TIBCO Mashery® Local Virtual Appliance Installation and Configuration Guide

Software Release 4.2
November 2017

Two-Second Advantage®

Important Information

Contents

TIBCO Documentation and Support Services

Documentation for this and other TIBCO products is available on the TIBCO Documentation site. This site is updated more frequently than any documentation that might be included with the product. To ensure that you are accessing the latest available help topics, visit:

https://docs.tibco.com

Product-Specific Documentation

The following document for this product can be found on the TIBCO Documentation site:

- TIBCO Mashery® Local Installation and Configuration Guide

For information on TIBCO Cloud Integration with Mashery, refer to Integrating with Mashery.

TIBCO Mashery Professional customers will not have access to all of the features documented here. The following is a list of capabilities that are not available and as such will not be visible within the API Control Center for these customers:

- Distributed API Management (managing Organizations)
- Enriched Call Log Export
- HTTPS Client Profiles
- Mashery Local (Deploy)
- Event Triggers

Additionally, TIBCO Mashery Professional customers will not have access to the Mashery V2 API and as such will be able to use only the OAuth2 Accelerator feature.

Additionally, TIBCO Mashery Professional includes 8M QPM (Queries per month) and all traffic purchased is limited to a max of 100 QPS (Queries per second). TIBCO Mashery Professional customers can create a max of 25 APIs and 25 packages.

How to Contact TIBCO Support

For comments or problems with this manual or the software it addresses, contact TIBCO Support:

- For an overview of TIBCO Support, and information about getting started with TIBCO Support, visit this site:

  http://www.tibco.com/services/support

- If you already have a valid maintenance or support contract, visit this site:

  https://support.tibco.com

  Entry to this site requires a user name and password. If you do not have a user name, you can request one.

How to Join TIBCO Community

TIBCO Community is an online destination for TIBCO customers, partners, and resident experts. It is a place to share and access the collective experience of the TIBCO community. TIBCO Community offers forums, blogs, and access to a variety of resources. To register, go to the following web address:

https://community.tibco.com

Installation

This guide provides an overview of the installation, requirements and configuration for Mashery® Local Virtual Appliance.

Mashery Local is an on-premise traffic manager in a virtual appliance that manages and runs your API-related network traffic behind your firewall for enhanced API response time and security. Mashery Local securely interacts with the Mashery Cloud hosted Developer Portal, Administration Dashboard and API Reporting and Analytics modules.

Mashery Local includes commercial support for Project Mashling, an open-source, event-driven microgateway. You can publish an endpoint exposed by Mashling to Mashery for access to broader API management functions. For more information about Mashling, please see https://www.mashling.io/home or https://community.tibco.com/products/project-mashling.

Assumptions

This guide assumes that you are using VMware ESX servers with the vSphere client on Microsoft Windows. If you have an internal cloud, established best practices will be applied (for example disk alignment). If you are using different servers and clients, the underlying concepts implied by the installation and configuration steps still apply.

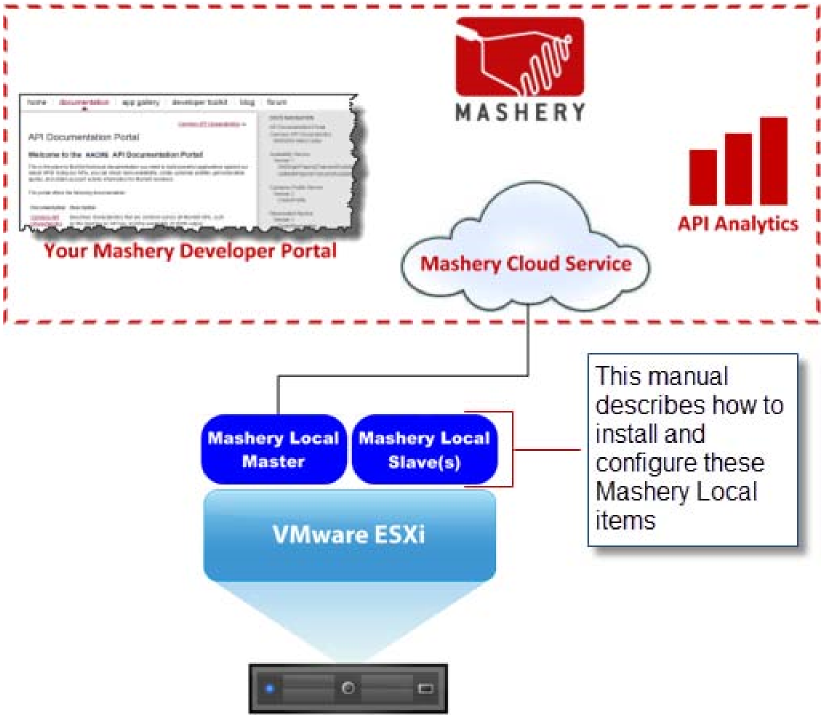
Conventions

This guide uses the following conventions:

- Keys you press simultaneously appear with a plus (+) sign between them (for example, **Ctrl+P** means press the **Ctrl** key first, and while holding it down, press the **P** key).
- Field, list, folder, window, and dialog box names have initial caps (for example, City, State).
- Tab names are **bold** and have initial caps (for example, **People** tab).
- Names of buttons and keys that you press on your keyboard are in bold and have initial caps (for example, **Cancel**, **OK**, **Enter**, **Y**).

Deployment Topology

The following diagram depicts a typical deployment topology for Mashery Local.



This manual describes how to install and configure these Mashery Local items

Hardware and Software Requirements

The following table describes the required hardware and software products and their purpose.

Mashery Local Hardware and Software Requirements

Requirement	Description
Server	• Must have VMware ESX/ESXi (4.x or greater) installed, configured and running. • Must have at minimum 2GB of RAM and 50GB of disk space per VM Instance. • Must have 1 Gbps Network connection.
Windows Client	• Must have VMware vSphere client (4.x or greater) installed. • Must have 1 Gbps Network connection to Server. • Must be routable to from upstream network equipment such as load balancers, firewalls, or routers.

Requirement	Description
TIBCO Mashery Local Virtual Appliance	Minimum Requirements: • 1 virtual CPU • 1.7GB RAM • 50GB hard drive space • 2 Virtual Network Interfaces (External eth0/Internal eth1). See the best practice recommendation in step 12 of Deploying the Mashery Local OVF Template regarding recommended roles for eth0 and eth1. • For automatic configuration, both network interfaces must have DHCP running on them. You can also configure the IP address manually by logging in via the console, editing `/etc/sysconfig/network-scripts/ifcfg-eth0`, editing `/etc/sysconfig/network-scripts/ifcfg-eth1`, and then performing a service network restart. • Masters and Slave must all be on the same virtual network on the internal interface. It is not recommended to run both the master and slave nodes in the same ESX host. In addition to availability issues, there are some processes with high IO usage that can cause temporary performance impacts for all VMs on the host. Registering a new slave node causes a performance impact on all nodes running on the same VM host. This is because there is a fortnightly database backup that impacts performance of the master node and any slaves on the same VM host. The performance impact exists for this operation, even when the master and slave nodes are on separate VM hosts. • Built for ESX 4.X and HW level 7. VMware limitations and supported hardware and software for this build include: – **Limitations**: 255 GB memory, 8 processors, 10 network adapters – **Supported hardware and software**: ESX/ESXi 4.x, vCenter 4.x, vCloud Director 1.0, Server 2.0, Workstation 6.5.x, Workstation 7.x

Requirement	Description
Web Console Configuration	- Must have IE 8 or later, Chrome, Safari, or Firefox to perform the web console configuration on the Mashery Cluster Manager UI within Mashery Local. - Necessary services and API keys have been configured in your Developer Administration Dashboard. Necessary services and API keys include service definitions in the API Settings area of the product and then the API Keys for developers to make API calls. Service definitions for Mashery define the API endpoints. The keys give you access to the endpoints. - Mashery Client Service Management (CSM) must have enabled your service to include Mashery Local and provided you a Cloud Key and Secret. For more information, contact TIBCO Support.
VMI Bundle from Mashery	Mashery CSM has provided the Mashery Local download link.

Expanding the Disk Space of a Mashery Local Instance

If the default disk space for your Master or Slave instance is not sufficient, a Mashery Local user with an Administrator role is able to expand the disk space using the following sudo commands:

```
monit stop proxy
monit stop mysql
```

Then run the sequence:

```
sudo pvresize /dev/sdb
sudo lvextend -l +100%FREE /dev/mnt_vg/mnt
sudo resize2fs /dev/mnt_vg/mnt
```

Then run:

```
monit start mysql
 monit start proxy
```

For `pvresize`, `lvextend`, `resize2fs`, and `monit`, these sudo commands should be run with root (administrative privileges).

Overview of Installation and Configuration Process

This section provides a roadmap of the installation process for Mashery Local.

Procedure

1. Ensure that you have adequately addressed the minimum hardware and software requirements, and installed the prerequisite software, as described in Hardware and Software Requirements.
2. Deploy the Mashery Local VMI bundle (OVF Template) as described in Deploying the Mashery Local OVF Template.
3. Configure a Mashery Local Master as described in Configuring a Mashery Local Master.
4. Configure slaves to the Mashery Local Master as described in Configuring Slaves to the Local Master. It is best practice to set up production with no less than 2 slaves per master.
5. Configure the load balancer as described in Configuring the Load Balancer.

6. Perform advanced configuration such as enabling notifications, LDAP, and API and JMX reporting access, as described in Advanced Configuration.

Mashery Local Failover Strategy Recommendations

Many of TIBCO customers rely on TIBCO Mashery Local to manage and distribute their revenue-generating and business-critical API traffic. The nature of the usage warrants that Mashery Local is always on without any downtime. Proper planning for redundancy and failover is recommended when high availability is expected of a mission-critical system.

The current Mashery Local architecture relies on four entities:

1. Mashery Cloud - This is where you make all your service configuration changes, through the Mashery Control Center API dashboard.
2. Mashery On-Prem Manager (MoM) - This is your Mashery Local's gateway to the Mashery Cloud. You must have received a secured key and secret, which provides each of your clusters its unique identity. You should always have a separate MoM key for each cluster, even if they are connecting to the same Mashery area.
3. A Mashery Master node synching with Cloud for API keys, OAuth Tokens, Service Configuration, User details, etc.. The time taken for synchronization of your configuration and token data is a function of the amount of data. Each customer implementation is unique and each network topology is different, so there isn't any formula that can correctly project the amount of time taken.
4. Slaves within the cluster that replicate the API Key, OAuth Token and Service Configuration data from Master. This happens within the cluster and in your environment, so the replication speed is slightly faster than Cloud Sync, but yet depends on various other environmental and data components.

TIBCO Recommendations to Achieve High Availability for TIBCO Mashery Local Deployment

Failover and redundancy can be achieved at many levels and should be considered while building your high availability strategy. Customers should also maintain updated runbooks so that their environment specific nuances are captured for their internal teams for faster deployment and recovery of systems. Failover systems should be tested and monitored in periodic intervals to ensure that they are in sync with production. Failure to do so will result in loss of traffic at the time of need. The following are some recommendations for redundancy – redundancy within a cluster and cross datacenter redundancy.

Redundancy within a Cluster:

Each Cluster has two type of Nodes – a Master Node that syncs the cluster with Cloud and many Slave Nodes that replicate from the Master. Though both type of Nodes are capable of serving traffic, TIBCO's recommendation is to keep the Master out of rotation in high traffic, high OAuth type implementations.

Keep one Slave extra than what is needed for optimum capacity to achieve within cluster redundancy.

If Master runs into problem due to disk, VM, or Network issues, then you can easily promote the spare Slaves to a Master and point the rest of the Slaves to the new Master. This can be achieved in minutes and will have a very low impact on the traffic. Except for newly synched up OAuth tokens, Slaves should be able to successfully service traffic during the promotion and pointing to the new Master. Fix the old Master Node and you can bring it back as a Slave into the cluster after re-imaging the VM.

If a Slave runs into problems, then take that Slave out of rotation from load balancer level. That way, you will not experience any traffic loss. Fix the issue and bring the slave back into rotation.

Cross Datacenter Redundancy:

If there is an issue with the datacenter, or if the whole cluster is having problems, having an Active-Active or Active-Passive cluster strategy is very beneficial in this scenario:

- **Active-Active Strategy**: Both Clusters with their unique Mashery On Prem Manager (MoM) key would connect to the same Mashery area and continue to sync. Nodes in both clusters can be used to serve traffic but both would have enough capacity (Disk Space, Caching configuration, etc.) to serve total traffic from both clusters combined and act like a failover if needed.

- **Active-Passive Strategy**: Both Clusters with their unique Mashery On Prem Manager (MoM) key would connect to the same Mashery area and will sync. Nodes from only one cluster would serve traffic. If needed traffic can be routed to the other non-traffic serving cluster without any blip in service. Both clusters should be identical in their configuration and capacity (Disk Space, Caching configuration, etc.) to serve total traffic.

Please note that TIBCO Mashery's license policy in cluster-based, so please discuss this with your Sales or Mashery Customer Success team. Having cluster redundancy is absolutely essential to avoid any traffic loss. Master sync and Slave replication takes time when done from scratch, and without cluster failover, you will encounter traffic loss.

Cluster Management in Mashery Local

The following sections describe how to set up and manage a Mashery Local Cluster:

- Setting up a new Mashery Local Cluster
- Adding a Slave to a Running Mashery Local Cluster
- Changing the Master in a Mashery Local Cluster

Setting up a New Mashery Local Cluster

The following section describes how to set up a new Mashery Local Cluster.

Procedure

Prepare mysqldump from Existing Mashery Local Cluster. (Optional)

1. For Mashery Local prior to 4.2.0, root can generate mysqldump in the Master Instance. Because proxy service needs to be stopped when mysqldump is being generated, this Mashery Local cluster cannot handle traffic.

```
monit stop proxy
mysqldump -u masherybackup -p'password_for_masherybackup' --opt --master-data --
single-transaction onprem > /mnt/onprem.sql
monit start proxy


md5sum /mnt/onprem.sql
```

   For Mashery Local 4.2.0 or later, the Administrator can generate mysqldump in the Slave or Master Instance.

   The Administrator can generate mysqldump in the Slave Instance. Because proxy service needs to be stopped when mysqldump is being generated, this Slave Instance cannot handle traffic.

```
sudo monit stop proxy
mysqldump -u masherybackup -p'password_for_masherybackup' --opt --dump-slave --
single-transaction onprem > /mnt/dump/onprem.sql
sudo monit start proxy

md5sum /mnt/dump/onprem.sql
```

   The Administrator can generate mysqldump in the Master Instance. Because proxy service needs to be stopped when mysqldump is being generated, this Mashery Local cluster cannot handle traffic.

```
sudo monit stop proxy
mysqldump -u masherybackup -p'password_for_masherybackup' --opt --master-data --
single-transaction onprem > /mnt/dump/onprem.sql
sudo monit start proxy


md5sum /mnt/dump/onprem.sql
```

Set up Mashery Local Master Instance

2. Perform the following steps:
   a) Create and configure a Mashery Local Master instance as described in the topic "Configuring a Mashery Local Master", in this Guide.

