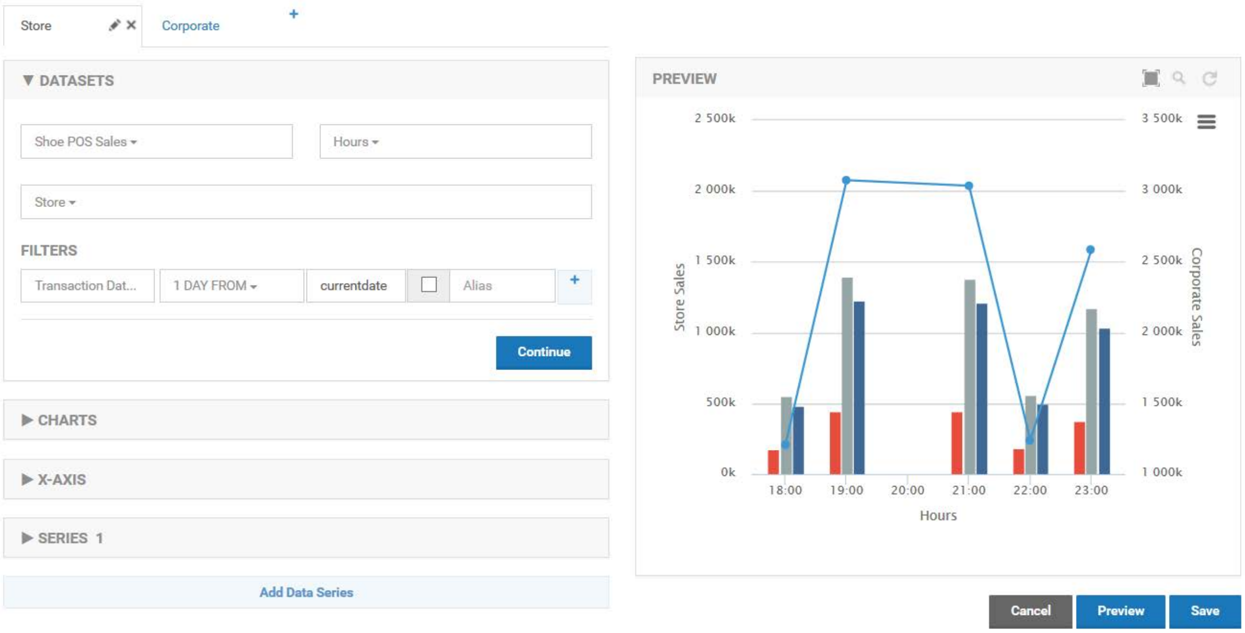
The "Style Subtotal" metric poses a challenge because there is not a single element in the message that provides this number. It is calculated by multiplying the sho:quantity by the sho:unitPrice. When testing the expression both subtotals are shown. During actual execution of the policy these subtotals, along with the "Style Quantity" metric expression results, will be grouped together in order to not lose their correlation. So instead of three separate arrays for style, quantity, and price there will be a single array of structures where each structure will have a style, quantity, and price.

With a Data Set and policy now defined, once the policy is attached to the POS API in Policy Manager the Network Director can now begin capturing and storing metrics. The data will be aggregated according to the Data Set settings.

The Charts

ACME has created charts to visualize each of the measurements stated in the use case introduction. In a real world scenario their charts may be tracking data by the month but for in these examples the charts will track by hours so that users trying to use the examples as learning tools can see faster results. A filter is applied to all charts that will only display the last days' worth of data.

The first chart is a combination chart that displays company wide sales and the sales for each store.



