

Oracle® Fusion Middleware

Oracle API Gateway Key Property Store User Guide
11g Release 2 (11.1.2.4.0)

July 2015

Copyright 1999, 2015, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this software or related documentation is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications which may create a risk of personal injury. If you use this software in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of this software. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software in dangerous applications.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

This software and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services. This documentation is in prerelease status and is intended for demonstration and preliminary use only. It may not be specific to the hardware on which you are using the software. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to this documentation and will not be responsible for any loss, costs, or damages incurred due to the use of this documentation.

The information contained in this document is for informational sharing purposes only and should be considered in your capacity as a customer advisory board member or pursuant to your beta trial agreement only. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described in this document remains at the sole discretion of Oracle.

This document in any form, software or printed matter, contains proprietary information that is the exclusive property of Oracle. Your access to and use of this confidential material is subject to the terms and conditions of your Oracle Software License and Service Agreement, which has been executed and with which you agree to comply. This document and information contained herein may not be disclosed, copied, reproduced, or distributed to anyone outside Oracle without prior written consent of Oracle. This document is not part of your license agreement nor can it be incorporated into any contractual agreement with Oracle or its subsidiaries or affiliates.

Contents

Preface	6
Who should read this document	6
How to use this document	6
1 Introduction to KPS	8
KPS architecture	8
KPS data stores	9
KPS client applications	9
When to use a KPS	9
2 Get started with KPS	10
Example KPS table	10
Before you begin	11
Define KPS configuration with Policy Studio	11
Step 1: Define where data will be stored	11
Step 2: Define the KPS table	12
Step 3: Define a policy that accesses the table	14
Step 4: Deploy the configuration	16
Add KPS data using API Gateway Manager	17
Access KPS data from a policy	18
Enable API Gateway tracing	19
3 Configure KPS in Policy Studio	20
Configure a KPS collection	20
Configure a KPS table	21
KPS aliases	22
KPS data sources	22
KPS table structure	22
Query tables using properties and keys	23
Primary key	23
Secondary key	24
Auto-generated properties	24
Encrypted properties	24
4 Access KPS data using selectors	26
KPS selector syntax	26
KPS selector examples	27

5 Manage a KPS using kpsadmin	28
Start kpsadmin	28
Start in verbose mode	28
Select kpsadmin operations	28
Table operations	29
Table administration operations	29
Collection administration operations	30
Cassandra administration operations	31
General administration operations	31
Example switching data source	31
Step 1: Backup collection data using kpsadmin	31
Step 2: Create a new data source	32
Step 3: Deploy the configuration	33
Step 4: Restore collection data using kpsadmin	33
6 Configure Apache Cassandra KPS storage	34
Cassandra configuration	34
Cassandra ports	35
cassandra.yaml	35
client.yaml	35
jvm.xml	36
Secure ports	36
Cassandra logging	36
Cassandra configuration steps	37
Step 1: Configure the Cassandra topology	37
Step 2: Start API Gateways with embedded Cassandra servers	43
Step 3: Configure the Cassandra replication factor	43
Step 4: Configure request consistency levels	45
Disable Cassandra storage	45
7 Configure database KPS storage	47
Shared database storage	47
Step 1: Create a KPS database table	47
Step 2: Set up an external connection to the database	48
Step 3: Use the external connection in a KPS collection	48
Per-table database storage	51
Map a database table using a single key	51
How to map a database table using a composite key	56
8 Configure file-based KPS storage	59
Configure a file-based KPS collection	59
Appendix A: KPS FAQ	60
KPS and API Gateway	60

What is KPS used for in API Gateway?	60
What is KPS not suitable for?	60
What are the transaction semantics of KPS?	60
What is the KPS collection alias prefix for?	60
Why are some collections hidden? How can you show them?	61
How do I change API Gateway group passphrase?	61
KPS storage options	62
Is Apache Cassandra storage required? Can you use file or database?	62
How do you switch storage for a KPS collection?	62
Why use database storage?	62
Why use file storage?	62
When can you use kpsadmin? When should you use storage-specific tools?	63
Apache Cassandra	63
Why use Cassandra as a KPS storage option?	63
What version of Cassandra does the API Gateway use?	63
What does all host polls marked down mean?	63
Can you use an external Cassandra instance?	63
How do you set Cassandra consistency levels?	64
How do you disable Cassandra?	64
Appendix B: Troubleshoot KPS error messages	65
All platforms	65
All host polls marked down	65
Nodetool reports failed to connect	65
May not be enough replicas present to handle consistency level	66
Windows only	66
Node running in client mode	66
UTF-8 characters not displaying correctly in kpsadmin	67
FSUTIL utility bug	67
Path length issue >255 characters	67
Could not drop kps keyspace using cassandra-cli	68
Appendix C: Apache Cassandra operations for API Gateway	69
nodetool repair	69
Backup and recovery	69
Back up and restore Cassandra node data	69
Back up API Gateway KPS configuration	70
Back up API Gateway Cassandra configuration and data	70
Replace dead node	70
Further information	72
Glossary	73

Preface

This document describes how to configure and manage the API Gateway Key Property Store (KPS). The KPS enables you to manage API Gateway data referenced from policies running on the API Gateway.

Who should read this document

The intended audience for this document is KPS administrators and policy developers. For more details on API Gateway user roles, see the *API Gateway Concepts Guide*. This document assumes that you are familiar with the following:

- Database concepts such as tables, rows, and keys
- API Gateway configuration and deployment
- API Gateway selectors
- Using command line tools
- Database configuration where database storage is required

For more details on API Gateway configuration and selectors, see the *API Gateway Policy Developer Guide*.

How to use this document

This document should be used with the other documents in the API Gateway documentation set. Before you begin, review this document thoroughly. The following is a brief description of the contents of each chapter:

- [Introduction to KPS on page 8](#) provides an overview of the KPS architecture and features.
- [Get started with KPS on page 10](#) explains how to develop an example KPS table for managing simple user information.
- [Configure KPS in Policy Studio on page 20](#) provides more detail on how to define general KPS configuration using the Policy Studio graphical tool.
- [Access KPS data using selectors on page 26](#) explains how to access data in policies on the API Gateway at runtime.
- [Manage a KPS using kpsadmin on page 28](#) explains how to manage a KPS, independent of data source.
- [Configure Apache Cassandra KPS storage on page 34](#) explains how to store KPS data in the default Cassandra server embedded in the API Gateway.

- [Configure database KPS storage on page 47](#) explains how to store KPS data in a relational database (for example, Oracle, MySQL, IBM DB2, or Microsoft SQL Server).
- [Configure file-based KPS storage on page 59](#) explains how to store KPS data in a directory on the file system.

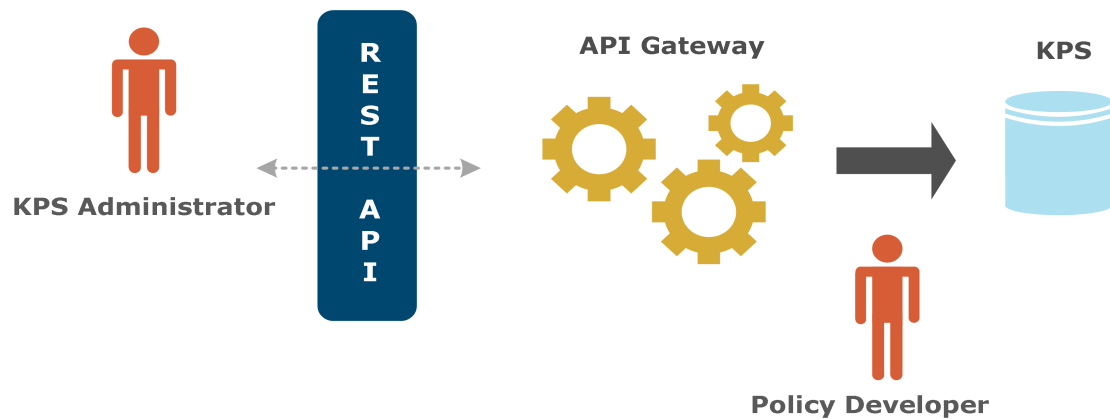
Introduction to KPS

1

A Key Property Store (KPS) is a table of data that can be referenced from policies running on the API Gateway. Data in a KPS table is assumed to be read frequently and seldom written, and can be changed without incurring an API Gateway service outage. KPS tables are shared across an API Gateway group.

KPS architecture

The following diagram shows a simple role-based architecture:



A KPS is typically used to store property values that are used in policies running on an API Gateway. KPS data is injected into policies using *selectors* that are first created in Policy Studio by *policy developers*. Selectors are evaluated and expanded dynamically at runtime. For example, a KPS table could contain authorization tokens for different users. A policy could look up the token for the current user and insert it into an HTTP request.

KPS tables are organized into *collections*. The tables in a collection typically have some sort of relationship to one another. For example, the OAuth collection contains a set of tables that store all OAuth-related data. Every KPS table is assigned an alias so that it can be easily referred to in a policy or a REST request. KPS collections and tables can be created by policy developers using Policy Studio.

KPS administrators can use the API Gateway Manager web console to view and modify KPS data at runtime. This is a business or operational role that manages dynamic policy configuration data in a KPS (for example, customer details, authorization levels, or quotas). This means that this information does not need to be configured at design time by policy developers.

For more details on API Gateway architecture, components, and roles, see the *API Gateway Concepts Guide*.

KPS data stores

KPS data can be stored in one of the following locations:

- Embedded Apache Cassandra database: Data can be distributed across multiple nodes to provide high availability (this is the default KPS data store).
- Relational database: Accessible to all API Gateway instances in the API Gateway group.
- JSON files: On the local file system.

KPS client applications

API Gateway provides the following client applications:

- Policy Studio: Enables policy developers to create KPS collections and tables, and to configure data sources.
- API Gateway Manager: Includes a visual web-based interface to enable KPS administrators to view and modify KPS data at runtime.
- `kpsadmin` command: Supports KPS data entry and other administrative functions. It is designed for use in a development environment.
- KPS REST API: Enables remote programmatic clients to read and write KPS data.

When to use a KPS

KPS provides a flexible data storage service that can be used to store any configuration data. Its primary function is to make this data available to selectors at runtime, and it is optimized for this purpose. This makes it most suitable for data with the following characteristics:

- Data is common to all API Gateways in an API Gateway group. KPS is not suitable for data that is specific to one particular API Gateway.
- The data schema is relatively simple. Each KPS table is assumed to be independent of all others, and referential integrity across tables is not supported.
- Data can change while API Gateways are running. Updating Cassandra-backed or database-backed KPS tables does not require an API Gateway restart. Changeable data should therefore be stored in KPS instead of hard-coded into policies.
- Queries always involve looking up a key value in a table to retrieve a single object. This is the usage model supported by selectors. Ad hoc queries that involve searching for non-key properties are not supported.
- Multi-operation transactions are not required. Each read or write to a KPS table is considered a standalone operation. Locking or rollback across multiple operations are not supported.

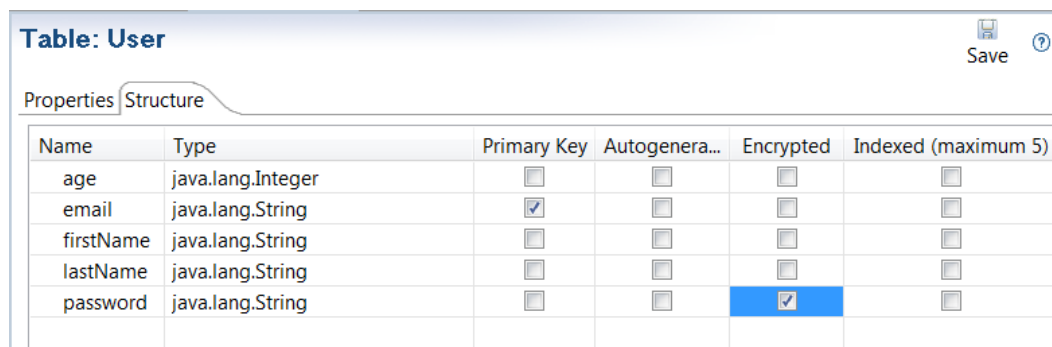
Get started with KPS

2

This topic explains how to develop an example KPS table for managing simple user information.

Example KPS table

The final structure of the example table is displayed in Policy Studio as follows:



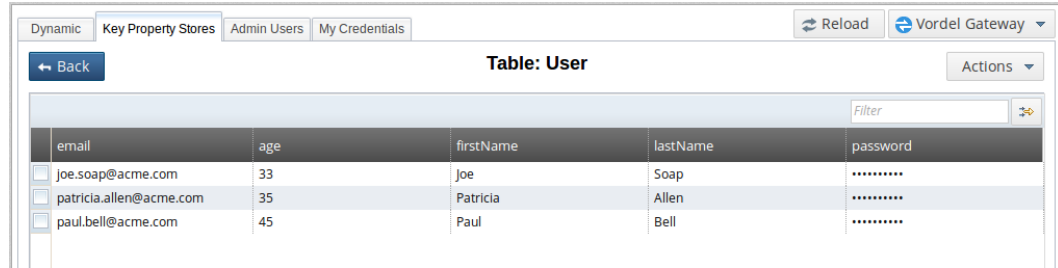
The screenshot shows the 'Table: User' configuration in Policy Studio. It has two tabs: 'Properties' and 'Structure'. The 'Structure' tab is active, displaying a table with the following columns: Name, Type, Primary Key, Autogenera..., Encrypted, and Indexed (maximum 5). The rows are: age (java.lang.Integer), email (java.lang.String), firstName (java.lang.String), lastName (java.lang.String), and password (java.lang.String). The 'email' row has the 'Primary Key' checkbox checked, and the 'password' row has the 'Encrypted' checkbox checked.

