



API Gateway Multi-Regional Deployment

API Gateway Multi-Regional Deployment

December 2015

Copyright

Copyright © 2015 Akana, Inc. All rights reserved.

Trademarks

Akana, SOA Software, Policy Manager, Portfolio Manager, Repository Manager, Service Manager, Community Manager, Akana Intermediary for Microsoft and SOLA are trademarks of Akana, Inc. All other product and company names herein may be trademarks and/or registered trademarks of their registered owners.

Akana, Inc.

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

Disclaimer

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 The content of this document may be out of date, and Akana makes no commitment to update this content.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

Contents

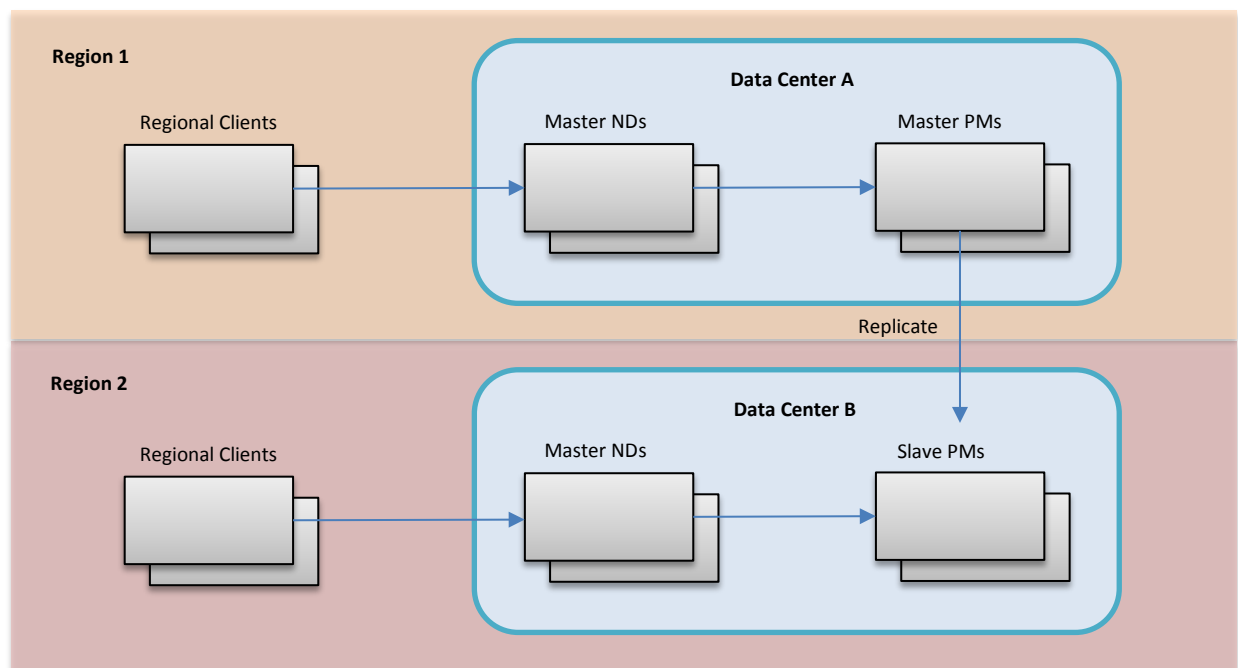
Chapter 1 Overview	4
Chapter 2 Multi-Regional Architecture	5
Master-Slave Deployment	5
Incorporation of NoSQL Databases that use Sharding	6
Propagating Network Director Configuration Information	7
How Policy Managers Gain Read Access to Network Director Data Written to Local NoSQL Shards.....	7
Chapter 3 Setting up the Data Stores	9
Configuration Servers	9
Query Routers.....	10
Database Servers	11
Shards	14
Database Driver Settings.....	17
Troubleshooting.....	18
Verify Shard Configuration.....	18
Verify Proper Data Distribution across Shards	18
Check MongoDB Process Logs.....	19
Chapter 4 API Gateway Installation and Configuration	20
Master Data Center.....	20
Install Policy Manager	20
Create a Policy Manager Cluster	21
Install Network Director	28
Slave Data Center.....	29
Policy Manager Provisioning	29

Chapter 1 | Overview

A single API Gateway deployment can span multiple regions around the globe to improve regional performance as well as provide a more seamless disaster recovery. To achieve better performance all messages from a given region should be brokered by a Network Director in the same region. For disaster recovery, it must be possible to access both the Network Director and Policy Manager in a different region if the local region has suffered a disaster.

The API Gateway does not support a full multi-master deployment across regions. Instead it supports a multi-master deployment for the Network Director processes and a master-slave deployment for the Policy Manager processes. In each region of course the Network Directory and Policy Manager processes can be deployed in clusters for scaling.

For the sake of discussion in this document the regional locations that API Gateway components are deployed to will be referred to as data centers. The following diagram illustrates the mixture of multi-master Network Directors with master-slave Policy Managers:



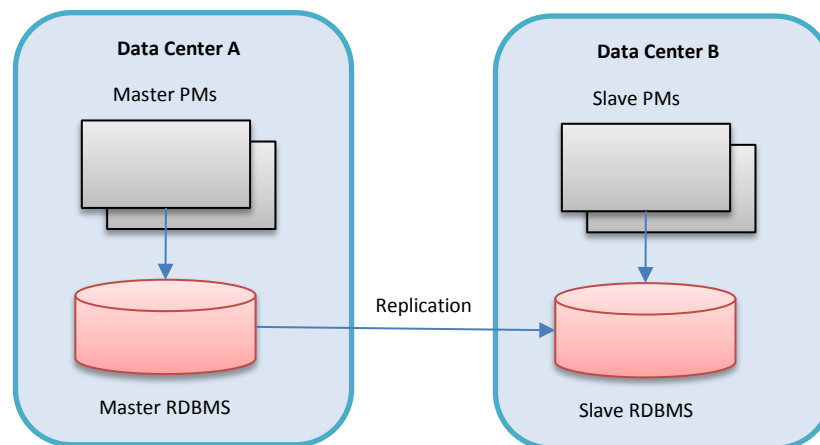
Each region has its own data center with master Network Directors that regional clients will interact with. These Network Directors will communicate with Policy Managers in their same region. Only one data center (Data Center A) will have master PM's. All other data centers will have slaves.

Chapter 2 | Multi-Regional Architecture

The key to a multi-regional software deployment is the speed of access to and integrity of persisted data. The API Gateway has traditionally persisted its data in a relational database. Although some relational database vendors claim multi-master capabilities these solutions have not proven to be 100% reliable. Instead, a more traditional master-slave deployment has proven to be the best option. This is why there is a single master Policy Manager in the multi-regional architecture.

Master-Slave Deployment

- One data center will host the master RDBMS that the cohosted Policy Manager processes will read and write to.
- The RDBMS will utilize replication to propagate changes made through the master Policy Manager to the slave databases in the other data centers.
- The Policy Manager process(es) in data centers with slave databases will be slaves themselves, only supporting read activities, no writes.



In case of a disaster that renders the data center hosting the master Policy Manager(s) inoperable the fail-over capabilities of the RDBMS will be utilized to promote a different data center's database to the new master. The Policy Manager(s) in this data center will now become the master(s) as well.

The limitation of using a master-slave RDBMS for persisting data becomes an issue for the Network Directors. The goal of our architecture is for the Network Directors in each data center to be masters, meaning that they have the exact same functionality as any other Network Director in other data centers. This functionality includes the persisting of data. Examples of data that a Network Director persists, or writes, are:

- Metrics

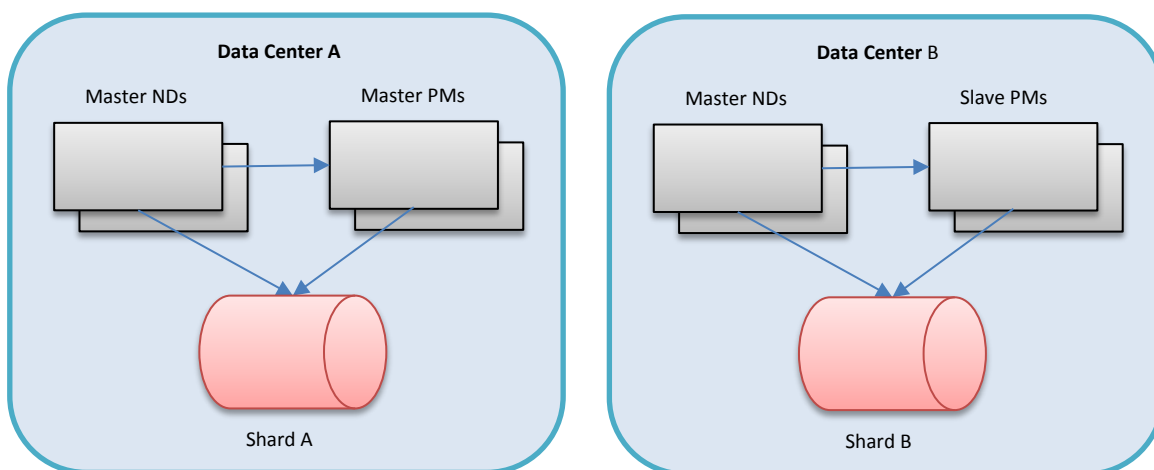
- Usage logs and recorded messages
- Security audit trails
- Alerts
- Container State

Incorporation of NoSQL Databases that use Sharding

Using the master-slave RDBMS architecture for writing this data would require the Network Directors in all data centers to communicate directly with the single master Policy Manager/RDBMS. This would lead to obvious problems with performance which is only compounded by the volume of data needing to be written during heavy client traffic periods.

