Internal Virtualization in the Layer 7 Gateway

Contents

Introduction 1
Best Practices 2
Distinct URL for Services 2
Practices 2
Example Configurations 4
Distinct Management of Services 6
Practices 6
Example Configurations 6
Distinct Identity Providers 7
Practices 7
Example Configurations 7
Distinct Schema Imports 7
Practices 7
Example Configurations 8
Distinct Keystores 8
Practices 8
Example Configurations 9
Migration 9
Procedure 9

Introduction

This whitepaper describes the mechanisms Layer 7 Technologies provides in addition to the Gateway's native virtualization support based on standard technologies such as VMWare and Xen. The Layer 7 Gateway accommodates situations in which a hardware appliance is required for traffic pertaining to different domains or groups of users or services.

Note: In the examples throughout this document, the production and test environments are shared, but this is not considered best practice and is only included here for a worst case scenario where all environments are required to be shared.

If you require further assistance, please email *support@layer7tech.com*.

Best Practices

Distinct URL for Services

Providing a distinct URL for a service comprises of three components: a distinct Host, Port, and Path.

These criteria can be used for branching within policies. In addition, multiple services can be made to effectively support and share the same path at different ports or hosts for different groups of users.

Practices

All non-production services should be published at a custom URL that reflects the environment. For example, if a production environment service is published at /ACMEWarehouseWS, the test service would be /**test**/ACMEWarehouseWS.

A key part of this approach is using the Gateway service resolution as explained in the *Layer7 Installation and Maintenance Manual*. To override the default behavior, use the Resolve Service assertion in the "Service Availability" folder. The Resolve Service assertion is used to route to appropriate environment folders, according to the logic developed in your policy.

Below are the steps to implement this feature.

1. Create a Global Policy with type "message-received" (see Figure 1) which allows you to create logic to control the resolution path before a policy is executed. This option can be found in "Create Policy" under the menu "Tasks".

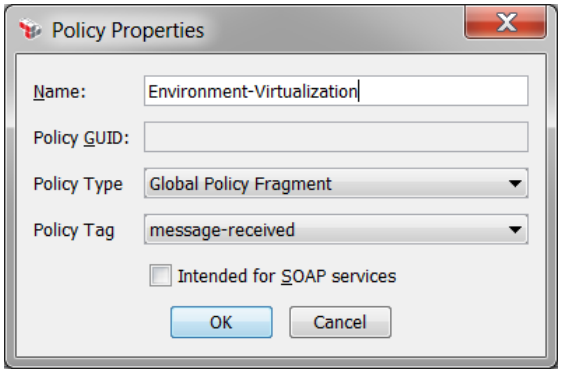

Figure 1: Global Policy for Resolving Environment Services

Note: In order for the Resolve Service assertion to execute prior to a service resolution, the assertion must be placed in a message-received global policy, or, in a policy fragment that is imported into that policy. Otherwise, the assertion will fail, since the service will have already been resolved by the time the service policy executes.

---

2. Differentiate the various environments by Host, Port or Path. For example, when you add a port through the "Manage Listen Ports" option, compare the request you get against the port, to determine to which environment it should be routed. Another method is to use the context variable *${request.tcp.localHost}*. This context variable is created when the request is received, and it returns the local Gateway side of the TCP connection's hostname.

3. Configure your policy based on the ports, hostname, or URL you have utilized, and execute the Compare Expression assertion to determine which environment will be used.

   The following is an example for a port comparison, but the same logic could be applied to a URL or hostname as well.