Name	Type	Primary Key	Autogenera...	Encrypted	Indexed (maximum 5)
age	java.lang.Integer	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
email	java.lang.String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
firstName	java.lang.String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
lastName	java.lang.String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
password	java.lang.String	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

This table structure is described as follows:

Column	Type	Description
age	java.lang.Integer	User age.
email	java.lang.String	User email address. This is selected in Policy Studio as a unique Primary Key , which is indexed implicitly by default.
firstName	java.lang.String	User first name.
lastName	java.lang.String	User surname.
password	java.lang.String	User password, which is selected as Encrypted in Policy Studio.

Example table data is displayed on the **Settings > Key Property Stores** tab in the API Gateway Manager web console:



The screenshot shows the Vordel Gateway Admin console interface. At the top, there are tabs for 'Dynamic', 'Key Property Stores', 'Admin Users', and 'My Credentials'. A 'Reload' button and a 'Vordel Gateway' dropdown menu are also visible. Below the tabs, there is a 'Back' button and a title 'Table: User'. A table with five columns (email, age, firstName, lastName, password) displays three rows of user data. A 'Filter' input field and an 'Actions' dropdown menu are located at the top right of the table.

email	age	firstName	lastName	password
joe.soap@acme.com	33	Joe	Soap	*****
patricia.allen@acme.com	35	Patricia	Allen	*****
paul.bell@acme.com	45	Paul	Bell	*****

Before you begin

The following prerequisite steps apply to this example:

1. Ensure that an API Gateway and an Admin Node Manager are running.
2. Start Policy Studio, and connect to the Admin Node Manager.

For more details, see the *API Gateway Installation Guide*.

Define KPS configuration with Policy Studio

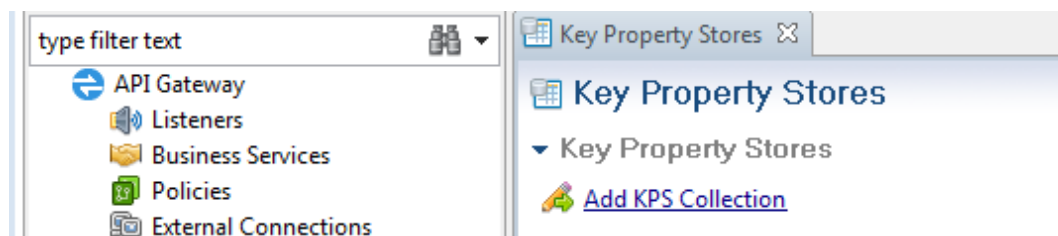
The main steps for configuring KPS tables in Policy Studio are as follows:

1. Define where the data will be stored.
2. Define the KPS table.
3. Define a policy that accesses the table.
4. Deploy the configuration.

Step 1: Define where data will be stored

You must first create a KPS collection in which to store the KPS table configuration. Perform the following steps:

1. In the Policy Studio tree, select **Key Property Stores**, and select **Add KPS Collection**.



2. In the **Add KPS Collection** dialog, name the collection **Samples**.

i A KPS Collection represents a grouping of KPS Tables

Name

Description

Alias prefix

Default datasource

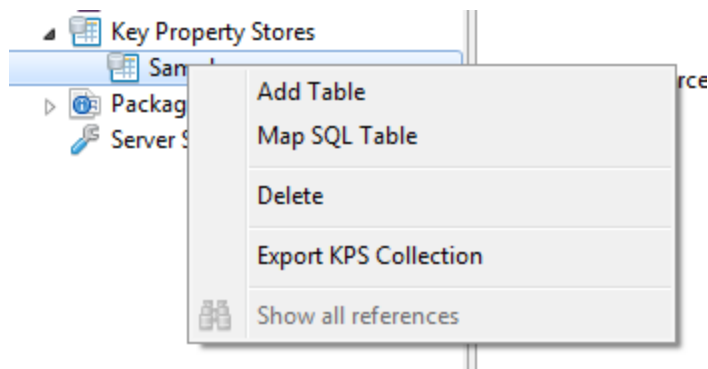
3. Select a **Default data source** of **Embedded (Cassandra)** for all tables in the collection.

Note Leave the **Alias prefix** field blank.

Step 2: Define the KPS table

To create a table, perform the following steps:

1. In the Policy Studio tree, right-click the newly-created **Samples** collection, and select **Add Table**:



2. In the dialog, enter a **Name** of `User`, and provide a **Description**.
3. Click **Add** to assign an alias of **User** to this table. A table must have at least one alias.

Name

Description

Aliases
User

4. Next define the table structure. This consists of the table columns and the data type stored in each column. Select the **User** table and **Structure** tab, and click **Add**:

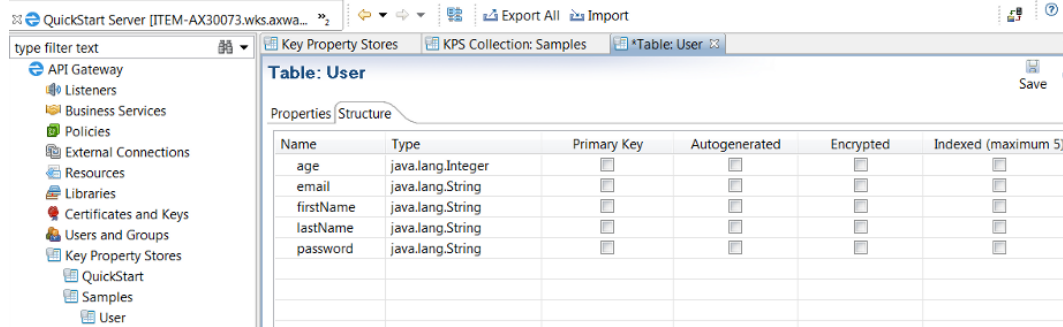
The screenshot shows the Oracle API Gateway console interface. On the left is a tree view of the project structure, with 'User' selected under 'Key Property Stores'. The main area displays the 'Table: User' configuration. The 'Structure' tab is selected, showing a table with columns: Name, Type, Primary Key, Autogenerated, and Encrypted. An 'Add' button is located at the bottom right of the table area.

5. Repeat to add the following columns for your table structure in the **Add Property** dialog:

- email
- password
- firstName
- lastName
- age

Note age has an `Integer` (numeric) **Type**. All the other columns are `String`.

5. When you select the **User** table, you should have the following structure:



- You want the **email** field to be the primary key for the table, so select **Primary Key** for this field.

Name	Type	Primary Key
age	java.lang.Integer	<input type="checkbox"/>
email	java.lang.String	<input checked="" type="checkbox"/>
firstName	java.lang.String	<input type="checkbox"/>

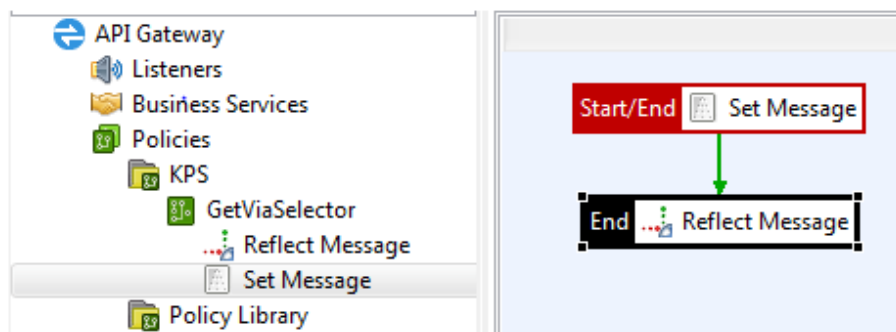
- You want the **password** field to be encrypted when stored in the data source, so select **Encrypted** for this field.

Name	Type	Primary Key	Autogenerated	Encrypted
age	java.lang.Integer	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
email	java.lang.String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
firstName	java.lang.String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
lastName	java.lang.String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
password	java.lang.String	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Step 3: Define a policy that accesses the table

To define a test policy that accesses the table, perform the following steps:

- Add a test policy with a **Set Message** filter from the **Conversion** filter category.



2. Right-click the filter, and set it as the **Start** filter for the policy.
3. Enter a filter **Content-Type** of `text/plain`.
4. Enter the following **Message Body** for use in the policy:

```

=====
User
===
Email: ${kps.User[http.querystring.id].email}
First Name: ${kps.User[http.querystring.id].firstName}
Last Name: ${kps.User[http.querystring.id].lastName}
Age: ${kps.User[http.querystring.id].age}
=====

```

These settings are displayed as follows in the **Set Message** filter:

Set the Message

Change the contents of the message body.



Name:

Content-Type:

Message Body: Populate ▼

```

=====
User
===
Email: ${kps.User[http.querystring.id].email}
First Name: ${kps.User[http.querystring.id].firstName}
Last Name: ${kps.User[http.querystring.id].lastName}
Age: ${kps.User[http.querystring.id].age}
=====

```

The message body value are specified using selectors, which are evaluated and expanded dynamically at runtime. For example, the user age is specified using the following selector string:

```
${kps.User[http.querystring.id].age}
```

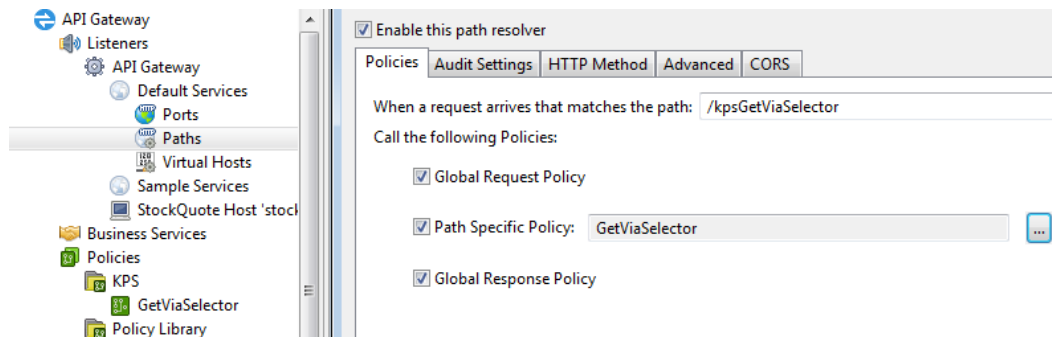
The selector parts are explained as follows:

Selector part	Description
<code>\${</code>	Indicates the start of the selector using a { bracket.

Selector part	Description
<code>kps</code>	Specifies that selector should query a KPS table.
<code>.User</code>	Specifies the alias of the table to query (in this case, <code>User</code>).
<code>[</code>	Indicates the start of a table property reference using a <code>[</code> bracket.
<code>http.querystring.id</code>	This is a dynamic query based on an HTTP query string parameter of <code>id</code> . The primary key value is retrieved from this parameter. The row with this key value is returned from the <code>User</code> table if it exists.
<code>]</code>	Indicates the end of a table property reference using a <code>]</code> bracket.
<code>.age</code>	Retrieves the <code>age</code> column.
<code>}</code>	Indicates the end of the selector using a <code>}</code> bracket.

Note Add a **Reflect Message** filter (**Conversion** category) to return a successful HTTP response status of 200.

5. Set up a path to this policy. In this example, the path is `/kpsGetViaSelector`:



Step 4: Deploy the configuration

When you are finished with your configuration changes, you must deploy them to the API Gateway. To deploy the new configuration, click **Deploy** in the Policy Studio toolbar:



This pushes the configuration to the API Gateway group.

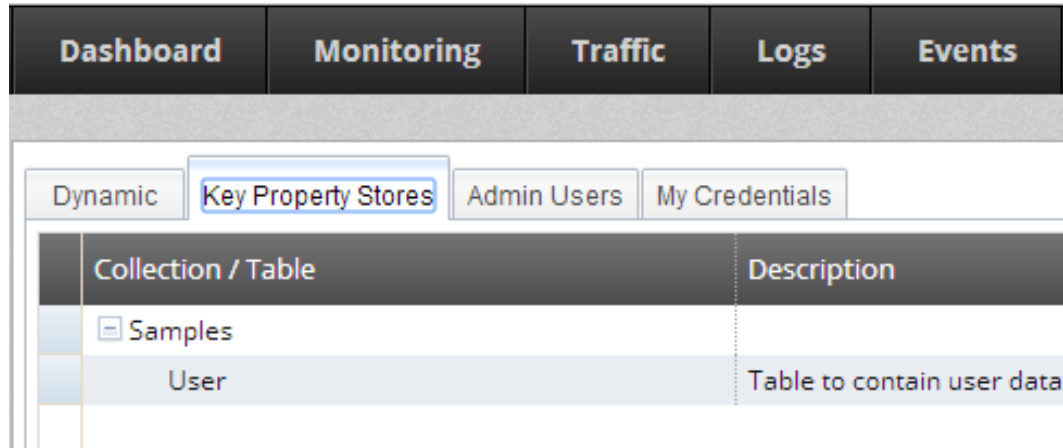
Tip If you deploy an incorrect configuration (for example, specify an incorrect primary key, property type, or name) you can use the `kpsadmin` command to drop the table in storage. For more details, see [Manage a KPS using kpsadmin on page 28](#).