To avoid this limitation the multi-regional architecture introduces a second persistence system, a NoSQL database, for the storage of the data that the Network Directors write. A NoSQL database provides more flexibility than an RDBMS in that it can support sharding.

- Sharding allows rows of the same table, or collection, to be stored in different servers in different data centers.
- The rows are segmented by a shard key.
- Each shard server is the master of rows written with a certain set of keys.
- In our architecture the shard key is the Network Director key.
- Each shard can be the master for multiple keys (to support Network Director clusters) but no more than one shard can be the master for a single key.



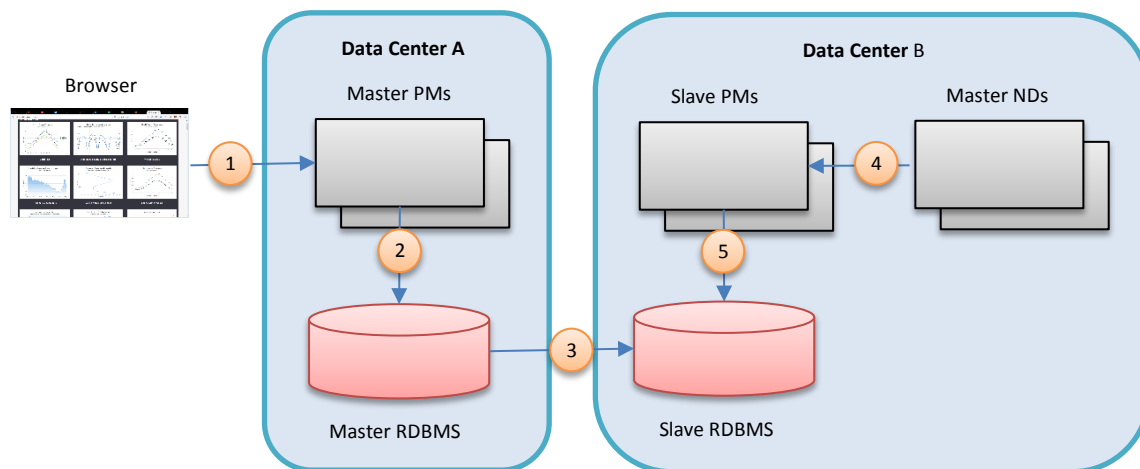
In the diagram above, each data center has its own shard of a NoSQL data store. For metrics and usage data the Network Director can write directly to the shard or use a RESTful service hosted by the Policy Manager which would write to the NoSQL shard. For the other data, the Network Director writes, there are services on the Policy Manager that the Network Director calls to insert the data into the shard. Note that even though the Policy Manager processes in data center B are slaves, they can still write to

their local shard. This architecture tries to ensure good performance by keeping all database writes local to the same data center.

The use of the NoSQL database is very well suited for the data the Network Director needs to write because the data tends to be historical data that does not have relationships to other data in the Policy Manager data model that must be kept in sync. The data does reference the Policy Manager data model to reference containers, services, operations, etc. but there aren't relational queries that Policy Manager needs to perform that would join with the historical data the Network Director is writing.

Propagating Network Director Configuration Information

There is still a need for a relational stateful model for Policy Manager however, which is why the RDBMS is still an important part of the architecture. The Network Director configuration information such as listener and virtual service configuration is still persisted in the RDBMS. Therefore the Network Directors will still be querying their configuration indirectly through the Policy Manager to the RDBMS. When changes need to be made to the Network Director configuration they must be made using another Policy Manager. The changes are then replicated to the slave database servers so that Network Directors needing that information in different data centers can read it from their local slave database server.

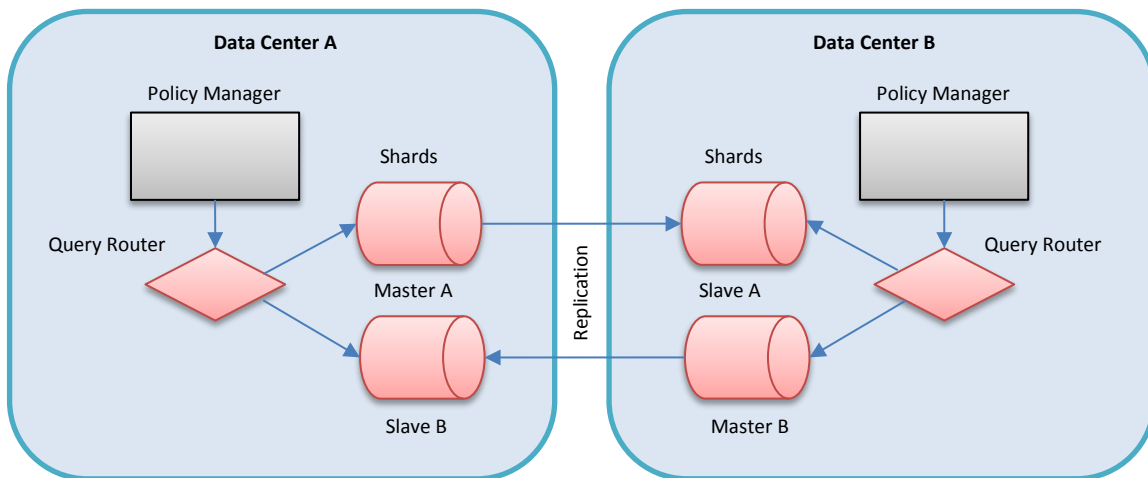


How Policy Managers Gain Read Access to Network Director Data Written to Local NoSQL Shards

Although each ND is the master for the records it writes to its local master NoSQL shard it will be necessary for Policy Managers in different data centers to at least gain read access to that data. There are two options for how this can be achieved.

- Option 1 – Through the use of a query router in each data center any query for ND written information by a local Policy Manager will be directed to the shards in the other data centers. The data will be queried from possibly multiple data centers on the fly. This option requires no duplication of data through replication. The data is always assured to be current. However, read performance may suffer due to having to route to different data centers. Furthermore, if a data center is down the data will not be accessible at all.

- Option 2 – Each master shard will replicate its content to other data centers. The other data centers will have slave shards. Through the use of a query router in each data center any query for ND written information by a local Policy Manager will be directed to the slave shards in the same data center. This requires the duplication of data through replication. It does provide the best performance for reads however queries may not get the most current information if the replication is configured to be infrequent. This is the recommended option for the multi-regional architecture.



Chapter 3 | Setting up the Data Stores

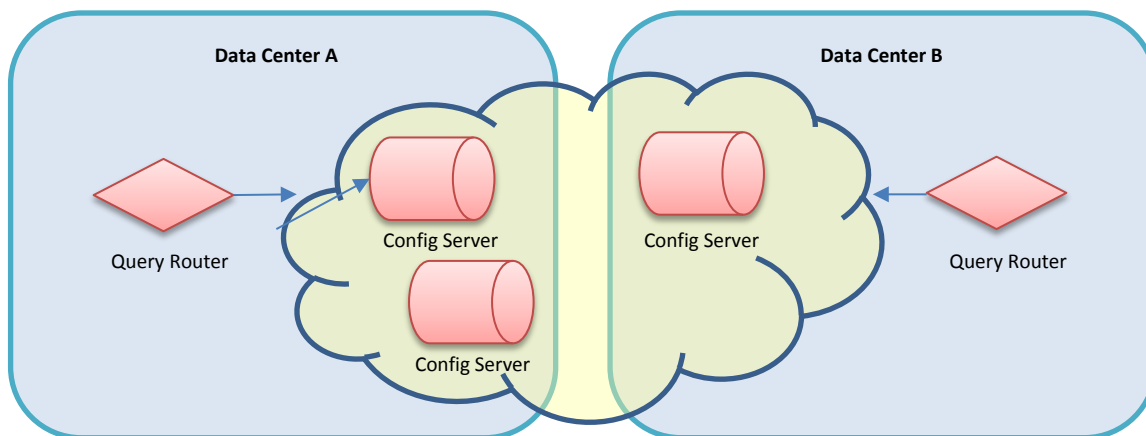
Both a relational database and a NoSQL database must be installed and configured. The relational database will be configured with a single master and replication to the slave data centers. Because the relational database vendor is a customer choice, the setup of the relational database master and slaves is out of scope for this guide. Please refer to the documentation from your vendor of choice.

The only NoSQL database vendor currently supported by the API Gateway is MongoDB. This section will outline the steps for setting up MongoDB instances in different data centers using sharding and replication.

Configuration Servers

A cluster of sharded MongoDB instances requires a set of three config servers. In order to make sharding configuration changes all three config servers must be available.