Shoe POS Sales ▾ | Hours ▾

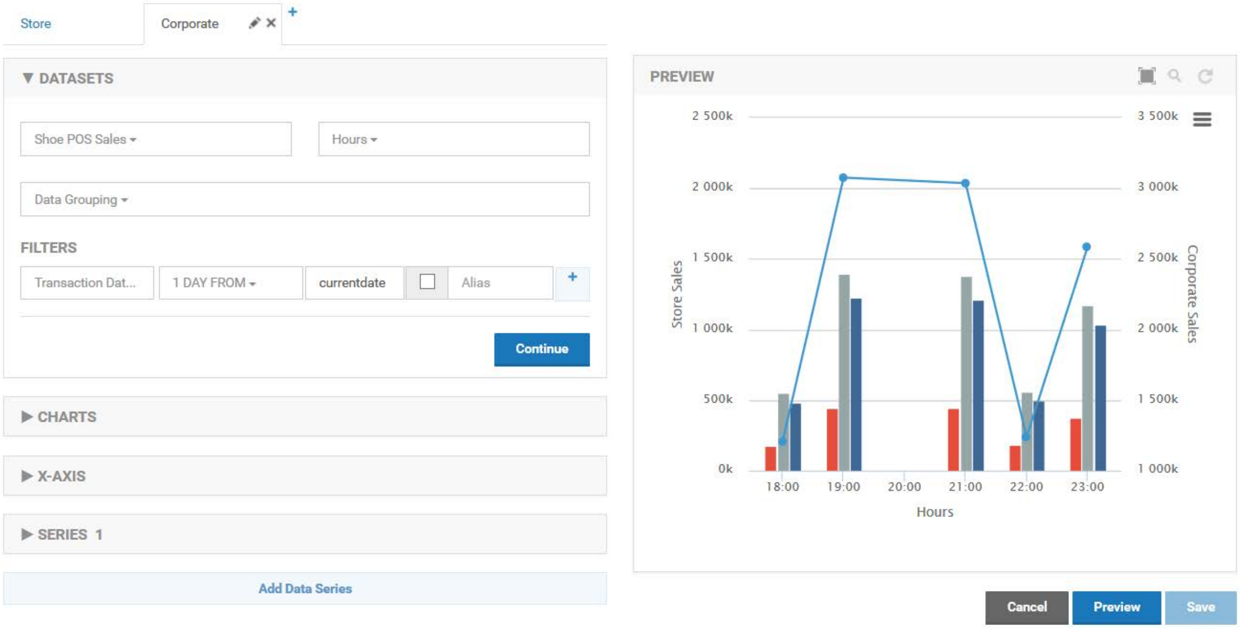
Store ▾

FILTERS

Transaction Dat... | 1 DAY FROM ▾ | currentdate | □ | Alias

The first chart in the combination is a standard column chart with a column per store representing the total sales for a given hour. The second chart is a standard line chart with a single line reflecting the sum of all transactions made using the POS API. There is no Drilldown capability defined so both charts will be displayed at the same time. The column chart is defined first so that the line of the line chart will be layered on top.

Both charts in the combination use the Shoe POS Sales Data Set as their source with the Hours *Duration*. The store chart uses the Store Dimension for *Data Grouping*. The company chart has no grouping as the measurements should be summed without breaking them down in any way.