Add KPS data using API Gateway Manager

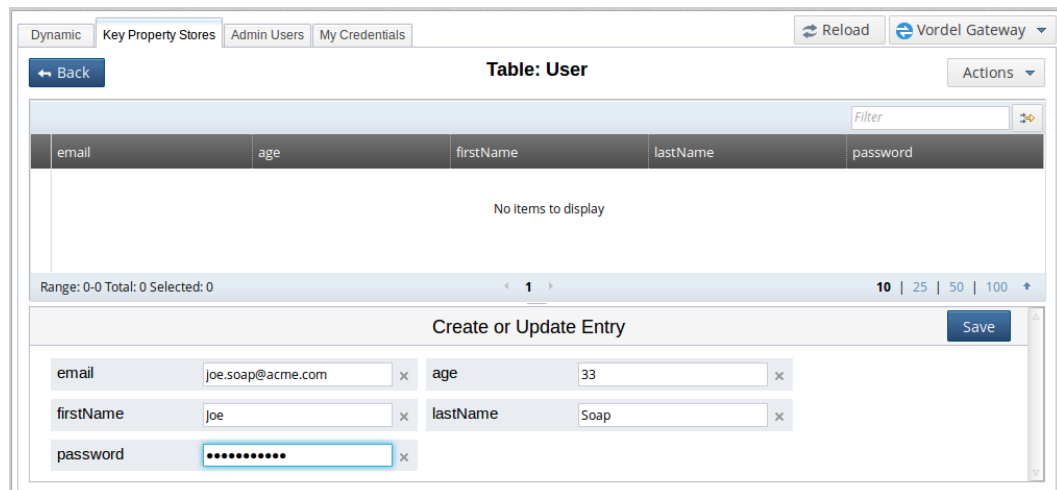
You can use API Gateway Manager to populate the `User` table with data.

Perform the following steps:

1. To access API Gateway Manager in your browser, go to `https://localhost:8090`.
2. Select the **Settings** > **Key Property Stores** tab.
3. Select the **Samples** > **User** table.



4. To enter new records, select **Actions** > **New Entry**:



5. Click **Save** to save a record.

For example, the table should look as follows:

email	age	firstName	lastName	password
<input type="checkbox"/> joe.soap@acme.com	33	Joe	Soap	*****
<input type="checkbox"/> patricia.allen@acme.com	35	Patricia	Allen	*****
<input type="checkbox"/> paul.bell@acme.com	45	Paul	Bell	*****

Access KPS data from a policy

For example, to access KPS data from a policy, go to the following URL in your browser:

```
http://localhost:8080/kpsGetViaSelector?id=patricia.allen@acme.com
```

This URL specifies the user ID (email) as `patricia.allen@acme.com`

You must specify an email that exists in your data. For example:

```

=====
User
===
Email: patricia.allen@acme.com
First Name: Patricia
Last Name: Allen
Age: 35
=====

```

Note If you enter an email that does not exist, you will see `[invalid field]` results. For example:

```

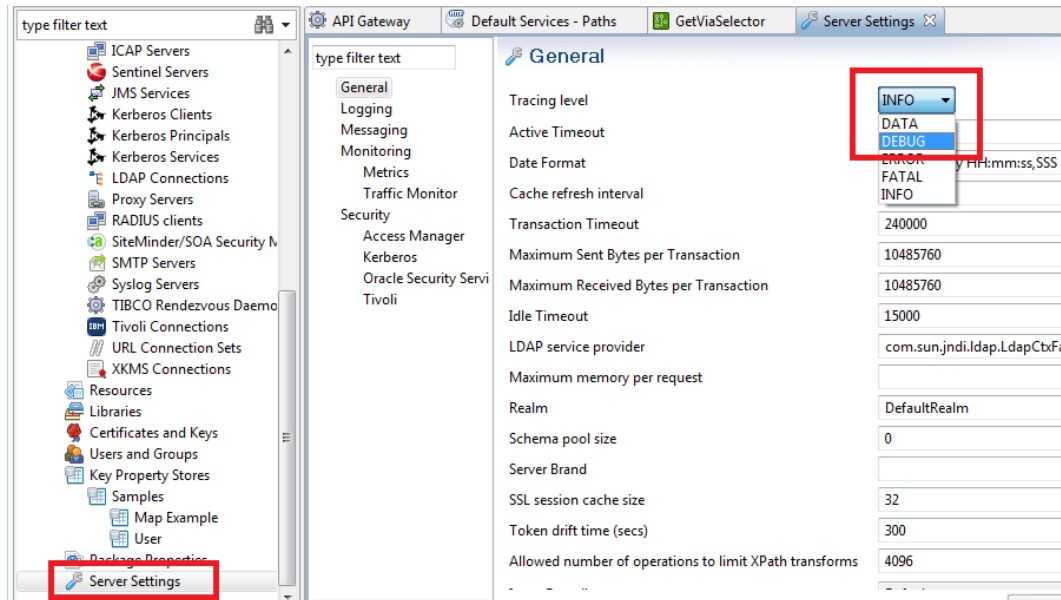
=====
User
===
Email: [invalid field]
First Name: [invalid field]
Last Name: [invalid field]
Age: [invalid field]
=====

```

Enable API Gateway tracing

To enable API Gateway debug tracing, perform the following steps in Policy Studio:

1. In the tree on the left, select **Server Settings > General**.
2. Select a **Tracing level** of **DEBUG**.
3. Click **Save**.
4. Click **Deploy**.



Note This setting enables debug tracing for the entire API Gateway, and not just for the KPS.

For more details on API Gateway tracing and logging, see the *API Gateway Administrator Guide*.

Configure KPS in Policy Studio

3

This topic describes how to define general KPS configuration in Policy Studio. For details on data source-specific configuration, see the following topics:

- [Configure Apache Cassandra KPS storage on page 34](#)
- [Configure database KPS storage on page 47](#)
- [Configure file-based KPS storage on page 59](#)

Configure a KPS collection

A KPS collection is a set of related KPS tables. To configure a KPS collection, perform the following steps:

1. Right-click **Key Property Stores** in the Policy Studio tree, and select **Add Key Property Store**.
2. Specify the following settings in the dialog:
 - **Name:** A collection must have a unique name in the API Gateway group.
 - **Description:** You can provide an optional description.
 - **Alias prefix:** You can also specify an alias prefix, but in normal usage, you can leave this field blank.
 - **Default data source:** A collection has a default data source where all data for all tables in the collection is stored. You can change this data source or assign a different data source to individual tables in the collection.
3. When the collection is created, you can specify a **Cache** for storage and retrieval of selector results. This will improve selector read performance for storage backends such as databases. For more details on API Gateway caching, see the *API Gateway Policy Developer Guide*

Note Only local caches are supported.

The following shows a KPS collection created in Policy Studio:

KPS Collection: Samples	
Properties	Data Sources
Name	Samples
Alias prefix	
Description	Samples collection
Default data source	Default Embedded Data Source
Cache	

Configure a KPS table

A KPS table is a user-defined table managed by the KPS in the API Gateway. To configure a KPS table, perform the following steps:

1. Right-click a KPS collection in the Policy Studio tree, and select **Add Table**.
2. Specify the following settings in the dialog:
 - **Name:** A KPS table must have a unique name in the collection.
 - **Description:** You can provide an optional description.
 - **Override the default data source with the following:** You can specify a different data source than the collection if required.
3. When the table is created, if required, you can use the **Override the default data source with the following** setting to specify a different data source than the collection:

Table: User

Properties | Structure

Name: User

Description: Table to contain user data.

Aliases: User

Override the default data source with the following:

4. Finally, click the **Structure** tab, and click **Add** to define the structure of the data stored in the table. For details on supported types and keys, see the following:
 - [KPS table structure on page 22](#)
 - [Query tables using properties and keys on page 23](#)

KPS aliases

KPS tables are accessed by alias. A table must have at least one alias. Aliases must be unique in an API Gateway group. You also can use the optional alias prefix for the collection to help ensure that the alias is unique.

The full alias of a table is the *collection alias prefix* and the *table alias* combined. For example, `samples` and `User` gives `samplesUser`. If unspecified, the default value of the alias prefix for the collection is an empty string (for example, `User` only).

KPS data sources

A KPS collection has one active data source associated with it. All tables in the collection use this data source by default. You can configure a table to use a different data source if required (see [Configure a KPS table on page 21](#)).

KPS table structure

When creating a table, you must define the structure of the data that is stored in that table. This consists of property (column) names and types. You can choose from the following types.

Type	Description
String	Java type.
Boolean	
Byte	
Integer	
Long	
Double	
List	Java <code>List</code> of any one of the above Java types.
Map	Java <code>Map</code> . The key can be any one of the above Java types. The value can be any one of the above Java types.

Query tables using properties and keys

You can directly access records in a table by specifying certain property names and values, without needing to read all the records in the table. You can use only properties of `String`, `Long`, and `Integer` types for this purpose. These properties are known as *indexable properties*.

Indexed properties include primary keys, secondary keys (which are indexed implicitly), and other properties that you explicitly select as **Indexed** in Policy Studio.

Table: User Save ?

Properties Structure

Name	Type	Primary Key	Autogenera...	Encrypted	Indexed (maximum 5)
age	java.lang.Integer	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
email	java.lang.String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
firstName	java.lang.String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
lastName	java.lang.String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
password	java.lang.String	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Primary key

You can directly access any record using its *primary key*. All records in the table must be accessible using a unique primary key. You must select one **Primary Key** per table in Policy Studio. The specified property must be an indexable property. Primary key values cannot be null.

Secondary key

You can optionally access any record directly using a unique secondary key. The secondary key can be a simple key (for example, `email`) or a composite key (for example, `appId` or `companyId`). The specified properties must be indexable properties. The secondary key (and parts of a composite secondary key) cannot be null. You can specify one secondary key per table. A common use case is to specify an internal unique ID as a primary key, and an external user-facing ID as a secondary key. For example, `id` for internal primary key, and `email` for external ID as a secondary key.

Selector access

You can access records in a KPS table using an API Gateway selector. If a secondary key is defined for the table, you must specify all secondary key values in the selector. If no secondary key is defined, you must specify the primary key value instead. In Policy Studio, you can specify a secondary key or a primary key in the **Use the following property name(s) for looking up a table from a selector** field.

For examples of accessing KPS tables using selectors, see the following:

- [Get started with KPS on page 10](#)
- [Access KPS data using selectors on page 26](#)
- [Configure database KPS storage on page 47](#)

Auto-generated properties

In Policy Studio, you can select that `String` fields are **Autogenerated**. When a record is created, a Java `java.util.UUID` is assigned to the field if it is empty.

Note Values for auto-generated fields can be supplied and modified by users. KPS only generates a value at creation time if no value is already present.

Encrypted properties

In Policy Studio, you can select that `String` fields are **Encrypted** in storage. However, fields selected as **Indexed** (including primary and secondary key fields) cannot be encrypted. You can enter values for encrypted fields using the API Gateway Manager or the `kpsadmin` command. These values are forwarded to the API Gateway in the clear using the KPS REST service, and encrypted before being written to storage.

Note The KPS REST service must always run over HTTPS (the default). You must set an encryption passphrase for the API Gateway group, because this is used in the encryption process. For more details, see the *API Gateway Administrator Guide*.

When KPS tables are accessed using API Gateway selectors at runtime, encrypted fields are automatically decrypted. Selectors do not need to be aware that particular fields in a table are encrypted in storage.

When KPS tables are read using the REST API, data is always returned in its encrypted state. Sometimes you may need to view decrypted data to help debug problems on an API Gateway. You can do this using debug mode in `kpsadmin`. This requires you to enter the passphrase for the API Gateway group.

If the in-built KPS encryption mechanism does not suit your needs, you can encrypt and decrypt data outside the KPS. In this case, you should not select properties in KPS tables as encrypted in Policy Studio. Encrypted data must be string-encoded for storage (for example, base64-encoded). Selectors that access the data must decrypt it themselves (for example, using a dedicated decryption filter in Policy Studio).

Access KPS data using selectors

4

Data in KPS tables can be accessed using selectors that execute in policies on the API Gateway at runtime. This topic explains KPS selector syntax and provides some example selectors.

KPS selector syntax

KPS selector syntax is as follows:

```
#{kps.alias[key].property}
```

The parts in the selector are described as follows:

Selector part	Description
<code>#{</code>	Indicates the start of the selector using a <code>{</code> bracket.
<code>kps</code>	Specifies that selector should query a KPS table.
<code>.alias</code>	Specifies the full alias of the KPS table, including the collection alias prefix if any (for example, <code>User</code>).
<code>[</code>	Indicates the start of a table property reference using a <code>[</code> bracket.
<code>key</code>	The key value to query the table (for example, <code>http.querystring.id</code>).
<code>]</code>	Indicate the end of a table property reference using a <code>]</code> bracket.
<code>.property</code>	The field to retrieve from the returned row (for example, <code>age</code>).
<code>}</code>	Indicate the end of the selector using a <code>}</code> bracket.

You can also use a composite key, for example:

```
#{kps.alias[key1][key2].property}
```

KPS selector examples

For an example of accessing KPS data from a selector using a primary key, see [Get started with KPS on page 10](#). For examples of selectors that use both primary and composite keys, see [Configure database KPS storage on page 47](#).

The following table shows more examples of KPS selectors:

Selector	Description
<pre data-bbox="349 575 795 638"> \${kps.User [http.querystring.id].firstName} </pre>	<ul data-bbox="893 575 1339 735" style="list-style-type: none"> • Get row from KPS table with <code>User</code> alias • Use key supplied in HTTP query string (<code>id</code>) • Return <code>firstName</code> field of row
<pre data-bbox="349 772 755 835"> \${kps.User ["kathy.adams@acme.com"].age} </pre>	<ul data-bbox="893 772 1339 966" style="list-style-type: none"> • Get row from KPS table with <code>User</code> alias • Use constant key <code>"kathy.adams@acme.com"</code> with quotation marks • Return <code>age</code> field of row
<pre data-bbox="349 1008 836 1102"> \${kps.User [http.querystring.firstName] [http.querystring.lastName].email} </pre>	<ul data-bbox="893 1008 1339 1165" style="list-style-type: none"> • Get row from KPS table with <code>User</code> alias • Use key supplied in HTTP query string (<code>firstName</code> and <code>lastName</code>) • Return <code>email</code> field of row

For more details on selectors, see the *API Gateway Policy Developer Guide*.