- The MongoDB query routers will make use of these configuration servers for query routing.
- At least one config server will need to be available to a query router at all times.
- To ensure they are available for query routers in case of a disaster it is recommended the config servers be installed in the different data centers along with the query routers in each data center.
- As noted though, the three config servers need to be available for configuring the sharded cluster.



The following outlines the steps for configuring a config server.

Step 1 - Create a directory where the config server will store its data.

Step 2 - Create a configuration file the config server will read on startup. This file will include the path to the data directory created in step 1, the role of this process in the cluster, and the network address of the server. An example file is shown below.

```

01) storage:
02)   dbPath: /data/configdb
03) sharding:
04)   clusterRole: configsvr
05) net:
06)   bindIp: hostA
07)   port: 27019

```

Note on line 04 that the cluster role is “configsvr” telling mongod to run as a config server and not a database server. Lines 06 and 07 identify the host and port that the config server will be listening on. The default port for all MongoDB processes is 27017.

Step 3 – Start the config server process referencing the config file as follows (where confserver.cfg is the name of the file created in Step 2):

```
mongod – config confserver.cfg
```

Repeat these steps for each of the config servers in the different data centers.

Query Routers

A query router must be configured in each data center that will route inserts and queries to the correct database servers in the MongoDB cluster. The query routers must all use the same set of config servers in order for all queries and inserts to work on consistent data across the data centers. The following are the steps to configure a query router:

Step 1 - Create a configuration file that the query router will read on startup. The configuration file will include the addresses of the config servers and the network address of the server. An example file is shown below.

```

01) sharding:
02)   configDB: hostA:27019, hostB:27019, hostC:27019
03) net:
04)   bindIp: hostA
05)   port: 27018

```

Line 02 identifies the host:port of each config server created separated by commas. Lines 04 and 05 identify the host and port the query router will be listening on. The default port for all MongoDB processes is 27017.

Step 2 – Ensure all MongoDB databases that should be part of this deployment are stopped.

Step 3 – Start the query router process referencing the config file as follows (where queryrouter.cfg is the name of the file created in Step 1):

```
mongos – config queryrouter.cfg
```

Where queryrouter.cfg is the name of the file created in Step 1.

Repeat these steps for each of the query routers in the different data centers.

Database Servers

The server instances that hold the application data are called database servers. A set of multiple database servers that are reachable from a single query router make up a cluster. Database servers in a cluster can act as shards, or partitions of the application data, that can be used for horizontal scaling. Database servers in a cluster can also act as backups, or replicas, to either provide fault tolerance or read performance improvements.

- For fault tolerance each (master), the database should replicate its data to a (slave) database in the same data center. This slave database will become the master should the original master server go down.
- To enhance read performance in all data centers, the goal is to ensure that from a single data center no read or write is performed against a server in a different data center. All interactions with the data store will be local to a single data center. At the same time, the API Gateway in a single data center should have read access to all data written by all data centers.
- To achieve this each master database will replicate its data to slave databases in all the other data centers. When reads are performed, the query router of the local data center will direct the reads to the local master database and the local slave databases. The only data traffic between data centers will be replication traffic.

There will be one replica set per master database. The following are the steps for creating a database replica set.

Step 1 - Create a directory where the database server will store its data. Do this on the host for each database server.

Step 2 - Create a configuration file the database server will read on startup. The configuration file will include the path to the data directory created in Step 1, the role of this process in the cluster, and the network address of the server. An example file is shown below:

```
01) storage:
02)   dbPath: /data/configdb
03) sharding:
04)   clusterRole: shardsvr
05) replication:
06)   oplogSizeMB: 10240
07)   replSetName: ShardA
08) net:
09)   bindIp: hostA
10)   port: 27017
```

- Lines 03 – 04 are the sharding configuration. That will be discussed in the next section.
- Lines 05 – 07 are the replication configuration. The oplogSizeMB on line 06 is the amount of space available for performing replication and should be roughly 5% of the total space used by the database.
- Line 07 gives a name to the replica set this database will be part of. There will be one master database per data center and one replica set per master, so name the replica set after the master database's data center.

- Lines 09 and 10 identify the host and port the shard server will be listening on. The default port for all MongoDB processes is 27017.

Step 3 – Start the database server process referencing the config file as follows:

```
mongod – config shardserver.cfg
```

Step 4 – In each of the other data centers create a slave database. For fault tolerance, you can create more than one in each data center including the same data center as the master database. However if you created a slave in the same data center as the master, it is possible that the secondary is used when performing reads which may have stale data. On each of the hosts for the slave servers create new data directories for the replicated data.

Step 5 – Create configuration files for each of the slave shards in the local and remote data centers. The configuration files will include the path to the data directory created in Step 2, identify the replica set they are part of, and the network address of the server. An example file is shown below:

```
01) storage:
02)   dbPath: /data/slaveconfigdb
03) sharding:
04)   clusterRole: shardsvr
05) replication:
06)   oplogSizeMB: 10240
07)   replSetName: ShardA
08) net:
09)   bindIp: hostC
10)   port: 27020
```

Note on line 07 the replica set name must be the same as the one named for the master database.

Step 6 – Start the slave databases.

Step 7 – Connect to the admin shell of the master database as follows:

```
mongo hostA:27017/admin
```

Step 8 – From the shell designate the current database as the master in the replicate set with the following command:

```
> rs.initiate()
```

Step 9 – From the shell add the slave databases to the replica set with the following:

```
> rs.add("hostC:27020")
```

Where the argument is the host and port of the slave database, do this for each database in the replica set.

Step 10 – From the shell, retrieve the replica set's configuration as it currently exists with the following:

```
> cfg = rs.config()
```

We are going to assign the config to a variable 'cfg' so that we can alter the configuration. The output of the command should look something like the following:

```
{
  "_id" : "ShardA",
  "version" : 3,
  "members" : [
    {
      "_id" : 0,
      "host" : "hostA:27017",
      "priority" : "1"
    },
    {
      "_id" : 1,
      "host" : "hostB:27017",
      "priority" : "1"
    },
    {
      "_id" : 2,
      "host" : "hostC:27017",
      "priority" : "1"
    }
  ]
}
```

Step 11 – If you choose to create slave replicas in the same data center as the master, we can avoid replicas in other data centers ever taking over the master role if the master goes down. To ensure the remote replicas do not become the master, change the priority property of each of the remote members to '0' with the following:

```
> cfg.members[2].priority = 0
```

Where 2 in this case is the _id of our remote slave database, execute this command for each of the remote slave databases in the configuration.

Step 12 – From the shell commit the configuration change with the following:

```
> rs.reconfig(cfg)
```

If there is an even number of replica servers in a replica set an arbiter must be created. An arbiter is another mongod process but it does not hold data. It must be run on a machine that is not already running a replica. To add an arbiter, perform the following steps.

Step 1 - Create a directory where the arbiter server will store its data.

Step 2 - Create a configuration file the arbiter server will read on startup. The configuration file will include the path to the data directory created in Step 1, the role of this process in the cluster, and the network address of the server. An example file is shown below.

```

01) storage:
02)   dbPath: /data/configdb
03)   journal.enabled: false
04)   smallFiles: true
05) sharding:
06)   clusterRole: shardsvr
07) replication:
08)   replSetName: ShardA
09) net:
10)   bindIp: hostD
11)   port: 27017

```

- Lines 03 – 04 are specific to arbiters.
- Lines 05 – 06 are the sharding configuration. That will be discussed in the next section.
- Lines 07 – 08 are the replication configuration. Line 08 gives a name to the replica set this arbiter will be part of.
- Lines 10 and 11 identify the host and port the shard server will be listening on. The default port for all MongoDB processes is 27017.

Step 3 – Start the arbiter server process referencing the config file as follows:

```
mongod –config arbiterserver.cfg
```

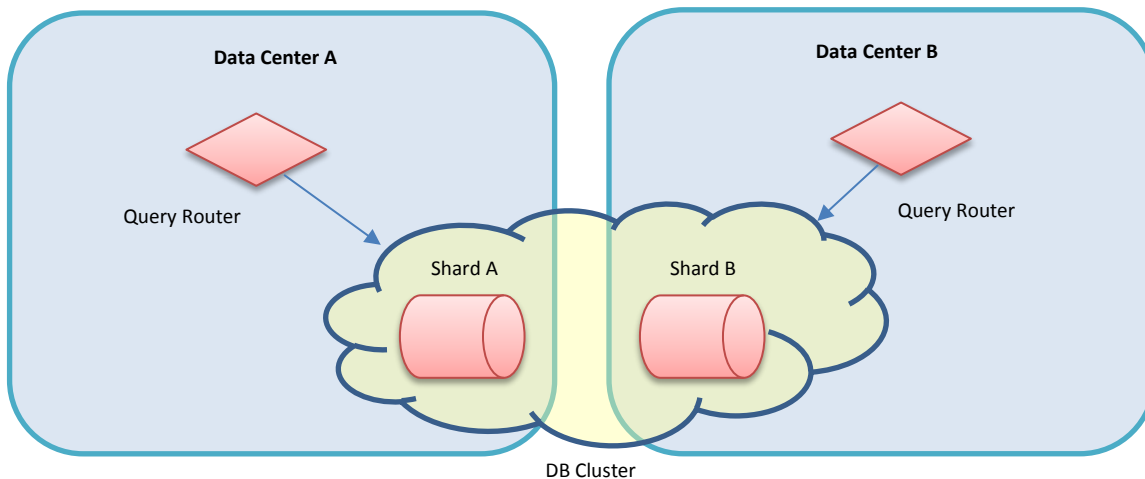
Step 4 – From the shell of the primary server add the arbiter to the replica set with the following command:

```
> rs.addArb("hostD:27017")
```

Where the only parameter is the host and port the arbiter was deployed on.