      If special tuning is needed on MySQL, for example, expanding buffer pool size in "/etc/my.cnf":

```
innodb_buffer_pool_size = 512M
```

      Mashery Local customers should consult TIBCO Support and request assistance in tuning MySQL. After tuning of MySQL, the MySQL service should be restarted by the Administrator:

```
sudo service mysqld restart
```

When disk expansion is needed, the Administrator should follow the instructions in the topic "Expanding the Disk Space of a Mashery Local Instance" in this Guide.

b) Import mysqldump from a previous Mashery Local Cluster. (Optional)

Importing mysqldump from an existing Mashery Local Cluster can minimize the amount data to synchronize from Cloud, greatly reducing the amount of time required to setup the Mashery Local Instance.

For example, suppose the remote Mashery Local instance is "remote_host".

For Mashery Local prior to 4.2.0, the administrator can copy mysqldump from remote host:

```
scp root@remote_host:/mnt/onprem.sql /mnt/dump
```

For Mashery Local 4.2.0 or later, the administrator can copy mysqldump from remote host:

```
scp administrator@remote_host:/mnt/dump/onprem.sql /mnt/dump
```

Verify the checksum of mysqldump file:

```
md5sum /mnt/dump/onprem.sql
```

Stop "proxy" service:

```
sudo monit stop proxy
```

Import mysqldump:

```
screen
mysql -u masheryonprem -p'password_for_masheryonprem' onprem < /mnt/dump/
onprem.sql
```

To detach from "screen" process, press **Ctrl-A** then **Ctrl-D**. To reattach to "screen" process, run the following command:

```
screen -r
```

After importing is done, restart MySQL:

```
sudo service mysqld restart
```

Start "proxy" service:

```
sudo monit start proxy
```

c) Register the Mashery Local Instance as Master.

Follow the instructions in the topic "Configuring a Local Mashery Master" in this guide to register the Mashery Local Instance as Master, to finish the settings for the Master in Cluster Manager.

> When synchronizing the Master for the first time, allow the Master to finish the synchronization. This ensures the Master Instance is set up properly and synchronization with the Cloud is normal.

d) Stop "proxy" Service (Optional).

After "proxy" service is stopped, there will be no activity in MySQL. This enables Mashery Local slaves to replicate faster. To stop "proxy" service, run the following command:

```
sudo monit start proxy
```

Set up the Mashery Local Slave Instance

3. Perform the following steps:

a) Create and configure a Mashery Local Slave instance as described in the topic "Configuring a Mashery Local Slave", in this Guide.

If special tuning is needed on MySQL, for example, expanding buffer pool size in "/etc/my.cnf":

```
innodb_buffer_pool_size = 512M
```

Mashery Local customers should consult TIBCO Support and request assistance in tuning MySQL. After tuning of MySQL, the MySQL service should be restarted by the Administrator:

```
sudo service mysqld restart
```

When disk expansion is needed, the Administrator should follow the instructions in the topic "Expanding the Disk Space of a Mashery Local Instance" in this Guide.

b) Register the Mashery Local Instance as a Slave.

Follow the instructions in the topic "Configuring a Local Mashery Slave" in this guide to register the Mashery Local Instance as Slave, to finish the settings for the Slave in Cluster Manager.

c) Import mysqldump from a previous Mashery Local Cluster. (Optional)

Importing mysqldump from an existing Mashery Local Cluster can minimize the amount data to synchronize from Cloud, greatly reducing the amount of time required to setup the Mashery Local Instance.

For example, suppose the remote Mashery Local instance is "remote_host".

For Mashery Local prior to 4.2.0, the administrator can copy mysqldump from remote host:

```
scp root@remote_host:/mnt/onprem.sql /mnt/dump
```

For Mashery Local 4.2.0 or later, the administrator can copy mysqldump from remote host:

```
scp administrator@remote_host:/mnt/dump/onprem.sql /mnt/dump
```

Verify the checksum of mysqldump file:

```
md5sum /mnt/dump/onprem.sql
```

Stop "proxy" service:

```
sudo monit stop proxy
```

Stop MySQL Slave:

```
mysql -u masheryonprem -p'password_for_masheryonprem' onprem
stop slave
```

Import mysqldump:

```
screen
mysql -u masheryonprem -p'password_for_masheryonprem' onprem < /mnt/dump/
onprem.sql
```

To detach from "screen" process, press **Ctrl-A** then **Ctrl-D**. To reattach to "screen" process, run the following command:

```
screen -r
```

After importing is done, restart MySQL:

```
sudo service mysqld restart
```

Start "proxy" service:

```
sudo monit start proxy
```

Ensure MySQL Slave replicates well from MySQL Master:

```
mysql -u masheryonprem -p'password_for_masheryonprem' onprem
show slave status\G
```

d) Start "proxy" Service in Master after all Slaves are set. (Optional)

Run the following command:

```
sudo monit start proxy
```

Adding a Slave to Running Mashery Local Cluster

The following section describes how to add a Slave to a running Mashery Local Cluster.

Procedure

1. Register the Mashery Local Instance as new Slave.

   Follow the instructions in the topic "Configuring a Local Mashery Slave" in this guide to register the Mashery Local Instance as Slave, to finish the settings for the Slave in Cluster Manager.

2. Exclude existing Slave or Master Instance from taking traffic.

   In order to prepare mysqldump, it is recommended that the Administrator exclude an existing Slave Instance from Load Balancer, so that the Master Instance and other Slave Instances can keep taking traffic. In the case of high availability setup, the Administrator can also switch traffic from the primary cluster to a backup cluster, then stop proxy service in the Master Instance of the primary cluster.

3. Stop "proxy" Service in existing Slave or Master Instance.

   Use the following command:

   ```
   sudo monit stop proxy
   ```

4. Initialize New Slave Database.

   a) Initialize New Slave Database by importing mysqldump file.

      The Administrator should prepare mysqldump in the existing Slave Instance.

      1. The administrator generates mysqldump in existing Slave instance:

         ```
         mysqldump -u masherybackup -p'password_for_masherybackup' --opt --dump-
         slave --master-data --single-transaction onprem > /mnt/dump/onprem.sql


         md5sum /mnt/dump/onprem.sql
         ```

      2. Or, the Administrator generates mysqldump in existing Master instance:

         ```
         mysqldump -u masherybackup -p'password_for_masherybackup' --opt --master-
         data --single-transaction onprem > /mnt/dump/onprem.sql


         md5sum /mnt/dump/onprem.sql
         ```

      3. Copy mysqldump to new Slave instance:

         ```
         scp administrator@remote_host:/mnt/dump/onprem.sql /mnt/dump


         md5sum /mnt/dump/onprem.sql
         ```

      4. Stop MySQL Slave:

         ```
         mysql -u masheryonprem -p'password_for_masheryonprem' onprem
         stop slave
         ```

      5. Import mysqldump to new Slave instance:

         ```
         screen
         mysql -u masheryonprem -p'password_for_masheryonprem' onprem < /mnt/dump/
         onprem.sql
         ```

   b) Initialize New Slave Database by streaming mysqldump.

      1. Stop Slave in new Slave Instance:

         ```
         mysql -u masheryonprem -p'password_for_masheryonprem' onprem
         stop slave
         ```

      2. Import mysqldump to new Slave Instance via Stream:

         - Stream from existing Slave Instance:

           ```
           screen
           mysqldump -h remote_host_internal_ip -u mashonpremrepl -
           p'password_for_mashonpremrepl' \
               --opt --dump-slave --single-transaction onprem \
               | mysql -u masheryonprem -p'password_for_masheryonprem' onprem
           ```

         - Stream from existing Master Instance:

           ```
           screen
           mysqldump -h remote_host_internal_ip -u mashonpremrepl -
           p'password_for_mashonpremrepl' \
               --opt --master-data --single-transaction onprem \
               | mysql -u masheryonprem -p'password_for_masheryonprem' onprem
           ```

5. Restart MySQL Service:

   ```
   sudo service mysqld restart
   ```

6. Start "proxy" Service:

   ```
   sudo monit start proxy
   ```

Changing the Master in a Mashery Local Cluster

The following section describes how to change the Master in a Mashery Local Cluster.

Procedure

1. Shut down the old Master in the Mashery Local Cluster.
   Follow the steps in the topic Shutting down a Master.
2. Promote one Slave to be the new Master.
   Follow the steps in the topic Promoting a Slave to Master.
3. Repoint other Slaves to the new Master.
   Follow the steps in the topic Repointing Other Slaves to a New Master.

Installing and Configuring Mashery Local

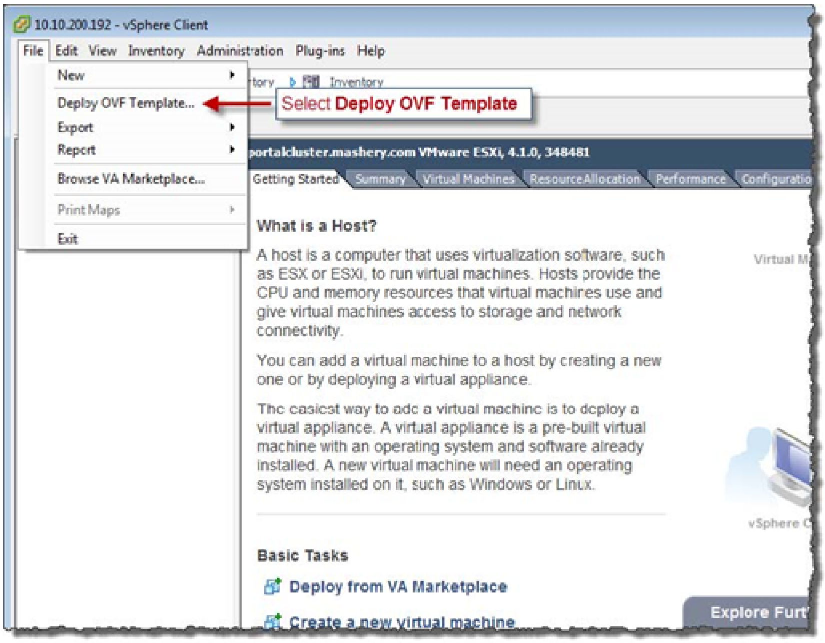
The following sections describe how to install and configure a basic environment complete with a master, slaves and load balancing.

Deploying the Mashery Local OVF Template

To deploy the Mashery Local OVF Template:

Procedure

1. Start the vSphere Client application on a computer that has access to the target ESX server.
2. Select `File` >`Deploy OVF Template`.
   The vSphere **Source** window appears.



What is a Host?

A host is a computer that uses virtualization software, such as ESX or ESXi, to run virtual machines. Hosts provide the CPU and memory resources that virtual machines use and give virtual machines access to storage and network connectivity.

You can add a virtual machine to a host by creating a new one or by deploying a virtual appliance.

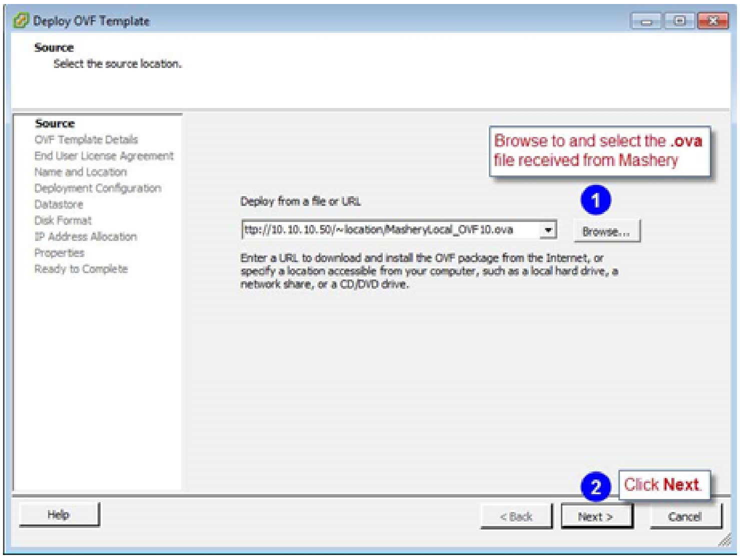
The easiest way to add a virtual machine is to deploy a virtual appliance. A virtual appliance is a pre-built virtual machine with an operating system and software already installed. A new virtual machine will need an operating system installed on it, such as Windows or Linux.

Basic Tasks

Deploy from VA Marketplace

Create a new virtual machine

3. Navigate to and select the `.ova` file you received from Mashery.



Source
Select the source location.

Deploy from a file or URL: ttp://10.10.10.50/~location/MasheryLocal_OVF10.ova

Enter a URL to download and install the OVF package from the Internet, or specify a location accessible from your computer, such as a local hard drive, a network share, or a CD/DVD drive.

Browse to and select the **.ova** file received from Mashery

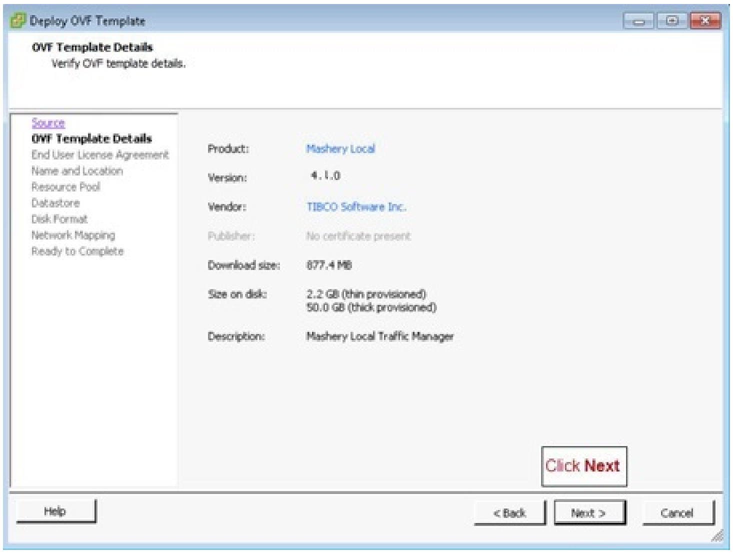
Click **Next**.