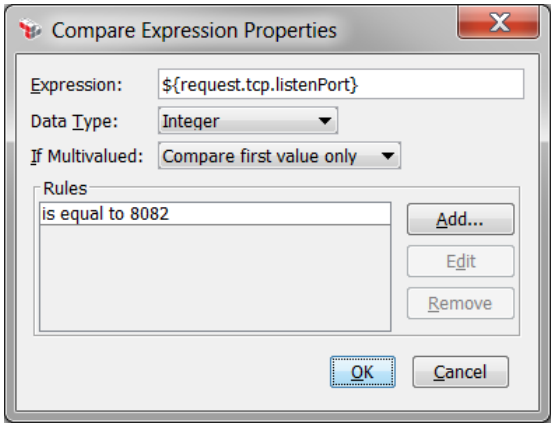

Figure 2: Compare Port for Environment Distinction

4. Insert the Resolve Service assertion in the global policy created, to override the default behavior of service resolution. For example, in Figure 2, if the condition matches with port 8082 (dedicated for development), the URI path would display the *development* environment reference (as illustrated in Figure 3).

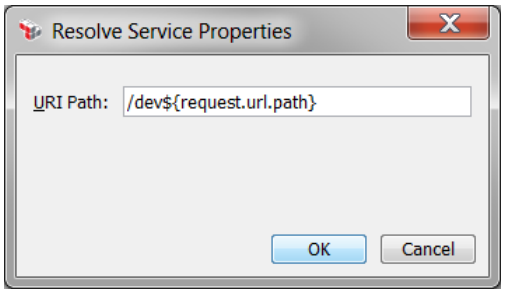

Figure 3: Resolve the Service to Environment

5. Define the various folders within the Policy Manager that separate the environments, and assign appropriate roles to users in order to control access. All services should be published at a custom URL that reflects the environment. For example, a test service would be published in folder called "test" such as /test/ACMEWarehouseWS, and a "development" service would be published in folder called "dev", such as /dev/ACMEWarehouseWS.

Some items such as cluster properties will be inseparable, since this is the same server. Thus, it is advisable to restrict access with RBAC by only allowing a single administrator to modify elements that span across configurations.

To restrict access, follow these steps.

1. Limit access to ports in the filter policy by comparing the *${request.url.port}* context variable against the unique environment ports defined. If more than one port is configured for the environment, the tests can be wrapped in an "At least one" folder (see Figure 6 for an example).
2. Limit access to a host name or IP address in the filter policy by testing the *${request.url.host}* context variable. If testing for both host name and IP the tests should be wrapped in an "At least one" folder (see Figure 6 for an example).
3. Non-production environments should be filtered to IP addresses in known test environments. This could also be managed using external firewalls to block port traffic for these environments.
4. Create a globally-writable folder labeled "Transition Folder" loaded with several service templates that can be used to test service migration. This is covered in more detail in the *Migration* section below.

Example Configurations

Figure 4 illustrates a port configuration for 3 environments for DEV, TEST and PROD.

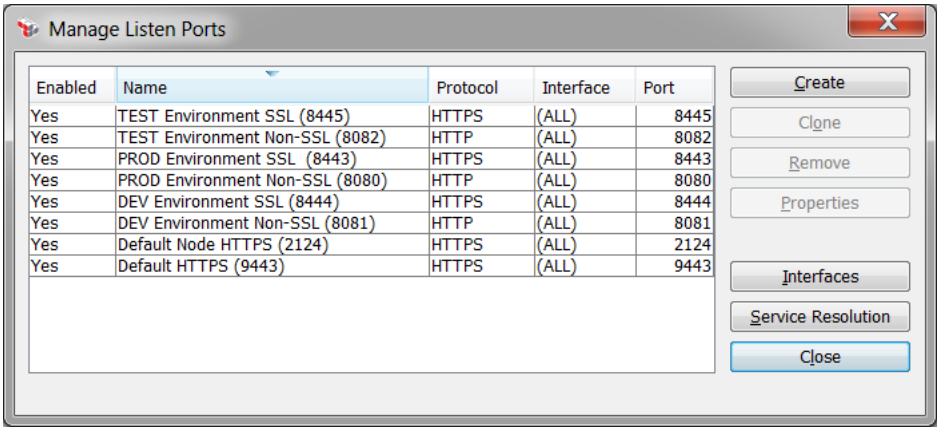

Figure 4: Manage Listen Ports

Figure 5is an example of how various environments in your services folder can be defined to filter inbound service requests to a specific environment. The environment filters are included in the global policy fragment that gets executed before a message is received.