Shards

- Each data center will have at least one master MongoDB database server instance.
- Each master server is a shard.
- The collection of shards provides the complete set of application data.
- So far we have been describing a single master database, or shard, in each data center, but it is possible to have multiple shards in a single data center.
- The query routers will direct inserts and queries to the server instances based on shard keys. In this architecture the shard key will be the key of the container generating the record to be stored.



To configure the shards, complete the following steps:

Step 1 – Ensure the proper shard configuration exists in each database server’s configuration file (including the replica set slaves). The following properties should be found in each file:

```
01) sharding:
02)   clusterRole: shardsvr
```

Line 02 identifies this database server as a shard server and not a standalone database.

Step 2 – Connect to the admin shell of the local query router as follows:

```
mongo hostA:27018/admin
```

Step 3 – From the shell perform the following command for every shard to add to the cluster (including replica set slaves):

```
> db.runCommand({addShard:"ShardA/hostA:27017", name: "shardA"})
```

The parameters are the shard address and shard name. The shard address is in the format of <replica name>/<hostname>:<port>.

Step 4 – From the shell define shard tags. A shard tag is essentially an alias for the shards. You can define more than one shard in a tag but in our example there will only be one tag per shard. Execute the following command for each shard with its own tag name:

```
> sh.addShardTag("shardA", "ShardA")
```

Step 5 – From the shell enable sharding for each of the API Gateway databases as follows:

```
> sh.enableSharding("PM")
> sh.enableSharding("PM_AUDIT")
> sh.enableSharding("METRIC_RAW_DATA")
> sh.enableSharding("METRIC_ROLLUP_DATA")
```

Where PM, AUDIT_LOG, METRIC_RAW_DATA, and METRIC_ROLLUP_DATA are the databases the API Gateway uses.

Step 6 – From the shell enable sharding for each of the collections the API Gateway uses as follows:

```
> sh.shardCollection("PM.ALERTS", { "containerKey" : 1 })
> sh.shardCollection("PM.CONTAINER_STATE", { "containerKey" : 1 })
> sh.shardCollection("PM.SECURITY_AUDIT_TRAIL", { "containerKey" : 1 })
> sh.shardCollection("PM.ALERT_AUDIT_TRAIL", { "containerKey" : 1 })
> sh.shardCollection("PM_AUDIT.AUDIT", { "containerKey" : 1 })
> sh.shardCollection("PM_AUDIT.TRANSACTION_METRIC", { "containerKey" : 1 })
> sh.shardCollection("METRIC_RAW_DATA.OPERATIONAL_METRIC", { "containerKey" : 1 })
> sh.shardCollection("METRIC_ROLLUP_DATA.OPERATIONAL_METRIC", { "_id" : 1 })
```

Where the first argument of each command is the collection name prefixed by the database name, the second argument is a structure identifying the name of the property to use as the key and the type of property it is.

Step 7 – From the shell designate the range of shard tags that should be directed to each shard. You can have a shard per Network Director/Policy Manager container, one for the data center, or any combination in between. It is easier to have only one for the data center. The easiest approach is to create each of the Network Director/Policy Manager containers with numeric keys (i.e., use keys 100 – 199 for data center A and 200 – 299 for data center B) so one range can be configured only once. To define shard tag ranges perform the following commands for each shard:

```
> sh.addTagRange("PM.ALERTS", { "containerKey": "100" }, { "containerKey": "200" }, "ShardA")
> sh.addTagRange("PM.CONTAINER_STATE", { "containerKey": "100" }, { "containerKey": "200" }, "ShardA")
> sh.addTagRange("PM.SECURITY_AUDIT_TRAIL", { "containerKey": "100" }, { "containerKey": "200" }, "ShardA")
> sh.addTagRange("PM.ALERT_AUDIT_TRAIL", { "containerKey": "100" }, { "containerKey": "200" }, "ShardA")
> sh.addTagRange("PM_AUDIT.AUDIT", { "containerKey": "100" }, { "containerKey": "200" }, "ShardA")
> sh.addTagRange("PM_AUDIT.TRANSACTION_METRIC", { "containerKey": "100" }, { "containerKey": "200" }, "ShardA")
> sh.addTagRange("METRIC_RAW_DATA.OPERATIONAL_METRIC", { "containerKey": "100" }, { "containerKey": "200" }, "ShardA")
> sh.addTagRange("METRIC_ROLLUP_DATA.OPERATIONAL_METRIC", { "_id": { "containerKey": "100" } }, { "_id": { "containerKey": "200" } }, "ShardA")
```

Where the first argument of each command is the collection name prefixed by the database name, the second argument is the lower inclusive bound of the range. The third argument is the exclusive upper bound of the range. In this example we are directing all records from containers with keys of 100 – 199 to be part of this shard. The final argument is the tag of the shard created in step 6.

NOTE: It is important to make sure the keys of all containers in the system (including Policy Manager containers) are covered by a tag range. If not the query routers will choose the server to store the records without guidance and may store them on servers in a different data center entirely.

Execute the addTagRange commands for each tag, or shard, with a different range. Do not create ranges that overlap with other shard tags.

Perform the shard configuration steps for each replica set. Each replica set represents one shard cluster.

Database Driver Settings

When a read is performed through a query router, the default settings would have the query router route the query to each of the master shards in the cluster. This would not be ideal as it would cause the queries to span data centers. To avoid this we want to configure the database driver with a read preference.

The read preference is configured as a query parameter of the URI of the MongoDB connection used by a database client. Creating the URI will be part of the configuration steps in the next section. The database URI and the possible settings are described in detail at <https://docs.mongodb.org/manual/reference/connection-string/>.

The readPreference setting supports a variety of options. The option we will be using is the “nearest” option. This tells the query router to direct all reads to the closest member of a replica set regardless of whether the member is a master or slave shard. See <https://docs.mongodb.org/manual/reference/read-preference/#nearest> for more information on the “nearest” read preference.

NOTE: If you chose to have a slave replica in the same data center as the primary then, based on this read preference, if the database driver determines that the slave replica is closer than the master then the reads will be performed against the slave.

An example of a MongoDB URI with this setting is as follows:

```
monogodb://routerHost:27017/AKANA?readPreference=nearest
```

To be tolerant of query router outages you can deploy multiple query routers in a data center, creating a cluster of query routers. The database driver will fail over to a different query router in the cluster if the primary goes down. The driver must be configured with the addresses of each of the query routers to make use of this functionality by providing a comma separated list of host:port's in the MongoDB URI as follows:

```
monogodb://routerHost1:27017,routerHost2:27017,routerHost3:27017/AKANA
```

There are several other options for the database driver that can be set using the MongoDB URI. An exhaustive list of those options can be found in the MongoDB reference manual at <https://docs.mongodb.org/manual/reference/connection-string/>. For convenience, some of the more commonly used settings are listed below:

- `connectTimeoutMS` – The time in milliseconds to attempt a connection before timing out. The default is never to timeout, though different drivers might vary.
- `socketTimeoutMS` – The time in milliseconds to attempt a send or receive on a socket before the attempt times out. The default is never to timeout, though different drivers might vary.
- `maxPoolSize` – The maximum number of connections in the connection pool. The default value is 100.
- `minPoolSize` – The minimum number of connections in the connection pool. The default value is 0.
- `waitQueueMultiple` – A number that the driver multiplies the `maxPoolSize` value to, to provide the maximum number of threads allowed to wait for a connection to become available from the pool.

- `waitQueueTimeoutMS` – The maximum time in milliseconds that a thread can wait for a connection to become available.

An example of using these options is as follows:

```
monogdb://routerHost1:27017/AKANA?connectTimeoutMS=15000&socketTimeoutMS=1500000&maxPoolSize=100&waitQueueMultiple=5&waitQueueTimeoutMS=10000
```

Troubleshooting

The following subsections outline steps to help diagnose problems with the data store configuration.

Verify Shard Configuration

Step 1 – Connect to the admin shell of the local query router as follows:

```
mongo hostA:27018/admin
```

Step 2 – From the shell change the current database to the config database as follows:

```
> use config
```

Step 3 – Print a formatted report of the sharding configuration and the information regarding existing chunks in a sharded cluster as follows:

```
> sh.status()
```

The default behavior suppresses the detailed chunk information if the total number of chunks is greater than or equal to 20. For more information about this command and its output please visit <https://docs.mongodb.org/v3.0/reference/method/sh.status/>.