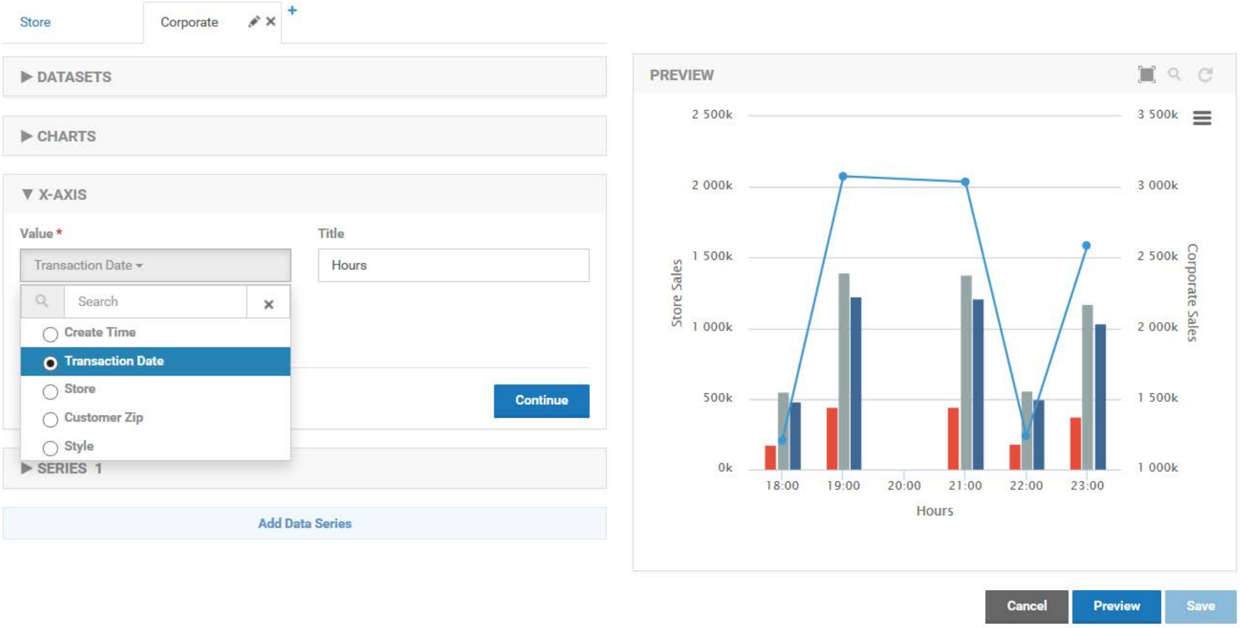
Shoe POS Sales ▾ | Hours ▾

Data Grouping ▾

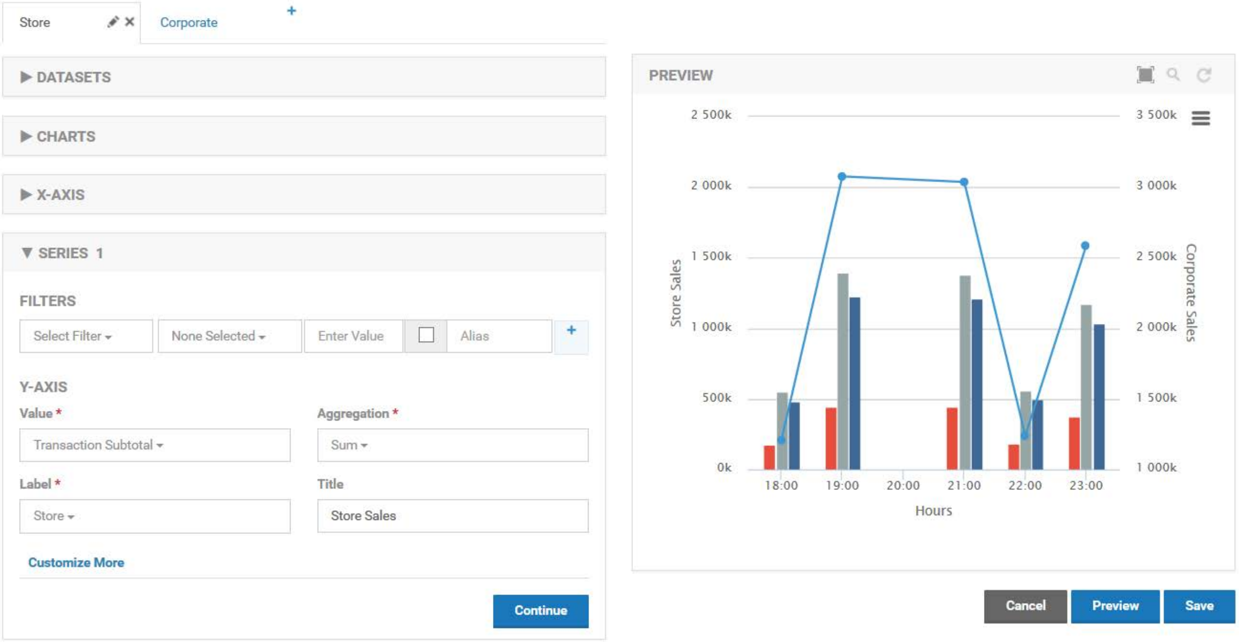
FILTERS

Transaction Dat... | 1 DAY FROM ▾ | currentdate | □ | Alias

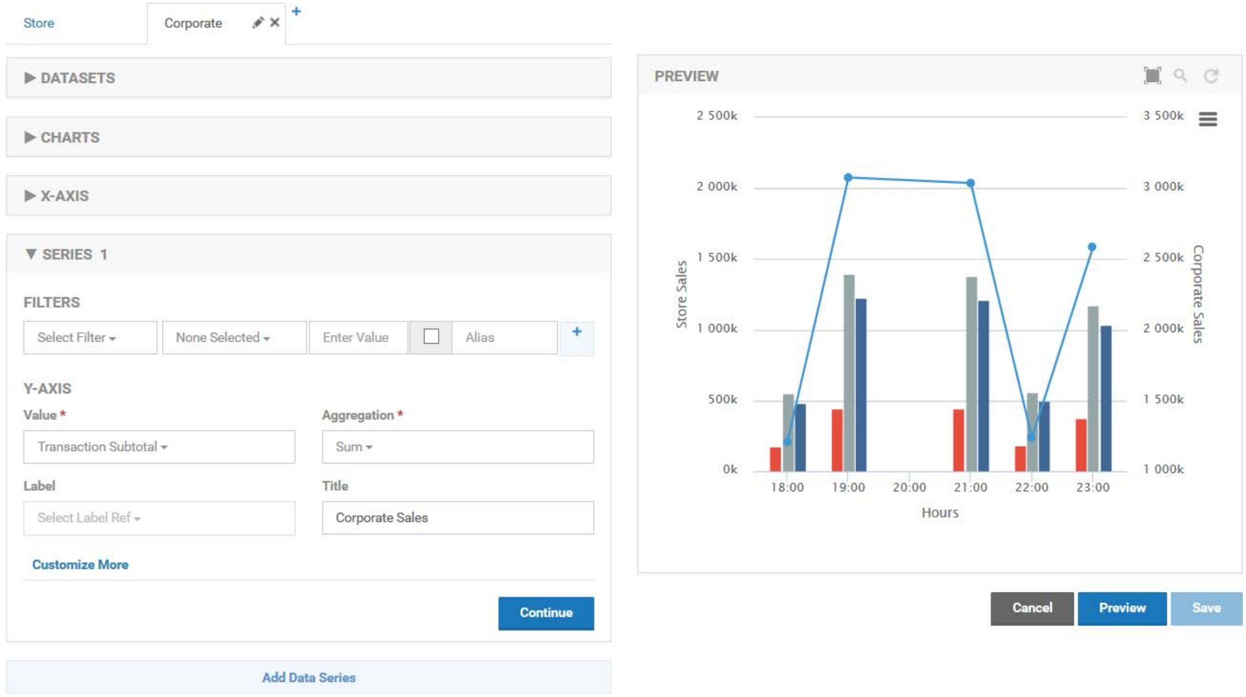
The measurements will be displayed over time. The X-Axis *Value Ref* will therefore use a date field. The Transaction Date is used as opposed to the Create Time to be more reflective of the time at the terminals.



The data series for store sales uses the Transaction Subtotal metric for the Y-Axis *Metric*. The *Aggregation Operator* is Sum. The *Label Ref* is the Store dimension so there will be a separate column for each store's subtotal.



The data series for corporate sales also uses the Transaction Subtotal metric for the Y-Axis *Metric*. The *Aggregation Operator* is Sum as well. Since there is no *Data Grouping* there is no selection for *Label Ref* and there is only one line drawn for the subtotal.