To complete this step, you will need to have previously downloaded the .ova file and stored it in a location that is accessible to the computer running vSphere.

vSphere displays the OVF template's details. For Mashery Local, the details are as follows:

- Version: 4.2.0 (or later)
- Download size: 877.4 MB
- Size on disk: 2.2 GB (thin provisioned), 50 GB (thick provisioned)

4. Click **Next**.
   The End User License Agreement appears.



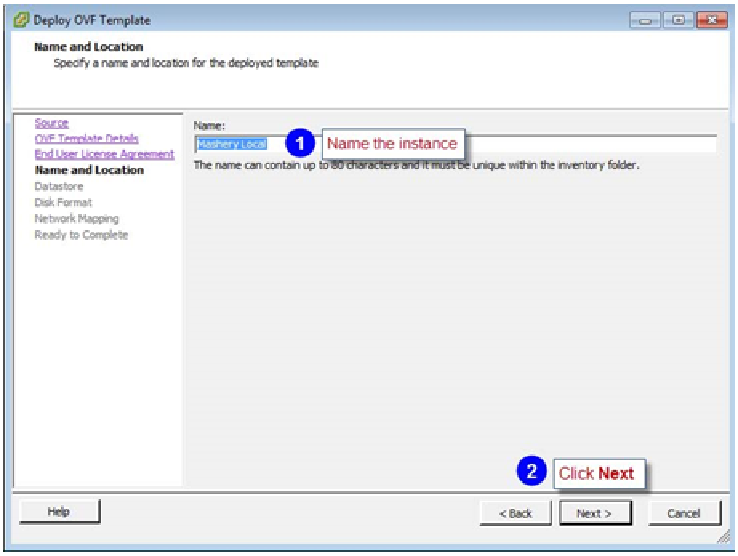
OVF Template Details
Verify OVF template details.

Product: Mashery Local
Version: 4.1.0
Vendor: TIBCO Software Inc.
Publisher: No certificate present
Download size: 877.4 MB
Size on disk: 2.2 GB (thin provisioned)
50.0 GB (thick provisioned)
Description: Mashery Local Traffic Manager

Click **Next**

5. Read and accept the End User License Agreement and click **Next**.
   The vSphere **Name and Location** window appears.
6. Run the following command to start the installation:
   Enter a name that is meaningful to your managing your virtual environment. This name has no relevance to what your users and partners will see in Mashery products but is meaningful to your administration of your ESX environment. Your operations team likely has naming conventions that should be followed. Then, click **Next**.

The **Datastore** window appears.



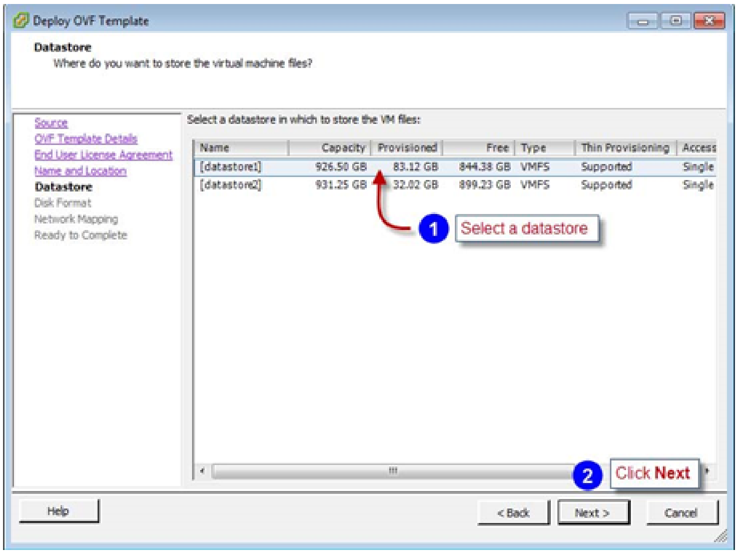
Name and Location

Specify a name and location for the deployed template

Name: Mashery Local

The name can contain up to 80 characters and it must be unique within the inventory folder.

7. Select the appropriate VM datastore, and then click **Next**.



Datastore

Where do you want to store the virtual machine files?

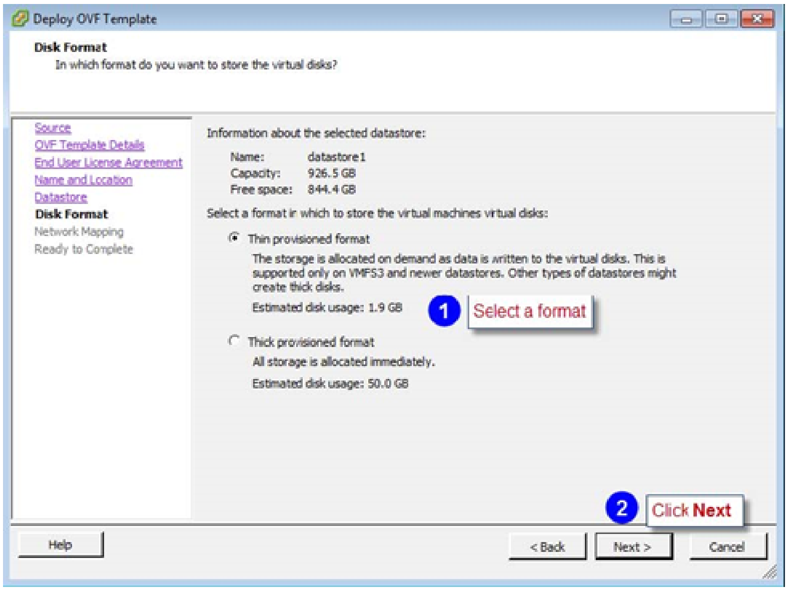
Select a datastore in which to store the VM files:

Name	Capacity	Provisioned	Free	Type	Thin Provisioning	Access
[datastore1]	926.50 GB	83.12 GB	844.38 GB	VMFS	Supported	Single
[datastore2]	931.25 GB	32.02 GB	899.23 GB	VMFS	Supported	Single

Ensure that you pick a datastore with enough free space. See Hardware and Software Requirements for guidance.

The **Disk Format** window appears.

8. Select the **Thin provisioned format** or **Thick provisioned format** option for **Disk Format** according to your needs and best practices, and then click **Next**. See this VMware performance study for guidance. Thin clients, although faster, need to be monitored closely to ensure you do not run out of space and sometimes require attention to shrink them back down. Remember also that a large number of thin provisioned VMs can affect performance due to frequent metadata updates.



Disk Format

In which format do you want to store the virtual disks?

Information about the selected datastore:

Name: datastore1
Capacity: 926.5 GB
Free space: 844.4 GB

Select a format in which to store the virtual machines virtual disks:

● Thin provisioned format

The storage is allocated on demand as data is written to the virtual disks. This is supported only on VMFS3 and newer datastores. Other types of datastores might create thick disks.

Estimated disk usage: 1.9 GB

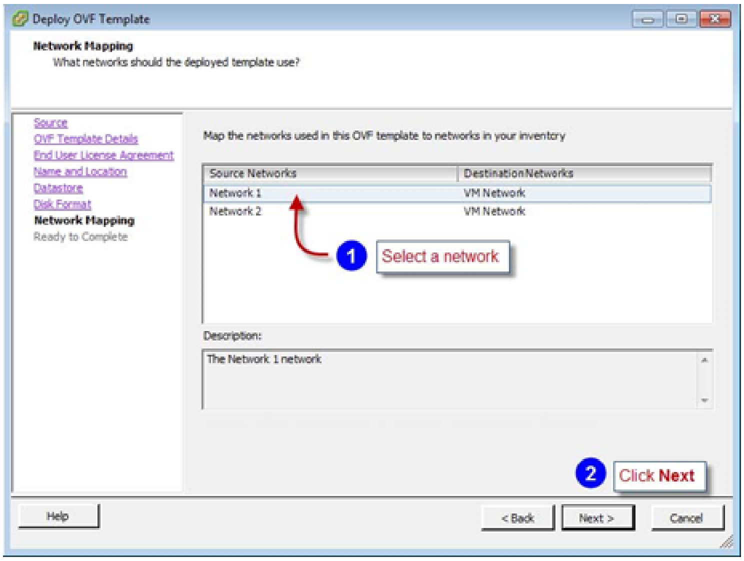
○ Thick provisioned format

All storage is allocated immediately.

Estimated disk usage: 50.0 GB

The **Network Mapping** window appears.

9. Select the appropriate Network Mapping for your environment, and then click **Next**.



Network Mapping

What networks should the deployed template use?

Map the networks used in this OVF template to networks in your inventory

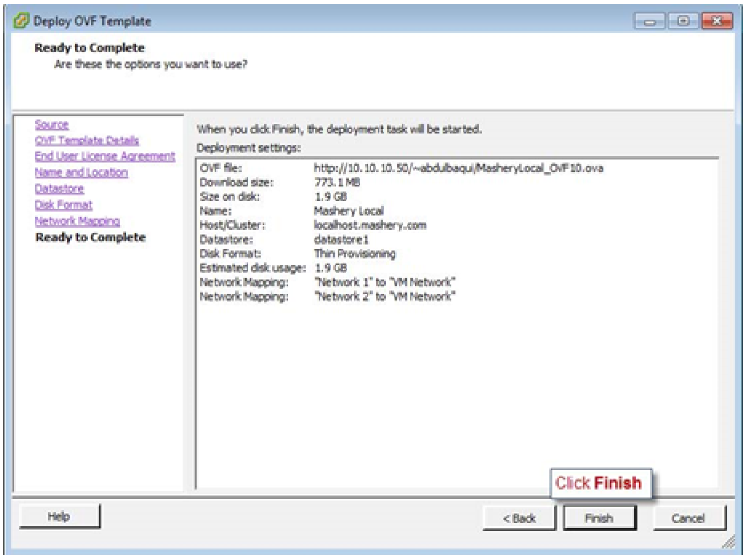
Source Networks	Destination Networks
Network 1	VM Network
Network 2	VM Network

Description:

The Network 1 network

The **Ready to Complete** window appears.

10. Click **Finish** to complete the provisioning.



Ready to Complete

Are these the options you want to use?

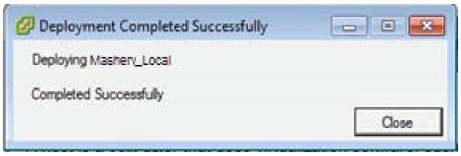
When you click Finish, the deployment task will be started.

Deployment settings:

OVF file: http://10.10.10.50/~abdulbaqui/MasheryLocal_OVF10.ova
Download size: 773.1 MB
Size on disk: 1.9 GB
Name: Mashery Local
Host/Cluster: localhost.mashery.com
Datastore: datastore1
Disk Format: Thin Provisioning
Estimated disk usage: 1.9 GB
Network Mapping: "Network 1" to "VM Network"
Network Mapping: "Network 2" to "VM Network"

Click Finish

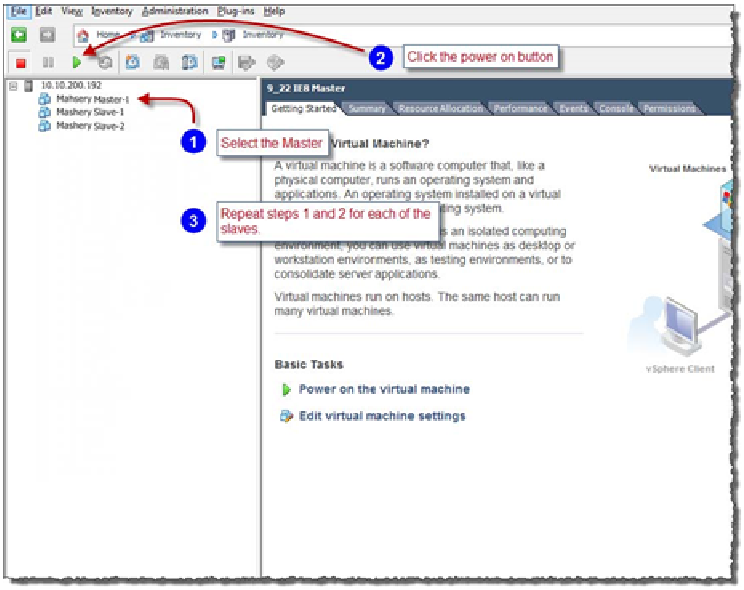
The vSphere Client reports as follows when the deployment is complete. Click **Finish**.



Deploying Mashery_Local

Completed Successfully

11. Select the **Power On** icon for the instance to boot the instance.



1 Select the Master

2 Click the power on button

3 Repeat steps 1 and 2 for each of the slaves.

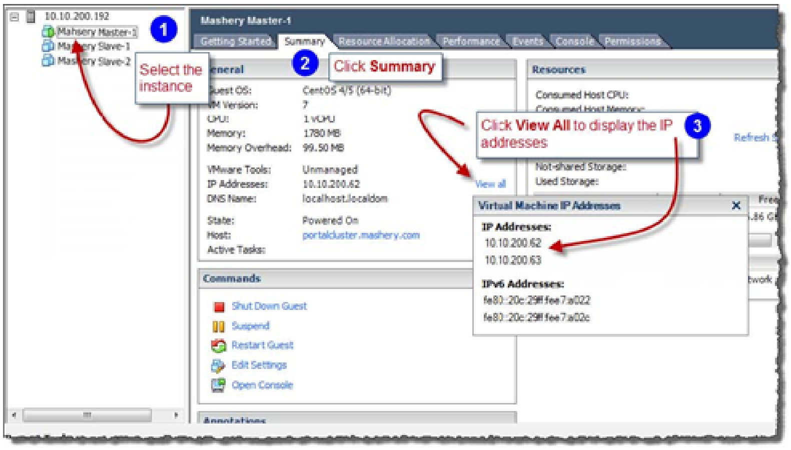
Virtual Machine?

A virtual machine is a software computer that, like a physical computer, runs an operating system and applications. An operating system installed on a virtual …ting system.

…s an isolated computing environment, you can use virtual machines as desktop or workstation environments, as testing environments, or to consolidate server applications.

Virtual machines run on hosts. The same host can run many virtual machines.

12. You need to get the IP addresses of the instance for use when performing further configuration steps within the Mashery Local appliance. Select the instance and then click **View All** to find the IP address and make a note of them.