Step 4 – Verify that the shards have the correct hosts and ports.

Step 5 – Verify that the shards are associated with the correct replica sets.

Step 6 – Verify that the database is sharded.

Step 7 – Verify the shard key is correct.

Step 8 – Verify that the tags are split amongst the shards correctly.

Verify Proper Data Distribution across Shards

Step 1 – Connect to the admin shell of the local query router as follows:

```
mongo hostA:27018/admin
```

Step 2 – From the shell change the current database to the database to verify as follows:

```
> use METRIC_ROLLUP_DATA
```

Step 3 – Print the data distribution statistics for the [sharded](#) collection to verify as follows:

```
> db.OPERATIONAL_METRIC.getShardDistribution()
```

For more information about this command and its output please visit <https://docs.mongodb.org/v3.0/reference/method/db.collection.getShardDistribution/>.

Step 4 – Verify the statistics for each of the shards matches with your expectations based on the data being stored.

Check MongoDB Process Logs

By default each MongoDB process (query router, configuration server, database server, and arbiter) will write diagnostic and error information to the shell it was started from. Each process can be configured to write this information to a log file instead.

To instruct a process to write to a log add a systemLog section its configuration file like below:

```
01) systemLog:  
02)   path: /logs/shardA
```

Line 01 starts the systemLog section. Line 02 provides a location log files will be written. There are several other options for logging that are described in detail at <https://docs.mongodb.org/manual/reference/configuration-options/>.

Flush a Query Router's Configuration - If there is a concern that a query router is using stale information because its cache is out of date you can flush the cache and force the query router to re-read the config database. To flush a query router's cache perform the following steps.

Step 1 – Connect to the admin shell of the query router as follows:

```
mongo hostA:27018/admin
```

Step 2 – From the shell flush the cache as follows:

```
> db.adminCommand("flushRouterConfig")
```

Chapter 4 | API Gateway Installation and Configuration

In this chapter:

- We'll walk through the steps for installing and configuring the Policy Manager and Network Director processes in two different data centers.
- We'll designate one data center as the master data center and the other as the slave to denote the location of the master relational database.
- Basic installation steps will be omitted as they can be found in the standard installation instructions for the product.

Master Data Center

The master data center is home to the master relational database. This is the only database instance that can support writes. The following subsections outline the steps for setting up the Master Data Center.

Install Policy Manager

First, as in a typical installation, we will install a container with the Policy Manager Console and Subsystems features.. Because we are going to use MongoDB for reading and writing data generated by the Network Director we must perform an additional step of configuring this container to use MongoDB.

To configure any Akana container to communicate with MongoDB, you must install the *Akana MongoDB Support* plugin. You can find the plugin on the *Available Features* page of the Akana Administration Console using the *Plug-in* filter.

(You are logged in as administrator\Admin Console) [Logout](#)

AVAILABLE FEATURES | INSTALLED FEATURES | CONFIGURATION | REPOSITORY | SYSTEM

Filter: Plug-in

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

[Install Feature](#)

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



Configure MongoDB Database Wizard

To configure the database connection, specify the Username / Password of the MongoDB database you will be connecting to, and specify the host name and port where the MongoDB database is installed in the MongoClientURI field.

MongoClientURI:

In this deployment the URI we will use the URI to the MongoDB query router installed in the local data center as outlined in Chapter 3. At the conclusion of this configuration (followed by a restart) we will have a Policy Manager process that can write to the master relational database and read/write from the local MongoDB server farm through the query router.

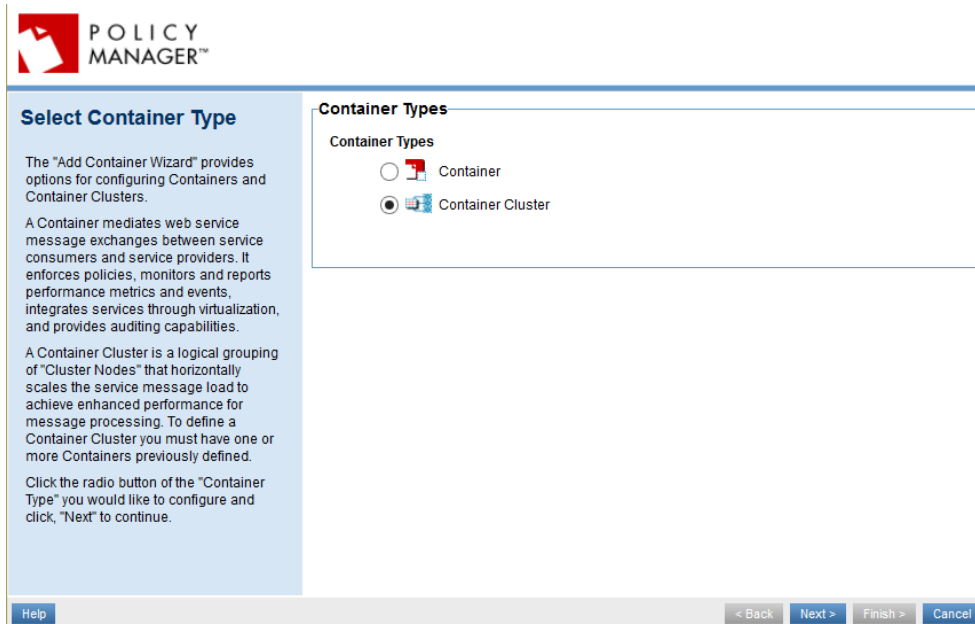
Create a Policy Manager Cluster

Ultimately we will have Network Directors in different data centers that we would like to communicate to only with locally installed Policy Manager containers. In order to do ensure this we need to create Policy Manager clusters in each data center so that the Policy Manager containers can be separated from the Network Directors' perspectives.

- 1 From the Policy Manager Management Console navigate to the *Akana Policy Manager* organization. Select the **Add Container** action.

The screenshot shows the Akana Policy Manager Management Console interface. At the top, there is a navigation bar with tabs: DASHBOARD, WORKBENCH, ALERTS, SECURITY, AUDITING, and CONFIGURE. Below this is a search bar and a breadcrumb trail: Details > Contacts > Identifiers > Categories > Monitoring > Authorization > Security. The main content area is divided into three sections. On the left is the 'Organization Tree' showing a hierarchy of organizations and their components. The middle section displays the 'Organization Overview' for 'Akana Policy Manager', including details like Parent Organization, Organization Name, Type, Description, and Statistics. On the right is the 'Actions' menu, which lists various actions available for the selected organization. The 'Add Container' action is highlighted with a mouse cursor. Below the Actions menu is the 'Workflow Tasks' section, which currently shows 'None Found'.

- This will start the *Add Container* wizard. On the first page of the wizard select the *Container Cluster* option and select **Next**.



POLICY MANAGER™

Select Container Type

The "Add Container Wizard" provides options for configuring Containers and Container Clusters.

A Container mediates web service message exchanges between service consumers and service providers. It enforces policies, monitors and reports performance metrics and events, integrates services through virtualization, and provides auditing capabilities.

A Container Cluster is a logical grouping of "Cluster Nodes" that horizontally scales the service message load to achieve enhanced performance for message processing. To define a Container Cluster you must have one or more Containers previously defined.

Click the radio button of the "Container Type" you would like to configure and click, "Next" to continue.

Container Types

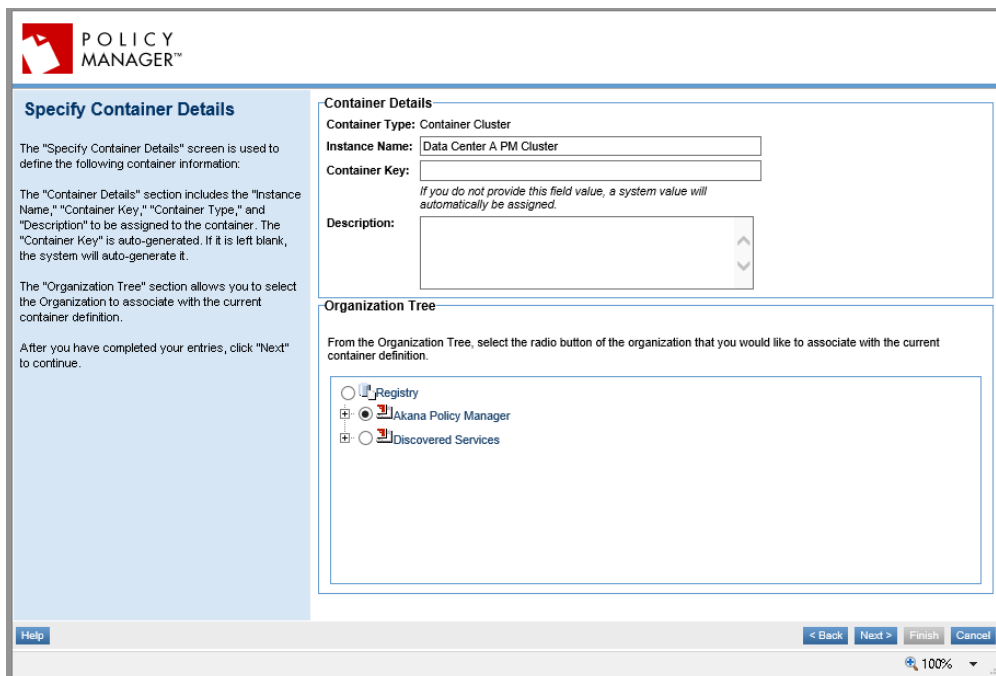
Container Types

☐ Container

☒ Container Cluster

Help < Back Next > Finish > Cancel

- On the next page provide a name of the PM cluster for this data center and an option key and description.