Manage a KPS using kpsadmin

5

The `kpsadmin` command-line tool provides KPS management functions, independent of data source. This tool is especially useful for development use.

Note In production, you should use data source-specific tools and administration procedures for data backup, restore, security, optimization, monitoring, and so on.

Start kpsadmin

From a command prompt, start `kpsadmin`. For example:

Windows

```
INSTALL_DIR\Win32\bin\kpsadmin.bat
```

UNIX

```
INSTALL_DIR/posix/bin/kpsadmin
```

Start in verbose mode

To run `kpsadmin` in verbose mode, use the `-v` option. `kpsadmin` will then show all REST messages that are exchanged with the API Gateway. This is useful for debugging. For example:

```
kpsadmin -v
```

Select kpsadmin operations

This section describes the `kpsadmin` operations that are available. When you first select an operation, you must enter the following:

- API Gateway group to use
- Admin API Gateway in that group that handles KPS requests

Note This is the Admin API Gateway used for KPS purposes only, and should not be confused with the Admin Node Manager.

- KPS collection to use in the group
- KPS table to use in the collection

You can change this selection at any time.

Table operations

The `kpsadmin` table operations are as follows:

Table operation	Description
Create Row	Create a row in the selected table.
Read Row	Read a row by primary key in the selected table.
Update Row	Update a row in the selected table. The row is specified by primary key.
Delete Row	Delete a row in the selected table. The row is specified by primary key.
List Rows	List all rows in the table.

Table administration operations

The `kpsadmin` operations for table administration are as follows:

Table Administration	Description
Clear	Clear all rows in the table.
Backup	Back up the table data. The generated backup UUID is required when restoring the data.
Restore	Restore table data. The table must be empty before you restore.
Re-encrypt	Re-encrypt encrypted data in the table. Use this option when the encryption passphrase has been changed for the API Gateway group. The table will be offline after a passphrase change. You must use this option to re-encrypt the data. You must enter the old API Gateway passphrase to proceed. Data is re-encrypted using the current API Gateway passphrase.

Table Administration	Description
Re-create	<p>Recreate a table. This is useful in development if you wish to change the table structure. This procedure involves dropping and recreating the table, so all existing data will be lost. The steps are as follows:</p> <ol style="list-style-type: none"> 1. Back up (optional). Backup the data if necessary using <code>kpsadmin</code>. 2. Deploy the correct configuration. First redeploy the correct configuration using Policy Studio. This may result in some KPS deployment errors. The changes you have made may no longer match the stored data structure. 3. Re-create the table with the correct configuration. Select the Re-create option using <code>kpsadmin</code>. 4. Restore (optional) Restore the data using <code>kpsadmin</code>. If you have made key or index changes, the data should import directly. If you have made more extensive changes (for example, renaming fields or changing types), you must upgrade the data to match the new table structure.
Table Details	Display information about a table and its properties.

Collection administration operations

The `kpsadmin` operations for collection administration are as follows:

Collection Administration	Description
Clear All	Clear all data in all tables in the collection.
Backup All	Back up all data in all tables in the collection.
Restore All	Restore all data in all tables in the collection.
Re-encrypt All	Re-encrypt all data in all tables in the collection.
Collection Details	Display information about all tables in the collection.

Cassandra administration operations

The `kpsadmin` operations for Cassandra administration are as follows:

Cassandra Administration	Description
Show Configuration	Show the current configuration for the KPS-embedded storage service (Apache Cassandra).

General administration operations

The `kpsadmin` operations for general administration are as follows:

General	Description
Change Table	Change the currently selected table.
Change Collection	Change the currently selected collection, and select a table in that collection.
Change Group or API Gateway	Refresh the configuration, and change the currently selected API Gateway group and KPS Admin API Gateway.
Debug Mode	Enable or disable debug mode. To enable, you must enter the API Gateway group passphrase. Encrypted data in KPS tables is then shown in the clear. This can be useful for debugging issues on the API Gateway.

Example switching data source

This example shows how to switch from Cassandra storage to file storage.

Step 1: Backup collection data using `kpsadmin`

To copy the current data in the collection to the new data source, back up the collection data using `kpsadmin option 21) Backup All`.

The backup UUID is highlighted in the following example:

```

Select option: 21
Getting Topology...

Select a Group:
1) My Group
Enter selection [1]:

Select an API Server:
1) My Gateway
Enter selection [1]:

Getting model...

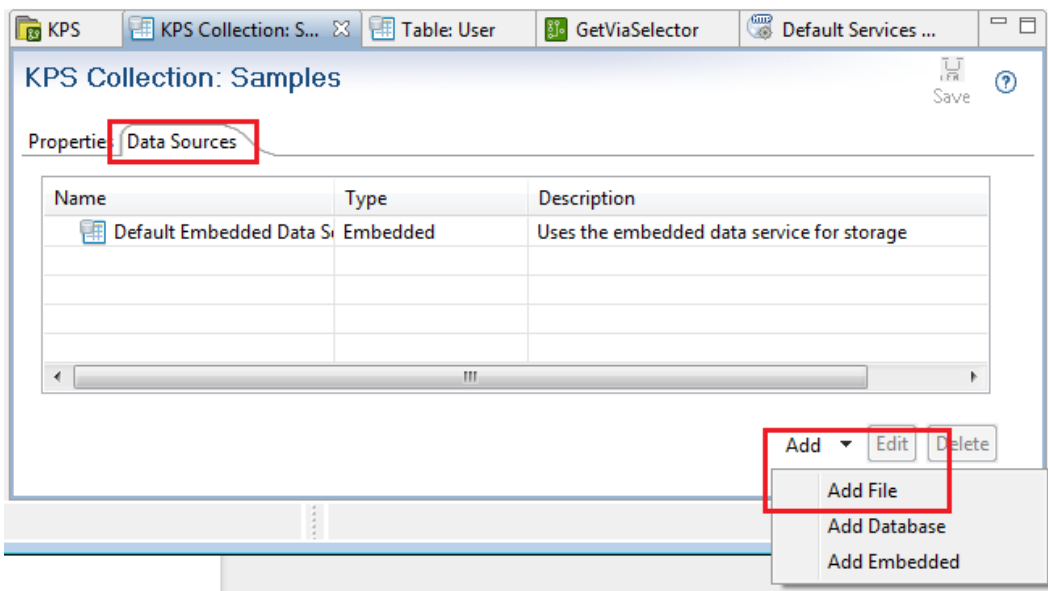
Select a Collection:
1) OAuth (internal)
2) Samples
3) API Server (internal)
Enter selection [1]: 2
Backing up all data in all tables in this collection.
Backup uuid is: 892cfd91-4b0a-417d-95f6-6c227100a00b
Are you sure you wish to continue y/n [n]: y
Starting backup to: 892cfd91_4b0a_417d_95f6_6c227100a00b_samples_user_json
Backup done.

```

Step 2: Create a new data source

To create the new data source, perform the following steps:

1. In the Policy Studio tree, select **Key Property Stores > Samples**.
2. Select the collection **Data Sources** tab.
3. Click **Add > Add File** at the bottom right.



4. Enter a file data source **Name** and **Description**.
5. Enter a **Directory Path** (for example, `${VINSTDIR}/kps/samples`).

Tip You can include `${VINSTDIR}` or `${VDISTDIR}` to indicate the API Gateway instance directory or install directory respectively. Make sure to use `\` on Windows or `/` on UNIX. If the directory does not exist, it is automatically created.

6. Select the collection **Properties** tab.

7. Change the collection **Default data source** to use the new data source:

KPS Collection: Samples

Save ?

Properties Data Sources

Name Samples

Alias prefix

Description Samples collection

Default data source Samples Collection File Storage

Cache

Step 3: Deploy the configuration

Click the **Deploy** button in the Policy Studio toolbar.

Step 4: Restore collection data using kpsadmin

If you made a backup in step 1, to restore the collection data, perform the following steps:

1. Using `kpsadmin`, select option 22) Restore All.
2. Enter the backup UUID noted in step 1. For example:

```
Select option: 22
Restoring all data in all tables in this collection.
Enter backup UUID: 892cfd91-4b0a-417d-95f6-6c227100a00b
Are you sure you wish to continue y/n [n]: y
Starting restore from: 892cfd91_4b0a_417d_95f6_6c227100a00b_samples_user_json
Restore done.
```

Configure Apache Cassandra KPS storage

6

Apache Cassandra provides a highly available (HA) data storage option for KPS. By default, each API Gateway runs an embedded Cassandra server. You must configure two or more API Gateways to provide a HA data service.

A single API Gateway, non-HA system does not require any configuration. For an overview of this configuration, see [Single node, out-of-the-box, non-HA configuration on page 37](#).

This guide describes use cases tested with API Gateway. For production deployments, see the Apache Cassandra and Datastax documentation. You should ensure that you are familiar with Cassandra configuration and administration requirements.

Note `nodetool repair` must be run at regular intervals to ensure that deleted data remains deleted in a cluster. For example use cases, see [Apache Cassandra operations for API Gateway on page 69](#).

Cassandra configuration

In the API Gateway, Cassandra configuration and data are stored in the following directory:

```
INSTALL_DIR/groups/<group-id>/<instance-id>/conf/kps/cassandra
```

You can configure API Gateway to run an embedded Cassandra server and client. The Cassandra configuration is stored as follows:

File	Description
<code>cassandra.yaml</code>	Cassandra server configuration file.
<code>client.yaml</code>	Cassandra client configuration file.
<code>jvm.xml</code>	JMX configuration required for Cassandra administration.
<code>commitlog</code>	Cassandra data directory.
<code>data</code>	Cassandra data directory.
<code>saved_caches</code>	Cassandra data directory.

Cassandra ports

This section describes the available ports in the Cassandra configuration files:

cassandra.yaml

The Cassandra server configuration file includes the following:

Property	Description	Default
<code>storage_port</code>	TCP port for server–server communication. The same port must be used across the cluster.	7000
<code>listen_address</code>	TCP Address for server–server communication. Each server must have a unique address.	localhost
<code>rpc_port</code>	TCP port for client communication.	9160
<code>rpc_address</code>	TCP address for client communication.	localhost
<code>seed_provider/ parameters/seeds</code>	At least one seed must be available when bootstrapping a new node for the first time.	localhost

client.yaml

The Cassandra client configuration file includes the following:

Property	Description	Default
<code>hosts</code>	Comma-separated list of <i>server address:port</i> pairs.	localhost:9160

jvm.xml

The JMX configuration file includes the following:

Property	Description	Default
<code>com.sun.management.jmxremote.port</code>	Cassandra JMX monitoring port. After the initial handshake, the JMX protocol requires that the client reconnects on a randomly chosen port (1024+). To enable JMX, remove the <code><if></code> clause from this file.	7199 1024+

Note You must ensure the following:

- `conf/kps/cassandra` directory is included as part of your API Gateway backup plan. For more details on backup, see the *API Gateway Administrator Guide*.
- Changes to `cassandra.yaml`, `client.yaml`, and `jvm.xml` require an API Gateway restart.
- `cassandra.yaml` is unique to each API Gateway instance because it contains a unique `listen_address` for each API Gateway.

Secure ports

The following firewall rules apply for securing ports:

- `storage_port:listen_address` must be available to other hosts in the cluster.
- `rpc_port:rpc_address` and `com.sun.management.jmxremote.port` can be restricted to local addresses.
- Use SSH to get to the machine to run Cassandra client and administration tools such as `nodetool` and `cassandra-cli`.

Cassandra logging

Cassandra logging is set to `ERROR` level by default. To enable Cassandra debug output, perform the following steps:

1. Edit the following file:

```
INSTALL_DIR/system/lib/log4j.properties
```

2. Update `ERROR` to `DEBUG` for `log4j.logger.org.apache.cassandra (server)` and `me.prettyprint.cassandra (client)`.

3. Restart API Gateway.

Cassandra configuration steps

The main Cassandra configuration steps are outlined as follows:

1. Configure the Cassandra topology in `cassandra.yaml`, `client.yaml`, and `jvm.xml`.
2. Start API Gateways with embedded Cassandra servers
3. Configure the Cassandra replication factor
4. Configure request consistency levels

Step 1: Configure the Cassandra topology

You must decide on the Cassandra topology, replication factor, and consistency levels for your application. This section describes recommended and tested topology configurations in `cassandra.yaml`, `client.yaml`, and `jvm.xml`. In development, you can always start with one node and reconfigure later.

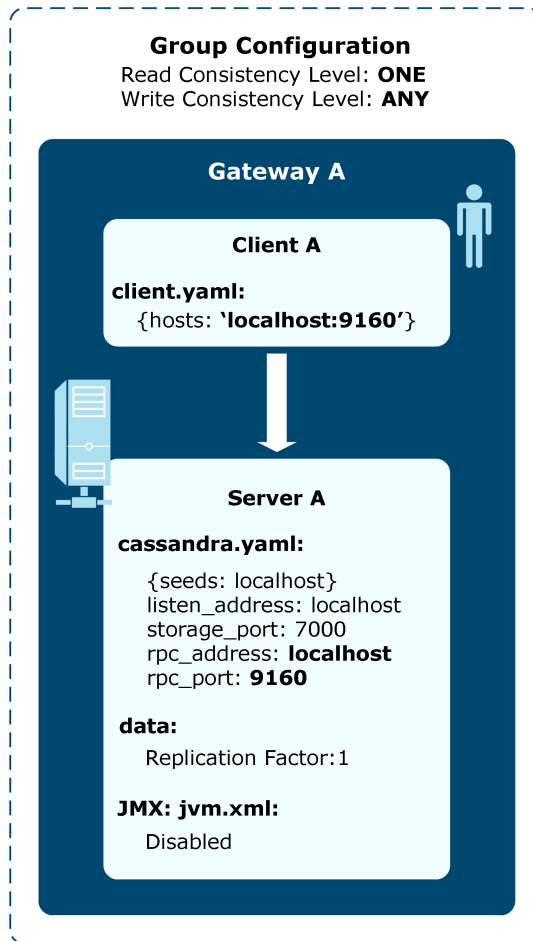
Single node, out-of-the-box, non-HA configuration

This is the out-of-the-box API Gateway configuration. No further configuration is required.