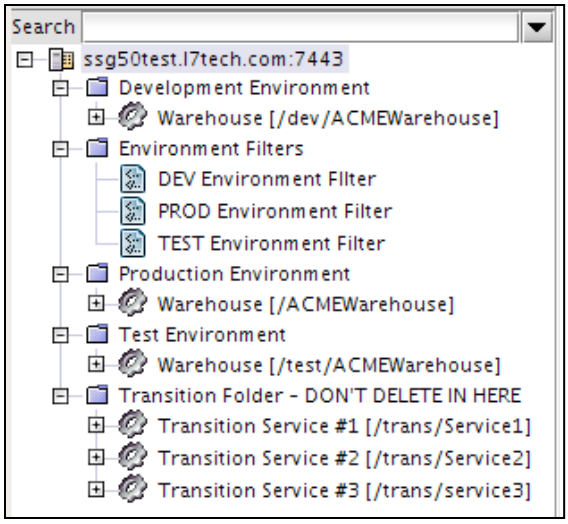

Figure 5: Environment and Transition Folders with Services

The policy fragment in Figure 6 is an example of how an environment can be filtered and included in the global "pre-message" policy created earlier. The restrictions on each environment can be different, depending on requirements. Thus, you do not necessarily need to follow this particular sample in Figure 6.

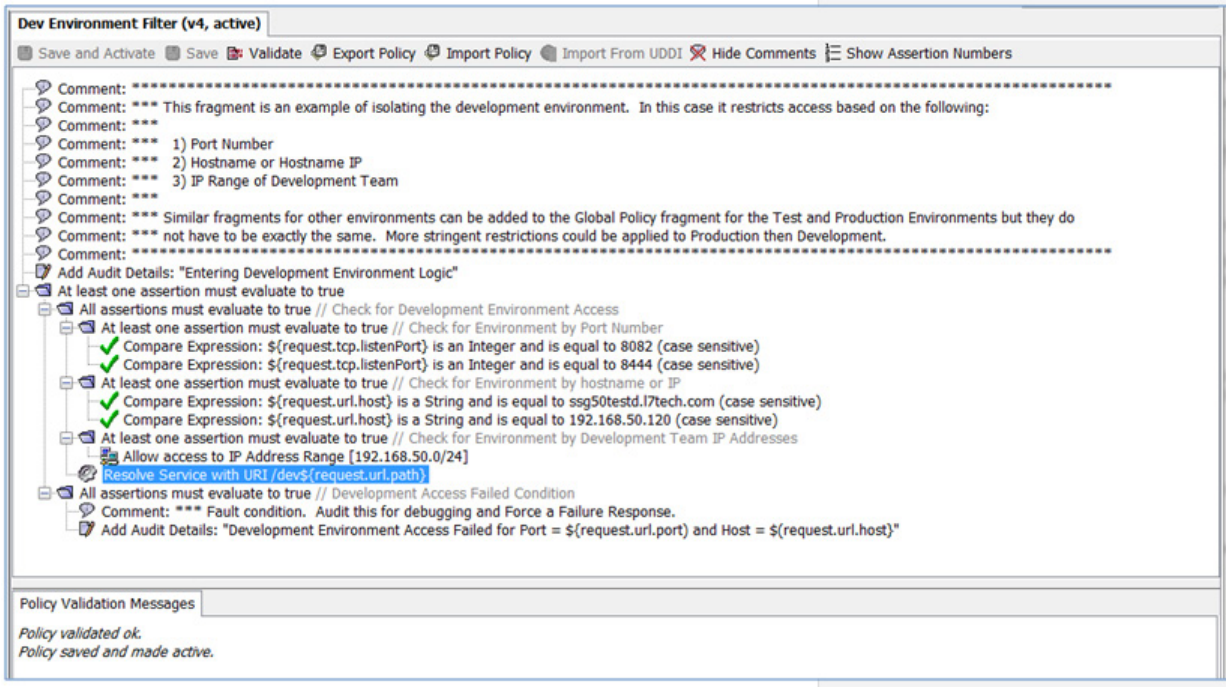

Figure 6: Policy Fragment to Restrict Access to Development

Distinct Management of Services

The current Role Based Access Control (RBAC) system allows assignment of users to view and manage roles for folders. This provides a secure way of restricting access to specific resource like Identity Providers, Folders, and Services.