1 Select the instance

2 Click **Summary**

3 Click **View All** to display the IP addresses

uest OS: CentOS 4/5 (64-bit)
M Version: 7
CPU: 1 vCPU
Memory: 1780 MB
Memory Overhead: 99.50 MB
VMware Tools: Unmanaged
IP Addresses: 10.10.200.62
DNS Name: localhost.localdom
State: Powered On
Host: portalcluster.mashery.com
Active Tasks:

Consumed Host CPU:
Consumed Host Memory:
Not-shared Storage:
Used Storage:

IP Addresses:
10.10.200.62
10.10.200.63

IPv6 Addresses:
fe80::20c:29ff:fee7:a022
fe80::20c:29ff:fee7:a02c

A recommended best practice is to select IP addresses as follows:

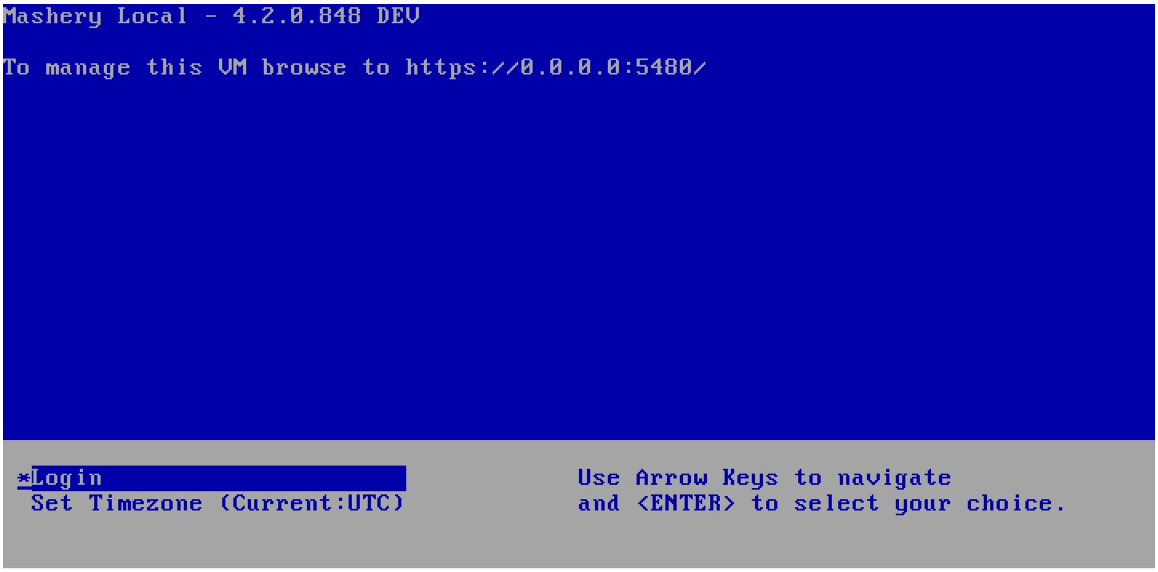
- **eth0**: use as a public interface where API calls would be coming in from the developers.
- **eth1**: use as an internal interface for the nodes to talk to each other and where nodes can interface with the backend systems.

This scheme allows you to deal with your firewall rules more easily and keep services on different subnets if needed for security reasons. Typically, the Admin UI would be handled via eth1 as it most likely would be from the internal subnet. The Slave would also be connecting to the Master on eth1 as it is internal traffic.

You are now ready to configure the Mashery Local instance. You may optionally create a number of slave instances before proceeding by repeating the previous steps for each.

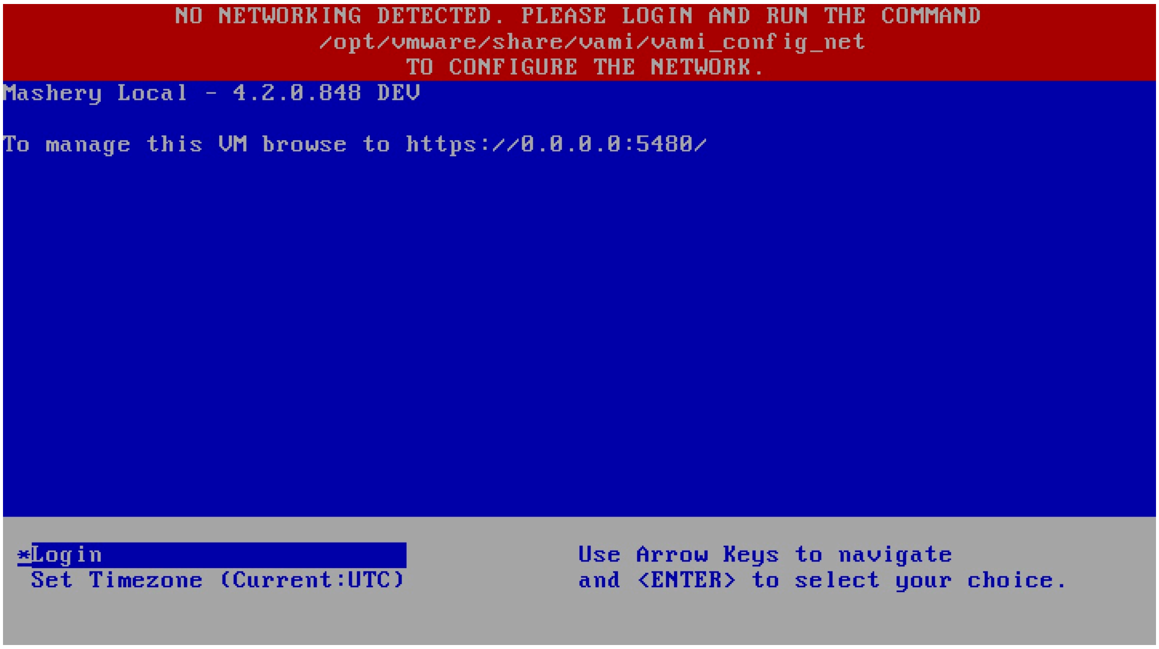
What to do next

Mashery Local VM Network Setup



```
Mashery Local - 4.2.0.848 DEV

To manage this VM browse to https://0.0.0.0:5480/

 *Login                                Use Arrow Keys to navigate
  Set Timezone (Current:UTC)           and <ENTER> to select your choice.
```

If networking is not detected, the user will see the following error message on the console:



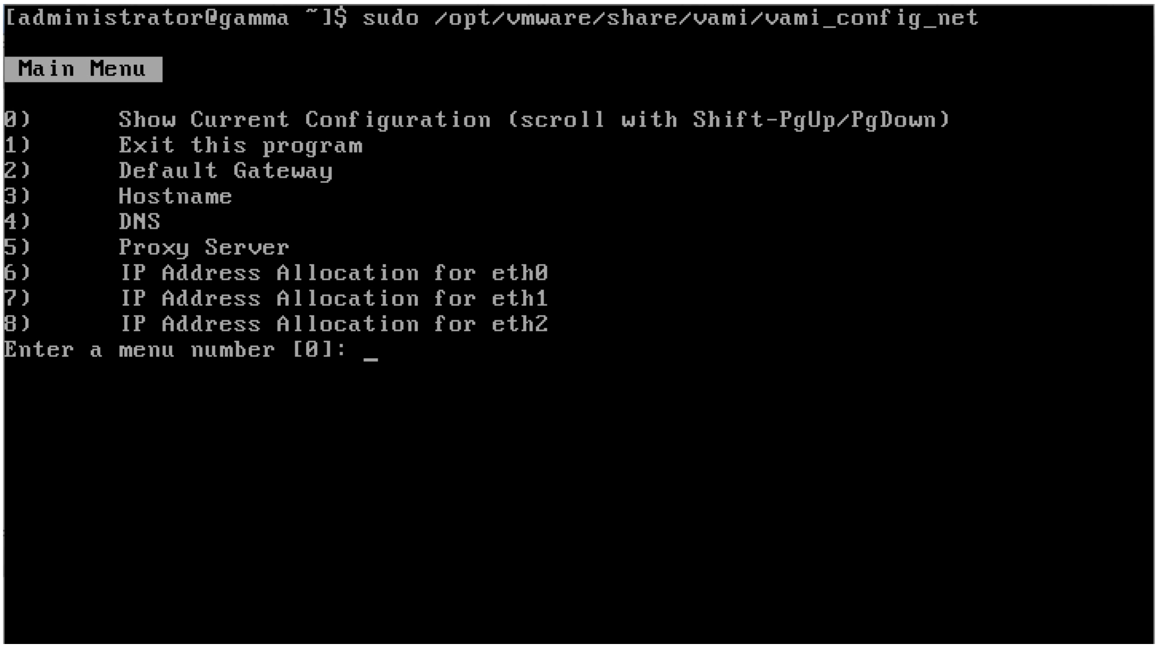
```
          NO NETWORKING DETECTED. PLEASE LOGIN AND RUN THE COMMAND
                   /opt/vmware/share/vami/vami_config_net
                         TO CONFIGURE THE NETWORK.
Mashery Local - 4.2.0.848 DEV

To manage this VM browse to https://0.0.0.0:5480/
```

```
 *Login                                  Use Arrow Keys to navigate
  Set Timezone (Current:UTC)             and <ENTER> to select your choice.
```

Whether or not the network is detected, the Administrator can still login and use "`vami_config_net`" to setup the network, for example:

```
sudo /opt/vmware/share/vami/vami_config_net
```

The following screen shot shows the menu options available for configuring the network:



```
[administrator@gamma ~]$ sudo /opt/vmware/share/vami/vami_config_net

 Main Menu

0)      Show Current Configuration (scroll with Shift-PgUp/PgDown)
1)      Exit this program
2)      Default Gateway
3)      Hostname
4)      DNS
5)      Proxy Server
6)      IP Address Allocation for eth0
7)      IP Address Allocation for eth1
8)      IP Address Allocation for eth2
Enter a menu number [0]: _
```

Configuring the Mashery Local Cluster

Mashery Local may run configured in a cluster of one master and multiple slaves.

To configure the Mashery Local cluster, you need to:

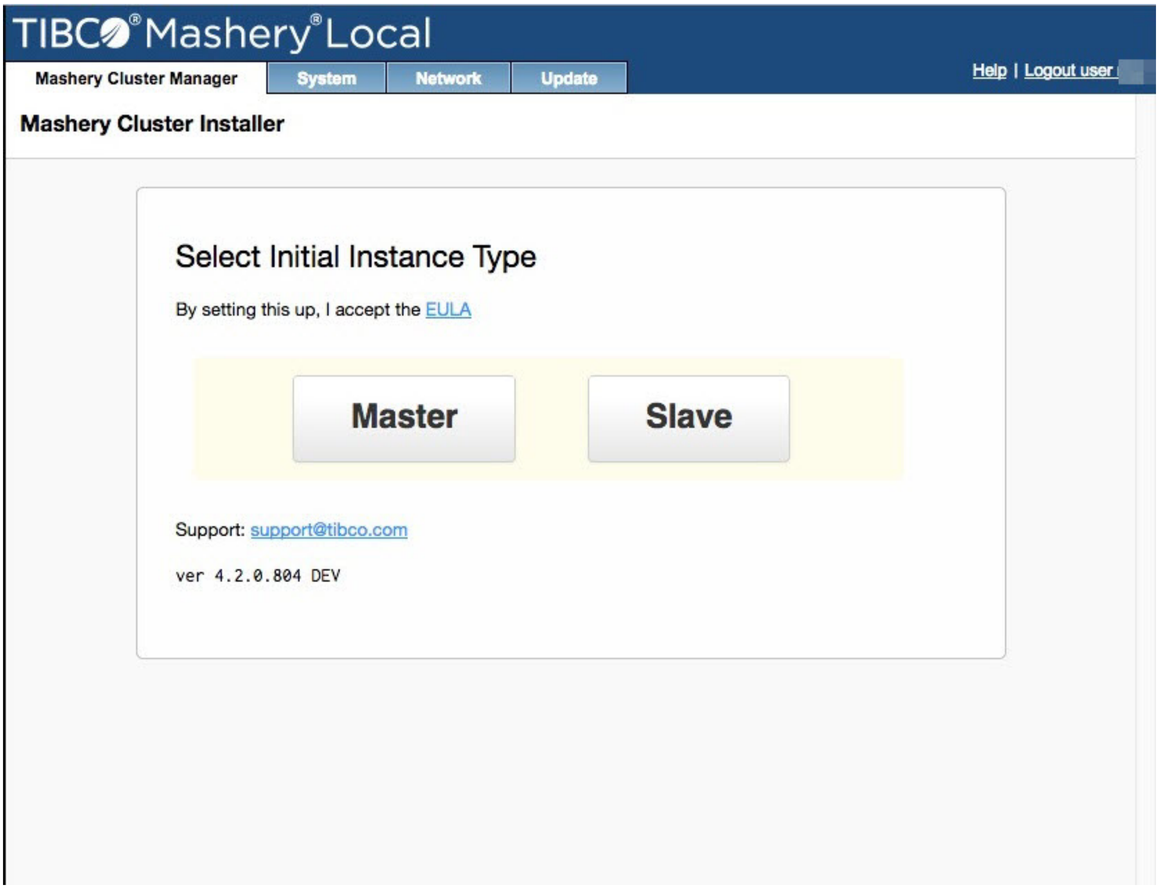
- Configure a Mashery local master
- Configure slave(s) to the local master

Configuring a Mashery Local Master

To configure a Mashery Local master:

Procedure

1. Browse to the Mashery Local Cluster Manager of the master by using the IP address of the instance that you found in step 12 of Deploying the Mashery Local OVF Template: `https://<IP_address_of_master>:5480`
2. Login with username **administrator** and the password provided by TIBCO Mashery.
   Click **Master**.