This configuration is described as follows:

- Cassandra server A is the seed node
- Server-server communication on port 7000 (must be the same across the cluster)
- Clients connect to their local server
- JMX is disabled (not needed for single node system)
- Consistent reads and writes
- No HA

The following diagram shows an example configuration:



Note When a Cassandra server starts up for the first time, it contacts a seed node to obtain information about other servers in the cluster. In a multi-server cluster, at least one seed node must be available when starting a Cassandra server for the first time.

Multiple nodes, single host, out-of-the-box, non-HA configuration

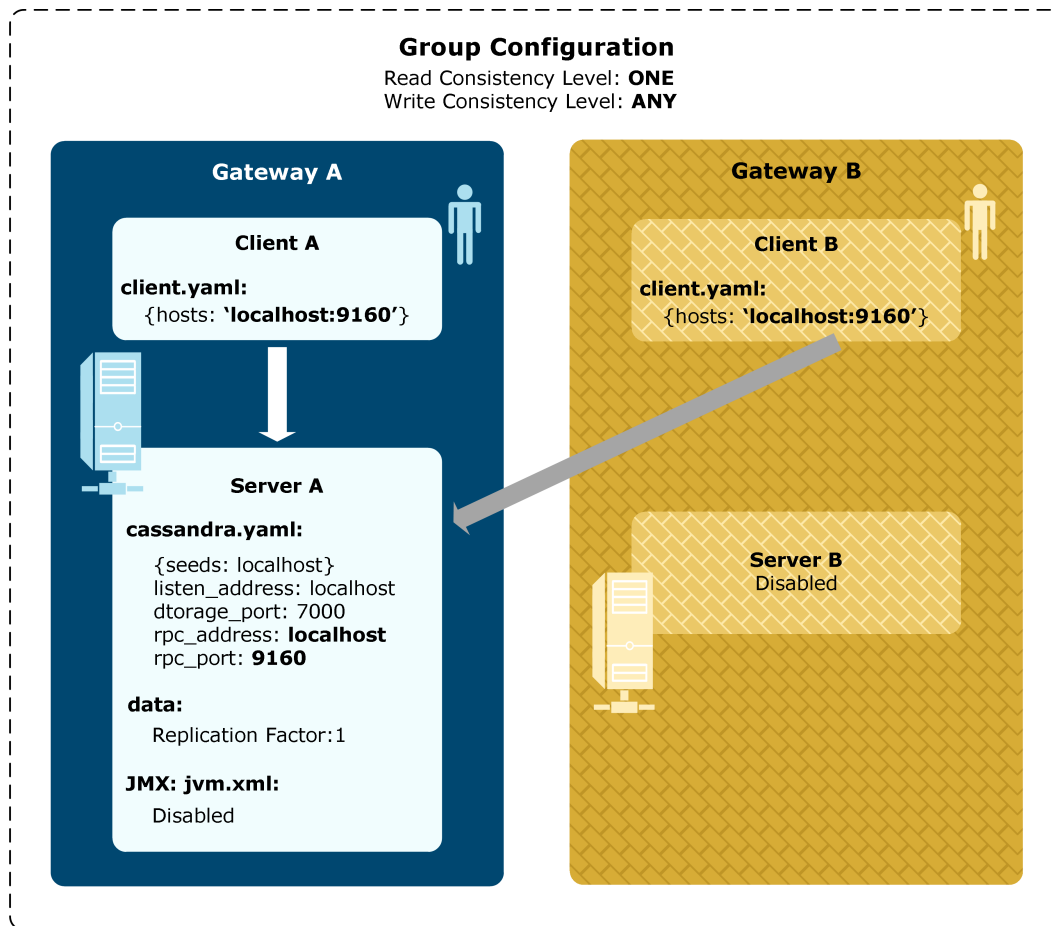
When running multiple API Gateways on the same host, the first API Gateway to run acts as a Cassandra server. Subsequent servers to run act as clients. This is sufficient when getting started. However, the Cassandra server designation depends on the run order.

Note On API Gateways that should run as Cassandra clients, rename or remove the `cassandra.yaml` file, and restart these API Gateways. If a client starts up before a server, you will get multiple errors such as:

```
me.prettyprint.hector.api.exceptions.HectorException:All host pools marked
down. Retry burden pushed out to client
```

However, this is expected. The client will not be able to field Cassandra requests until a server is available.

The following diagram shows an example configuration:



Note This is a client-server configuration. If the server API Gateway is not available, KPS data and functionality will not be available to client API Gateways. If you want higher availability with Cassandra, configure a suitable HA configuration as described in the following sections.

Multiple node prerequisite: IP address required per Cassandra server

For true HA, you should use separate hosts. However, for development, you can set up a test IP address. For example, you can configure multiple IP addresses on a single host as follows:

Windows

Edit or create the following file:

```
C:\Windows\System32\drivers\etc\hosts
```

Add the following lines:

```
127.0.0.1    127.0.0.N
```

Linux

Enter the following command:

```
sudo ifconfig lo:N 127.0.0.N netmask
255.0.0.0 up
```

Where *N* is 2, 3, 4.

The following example shows `ifconfig` output on Linux:

```
lo:2      Link encap:Local Loopback
          inet addr:127.0.0.2  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1

lo:3      Link encap:Local Loopback
          inet addr:127.0.0.3  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1

lo:4      Link encap:Local Loopback
          inet addr:127.0.0.4  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
```

Two-node HA eventual consistency

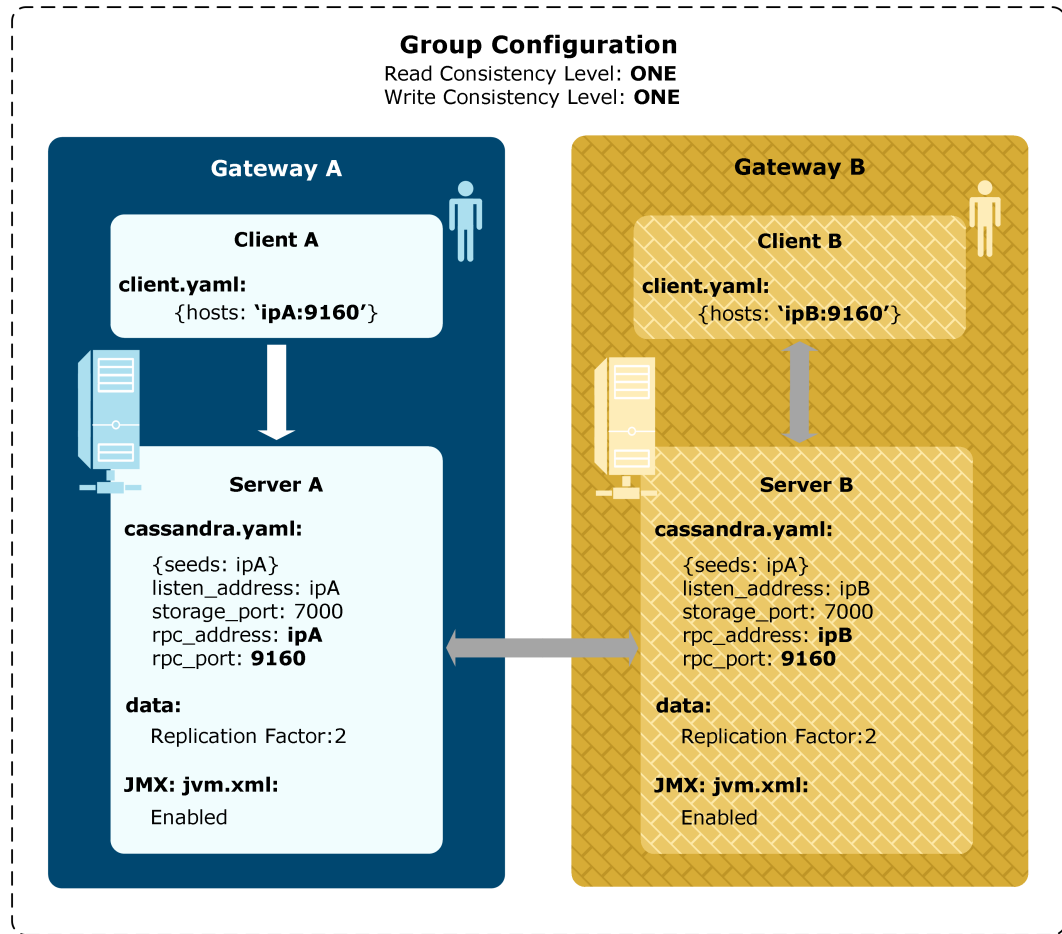
This is a two-API Gateway system, with IP addresses ipA and ipB. Note the [Multiple node prerequisite: IP address required per Cassandra server on page 39](#). This configuration is described as follows:

- Two Cassandra servers A and B defined in a cluster (server A is the seed node)
- Server-server communication on port 7000. This must be the same across the cluster
- Clients connect to their local server
- JMX is enabled for administration

The Cassandra parameter configuration includes the following:

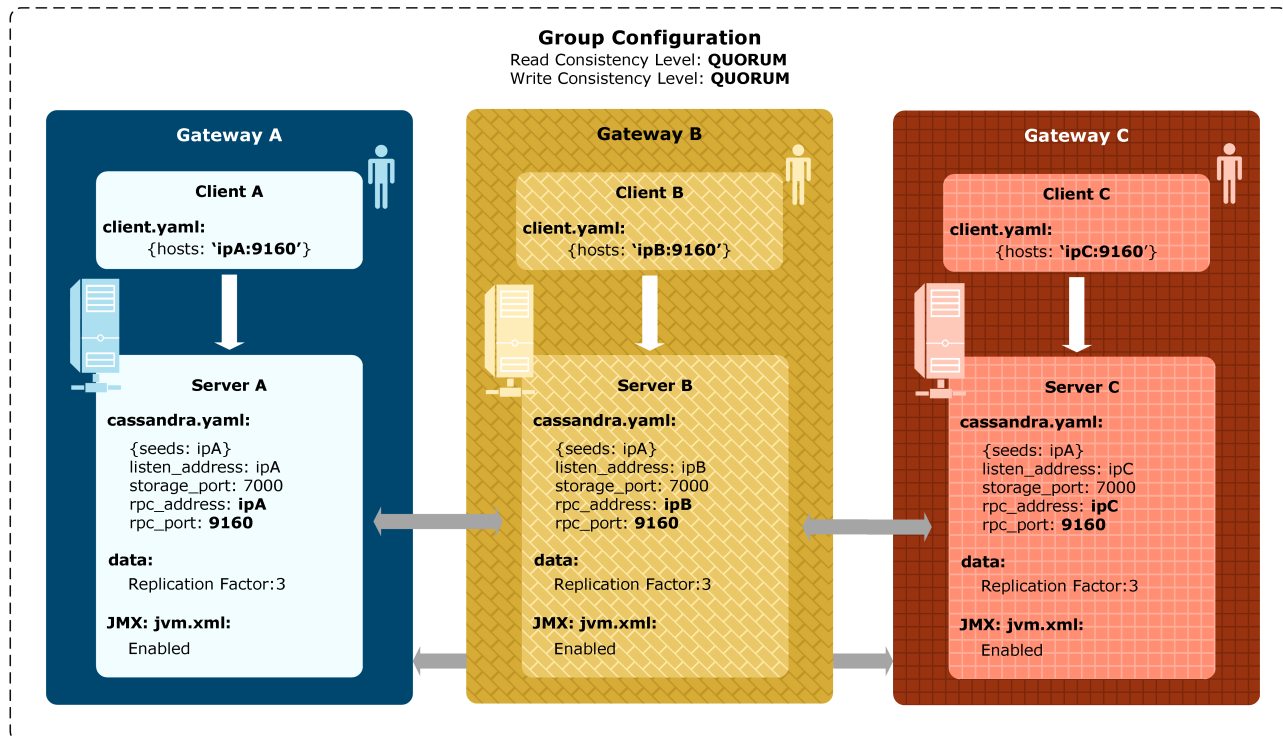
- Eventual Consistency: Read = Write = ONE
- Replication factor: 2
- Can survive the loss of one node
- Each node holds 100% of the data

The following diagram shows an example configuration:



Three-node HA full consistency

This is a three-API Gateway system, with IP addresses ipA, ipB, and ipC. Note the [Multiple node prerequisite: IP address required per Cassandra server on page 39](#). The following diagram shows an example configuration:



This configuration is described as follows:

- Three servers A, B, and C defined in a cluster (server A is the seed node)
- Server-server communication on port 7000 (must be the same across the cluster)
- Clients connect to their local server
- JMX enabled for administration

The Cassandra parameter configuration includes the following:

- Consistent: Read = Write = QUORUM
- Replication factor: 3
- Can survive the loss of one node
- Each node holds 100% of the data