Practices

- Assign all managers to the "Manage Transition Folder" role. This allows users to migrate services to the Transition folder for testing before migrating to final destination.
- Develop a process for requesting a new service that involves an administrator or a member of the "Manage Web Services" role.
- Restrict access to resources that are only necessary for that individual's role. For instance, in the Production environment, folders are not made accessible to the development or testing team.

Example Configurations

The following matrix illustrates RBAC settings for four typical users:

- admin - Admin required to publish services and move to folders
- ajones - Development environment administrator
- bsmith - Testing environment administrator
- cwilson - Production environment administrator

By design, the users that have management privileges automatically have the ability to view folders. Any identity with admin rights has the ability to manage all folders.

Table 1: Example configuration table

User	Admin	View Filter Folders	Manage Trans Folder	Manage Dev Folder	View Dev Folder	Manage Test Folder	View Test Folder	Manage Prod Folder	View Prod Folder
admin	x								
ajones (Dev)		x	x	x			x		x
bsmith (Test)		x	x		x	x			x
cwilson (Prod)		x	x		x		x	x	

Distinct Identity Providers

In the Layer 7 Policy Manager, identity providers are inherently distinct through their names. This is an advantage of Internal Virtualization in the Gateway, as identities can be leveraged from multiple environments for testing purposes.

Currently, there is no RBAC control over query access to identity providers.

Practices

- Name the identity providers so that it explicitly indicates a related environment.
- Use RBAC to control users able to manage dev, test, and production identity providers.
- Apply logic to use an appropriate identity provider based on a context variable that defines the environment in a policy (e.g. *${request.tcp.localHost} = devHost*).

Example Configurations

Figure 7 is an example of how a particular Identity Provider can be named. Based on the environment, the policy logic can be included to access the appropriate LDAP resource, so that policy changes are not required for transition between environments.

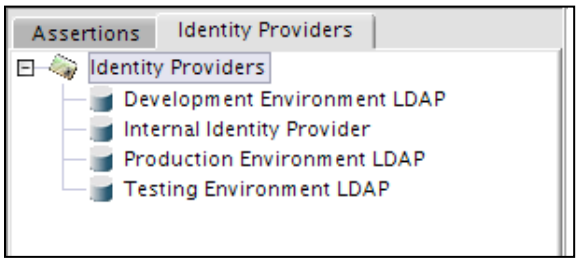

Figure 7: Example of LDAP Distinct Naming

Distinct Schema Imports

Non-production schema may import information from existing production schemas or may require new global schemas to be loaded. One advantage of Internal Virtualization in Gateway is the ability to leverage importing schemas from different environments during testing of schema validation.

Practices

- Non-production schemas should be named to indicate the associated environment as if they were a directory structure. See Figure 8 below for an example.

- Import statements into a policy-based schema that needs to be evaluated for proper global schema system IDs.