TIBCO® Mashery® Local
Mashery Cluster Manager
System
Network
Update
Help | Logout user
Mashery Cluster Installer
Select Initial Instance Type
By setting this up, I accept the EULA
Master
Slave
Support: support@tibco.com
ver 4.2.0.804 DEV

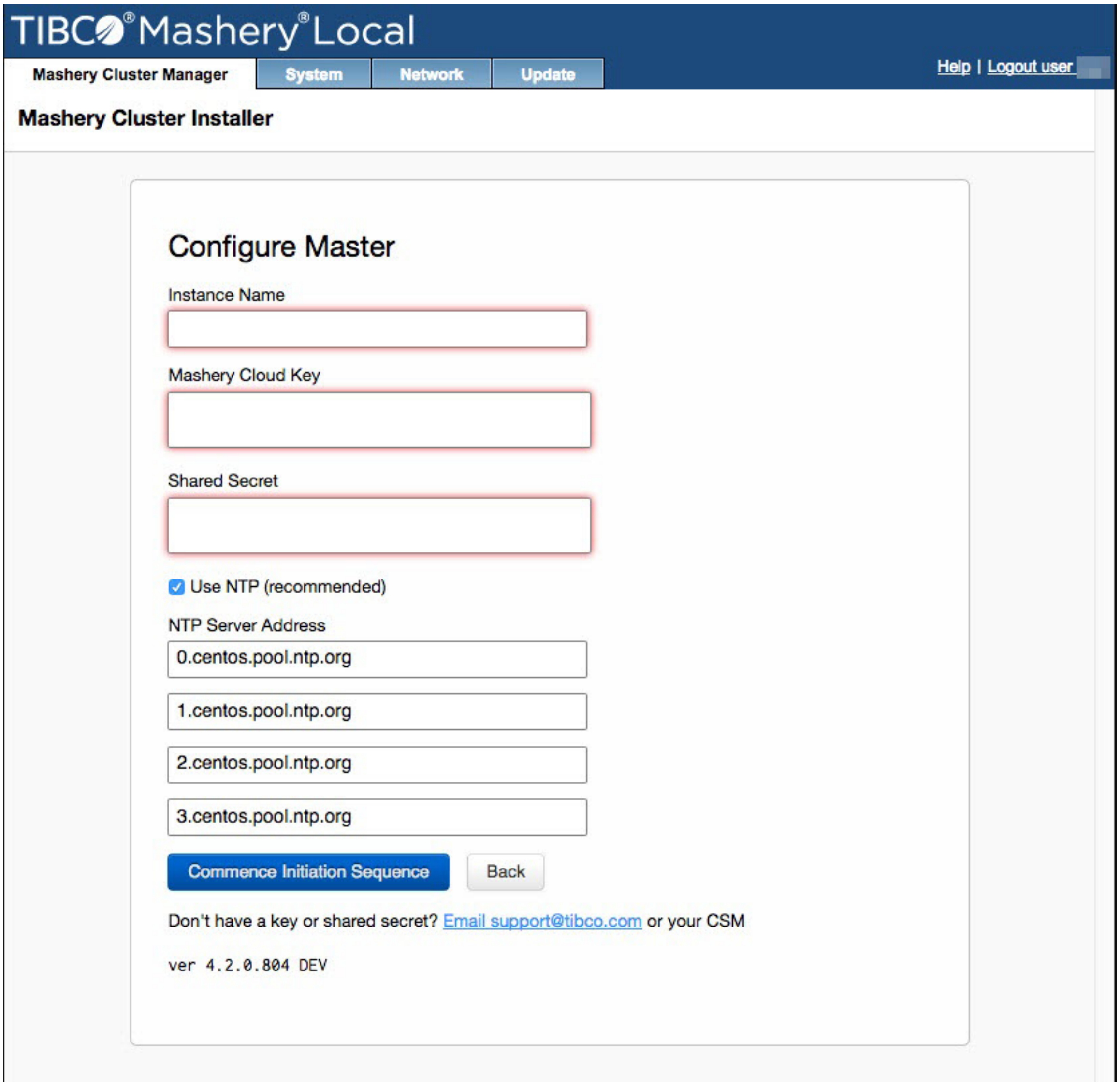
The **Configure Master** window appears.

Enter an instance name (this name may be equal or different from the name you provided VMware and will eventually display in the TIBCO Mashery Admin Dashboard) that is meaningful to your operation, the TIBCO Mashery Cloud Key and shared secret provided by TIBCO Mashery, and the NTP server address, if used.

The **Instance Name** you choose should be a unique name; other Master and Slave instances in your cluster cannot be given the same name.

The **Use NTP (recommended)** checkbox is selected by default, and four NTP servers can be configured.

> If you have multiple clusters, the Mashery Cloud Key and shared secret provided by TIBCO Mashery should be unique to each of your clusters. Mashery Local clusters should not share keys.



Mashery Cluster Installer

Configure Master

Instance Name

Mashery Cloud Key

Shared Secret

☑ Use NTP (recommended)

NTP Server Address

0.centos.pool.ntp.org

1.centos.pool.ntp.org

2.centos.pool.ntp.org

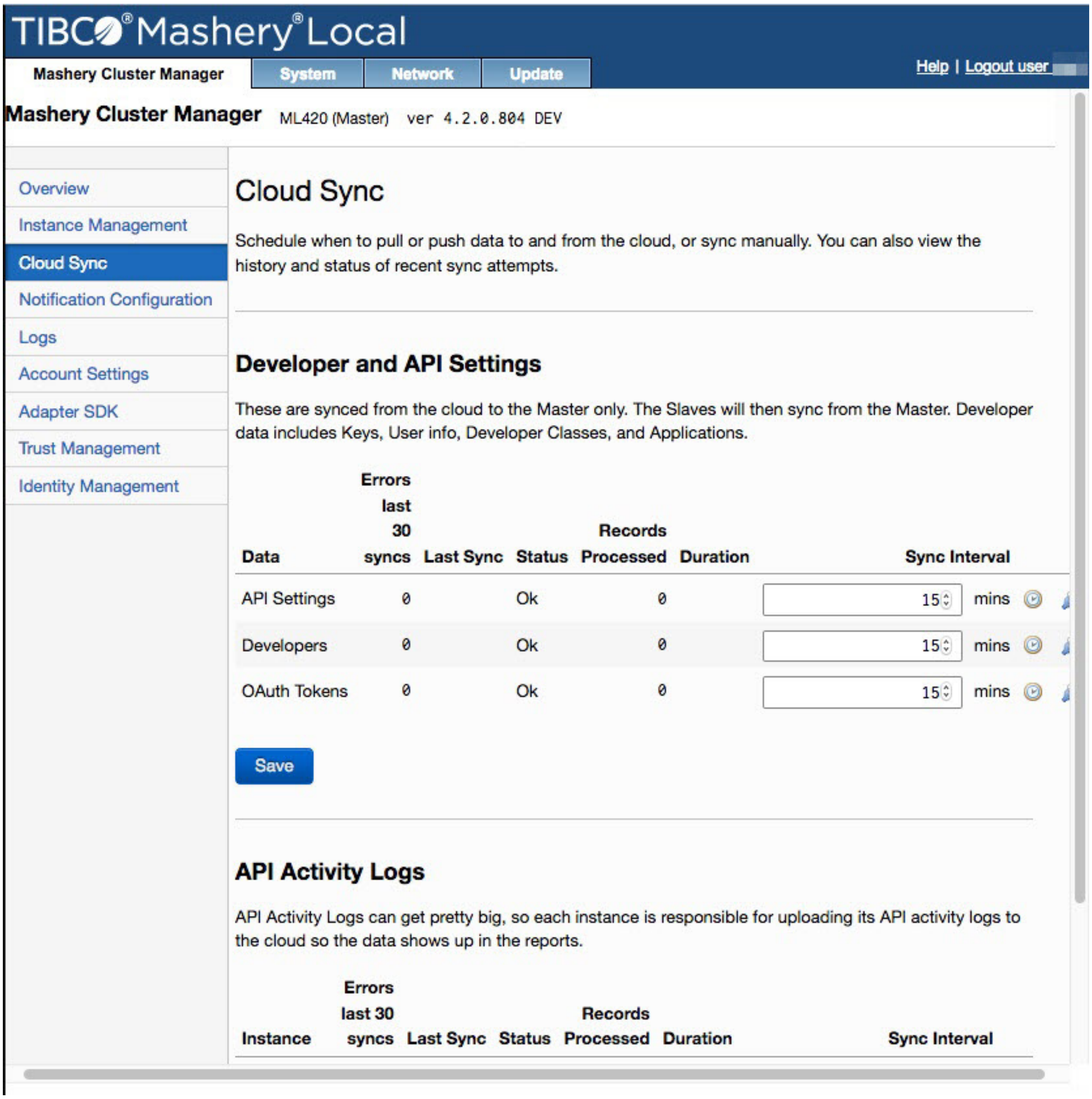
3.centos.pool.ntp.org

Don't have a key or shared secret? Email support@tibco.com or your CSM

ver 4.2.0.804 DEV

3. Click **Commence Initiation Sequence**.

   After the Master initializes with the TIBCO Mashery cloud service, a completion page appears.
4. Click **Continue**.
5. Navigate to the **Cloud Sync** page and perform manual syncs for **API Settings**, **Developers**, and **OAuth Tokens** by clicking the adjacent icons:



Mashery Cluster Manager ML420 (Master) `ver 4.2.0.804 DEV`

Cloud Sync

Schedule when to pull or push data to and from the cloud, or sync manually. You can also view the history and status of recent sync attempts.

Developer and API Settings

These are synced from the cloud to the Master only. The Slaves will then sync from the Master. Developer data includes Keys, User info, Developer Classes, and Applications.

Data	Errors last 30 syncs	Last Sync	Status	Records Processed	Duration	Sync Interval
API Settings	0		Ok	0		15 mins
Developers	0		Ok	0		15 mins
OAuth Tokens	0		Ok	0		15 mins

API Activity Logs

API Activity Logs can get pretty big, so each instance is responsible for uploading its API activity logs to the cloud so the data shows up in the reports.

Instance	Errors last 30 syncs	Last Sync	Status	Records Processed	Duration	Sync Interval

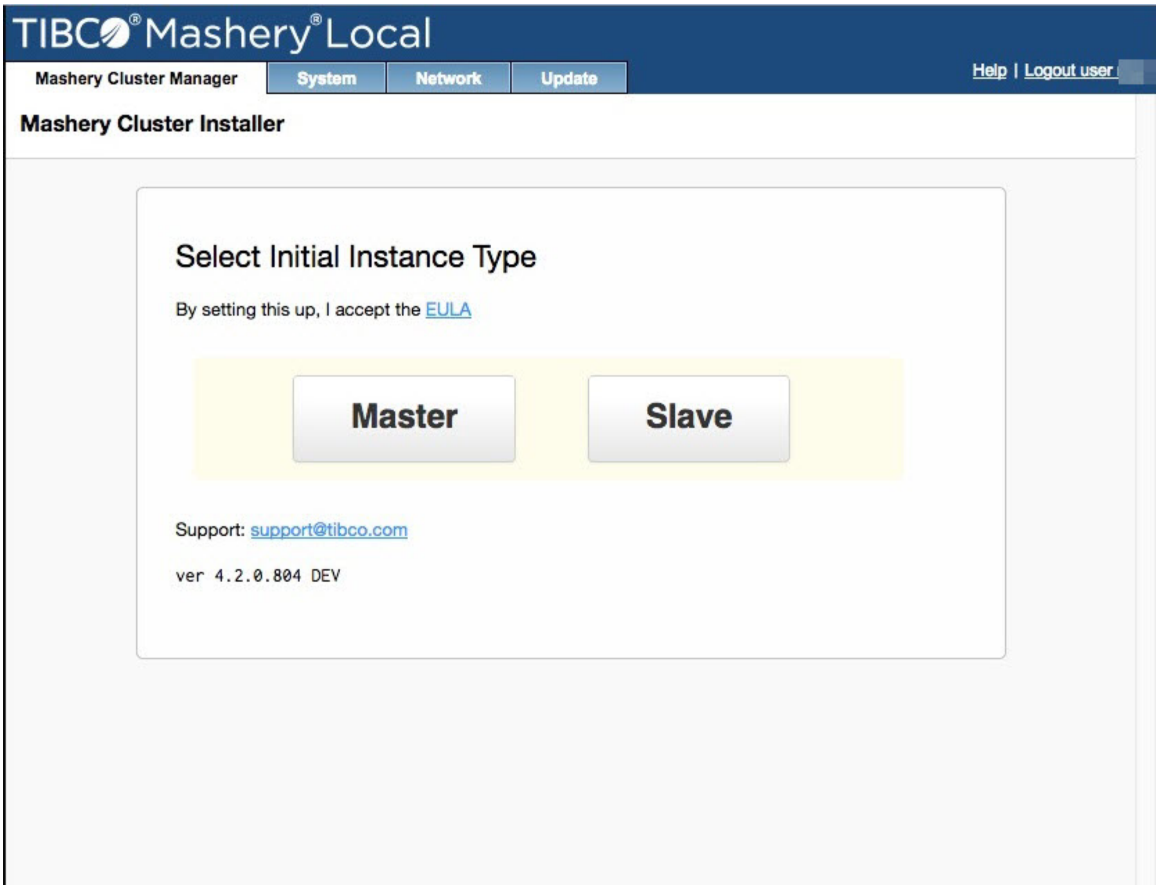
6. Test the instance as described in Testing the New Instance.
7. See the instructions in Advanced Configuration for how to enable notifications, LDAP, and API and JMX reporting access, if desired.

Configuring Slaves to the Local Master

Mashery Local may run configured in a cluster of one master and multiple slaves.
To configure slaves to the master:

Procedure

1. Browse to the Mashery Local Cluster Manager of the master by using the IP address of the instance that you found in step 12 of Deploying the Mashery Local OVF Template: `https://<IP_address_of_master>:5480`
2. Login with username **administrator** and the password provided by TIBCO Mashery.
3. Click **Slave**.