POLICY MANAGER™

Specify Container Details

The "Specify Container Details" screen is used to define the following container information:

The "Container Details" section includes the "Instance Name," "Container Key," "Container Type," and "Description" to be assigned to the container. The "Container Key" is auto-generated. If it is left blank, the system will auto-generate it.

The "Organization Tree" section allows you to select the Organization to associate with the current container definition.

After you have completed your entries, click "Next" to continue.

Container Details

Container Type: Container Cluster

Instance Name:

Container Key:

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

Description:

Organization Tree

From the Organization Tree, select the radio button of the organization that you would like to associate with the current container definition.

☐ Registry

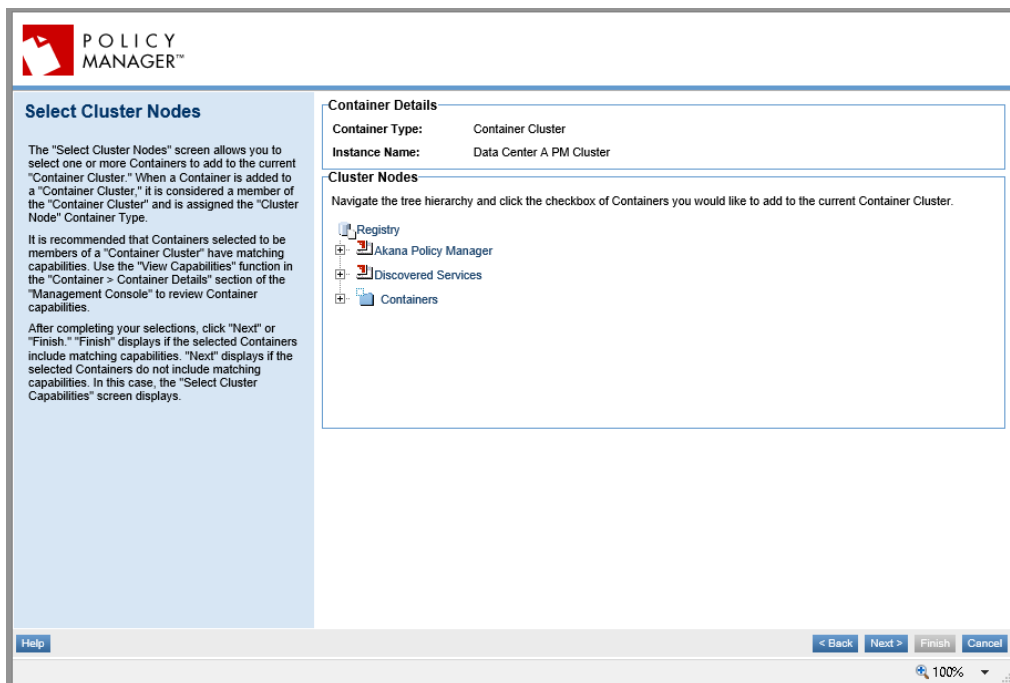
☒ Akana Policy Manager

☐ Discovered Services

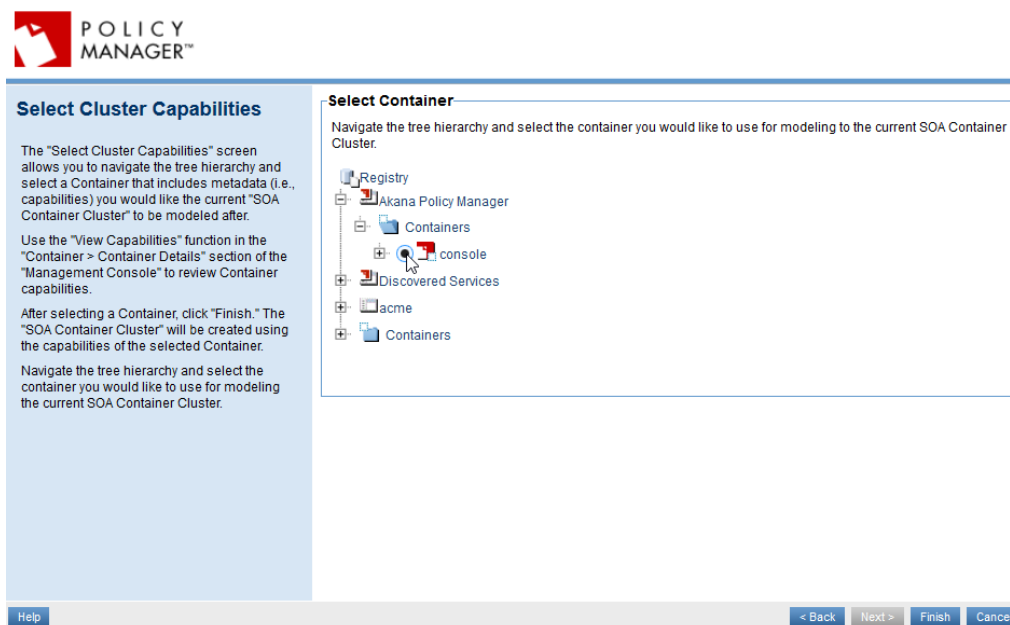
Help < Back Next > Finish > Cancel

100%

- The following page prompts you to add containers to the cluster. Do not select a container to add to the cluster at this time and select **Next**.



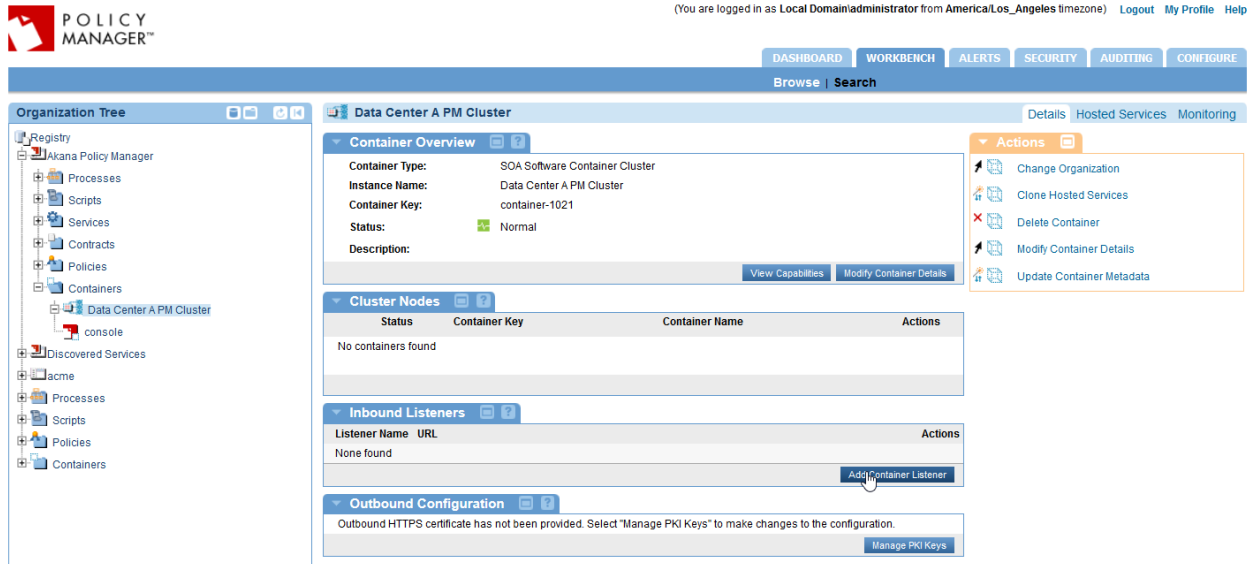
- 5 The last page of the wizard prompts you to select a container that has the capabilities you wish the cluster to also have.



- 6 Select the Policy Manager container that you have just installed and configured and select **Finish**.
- 7 Next we need to create a listener for the cluster container. This listener will be the address that the local Network Directors will use to access the Policy Manager container(s). If only one Policy Manager container will be installed in the data center then that Policy Manager container's listener information can be copied here. If more than one Policy Manager container will be used a load

balancer should be used between the Network Director(s) and the Policy Manager(s). We would then want to model the load balancer's interface for the cluster's listener.

- 8 To create a listener select the **Add Container Listener** button in the *Inbound Listeners* portlet.