Y-AXIS

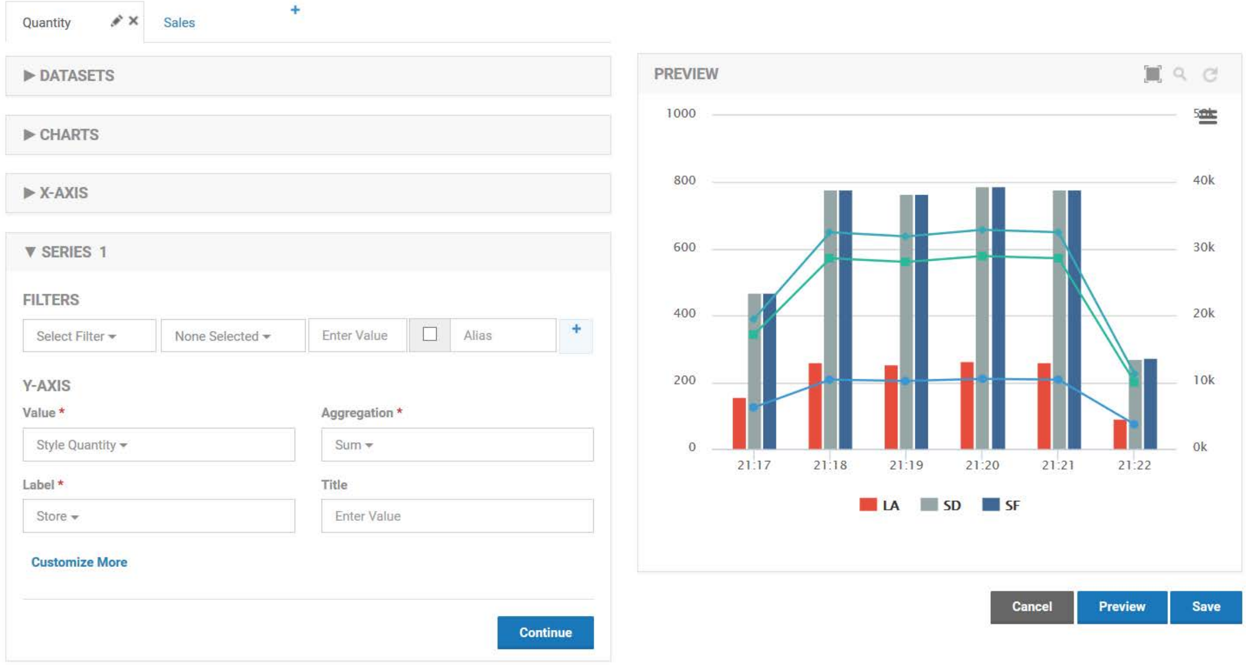
Value * Transaction Subtotal

Aggregation * Sum

Label: Select Label Ref

Title: Corporate Sales

The second chart is a combination chart that displays the quantity sold by store, by style, and by style for a given store. The first chart in the combination is a standard column chart with a column per store representing the total sales for a given hour.