Mashery Cluster Installer

Select Initial Instance Type

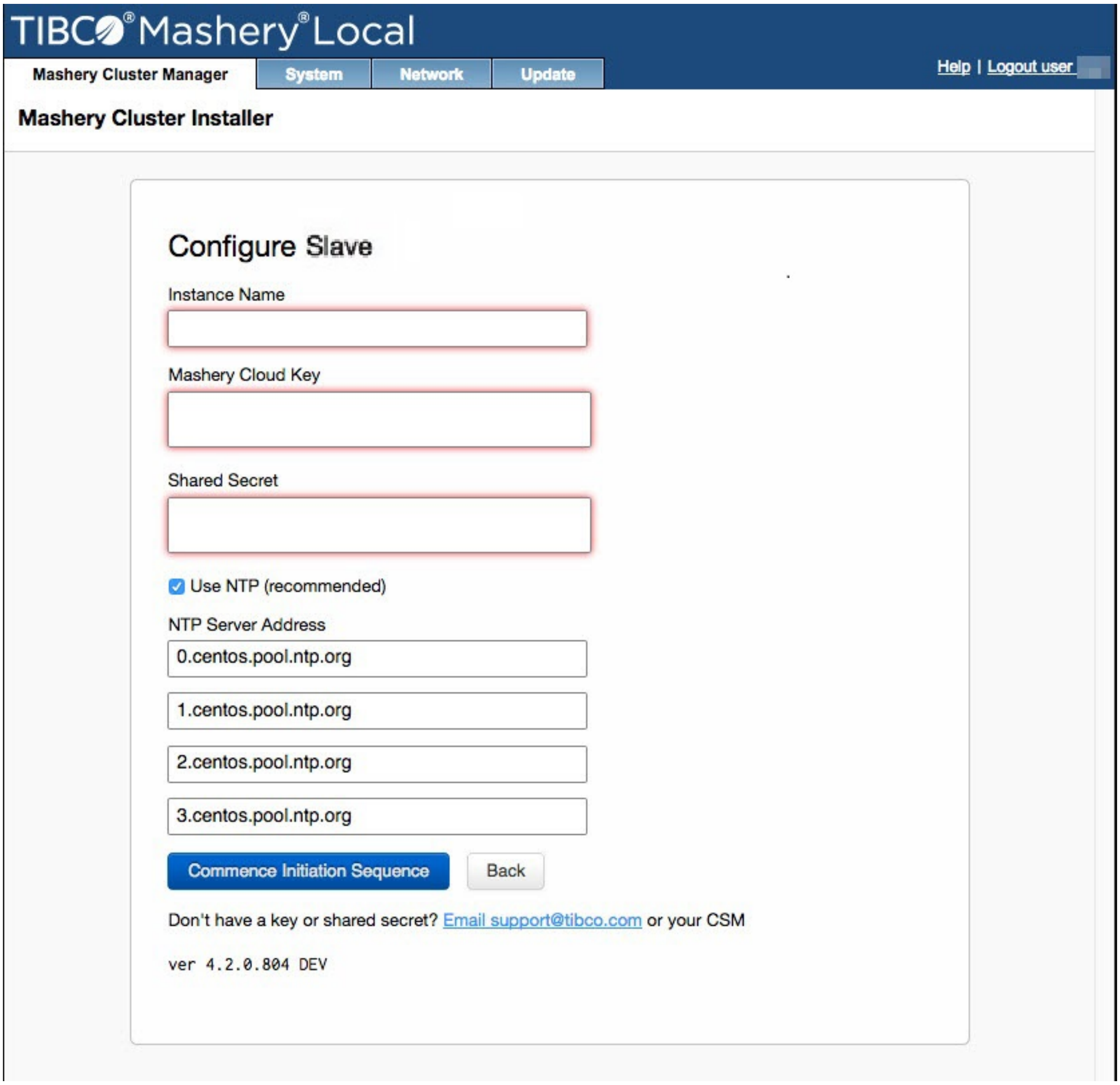
By setting this up, I accept the EULA

Support: support@tibco.com

ver 4.2.0.804 DEV

4. Enter an instance name (this name may be equal to or different from the name you provided VMware and will eventually display in the TIBCO Mashery Admin Dashboard) that is meaningful to your operation, the TIBCO Mashery Cloud Key and shared secret provided by TIBCO Mashery, and the NTP server address, if used.

   The **Use NTP (recommended)** checkbox is selected by default, and four NTP servers can be configured.



Mashery Cluster Installer

Configure Slave

Instance Name

Mashery Cloud Key

Shared Secret

☑ Use NTP (recommended)

NTP Server Address
0.centos.pool.ntp.org
1.centos.pool.ntp.org
2.centos.pool.ntp.org
3.centos.pool.ntp.org

Don't have a key or shared secret? Email support@tibco.com or your CSM

ver 4.2.0.804 DEV

5. Click **Register with Mashery and Master**.
6. Click **Continue**.
7. Test the instances as described in Testing a New Instance.
8. See the instructions in Advanced Configuration for how to enable notifications, LDAP, and API and JMX reporting access, if desired.

Configuring the Load Balancer

TIBCO Mashery recommends using a Load Balancer to best utilize the cluster, although this is not required because you may route your API traffic directly to each instance.

Each instance hosts a service called /mashping. Configure the Load Balancer to access the following address, without the host header:

```
http://<IP_address_of_instance>/mashping
```

If the Load Balancer and the cluster is working correctly, /mashping returns the following response:

```
HTTP/1.1 200 OK
Server: Mashery Proxy
Content-Type: application/json; charset=UTF-8
```

```
Transfer-Encoding: chunked
{"status":200,"time":1315510300,"message":"success"}
```

If /mashping returns any other response, then the load balancer should remove the instance from the cluster and either retry after a period of time or alert operations to investigate.

Mashery Local has two instance types: Master and Slave. Should the Load Balancer pull the Master out of the cluster pool, an Operations engineer should immediately investigate whether it can be recovered, and, if not, promote a Slave to Master. Taking offending Slaves out of rotation through the Load Balancer can mitigate any traffic impact. If no Master exists in the pool, data synchronization with the Mashery Cloud Service will not occur with the exception of API event activity. Access Tokens, Keys, Applications, Classes and Services will not be synchronized.

For steps on how to promote a Slave to Master, see Promoting a Slave to Master.

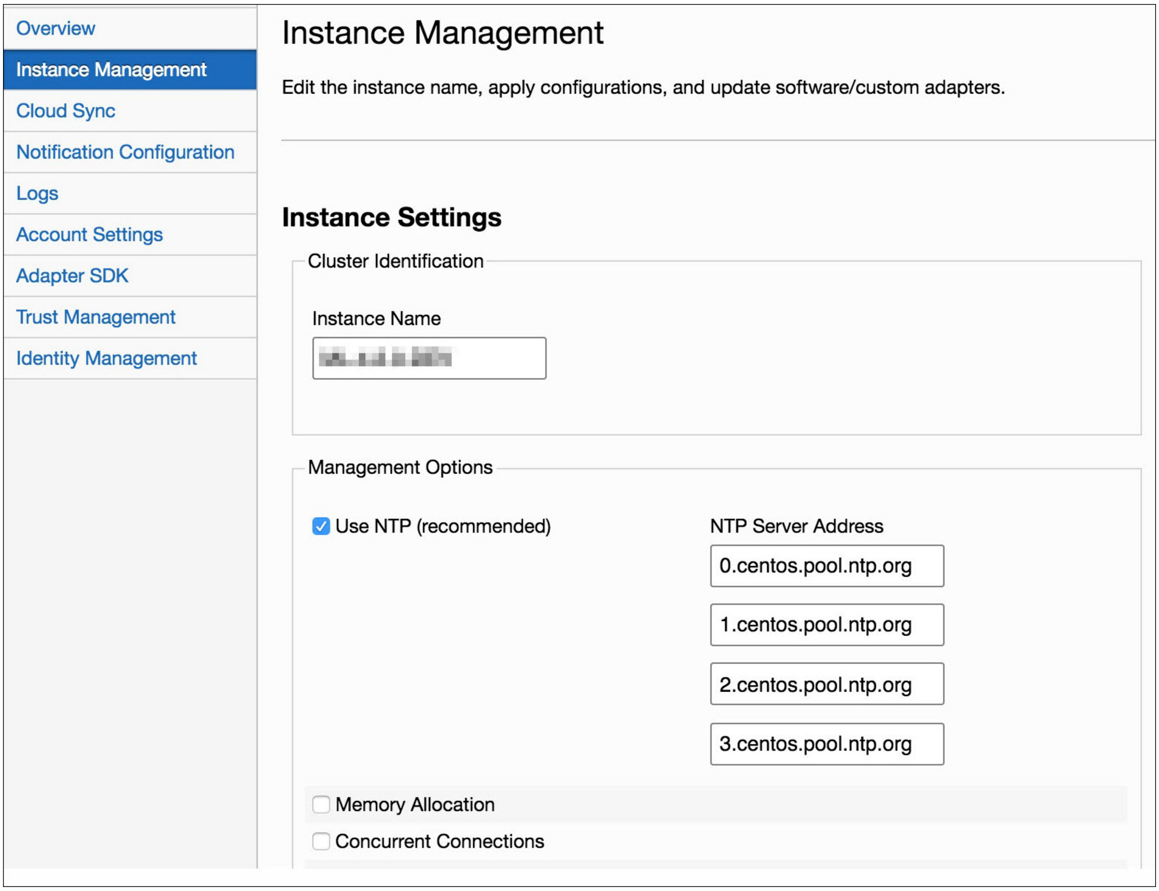
Configuring the Instance

The **Instance Management** tab allows you to configure additional settings for that particular instance. You can edit the instance name, configure instance settings, and update software and custom adapters. Additional system-level parameters can be tuned here such as application memory allocation, configuration cache size, maximum concurrent connections, and connection pool size for the database.

To configure an instance:

Procedure

1. Click **Instance Management**.



Instance Management

Edit the instance name, apply configurations, and update software/custom adapters.

Instance Settings

Cluster Identification

Instance Name

Management Options

☑ Use NTP (recommended)

NTP Server Address

0.centos.pool.ntp.org

1.centos.pool.ntp.org

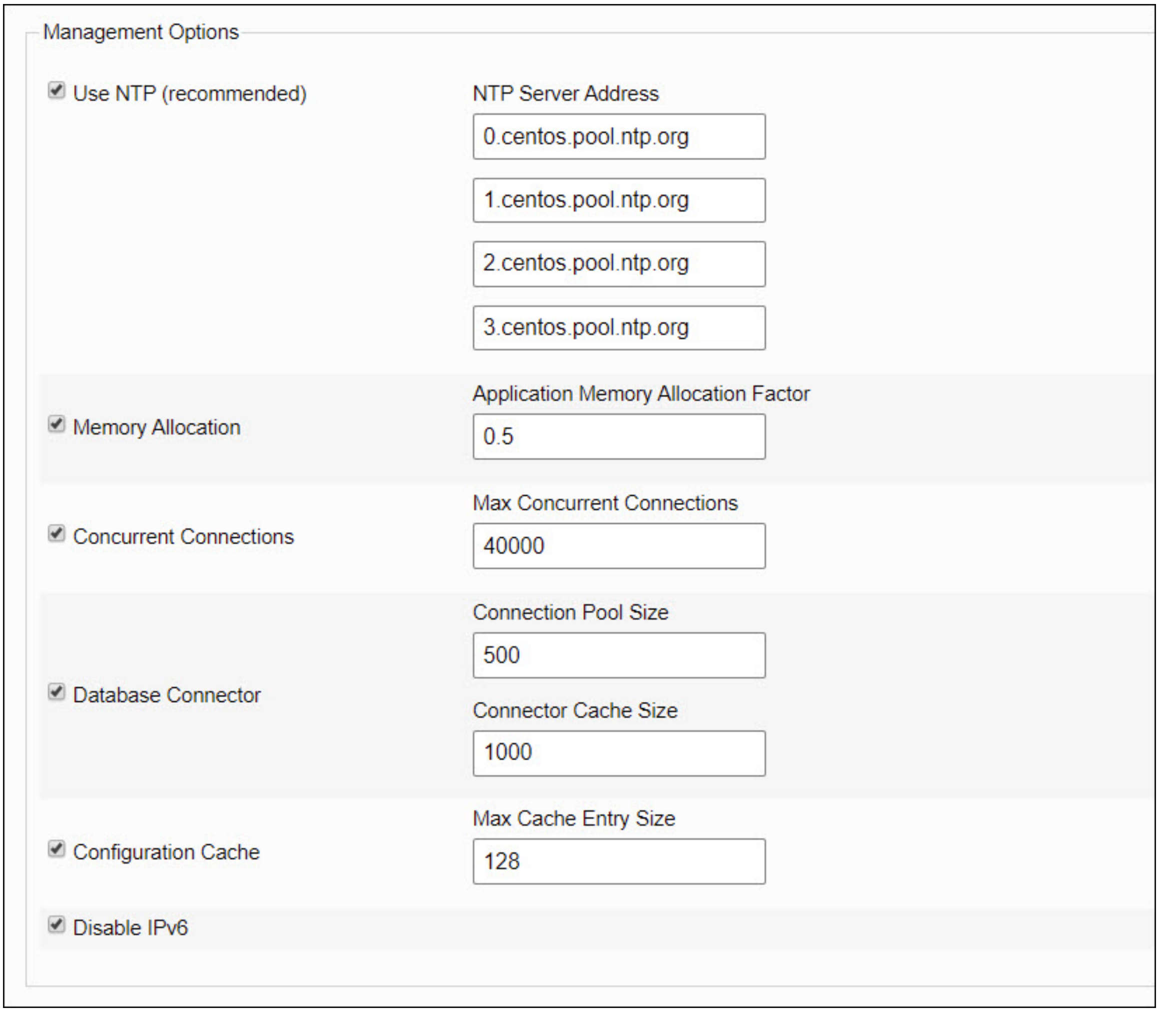
2.centos.pool.ntp.org

3.centos.pool.ntp.org

☐ Memory Allocation

☐ Concurrent Connections

2. Click the **Management Options** for which you want to configure the settings.
   A text box is displayed for the selected **Management Options.**
3. Enter the details for the following fields to configure the instance.



Management Options

☑ Use NTP (recommended)

NTP Server Address: 0.centos.pool.ntp.org, 1.centos.pool.ntp.org, 2.centos.pool.ntp.org, 3.centos.pool.ntp.org

☑ Memory Allocation

Application Memory Allocation Factor: 0.5

☑ Concurrent Connections

Max Concurrent Connections: 40000

☑ Database Connector

Connection Pool Size: 500

Connector Cache Size: 1000

☑ Configuration Cache

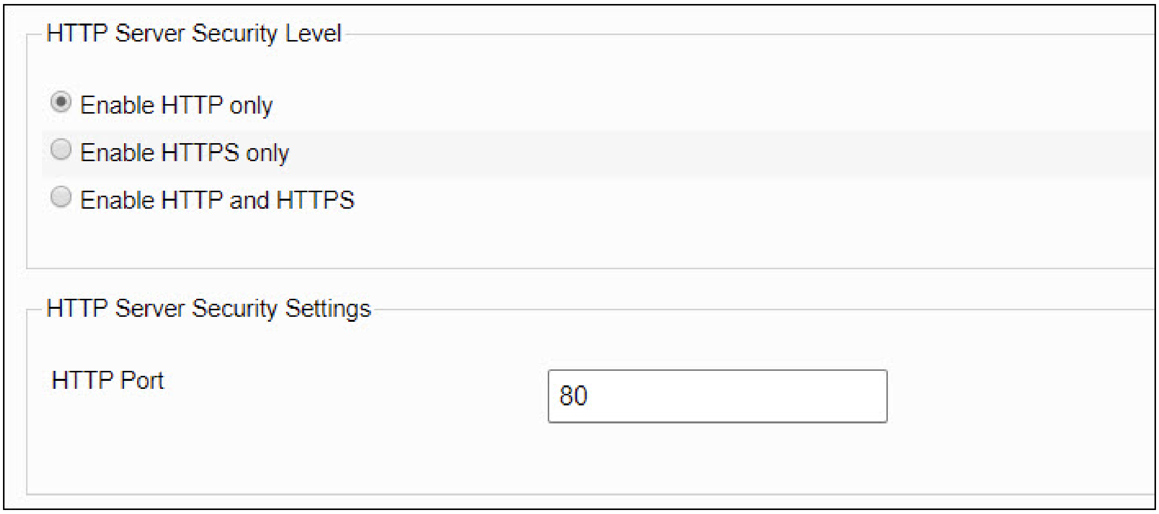
Max Cache Entry Size: 128

☑ Disable IPv6

Field	Description
Use NTP (recommended)	Enabled by default. Specify one to four NTP server addresses. NTP Address for time synchronizing is required if using NTP.
Memory Allocation	Specify application memory size as a fraction of the available memory, between 0 and 1.
Concurrent Connections	Sets the maximum number of concurrent connections to this service instance, must be at least 1.