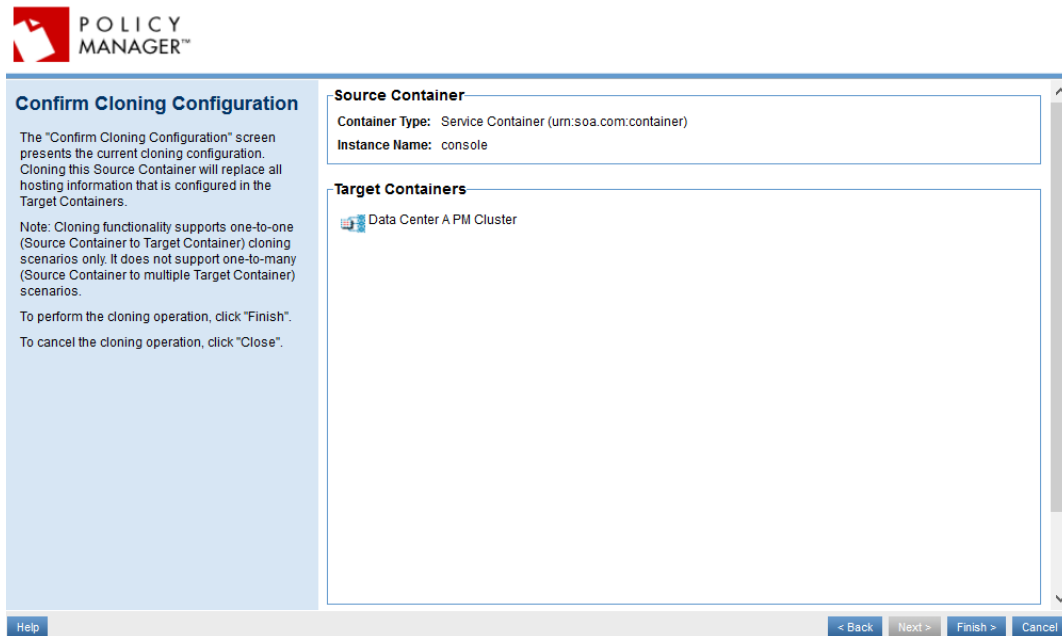


This will start the *Add Container Listener* wizard. For the sake of brevity we will not walk through all the pages of the *Add Container Listener* wizard. However, very importantly, the name of the listener created for the cluster **MUST** be the same name as the listener of the Policy Manager container(s) that will be in the cluster. If there are multiple Policy Manager containers in the cluster they **MUST** all share the same listener names.

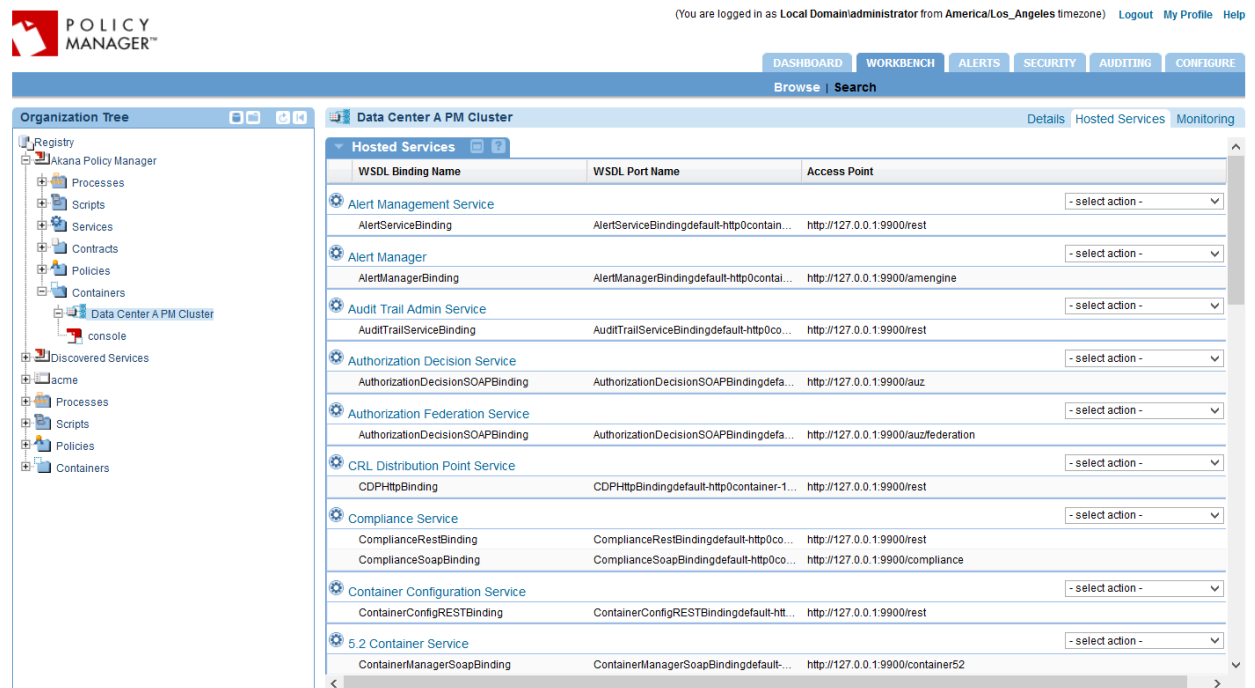
- 9 The next step is to host all the Policy Manager services on the cluster. To do this first select the original Policy Manager container in the tree. Then select the **Clone Hosted Services** action on the right.

This will start the *Clone Hosted Services* wizard. On the first page select the cluster as the target container and select **Next**.


The final page is just a confirmation page. Select **Finish** to confirm and complete.



- 10 At this point you should be able to select the cluster in the tree and the *Hosted Services* tab and see all the Policy Manager services listed.



- 11 Finally we want to place the original Policy Manager container in the cluster. Select the original container and the **Add Container to Cluster** action on the right.


 (You are logged in as Local Domain administrator from America/Los_Angeles timezone) [Logout](#) [My Profile](#) [Help](#)

[DASHBOARD](#) [WORKBENCH](#) [ALERTS](#) [SECURITY](#) [AUDITING](#) [CONFIGURE](#)

Browse | Search

Organization Tree console Details Hosted Services Monitoring

Container Overview
 Container Type: Service Container
 Instance Name: console
 Container Key: 3c868513-f3de-453d-baca-3f343934
 Status: Normal
 Description: Policy Manager Console and Services Container
 Administration Console: <http://10.2.20.143:9900/admin>

Inbound Listeners

Listener Name	URL	Actions
default-http0	http://TGullotta-E7440.local.akana.com:9900	- select action -

Outbound Configuration
 Outbound HTTPS certificate is present. Select "Manage PKI Keys" to make changes to the configuration.

Actions
[Add Container to Cluster](#)
[Change Organization](#)
[Clone Hosted Services](#)
[Delete Container](#)
[Manage Container Certificate](#)
[Modify Container Details](#)
[Update Container Metadata](#)

This opens the *Add Container to Cluster* popup page. Select the cluster and **Apply**.

Add Container to Cluster

Container Details

Category: Service Container (urn:soa.com:container)
 Instance Name: console
 Description: Policy Manager Console and Services Container

Select Container

Click the radio button of the Container Cluster you would like the current Container to be a member of.

Container Name	Description
<input checked="" type="radio"/> Data Center A PM Cluster	

[Help](#) [Cancel](#) [Apply](#)

Now you have a fully configured Policy Manager cluster with the first Policy Manager container as a cluster node.

Organization Tree

- Registry
- Akana Policy Manager
 - Processes
 - Scripts
 - Services
 - Contracts
 - Policies
 - Containers
 - Data Center A PM Cluster
 - console
 - Discovered Services
 - acme
 - Processes
 - Scripts
 - Policies
 - Containers

console

Container Overview

Container Type: Service Container

Member Of: Data Center A PM Cluster

Instance Name: console

Container Key: 3c868513-f3de-453d-baca-3f343934

Status: Normal

Description: Policy Manager Console and Services Container

Administration Console: http://10.2.20.143:9900/admin

Inbound Listeners

Listener Name	URL	Actions
default-http0	http://TGullotta-E7440.local.akana.com:9900	- select action -

Outbound Configuration

Outbound HTTPS certificate is present. Select "Manage PKI Keys" to make changes to the configuration.

Actions

- Delete Container
- Manage Container Certificate
- Modify Container Details
- Remove from Cluster
- Update Container Metadata

- Follow the previous steps for additional Policy Manager containers to provide increased scale or fault tolerance.

Install Network Director

Next we will install and configure a Network Director.

- First follow all the typical steps for installing and configuring a Network Director process. On the page for providing the *WS-MetadataExchange Options* be sure to enter the URL of the Policy Manager cluster set up in the previous section.

akana

WS-MetaDataExchange Options

The "WS-MetaDataExchange Options" screen allows you specify the URL of the Policy Manager "Metadata Exchange Service." Connecting to the "Metadata Exchange Service" enables communication between the current SOA Container instance and Policy Manager to retrieve key information (e.g., service hosting, database, etc.).

Specifying the "WS-MetaDataExchange" URL is a required installation task for "Network Director" and other Agent-based features.

In Policy Manager, the URL can be found by viewing the WSDL of the "Metadata Exchange Service". Multiple URL's can be specified for high availability. Additional entries must be comma separated.