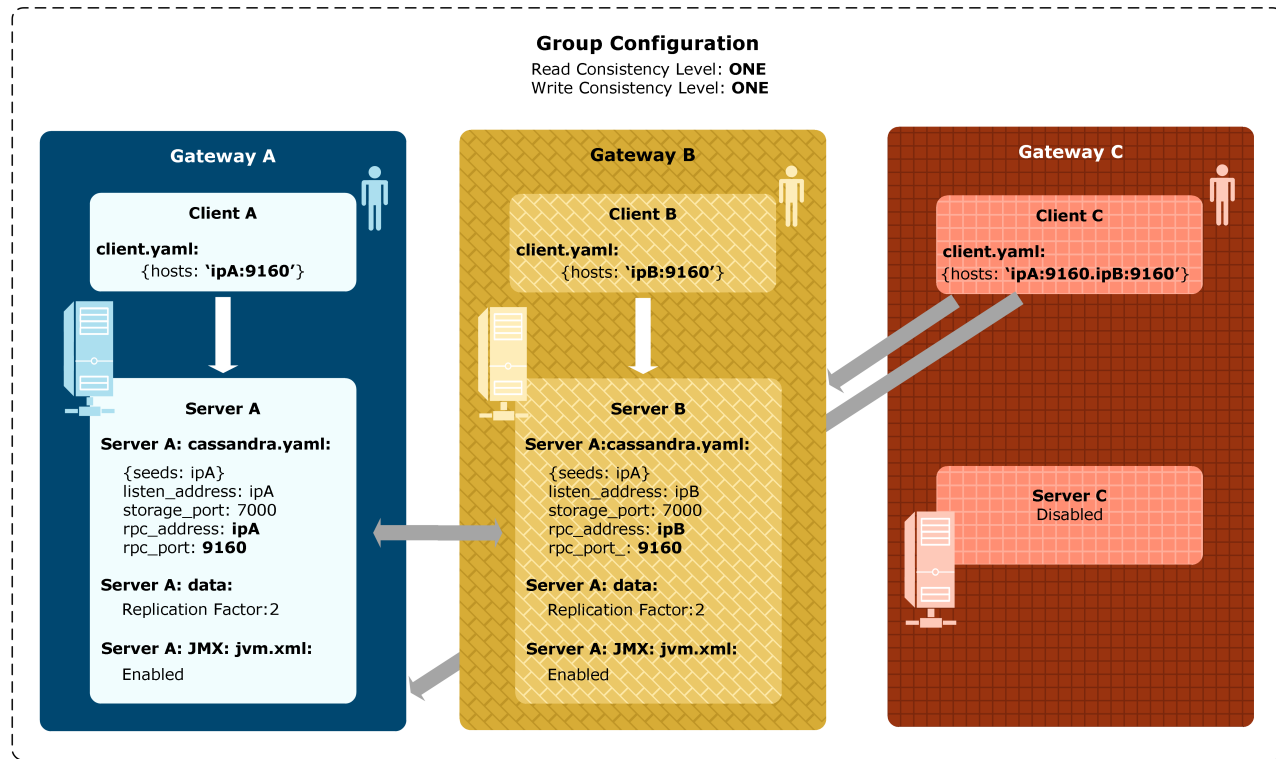
Two-node HA eventual consistency, with additional client-only node

This is similar to the previous two node systems with an additional client-only node. For example, you would choose this option for the following reasons:

- There are resource usage issues or restrictions on certain servers in the cluster (for example, disk, memory, CPU, JARs, classpath, and ports)
- You do not want to store data at rest on certain servers
- You want to separate the data layer from the application layer

Note If either Cassandra server goes down, the client automatically fails over to the other server.

The following diagram shows an example configuration:



Step 2: Start API Gateways with embedded Cassandra servers

On first time run, perform the following steps:

1. Ensure that the seed node is started first. Wait for this node to start.
2. Start up other nodes in turn. Wait for each node to start.

Tip You can verify the configuration by running `./kpsadmin` and selecting Option 30) Show configuration.

Step 3: Configure the Cassandra replication factor

To configure the replication factor, perform the following steps:

1. Run `cassandra-cli` in an API Gateway `bin` directory.
2. Execute the following commands:

```
./cassandra-cli -h 127.0.0.2
use kps;
update keyspace kps with strategy_
options = {replication_factor:2};
quit;
```

In this example, the replication factor is 2. Enter the correct replication factor for your Cassandra topology. For more details, see [Step 1: Configure the Cassandra topology on page 37](#).

3. Synchronise and verify this change across the cluster by executing `nodetool repair` against each node. In the following example, `nodetool` is executed against a two-node cluster running on IP addresses of 127.0.0.1 and 127.0.0.2:

```
./nodetool -h 127.0.0.2 repair kps
./nodetool -h 127.0.0.3 repair kps
./nodetool -h 127.0.0.2 ring kps
./nodetool -h 127.0.0.3 ring kps
```

You should see an effective ownership on each node of 100%. For example:

```
Datcenter: datacenter1
=====
Replicas: 2
```

Address	Rack	Status	State	Load	Owns	Token
127.0.0.2	rack1	Up	Normal	121.99 KB	100.00%	-6760287077468821940
127.0.0.3	rack1	Up	Normal	133.36 KB	100.00%	488569067738610269
						-6760287077468821940

Note Cassandra 1.2.18 `nodetool` requires the location of `cassandra.yaml`. You must update `apigateway/system/conf/cassandra-tools-jvm.xml` to specify location of `cassandra.yaml`. Do not use `$VINSTDIR` because the system does not know which API Gateway to resolve to. For example:

```
<ConfigurationFragment>

<!-- for the win32 JVM, ensure that the path to the jvm.dll is available. POSIX style
hosts do this using scripts. -->

<PathAdd name="PATH" value="$VDISTDIR/win32/jre/bin/server" />

  <JVMSettings>
    <ClassDir name="$VDISTDIR/system/lib/modules" />
    <ClassDir name="$VDISTDIR/system/lib/modules/cassandra/server" />
    <SystemProperty name="cassandra.config"
      value="file://$VDISTDIR/groups/group-2/instance-
1/conf/kps/cassandra/cassandra.yaml"/>
  </JVMSettings>
</ConfigurationFragment>
```

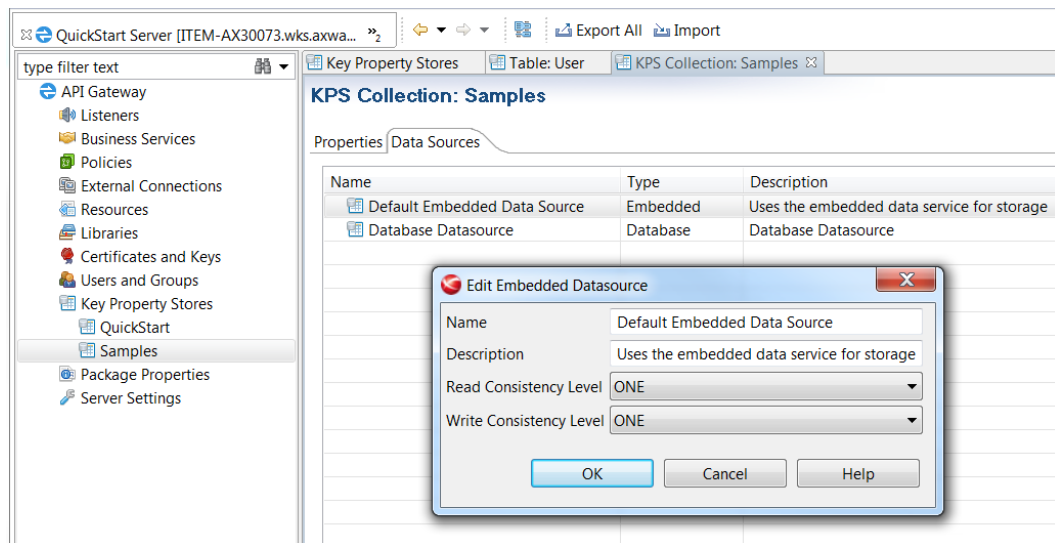
Step 4: Configure request consistency levels

In Policy Studio, set the request consistency levels specified in your Cassandra topology. For more details, see [Step 1: Configure the Cassandra topology on page 37](#).

To configure request consistency levels, perform the following steps:

1. Select a KPS collection in Policy Studio (for example, **Key Property Stores > Samples**).
2. Select the **Data Sources** tab.
3. Select the **Default Embedded Data Source**, and click **Edit** at the bottom.
4. In the dialog, select the appropriate **Read Consistency Level** and **Write Consistency Level** values for your topology.
5. Click **Deploy** in the Policy Studio toolbar.

The following example shows s Cassandra consistency level set to one for read and write for a collection called `Samples`:



Disable Cassandra storage

If you do not want to use Cassandra storage, and you are not using Cassandra for OAuth token storage, you can disable Cassandra storage.

To disable Cassandra storage, perform the following steps:

1. Run the following sample script against the API Gateway. This updates **the** API Gateway configuration by switching any Cassandra usage to file usage.

Windows

```
cd INSTALL_DIR\samples\scripts
```

```
run.bat cassandra/disableCassandra.py
```

UNIX

```
cd INSTALL_DIR/samples/scripts  
./run.sh cassandra/disableCassandra.py
```

2. Restart the API Gateway.
3. If you are running multiple API Gateways in the group, deploy the updated configuration to these API Gateways and restart them.

Configure database KPS storage

7

KPS data can be stored in a relational database. API Gateway supports the following databases:

- Oracle
- IBM DB2
- Microsoft SQL Server
- MySQL

The options for database storage are as follows:

- **Shared database storage:** Data for multiple KPS tables is stored in a single dedicated database table. Data is encoded in JSON before being stored. This approach is very flexible because it allows maps and lists to be stored in tables. It also minimizes administration overhead because only a single database table needs to be created and managed. This is the recommended approach. For more details, see [Shared database storage on page 47](#).
- **Per-table database storage** Each KPS table is backed by a single database table. Each property in the KPS table maps to a corresponding column in the database table. This approach allows existing tables to be reused, and allows more precise tuning at database level. However, it also has significant limitations (for example, supported data types, and adding and viewing data). For more details, see [Per-table database storage on page 51](#).

Shared database storage

For shared database storage, a single database table is used to store data for multiple KPS tables. This section describes the storage configuration steps.

Step 1: Create a KPS database table

Create a database table called `kps_object` by executing the appropriate SQL script for your database system. SQL scripts are available in the following location:

```
INSTALL_DIR/system/conf/sql/DB_NAME/kps.sql
```

Note For Oracle, ensure that the database is created with the AL32UTF8 character set encoding to support UTF8.

Step 2: Set up an external connection to the database

To access this table from the API Gateway, you must setup an external database connection from the API Gateway. For more details, see the *API Gateway Policy Developer Guide*.

The following shows example database connection settings in Policy Studio:

The screenshot shows the configuration interface for a database connection in Policy Studio. The fields are as follows:

- Name:** Test DB Connection
- URL:** jdbc:mysql://testserver:3306/sampledb?useUnicode=true&characterEncoding=UTF-8
- User Name:** root
- Password:**
 - Enter Password: *****
 - Wildcard Password:
- Advanced - Connection pool:**
 - Initial pool size: 0
 - Maximum number of active connections: 8
 - Maximum number of idle connections: 8
 - Minimum number of idle connections: 0
 - Maximum wait time (ms): 10000
 - Time between eviction (ms): -1
 - Number of tests: 3
 - Minimum idle time (ms): 1000

Note For MySQL, the table creation script specifies UTF8. You must also use the correct JDBC connection URL. Update the connection **URL** field to specify unicode & UTF8. For example:

```
jdbc:mysql://testserver:3306/kps?useUnicode=true&characterEncoding=UTF-8
```

Step 3: Use the external connection in a KPS collection

When creating a KPS collection, you can select database storage in the **Default datasource** field. The following example shows an SQL database selected:

i A KPS Collection represents a grouping of KPS Tables

Name

Description

Alias prefix

Default datasource **SQL Database** ▼

Alternatively, you can add a database storage option to the collection later. On the **Data Sources** tab, select **Add > Add Database**. For an example, see [Step 2: Add the database data source to the KPS collection on page 52](#).

For an example with file storage, see [Configure file-based KPS storage on page 59](#).

Database storage information

The following describes how KPS data is stored in a database:

- The maximum primary key length in a KPS row is 255 characters
- The maximum KPS table name length is 255 characters
- KPS rows are JSON encoded
- Optimistic locking is used and is enforced using a version column

Increase row size

You can increase the maximum KPS row size by changing the `largeValue` column. For example, to support image icons in MySQL, enter the following command:

```
alter table kps_object modify column largevalue mediumtext;
```

Logging for shared table storage

Shared database storage uses Apache OpenJPA to handle the communication between KPS and the back-end database. You can use OpenJPA logging to view the SQL requests transmitted to the database and their responses. This information can be useful for debugging. You can configure OpenJPA logging using Apache log4j properties.

Enable OpenJPA debug logging

To enable OpenJPA debug logging:

1. Edit the following file:

```
INSTALL_DIR/apigateway/system/lib/log4j.properties
```

2. Add the following settings:

```
log4j.rootLogger=DEBUG, A1, Vordel
# OpenJPA
log4j.category.openjpa.Tool=DEBUG
log4j.category.openjpa.Runtime=DEBUG
log4j.category.openjpa.Remote=DEBUG
log4j.category.openjpa.DataCache=DEBUG
log4j.category.openjpa.Metadata=DEBUG
log4j.category.openjpa.Enhance=DEBUG
log4j.category.openjpa.Query=DEBUG
log4j.category.openjpa.jdbc.SQL=DEBUG
log4j.category.openjpa.jdbc.SQLDiag=DEBUG
log4j.category.openjpa.jdbc.JDBC=DEBUG
log4j.category.openjpa.jdbc.Schema=DEBUG
```

3. Restart the API Gateway.
4. Verify that debug statements are written to the log.

Disable OpenJPA debug logging

To disable OpenJPA debug logging:

1. Edit the following file:

```
INSTALL_DIR/apigateway/system/lib/log4j.properties
```

2. Substitute `ERROR` for `DEBUG` in the `log4j.category.openjpa` settings.
3. Restart the API Gateway.
4. Verify that no debug statements are printed to the log.