Field	Description
Database Connector	The **Connection Pool Size** sets the maximum number of concurrent connections this instance will make to its database, between 1 and 100,000. The **Connector Cache Size** sets the cache size for the data connector, between 1 and 524,288.
Configuration Cache	Specify the maximum entry size (in MB) for configuration cache, at least 1.
Disable IPv6	Select this option to disable IPv6 if IPv6 traffic should not be allowed to the backend. By default, Mashery Local supports both IPv4 and IPv6.

4. Select the appropriate **HTTP Server Security Level**:



HTTP Server Security Level

● Enable HTTP only

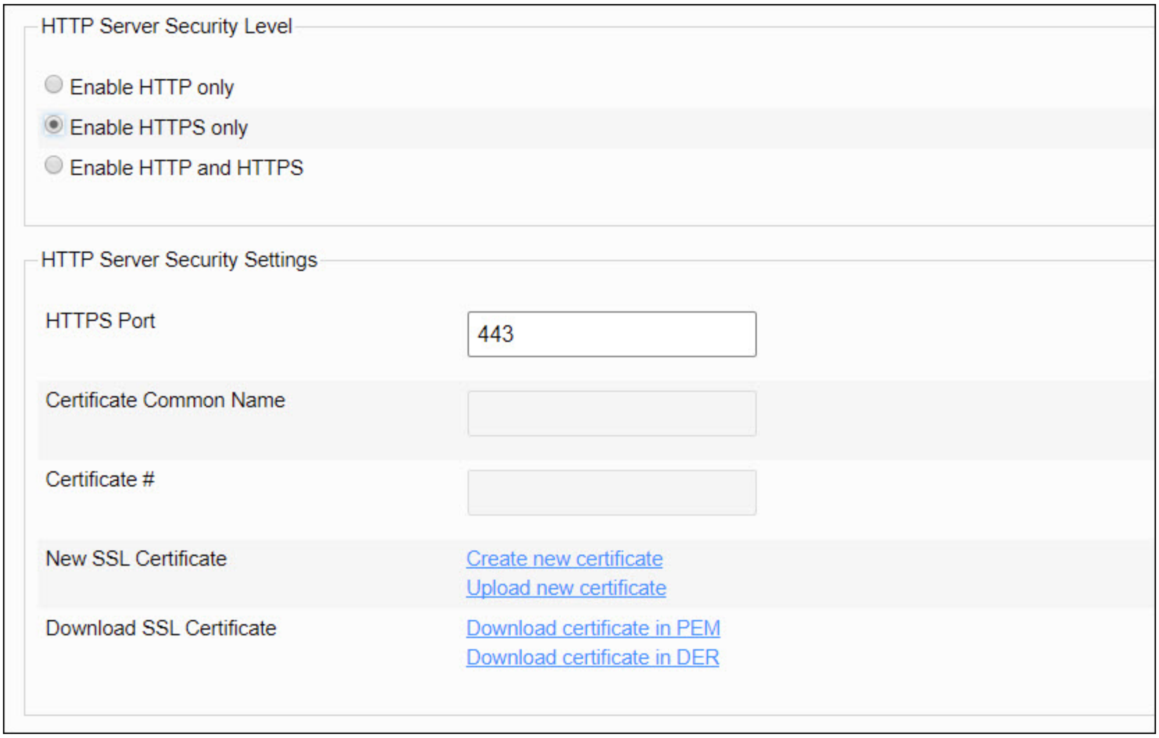
○ Enable HTTPS only

○ Enable HTTP and HTTPS

HTTP Server Security Settings

HTTP Port: 80

- **Enable HTTP only**: If selected, the default **HTTP Port** for **HTTP Server Security Settings** is `80`.



HTTP Server Security Level

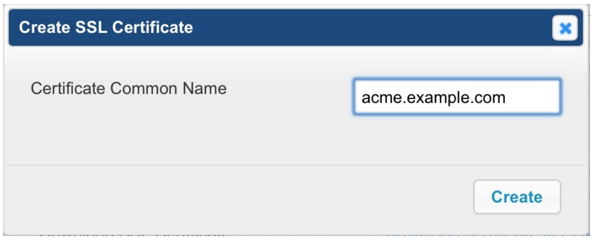
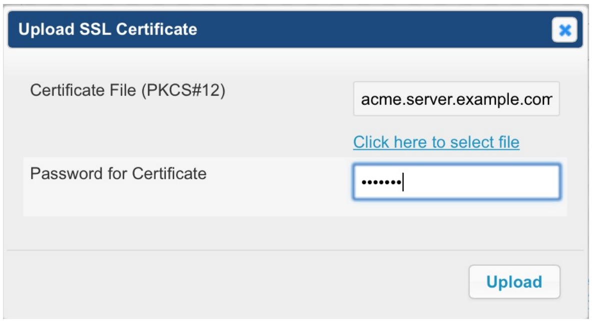
○ Enable HTTP only
● Enable HTTPS only
○ Enable HTTP and HTTPS

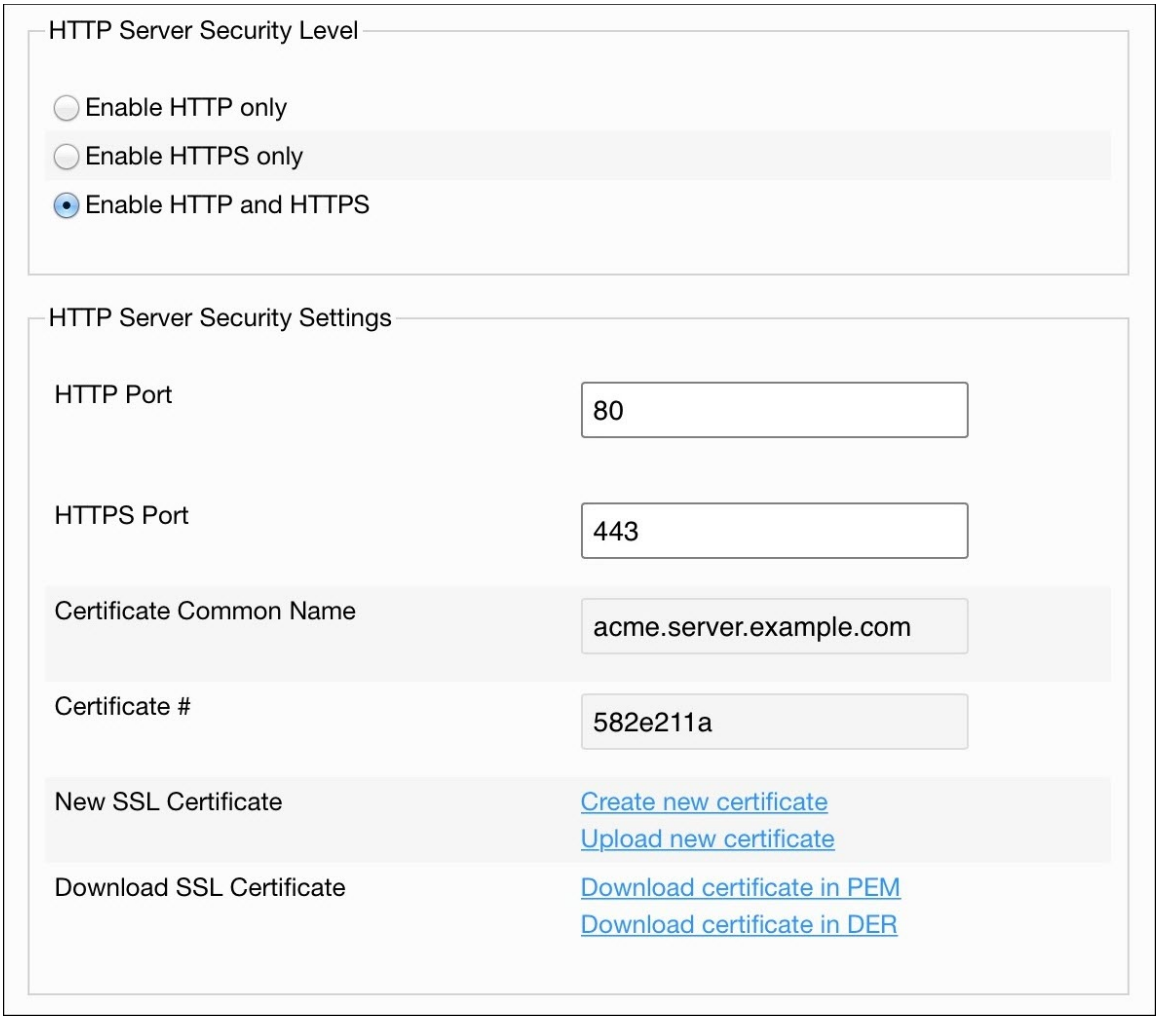
HTTP Server Security Settings

HTTPS Port	443
Certificate Common Name	
Certificate #	
New SSL Certificate	Create new certificate Upload new certificate
Download SSL Certificate	Download certificate in PEM Download certificate in DER

- **Enable HTTPS only**: If selected, enter the details for the following fields:

Field	Description
HTTPS Port	Specify the HTTPS port. The default is `443`.
Certificate Common Name (display only)	Automatically displays the name of the selected certificate.
Certificate # (display only)	Automatically displays the number of the selected certificate.

Field	Description
New SSL Certificate	Select from: • **Create new certificate**: If selected, enter a Certificate Common name in the **Create SSL Certificate** window, then click **Create**.  Certificate Common Name: acme.example.com • **Upload new certificate**: If selected, in the **Upload SSL Certificate** window, browse to the SSL certificate using the **Click here to select file** link, enter the **Password for Certificate**, then click **Upload**.  Upload SSL Certificate Click here to select file Upload Certificate File (PKCS#12): acme.server.example.com Password for Certificate: •••••••
Download SSL Certificate	Select from: • **Download certificate in PEM**: downloads the current certificate in PEM format. • **Download certificate in DER**: downloads the current certificate in DER format.



HTTP Server Security Level

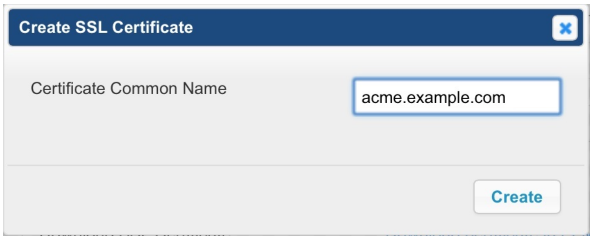
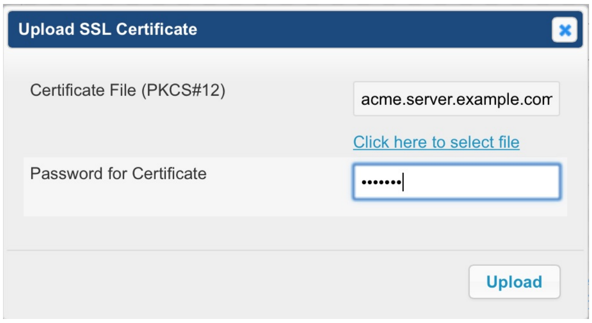
○ Enable HTTP only
○ Enable HTTPS only
● Enable HTTP and HTTPS

HTTP Server Security Settings

HTTP Port	80
HTTPS Port	443
Certificate Common Name	acme.server.example.com
Certificate #	582e211a
New SSL Certificate	Create new certificate Upload new certificate
Download SSL Certificate	Download certificate in PEM Download certificate in DER

- **Enable HTTP and HTTPS**: If selected, enter the details for the following fields:

Field	Description
HTTP Port	Specify the HTTP port. The default is `80`.
HTTPS Port	Specify the HTTPS port. The default is `443`.
Certificate Common Name (display only)	Displays the name of the selected certificate.
Certificate # (display only)	Displays the number of the selected certificate.

Field	Description
New SSL Certificate	Select from: - **Create new certificate**: If selected, enter a Certificate Common name in the **Create SSL Certificate** window, then click **Create**.  Certificate Common Name: acme.example.com - **Upload new certificate**: If selected, in the **Upload SSL Certificate** window, browse to the SSL certificate using the **Click here to select file** link, enter the **Password for Certificate**, then click **Upload**.  Upload SSL Certificate Click here to select file Upload Certificate File (PKCS#12): acme.server.example.com Password for Certificate: •••••••
Download SSL Certificate	Select from: - **Download certificate in PEM**: downloads the current certificate in PEM format. - **Download certificate in DER**: downloads the current certificate in DER format.

5. Click **Save**.

   You may be reminded that Mashery Local needs to restart proxy service.

   The instance is configured for the specified settings.

What to do next

For detailed steps on setting up HTTPS Server, please refer to the following sections in the Appendix:

- Setting up HTTPS Server using Self-Signed Certificate
- Setting up HTTPS Server using Customer-Provided Certificate

Shutting Down a Master

The following section describes how to shut down a Master in a Mashery Local Cluster.

Procedure

- Use `docker-compose down` to shut down the Master.

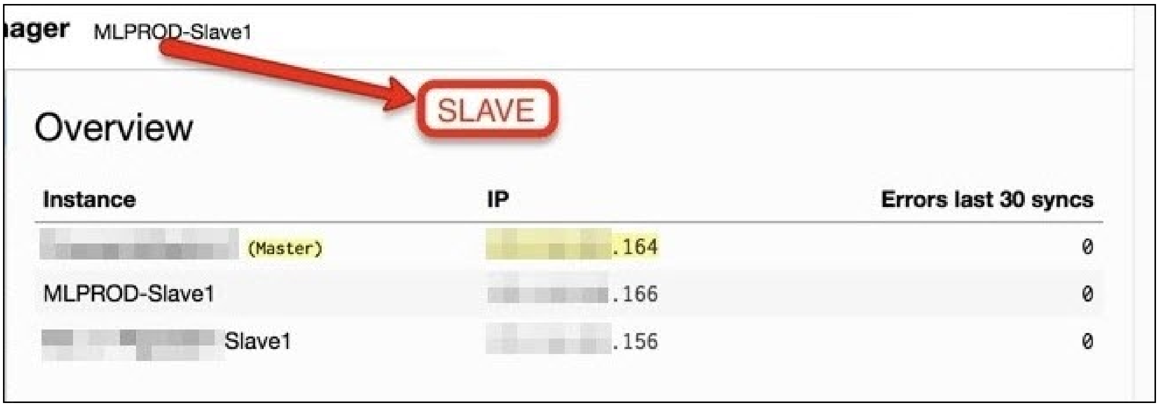
Promoting a Slave to Master

Promoting a Slave to Master is important from within a cluster, and having multiple clusters (using unique MoM keys) connecting to the same area is High Availability. Taking offending Slaves out of rotation through the Load Balancer can also mitigate any traffic impact.