Y-AXIS

Value * Style Quantity

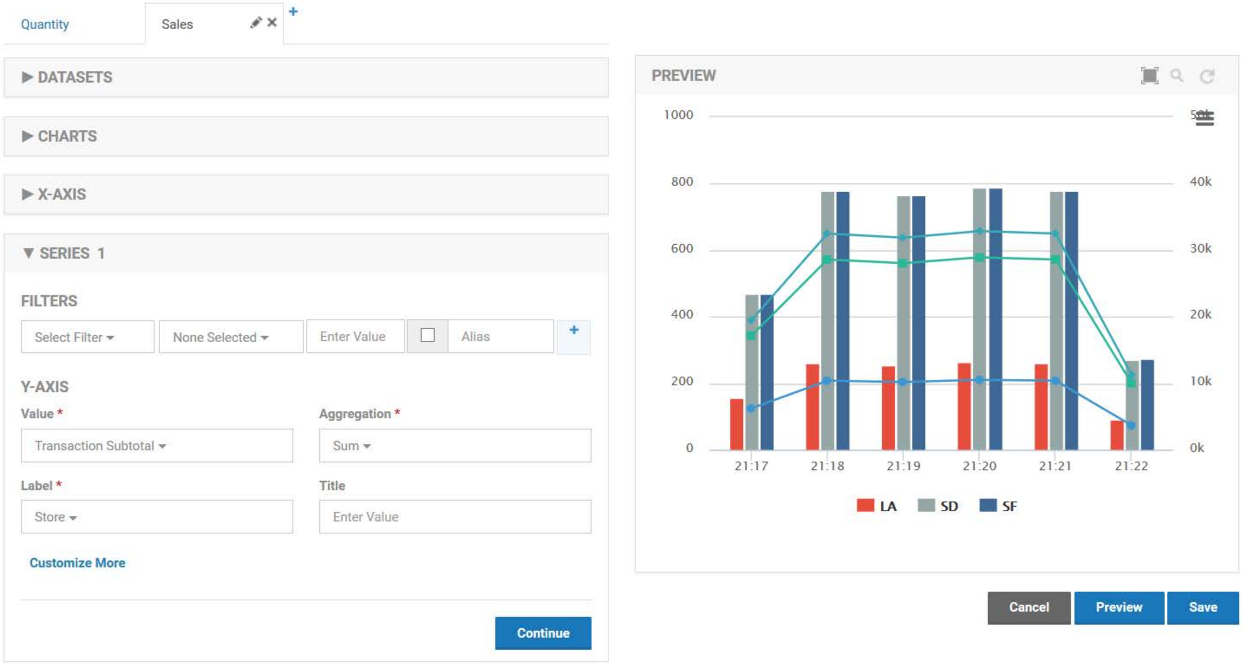
Aggregation * Sum

Label * Store

Title: Enter Value

The second chart is a standard line chart with a single line reflecting the sum of all transactions made using the POS API. There is no Drilldown capability defined so both charts will be displayed at the same time. The column chart is defined first so that the line of the line chart will be layered on top.

Both charts in the combination use the Shoe POS Sales Data Set as their source with the Hours *Duration*. The store chart uses the Store Dimension for *Data Grouping*. The company chart has no grouping as the measurements should be summed without breaking them down in any way.