For more information on Apache OpenJPA logging, see:

<http://openjpa.apache.org/builds/2.2.2/apache-openjpa/docs/main.html>

Per-table database storage

You can use this option to map a KPS table to a single database table. The structure of both tables must match, so a new database table is required for each new KPS table. When the KPS table is queried, an SQL statement is executed to retrieve the correct row from the underlying database table. This SQL statement is provided by the user in Policy Studio.

However, tables that use per-table database storage have significant limitations:

- Data cannot be added through KPS, but only directly through the database
- Data cannot be viewed in `kpsadmin` or API Gateway Manager, but can only be read by selectors at runtime
- Tables can only contain simple data types, not maps or lists

KPS tables can be queried using simple or composite keys. This section shows examples of both.

Map a database table using a single key

In this example, a KPS table is accessed using a single key property. This key is used to retrieve the correct row from the database table. This example uses the following `User` table and data created using a MySQL client:

```
CREATE TABLE User (  
    email VARCHAR(100),  
    password VARCHAR(100),  
    firstName VARCHAR(100),  
    lastName VARCHAR(100),  
    age INT  
);  
  
insert into User (email, password, firstName, lastName, age) values  
("ralph.jones@acme.com", "password", "Ralph", "Jones", 30);  
insert into User (email, password, firstName, lastName, age) values  
("kathy.adams@acme.com", "blah", "Kathy", "Adams", 35);
```

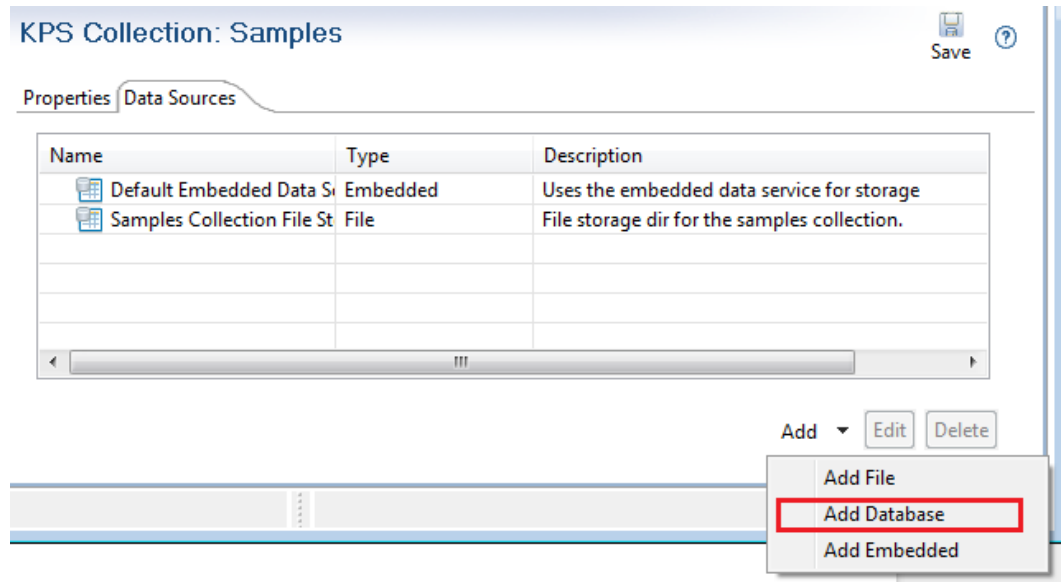
Step 1: Set up an external connection to the database

To access a database from the API Gateway, you must set up an external database connection (see [Step 2: Set up an external connection to the database on page 48](#)). This example uses the configuration from [Get started with KPS on page 10](#).

Step 2: Add the database data source to the KPS collection

To add a database data source, perform the following steps:

1. On the KPS collection **Data Sources** tab, select **Add > Add Database**.



2. Specify the **Database Connection** in the dialog (for example, Test DB Connection):

The screenshot shows a dialog box for adding a database connection. It contains three input fields:

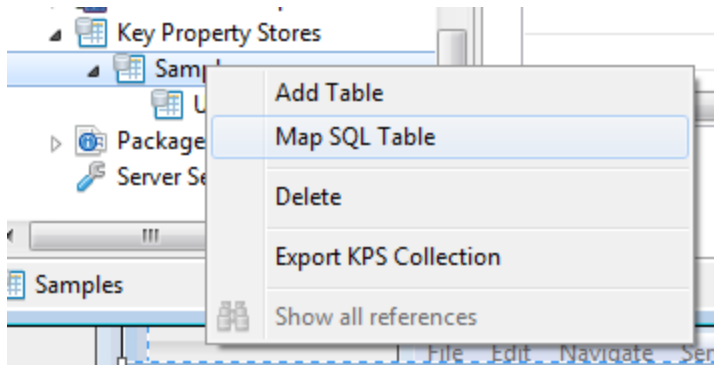
- Name:** Test Map SQL
- Description:** Test Map SQL
- Database Connection:** Test DB Connection

There is a three-dot menu button to the right of the 'Database Connection' field.

Step 3: Map the SQL table to a KPS table

From an existing KPS collection, perform the following steps:

1. Right-click the KPS collection, and select **Map SQL Table**:



2. Enter an alias in the dialog (for example `mapUser`) :

 A screenshot of the 'Map SQL Table' dialog box. The 'Name' field contains 'Map Example' and the 'Description' field contains 'Example of mapping SQL Query to a KPS Table'. Below these fields is a table with the following content:

Aliases
mapUser

 At the bottom right of the dialog are three buttons: 'Add', 'Edit', and 'Delete'. Below the table is a 'SQL Query' field containing the text: `select * from User where email = ?`

3. Enter a database-specific JDBC SQL query to retrieve the required data. For example:

```
select * from User where email = ?
```

4. On the **Properties** tab of the new KPS table, select the new database data source in **Override the default data source with the following**:

SQL Table: Map Example Save ?

Properties **Structure**

Name

Description

Aliases

Query

Override the default data source with the following

Step 4: Define the KPS table structure

You must define a KPS table structure into which data will be read. You must specify the fields that you expect to read with the SQL query. In this example, all fields in the table are read using an asterisk (*) in the SQL query. This lists all fields, so the order does not matter in this case. However, the names and type must match the result returned by the SQL query.

In this SQL query, `email` is the primary key. You specify `email` as the property to use in corresponding selector queries:

SQL Table: SQL Table: Map Example Save ?

Properties **Structure**

Name	Type
age	java.lang.Integer
email	java.lang.String
firstName	java.lang.String
lastName	java.lang.String
password	java.lang.String

Use the following property name(s) for looking up table from a selector (comma-separated, maximum 5):

For example, you can use the following selector:

```
${kps.mapUser
["kathy.adams@acme.com"].age}
```

Note This syntax uses ASCII quotation marks (").

This selector generates the following SQL query:

```
select * from User where email =
"kathy.adams@acme.com"
```

Step 5: Define the policy and path

You can reuse the policy and path from [Get started with KPS on page 10](#) with one change—use the `mapUser` KPS alias for this new table. For example:

Set the Message

Change the contents of the message body.

The screenshot shows the configuration for a message policy named 'Set Message'. The 'Content-Type' is set to 'text/plain'. The 'Message Body' field contains a template with placeholders for user information, where the `mapUser` function is highlighted with a red box.

```

=====
User
===
Email: ${kps.mapUser[http.querystring.id].email}
First Name: ${kps.mapUser[http.querystring.id].firstName}
Last Name: ${kps.mapUser[http.querystring.id].lastName}
Age: ${kps.mapUser[http.querystring.id].age}
=====

```

Step 6: Deploy and run

Click **Deploy** in the Policy Studio toolbar.

To run the policy in your browser, go to:

<http://localhost:8080/kpsGetViaSelector?id=kathy.adams@acme.com>

This URL specifies the user ID (`email`) as kathy.adams@acme.com.

For example, the result is as follows:

The screenshot shows a web browser window with the URL `localhost:8080/kpsGetViaSelector?id=kathy.adams@acme.com`. The browser displays the output of the policy, which is a text message with user details.

```

=====
User
===
Email: kathy.adams@acme.com
First Name: Kathy
Last Name: Adams
Age: 35
=====

```

How to map a database table using a composite key

This section modifies the example in [Map a database table using a single key on page 51](#). This section shows how a table with a composite secondary key can be accessed from a selector. In this version, the secondary key is {firstName, lastName}.

Step 1: Modify the KPS table

You must update the database-specific JDBC SQL query in the KPS table to retrieve the required data.

To modify the KPS table, perform the following steps:

1. On the **Properties** tab in the **Query** field, enter the following:

```
select * from User where firstName =  
? and lastName = ?
```

For example:

SQL Table: Map Example

Properties	Structure
Name	Map Example
Description	Example of mapping SQL Query to a KPS Table
Aliases	mapUser
Query	select * from User where firstName = ? and lastName = ?

2. On the **Structure** tab, change the selector properties to `firstName,lastName`:

Table: SQL Table: Map Example Save

Properties **Structure**

Name	Type	Primary Key	Autogenerated	Encrypted	Indexed (maximum 5)
age	java.lang.Integer	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
email	java.lang.String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
firstName	java.lang.String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
lastName	java.lang.String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
password	java.lang.String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Add Edit Delete

Use the following property name(s) for looking up table from a selector (comma-separated, maximum 5):

Step 2: Modify the policy

You must update the **Set Message** filter in your policy to use `firstName` and `lastName` parameters. For example:

Set the Message

Change the contents of the message body.



Name:

Content-Type:

Message Body: Populate ▾

```

=====
User
===
Email: ${kps.mapUser[http.querystring.firstName][http.querystring.lastName].email}
First Name: ${kps.mapUser[http.querystring.firstName][http.querystring.lastName].firstName}
Last Name: ${kps.mapUser[http.querystring.firstName][http.querystring.lastName].lastName}
Age: ${kps.mapUser[http.querystring.firstName][http.querystring.lastName].age}
=====

```

Enter the following in the **Message Body** field, using a single line for each entry (Email, First Name, Last Name, and Age):

```

=====
User
===
Email: ${kps.mapUser[http.querystring.firstName]
[http.querystring.lastName].email}

```

```
First Name: ${kps.mapUser[http.querystring.firstName]
[http.querystring.lastName].firstName}
Last Name: ${kps.mapUser[http.querystring.firstName]
[http.querystring.lastName].lastName}
Age: ${kps.mapUser[http.querystring.firstName]
[http.querystring.lastName].age}
=====
```

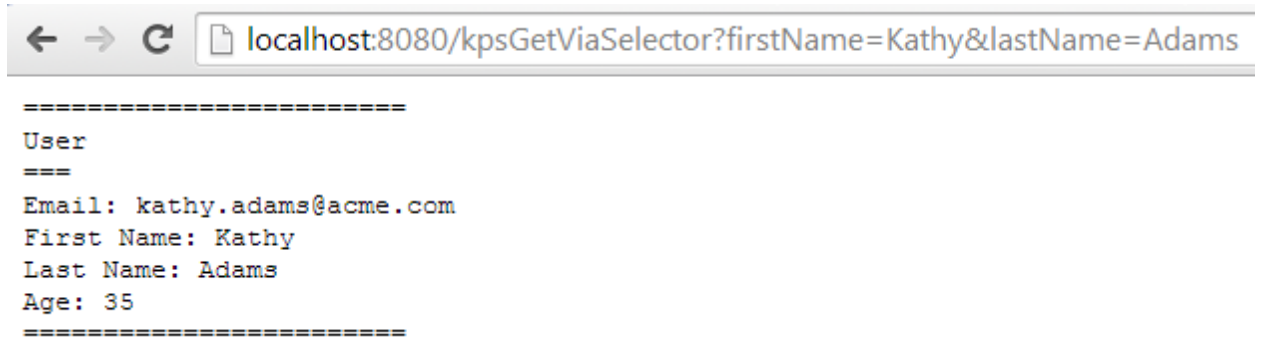
Step 3: Deploy and run

Click **Deploy** in the Policy Studio toolbar.

To run the policy in a browser, go to the following URL:

<http://localhost:8080/kpsGetViaSelector?firstName=Kathy&lastName=Adams>

For example, the result is as follows:



Configure file-based KPS storage

8

You can store KPS data in a directory on the file system. Each table is stored in a single JSON file. File-based storage is specified at the KPS collection or KPS table level.

Note File-based KPS storage is most suited to single API Gateway deployments. In a multi-API Gateway scenario, file replication or a shared disk is required to ensure that all API Gateways use the same data.

File-based KPS tables are read and cached by API Gateways when they start up. If data is modified, all API Gateways must be restarted to pick up the changes.

Configure a file-based KPS collection

You can configure a file-based KPS data source when creating a KPS collection, or add one later on the **Data Source** tab. For example, the following settings are available when editing file-based collection:

Name	<input type="text" value="file storage"/>
Description	<input type="text" value="sample file storage"/>
Directory Path	<input type="text" value="\${VDISTDIR}/mydata/samples"/>

These settings are described as follows:

Name	Description
Name	Collection-unique data source name.
Description	Optional description.
Directory Path	The directory name where table data for the collection is stored. If the directory does not exist, it is automatically created. If this directory is not specified, the directory path defaults to <code>\${VINSTDIR}/conf/kps</code> . The path can include <code>VDISTDIR</code> or <code>VINSTDIR</code> variables. These are resolved to the API Gateway instance and installation directories. For example, <code>\${VDISTDIR}/mydata/samples</code> . Remember to use the correct path separator (<code>/</code> on UNIX and <code>\</code> on Windows).