Enter the "Metadata Exchange Service" URL and click "Finish." The "Summary" screen displays.

WS-MetaDataExchange Options

URL: http://localhost:9900/wsmex

< Back Next > Finish Cancel

- If you want all metrics and usage log information to be written from the Network Director to the Policy Manager process over a secure link you must turn off direct database access and turn on remote Policy Manager access using the Network Director's administration console.
- Set the *usage.local.writer.enabled* option to "false" and the *usage.remote.writer.enabled* option to "true" in the *com.soa.monitor.usage* configuration category.

(You are logged in as administrator\Admin Console) [Logout](#)

Configuration Categories		com.soa.monitor.usage	
com.soa.jms	rollup.queue.capacity	10000	
com.soa.jsonxml	transaction.queue.capacity	10000	
com.soa.log	usage.batch.writer.discardOldestOnOverflow	true	
com.soa.metadata.wSDL	usage.batch.writer.rollupBatchSize	50	
com.soa.metrics	usage.batch.writer.usageBatchSize	50	
com.soa.monitor.usage	usage.batch.writer.writeInterval	1000	
com.soa.mp.core	usage.local.writer.enabled	true	
com.soa.mp.core.noncache	usage.queue.capacity	10000	
com.soa.platform.jetty	usage.remote.writer.enabled	false	
com.soa.policy.framework	usage.writer.client.adapter.retryOnUnrecoverableError	false	
com.soa.policy.handle.quota.bandwidth			
com.soa.policy.handle.quota.throughput			
com.soa.policy.handle.wssp.noncache			
com.soa.policy.handler.paging			
com.soa.policy.handler.spegno			
com.soa.rhino.scriptengine			
com.soa.scheduler			
com.soa.scheduler.jobs			
com.soa.script.framework			

- If you would like the data to be written directly to the local master MongoDB shard instead then leave that option set to "false". But in order for this to work you must now install the same *Akana MongoDB Support* plugin on the Network Director container. The URI of the query router should also be used for the Network Director to shield the Network Director from possible MongoDB configuration changes down the line.
- Follow the previous steps for additional Network Director containers to provide increased scale or fault tolerance.

Slave Data Center

The installation and configuration of the containers in the slave data center is very similar to the master data center. All of the steps in the Master Data Center section apply, with the exception that we will connect to the local slave RDBMS and local MongoDB query router and along with one other major difference in how the services of the slave Policy Manager will be provisioned, outlined below

Policy Manager Provisioning

When configuring the Policy Manager Services feature in the Akana Administration Console do NOT configure an administrator name and password or select the option to Provision Services. Both of these actions require writing to the relational database and since this container is configured to use a slave database they will not work. The administrator name and password have already been set up when configuring the master data center. Service provisioning will be done a different way as we will describe next. Restart the container upon completing the configuration wizard.

- 1 We will register the slave Policy Manager container with the system as we would for a Network Director. Log in to the master data center's Policy Manager Management Console and navigate to the Akana Policy Manager organization.
- 2 Select the **Add Container** action. This will start the *Add Container* wizard. On the first page select *Container* (which is the default).

The screenshot shows the Akana Policy Manager console interface. At the top, there's a navigation bar with tabs: DASHBOARD, WORKBENCH, ALERTS, SECURITY, AUDITING, and CONFIGURE. Below this is a search bar and a breadcrumb trail: Browse | Search. The main content area is divided into three sections. On the left is the 'Organization Tree' showing a hierarchy of Registry, Akana Policy Manager, and various sub-entities like Processes, Scripts, Services, Contracts, Policies, and Containers. The middle section is the 'Organization Overview' for 'Akana Policy Manager', showing details like Parent Organization, Organization Name, Type, and Statistics. The right section is the 'Actions' menu, where 'Add Container' is highlighted. Other actions include Add Organization, Add Organization Identity, Add Policy, Change Parent Organization, Create Physical Service, Create Virtual Service, Manage Organization Identities, Offer Contract, Rename Organization, Request Contract, and View Compliance Report. Below the Actions menu is a 'Workflow Tasks' section with an 'Include Subtree' checkbox and a 'None Found' status.

The screenshot shows the 'Add Container Wizard' in the Akana Policy Manager console. The left panel is titled 'Select Container Type' and contains descriptive text about Containers and Container Clusters. The right panel is titled 'Container Types' and shows two radio button options: 'Container' (selected) and 'Container Cluster'. The 'Container' option is highlighted with a mouse cursor. At the bottom of the wizard, there are navigation buttons: '< Back', 'Next >', 'Finish >', and 'Cancel'. A 'Help' button is also present in the bottom left corner.

- 3 On the second page enter the metadata address of the slave Policy Manager container (<scheme>://<host>:<port>/metadata).

Specify Metadata Import Options

The "Specify Metadata Import Options" screen is used to specify the location of the container's metadata for a Container. Two options for specifying the Metadata location are provided: Metadata URL and Metadata Path.

To use the "Metadata URL" option, specify a URL to the metadata document describing the Container. In the "Authentication Options" section you must also select one of three authentication options including: "Anonymous," "Logged in User," or "Specify Credentials."

To use the "Metadata Path" option, click "Browse" to select the file system path of the metadata document.

Select the radio button of the Metadata Import Option and configure as appropriate. After completing your entries, click "Apply." The metadata is retrieved and parsed.

Metadata Import Options

Select the mechanism for obtaining the container's metadata document.

☒ Metadata URL:

This option is used to enter the URL address that represents the location where metadata will be retrieved.

Authentication Options

☒ Anonymous
This option does not pass user credentials to the container to retrieve its metadata.

☐ Logged in User
This option passes the current logged in user's credentials to the container to retrieve its metadata.

☐ Specify Credentials
This option passes the supplied credentials in the Username, Password, and Domain fields to the container to retrieve its metadata.

Username:

Password:

Domain:

☐ Metadata Path: No file selected.

This option is used to enter the file system path of the metadata document.

Help < Back Next > Finish > Cancel

- 4 Complete the wizard, adding any trusted certificates if prompted.

At this point we now have the new Policy Manager container in the organization tree.

- 5 Next we will create a cluster for the slave database following procedures similar to those used in the master data center. When choosing a container whose capabilities we want to model in the cluster choose the slave PM container.
 - At this point we want to clone the services to be hosted by cluster.
 - When doing this with the master data center we chose the master PM container to clone services from.
 - In the slave case the slave PM container does not have any services to clone yet. Instead we will clone services from the Data Center A PM Cluster to the Data Center B PM Cluster.
 - Remember the listener names between the two clusters MUST match in order for the cloning to work properly.
- 6 Now that we have a slave PM cluster with hosted services we can now move the slave PM container into the slave PM cluster. Once the container is moved into the cluster the services from the cluster will be provisioned for the PM container.

POLICY MANAGER™

(You are logged in as Local Domain\administrator from America/Los_Angeles timezone) [Logout](#) [My Profile](#) [Help](#)

DASHBOARD **WORKBENCH** **ALERTS** **SECURITY** **AUDITING** **CONFIGURE**

Browse | Search

Organization Tree

- Registry
 - Akana Policy Manager
 - Processes
 - Scripts
 - Services
 - Contracts
 - Policies
 - Containers
 - Data Center A PM Cluster
 - console
 - Data Center B PM Cluster
 - console2
 - Discovered Services
 - Acme
 - Processes
 - Scripts
 - Policies
 - Containers

console2

Hosted Services

WSDL Binding Name	WSDL Port Name	Access Point
Alert Management Service		
AlertServiceBinding	AlertServiceBindingdefault-http0conta...	http://localhost:9902/rest
Alert Manager		
AlertManagerBinding	AlertManagerBindingdefault-http0cont...	http://localhost:9902/amengine
Audit Trail Admin Service		
AuditTrailServiceBinding	AuditTrailServiceBindingdefault-http0...	http://localhost:9902/rest
Authorization Decision Service		
AuthorizationDecisionSOAPBinding	AuthorizationDecisionSOAPBindingd...	http://localhost:9902/auz
Authorization Federation Service		
AuthorizationDecisionSOAPBinding	AuthorizationDecisionSOAPBindingd...	http://localhost:9902/auz/federation
CRL Distribution Point Service		
CDPHttpBinding	CDPHttpBindingdefault-http0containe...	http://localhost:9902/rest
Compliance Service		
ComplianceRestBinding	ComplianceRestBindingdefault-http0...	http://localhost:9902/rest
ComplianceSoapBinding	ComplianceSoapBindingdefault-http0...	http://localhost:9902/compliance
Container Configuration Service		
ContainerConfigRESTBinding	ContainerConfigRESTBindingdefault-...	http://localhost:9902/rest
5.2 Container Service		
ContainerManagerSoapBinding	ContainerManagerSoapBindingdefau...	http://localhost:9902/container52

This performs the same step as the provisioning configuration task in the administration console. It will record the services the container is hosting in the database.

- At this point, it is highly recommended that you restart the slave PM container so it will start listening for messages for those services.
- The slave PM container will now be fully functional in a read-only manner providing all necessary services for the local Network Directors to use.

Follow the previous steps for additional slave Policy Manager containers to provide increased scale or fault tolerance.