Quantity | Sales

▶ DATASETS

▶ CHARTS

▶ X-AXIS

▼ SERIES 1

FILTERS

Select Filter ▾ | None Selected ▾ | Enter Value | □ | Alias | +

Y-AXIS

Value *: Transaction Subtotal ▾

Aggregation *: Sum ▾

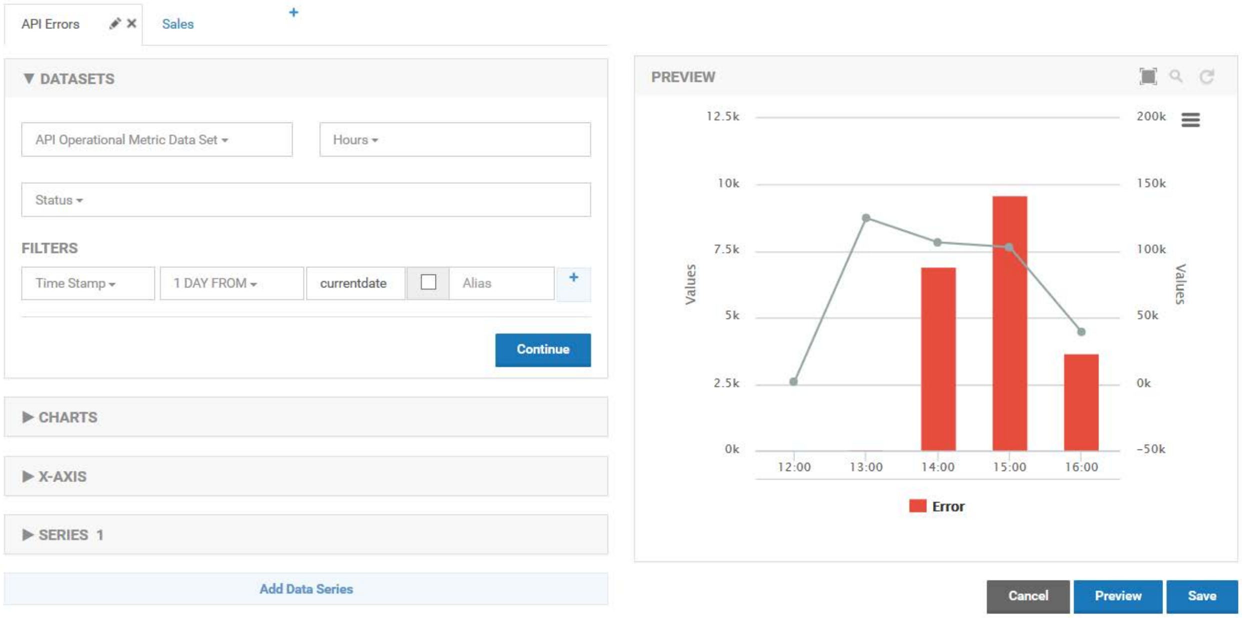
Label *: Store ▾

Title: Enter Value

Customize More

Continue

The third chart is a combination chart that displays the quantity sold company wide and overlays the number of errors that have been reported by the API. It provides a quantitative impact of errors on the business. The first chart in the combination is a standard column chart with a column for the number of errors the API encountered. It uses the API Operational Metric Data Set to graph errors.


API Errors | Sales

▼ DATASETS

API Operational Metric Data Set ▾ | Hours ▾

Status ▾

FILTERS

Time Stamp ▾ | 1 DAY FROM ▾ | currentdate | □ | Alias | +

Continue

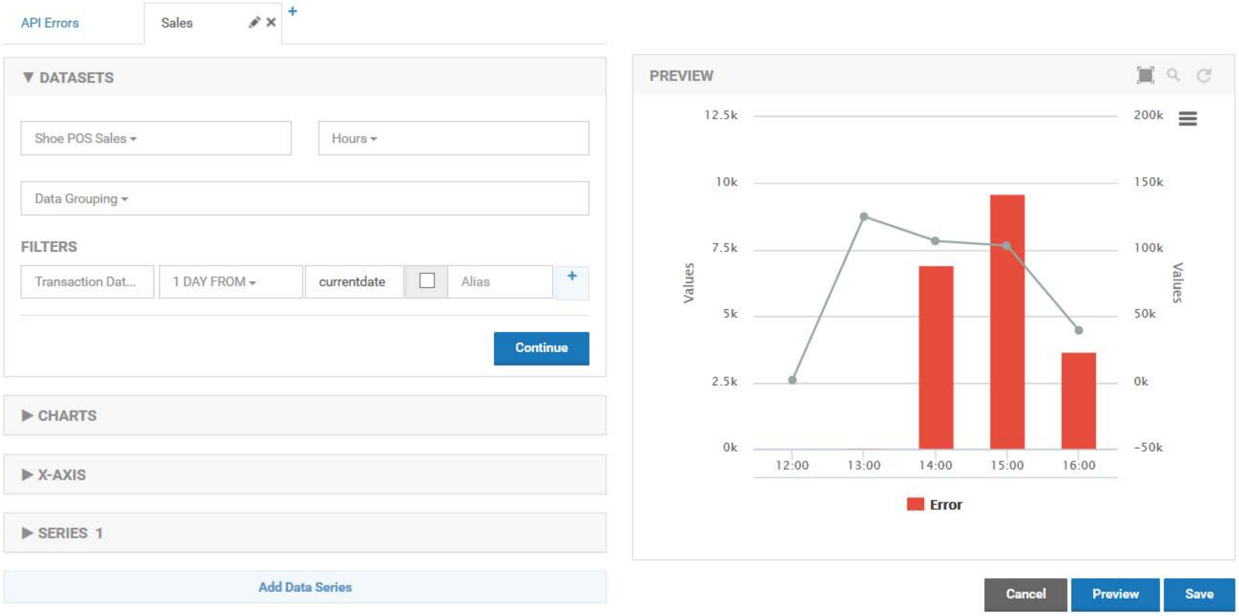
▶ CHARTS

▶ X-AXIS

▶ SERIES 1

Add Data Series

The second chart in the combination is a standard line chart with a line for the total quantity sold over time. It uses the Shoe POS Sales data set for the quantity. The X-Axis of the charts use different dimensions, Time Stamp vs Transaction Date, but as long as they have values that align the graphs overlay perfectly.



The Dashboard

ACME has created a single Dashboard that will display all the POS charts on a single page.