Example Configurations

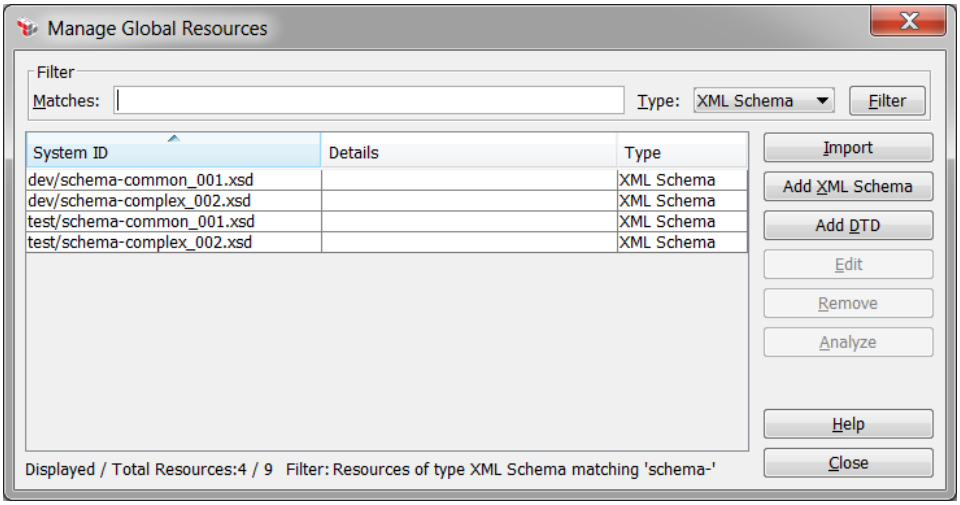

Figure 8: Global Schemas for Development and Test Environments

Distinct Keystores

The Gateway's private keys interface and configurable listeners already make it easy to associate SSL keystores to the various environments. There is no fine grained RBAC control over certificates.

Practices

- Set keystore aliases to reflect associated environments. See Figure 9 for an example.
- Develop a process for installing a new keystore or trusted certificate that involves a member of the administrator role defined through the “Manage Roles” option.

Example Configurations

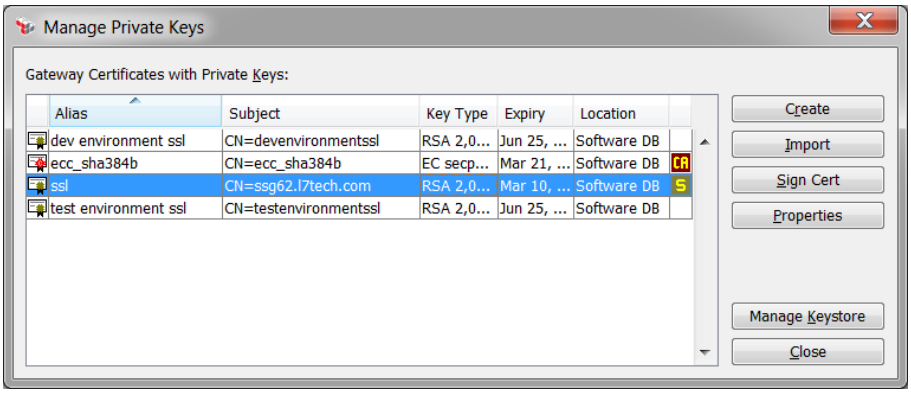

Figure 9: Configured keystores for Different Environments

Migration

To migrate services between environments leverage the Transition Folder to temporarily hold the service for policy editing and regression tests. The release process should be documented well and be detailed in defining the migration of environment independent factors such as cluster properties, identity providers, and keystores.

Procedure

1. Prepare a service in the Transition Folder by loading it with the correct WSDL for the service, and clear all assertions.
2. Select “Export Policy” from the “File” menu, and open a source service. Then export the policy to the local file system.
3. Open the prepared Transition Service Folder in the Services Panel. Import the policy from the local file system by selecting “Import Policy” from the File menu.
4. Review and confirm that the environment filter, identity assertions, routing, keystores and schema are configured as necessary for the target environment. These reviews should be defined as part of the release process for that environment.
5. Execute a pre-defined test plan as part of the release process as if it were in the new environment.
6. When the test plan has been completed successfully, export the policy to the local file system and import into the destination as explained above.
7. Test again. If a failure is found, roll back to the previous acceptable policy through the Policy Manager “Revision History” feature. You can also restore the policy from your own Configuration Management System.