Appendix A: KPS FAQ

This appendix answers the frequently asked questions on KPS and API Gateway, KPS storage, and Apache Cassandra.

KPS and API Gateway

This section includes frequently asked questions on KPS and API Gateway:

What is KPS used for in API Gateway?

In addition to use in your API Gateway policies, KPS is used as an option for OAuth token storage, and Client Application Registry. If you do not wish to use these features, or do not wish to use the embedded Apache Cassandra service as a storage mechanism for these features, see [Configure Apache Cassandra KPS storage on page 34](#) for details on how to disable Cassandra storage.

What is KPS not suitable for?

KPS is not suitable for complicated data models, ad hoc queries, or where full ACID transaction support is required. KPS does not enforce referential integrity.

What are the transaction semantics of KPS?

Individual KPS operations are atomic (A), isolated (I), and durable (D). Consistency (C) depends on the data storage mechanism chosen and the number of API Gateways in a group.

With file storage, data is consistent in a single API Gateway. With supported database storage, data is consistent. Cassandra storage allows consistency levels to be set per KPS table. KPS does not provide transactions across multiple operations. You cannot issue a set of KPS operations and roll them back.

What is the KPS collection alias prefix for?

This provides an optional namespace for a KPS collection to help ensure that tables in the collection have a unique alias. In most cases, you can leave this prefix empty.

Why are some collections hidden? How can you show them?

Some out-of-the-box KPS collections are hidden in Policy Studio to prevent accidental modification. To show hidden KPS collections, perform the following steps:

1. Edit the following file:

```
INSTALL_DIR/policy_studio/policystudio.ini
```

2. Set the `show.internal.kps.collection` system property to `true`. For example:

```
-vmargs  
-Xmx256m  
-Dfile.encoding=UTF-8  
-Dshow.internal.kps.collection=true
```

3. Restart Policy Studio.

Note You will need to edit these hidden collections if you wish to change Cassandra consistency levels (for example, when switching to an HA setup).

How do I change API Gateway group passphrase?

To change the group passphrase, perform the following steps:

1. Change the group passphrase in Policy Studio. For details, see the *API Gateway Administrator Guide*.
2. In `kpsadmin`, select the `Collection Administration, Re-encrypt All` option to re-encrypt the data in each collection.
3. You will be asked to enter the old API Gateway passphrase. This passphrase is used to decrypt the data. The data is then re-encrypted with the current API Gateway passphrase.

For more details, see [Manage a KPS using kpsadmin on page 28](#).

KPS storage options

This section includes frequently asked questions on KPS storage mechanisms:

Is Apache Cassandra storage required? Can you use file or database?

Apache Cassandra is the default out-of-the-box storage, but is not a requirement. You can switch to file or database storage. For more details, see [How do you switch storage for a KPS collection? on page 62](#).

How do you switch storage for a KPS collection?

The general steps to switch storage for a KPS collection are as follows:

1. Back up the data for a collection using `kpsadmin`.
2. Add and configure a new data source (file, database, or Cassandra) using Policy Studio.
3. Deploy the configuration in Policy Studio.
4. Restore the data using `kpsadmin`.

For more details, see the example in [Manage a KPS using kpsadmin on page 28](#).

Why use database storage?

You may already have a relational database system and expertise that you wish to use. For more details, see [Configure database KPS storage on page 47](#).

Why use file storage?

File storage is very easy to use. Data is written to very simple JSON encoded files. File storage is suitable for single API Gateway use in development and production. If you have more than one API Gateway sharing the files, you must restart these API Gateways after a KPS update. It is only suitable here for rarely changing, read-mostly data. For more details, see [Configure file-based KPS storage on page 59](#).

When can you use `kpsadmin`? When should you use storage-specific tools?

You can use `kpsadmin` anytime. However, it is especially useful for development use. In production, you should use storage-specific tools and procedures (for example, for data backup). For more details on `kpsadmin`, see [Manage a KPS using `kpsadmin` on page 28](#).

Apache Cassandra

This section includes frequently asked questions on the Apache Cassandra database:

Why use Cassandra as a KPS storage option?

Cassandra is a key-value store. Cassandra has a non-restrictive Apache 2.0 license. It provides HA, is embeddable in a Java application (API Gateway), and has an active community.

What version of Cassandra does the API Gateway use?

The API Gateway ships with a version of Cassandra 1.2.18. Non-required JARs and scripts have been removed. The `cassandra-cli` and `nodetool` utilities are included.

What does all host polls marked down mean?

This means that the embedded client cannot connect to the embedded Cassandra server. For details on how to resolve this issue, see [Troubleshoot KPS error messages on page 65](#).

Can you use an external Cassandra instance?

You can modify the `client.yaml` file in API Gateway Cassandra configuration to point to an external Cassandra instance. This will reduce resource usage requirements in the API Gateway process.

How do you set Cassandra consistency levels?

For details, see [Configure Apache Cassandra KPS storage on page 34](#).

How do you disable Cassandra?

For details, see [Configure Apache Cassandra KPS storage on page 34](#).

Appendix B: Troubleshoot KPS error messages

This appendix describes various KPS error messages, and explains how to resolve them.

All platforms

This section explains how to troubleshoot KPS errors on all platforms:

All host polls marked down

This error means that the embedded client cannot connect to the embedded Cassandra server.

To resolve this issue, perform the following steps:

- Check that all ports and addresses are correct in `cassandra.yaml` to verify that the endpoints are what you expect.
- Run `kpsdamin`, and select **Cassandra Administration, Show Configuration** to read this configuration. And verify the result is what you expect. For more details, see [Manage a KPS using kpsadmin on page 28](#).
- Enable Cassandra debug logging.

For more details on Cassandra configuration and logging, see [Configure Apache Cassandra KPS storage on page 34](#).

Nodetool reports failed to connect

When running the `nodetool` command against a remote host in the Cassandra cluster, the following error is output:

```
Failed to connect to 'REMOTE_IP:7199: Connection refused'
```

To resolve this issue, perform the following steps:

1. On each node, edit the `jvm.xml` file in the following directory:
`INSTALL_DIR/apigateway/groups/GROUP_ID/INSTANCE_ID/conf/kps/cassandra`
2. Add the following JVM argument:
`java.rmi.server.hostname=BIND_ADDRESS`

For more details on Cassandra configuration, see [Configure Apache Cassandra KPS storage on page 34](#).

May not be enough replicas present to handle consistency level

For example, this error is received in a two-node Cassandra system with an eventual consistency of read = write = ONE.

To resolve this issue, you must ensure that the replication is correctly. For example, you can use the following `nodetool` command:

```
./nodetool -h 127.0.0.2 ring kps
Datacenter: datacenter1
=====
Replicas: 1 xxxxx
Address      Rack  Status State
Load        Owns  Token 7564491331177403445
127.0.0.2   rack1 Up     Normal 1.19MB 35.10% -
1090016113642762867
127.0.0.3   rack1 Up     Normal 2.17MB 64.90% -
7564491331177403445
```

Note In this case, the number of replicas is 1 and `Owns` is not 100% per node. You should set the replication factor, synchronize the changes, and verify as described in [Configure Apache Cassandra KPS storage on page 34](#)

Windows only

This section explains how to troubleshoot KPS errors on Windows only:

Node running in client mode

On Windows, the software detects that a service is already running on the local machine, even though there is only one API Gateway running.

To resolve this issue, perform the following steps:

1. Edit `cassandra.yaml` and `client.yaml`.
2. Change `localhost` to IP name or address.
3. Restart API Gateway.

For more details on Cassandra configuration, see [Configure Apache Cassandra KPS storage on page 34](#).

UTF-8 characters not displaying correctly in kpsadmin

This is a known limitation of the Windows command console. To view UTF-8 data correctly, run `kpsadmin` in a Cygwin console instead. For more details on `kpsadmin`, see [Manage a KPS using kpsadmin on page 28](#).

FSUTIL utility bug

On Windows, the following error may occur:

```
The FSUTIL utility requires a local NTFS volume. at
org.apache.cassandra.utils.FBUtilities.exec(FBUtilities.java:573)
```

This is an Windows XP issue, which also applies to NTFS (not just FATxx). For more details, see:

<http://support.microsoft.com/kb/322275>

Path length issue >255 characters

On Windows, you may see an error such as the following:

```
java.io.IOException: Exception while executing the command: cmd /c
mklink...
command error Code: 1, command output: The system cannot find the path
specified.
```

To resolve this issue, perform the following steps:

1. Use a short path for Cassandra data.
2. Update the `cassandra.yaml` data directories:

```
data_file_directories: [dir-x]
commitlog_directory: dir-x
saved_caches_directory: dir-x
```

3. Restart the API Gateway.

For more details on Cassandra configuration, see [Configure Apache Cassandra KPS storage on page 34](#).

Could not drop kps keyspace using cassandra-cli

On Windows, you may see this error which is caused by Apache thrift.

To resolve this issue, you should drop individual column families and the `kps_schema` record. For example, for a `samples_user` table, paste the following into `cassandra-cli`:

```
use kps;
assume kps_schema validator as utf8;
assume kps_schema comparator as utf8;
assume kps_schema keys as utf8;
assume samples_user validator as utf8;
assume samples_user comparator as utf8;
assume samples_user keys as utf8;
drop column family samples_user;
del kps_schema['samples_user'];
```

Note The `assume` commands must be executed once.

Appendix C: Apache Cassandra operations for API Gateway

This appendix describes important Apache Cassandra operations for node repair, and backup and recovery.

nodetool repair

The `nodetool repair` command repairs inconsistencies across all Cassandra replicas for a given range of data. For more details, see:

http://www.datastax.com/documentation/cassandra/1.2/cassandra/operations/ops_repair_nodes_c.html

You should execute this command weekly, at off-peak times, and stagger execution on different nodes.

The following is a simple `crontab` command that executes repair every 10 minutes:

```
vagrant@node-3:~$ crontab
*/10 * * * * /home/vagrant/from/apigateway/posix/bin/nodetool -h node-3
repair kps >> /home/vagrant/nodetool.log
```

Note This is a simple example from a development test. In production, do not execute repair every 10 minutes.

Backup and recovery

This section explains how to back up and restore all KPS data.

Back up and restore Cassandra node data

To back up and restore Cassandra data (online and HA), use the following instructions:

http://www.datastax.com/documentation/cassandra/1.2/cassandra/operations/ops_backup_restore_c.html

Note When restoring from a snapshot, you should follow the node restart method.

Back up API Gateway KPS configuration

KPS configuration is stored in API Gatewaythe group configuration in the following directory:

```
INSTALL_DIR/apigateway/group-n/conf
```

You should regularly back up this directory.

Back up API Gateway Cassandra configuration and data

API Gateway group configuration and data is stored in the following directory:

```
INSTALL_DIR/apigateway/group-n/instance-m/conf/kps/cassandra
```

This contains the Cassandra configuration (`cassandra.yaml`, `client.yaml`, and `jvm.xml` files). It also includes the Cassandra runtime data in the `data`, `saved_caches`, and `commitlog` subdirectories.

You should regularly back up this directory.

Note For more details on API Gateway backup and disaster recovery, see the *API Gateway Administrator Guide*.

Replace dead node

The procedure for a standard Cassandra installation is as follows:

http://www.datastax.com/documentation/cassandra/1.2/cassandra/operations/ops_replace_node_t.html

The procedure for API Gateway is as follows:

1. Confirm that the node is dead using `nodetool status` (see the Cassandra procedure above).
2. Note the address of the dead node (this is used in the last step).
3. Install a new API Gateway, or restore a backup of an existing API Gateway. Do not start the API Gateway.
4. The API Gateway Cassandra directory is:

```
INSTALL_DIR/apigateway/group-n/instance-m/conf/kps/cassandra
```

Copy `cassandra.yaml`, `client.yaml` and `jvm.xml` from a backup or recreate in the API Gateway Cassandra directory.

5. Backup and remove the `data`, `saved_caches`, and `commitlog` directories if they exist. `cassandra-topology.properties` is required for multi-datacentre configuration. Oracle currently do not test or support this kind of configuration. For more details on the `cassandra.yaml` file, see the standard Cassandra procedure above.

Note `auto_bootstrap` should not be listed or should be set to `false`

6. Start the API Gateway with the `-Dcassandra.replace_address` option. Ensure JMX is enabled. For example:

```
./startinstance
-g TeamA
-n api2
-Dcassandra.replace_address=192.168.99.12
-DenableJMX
```

7. Remember to remove the `-Dcassandra.replace_address` option for subsequent starts.

Further information

This topic shows where to find more details on KPS-related information:

Feature	Documentation
KPS REST API	The KPS REST API is documented in the following directory: INSTALL_ DIR/apigateway/docs/restapi/APIGatewayAPI/api/rest/kps.html
Apache Cassandra	For details on Cassandra version 1.2.x, see the following: <ul style="list-style-type: none">• http://cassandra.apache.org/• http://www.datastax.com/documentation/cassandra/1.2
Apache Cassandra parameter configuration	For details on Cassandra parameter configuration, see the following: http://www.ecyrd.com/cassandrascalculator/
API Gateway database connections	For details on how to configure database connections, see the <i>API Gateway Policy Developer Guide</i> .
API Gateway caches	For details on how to configure local caches, see the <i>API Gateway Policy Developer Guide</i> .

Glossary

Apache Cassandra

An open source distributed database, designed to provide high availability by distributing data across multiple servers.

KPS

Key Property Store. Data management component in the API Gateway.

KPS alias

An alternative name for a KPS table that is unique in an API Gateway group.

KPS collection

A collection of related KPS tables.

KPS table

A user defined table that is managed by the KPS in the API Gateway.

Selector

A special syntax that enables API Gateway configuration settings to be evaluated and expanded at runtime based on metadata values (for example, from a KPS, message attribute, or environment variable).