To promote a Slave to Master:

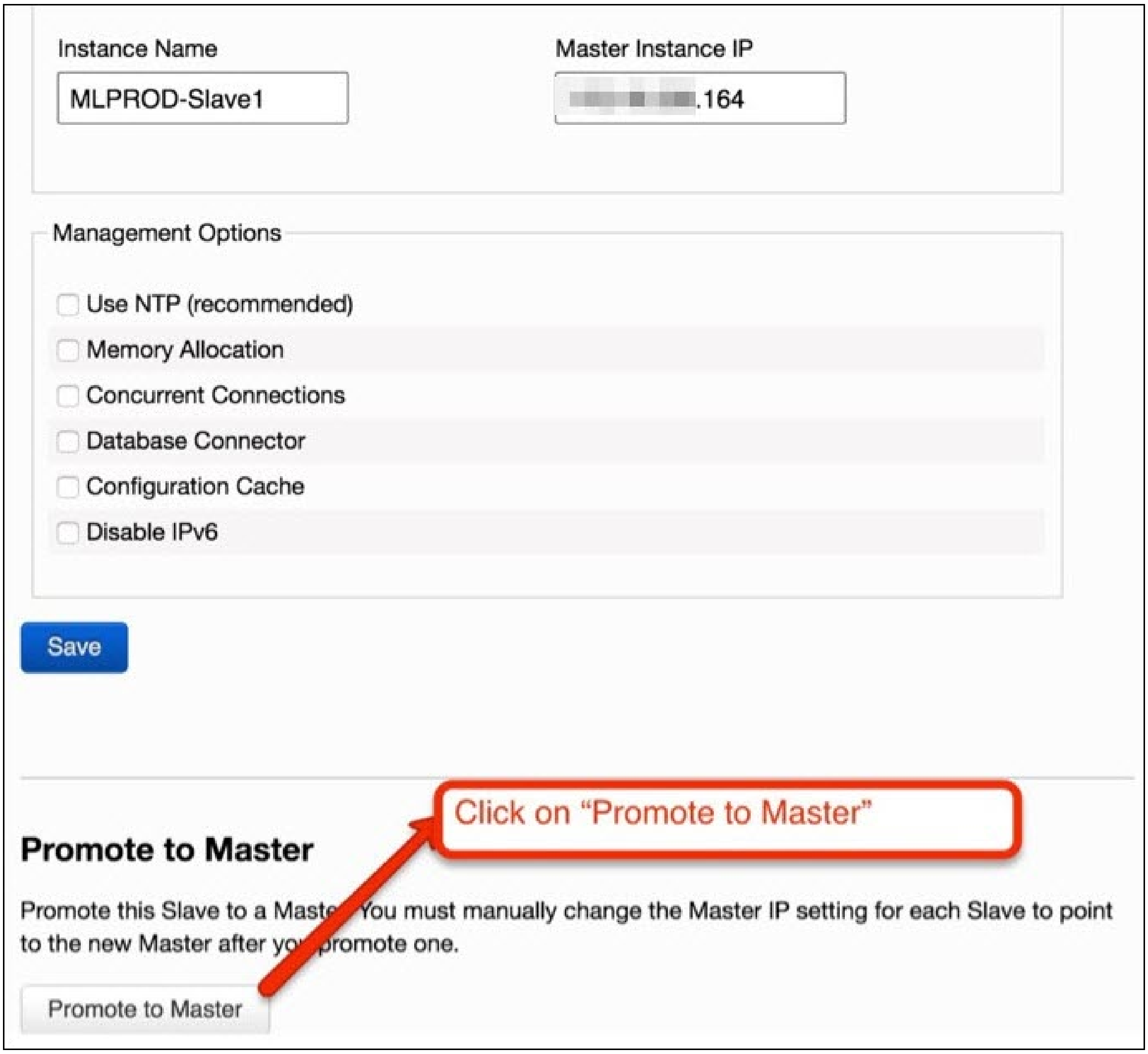
Procedure

1. Log into the Slave instance.



Instance	IP	Errors last 30 syncs
[redacted] (Master)	[redacted].164	0
MLPROD-Slave1	[redacted].166	0
[redacted] Slave1	[redacted].156	0

2. Click **Instance Management**.
3. In the **Promote to Master** section, click **Promote to Master**.



Instance Name: MLPROD-Slave1

Master Instance IP: [redacted].164

Management Options

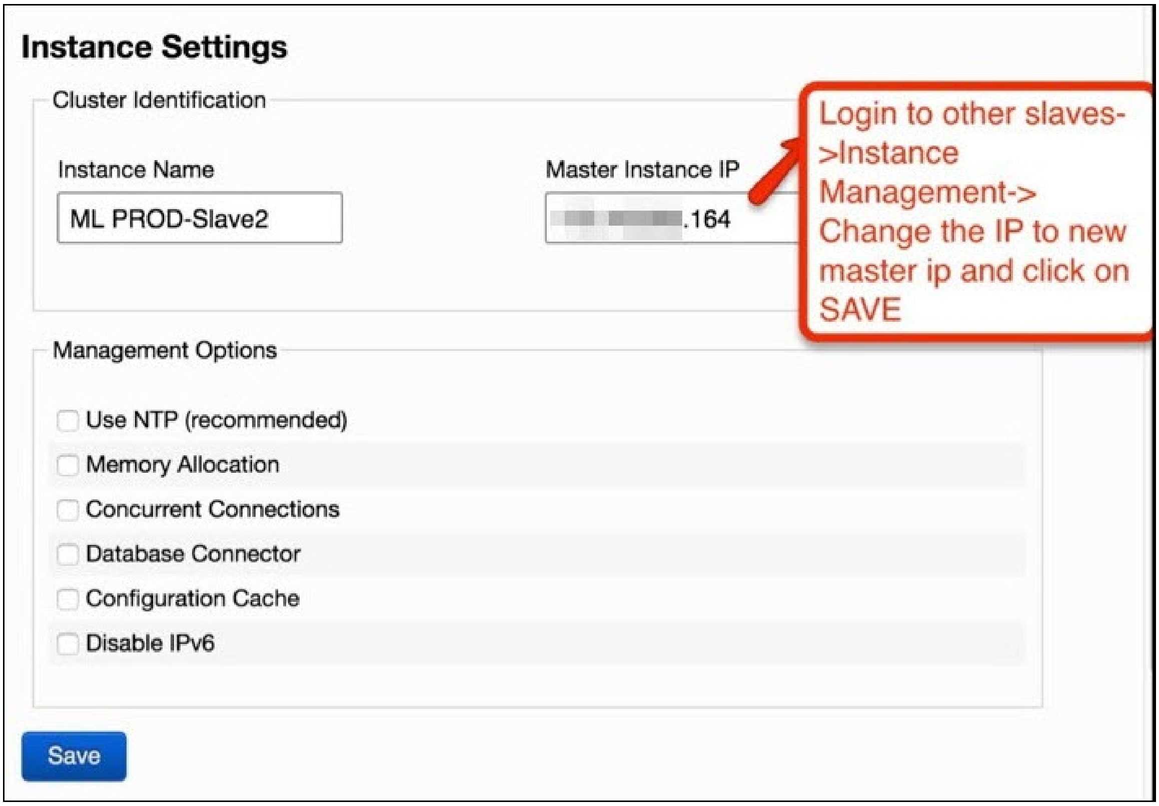
- ☐ Use NTP (recommended)
- ☐ Memory Allocation
- ☐ Concurrent Connections
- ☐ Database Connector
- ☐ Configuration Cache
- ☐ Disable IPv6

Click on “Promote to Master”

Promote to Master

Promote this Slave to a Master. You must manually change the Master IP setting for each Slave to point to the new Master after you promote one.

4. Log into the other Slaves, go to **Instance Settings** in **Instance Management**, and in the change the **Master Instance IP** address to the new Master's IP address.



Instance Settings

Cluster Identification

Instance Name: ML PROD-Slave2

Master Instance IP: [redacted].164

Login to other slaves->Instance Management-> Change the IP to new master ip and click on SAVE

Management Options

☐ Use NTP (recommended)
☐ Memory Allocation
☐ Concurrent Connections
☐ Database Connector
☐ Configuration Cache
☐ Disable IPv6

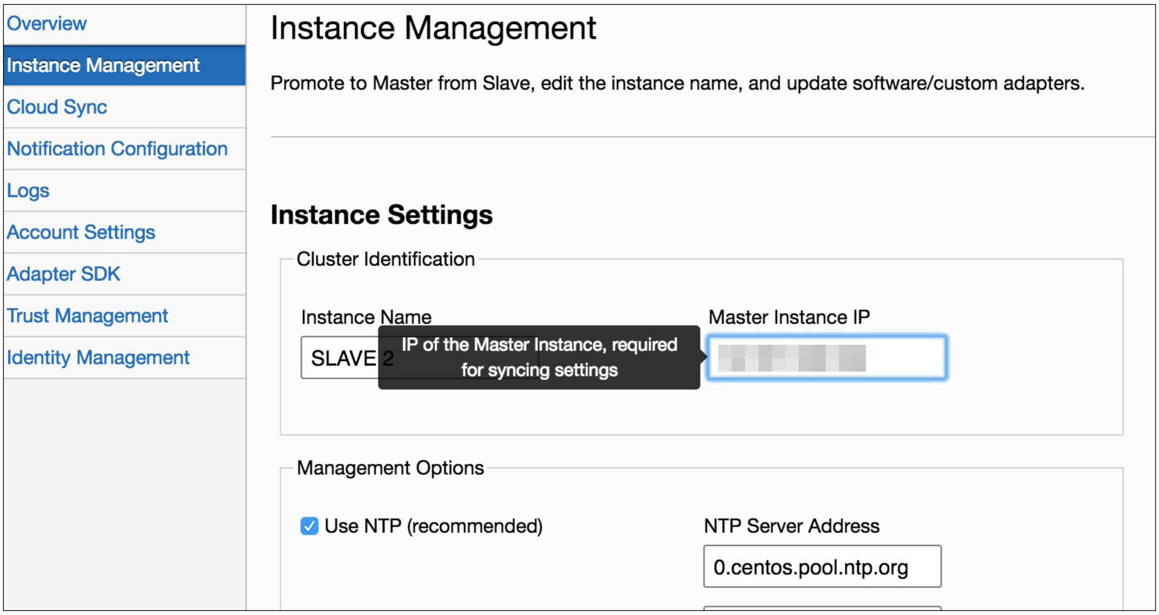
5. (Optional) Delete the old Master or shut down that Virtual Machine.

Repointing Other Slaves to a New Master

The following section describes how to repoint other slaves to a new Master.

Procedure

1. Log into the Slave Instance.
2. On the **TIBCO Mashery Local** page, click **Instance Management**.



Instance Management

Promote to Master from Slave, edit the instance name, and update software/custom adapters.

Instance Settings

Cluster Identification

Instance Name: SLAVE 2

Master Instance IP: [redacted]

IP of the Master Instance, required for syncing settings

Management Options

☑ Use NTP (recommended)

NTP Server Address: 0.centos.pool.ntp.org

3. In the **Instance Settings** section, enter the IP address of the new Master in the **Master Instance IP** field.
4. Click **Save**.

Administrator Sudo Commands for Cluster Management

Mashery Local Virtual Appliance (OVA) users with an Administrative role can use the following commands to manage their clusters:

```
sudo service javaproxy status
sudo service javaproxy start
sudo service javaproxy stop
sudo service javaproxy restart

sudo service mysqld status
sudo service mysqld start
sudo service mysqld stop
sudo service mysqld restart

sudo monit stop proxy
sudo monit start proxy

sudo monit stop mysql
sudo monit start mysql
```

Administrator Sudo Commands for Linux Package Management

Mashery Local users with an Administrator role have the ability to run the following "yum" commands for Linux Package Management:

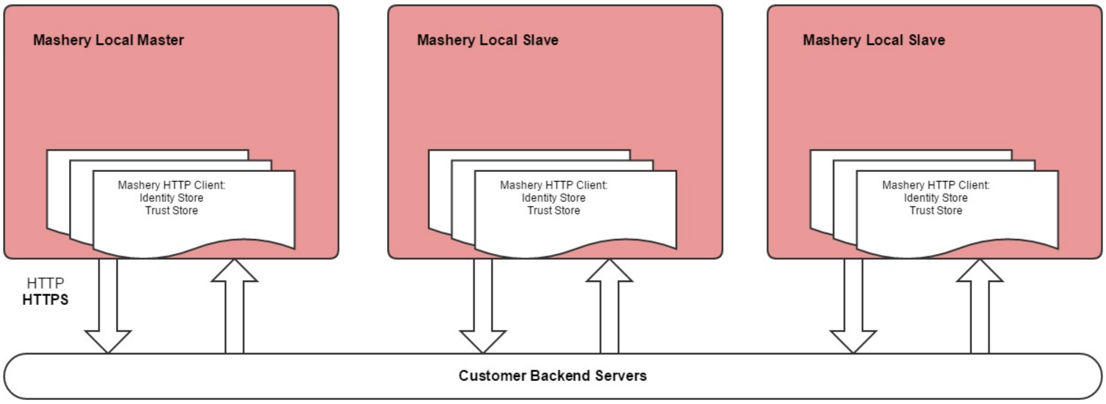
- sudo yum update
- sudo yum history
- sudo yum help
- sudo yum info
- sudo yum list
- sudo yum repolist
- sudo yum version

- sudo yum distribution-synchronization

The "yum update" command has the potential to install software that may not be compatible with the installed libraries in your Mashery Local instance. It is highly recommended to first check with TIBCO Support if "yum update" can be run safely in your Mashery Local instance. Further, any changes made to Mashery Local instances using "yum update" must be tested thoroughly in a non-production environment first.

HTTPS Client Feature Overview

The HTTP Client in Mashery Local is used for connecting to customer backend servers.



The HTTPS Client supports the following features:

- Verification of backend server certificate
- HTTPS Client authentication (Mutual SSL Authentication)

These features are configured via the HTTPS Client Profile feature. With an HTTPS Client profile, you can configure one or more trusted CA certificates. These CA certificates are used for verifying backend customer server certificates. You can configure only one identity, which will be used in HTTPS Client Authentication (Mutual SSL Authentication). The HTTPS Client profile can be applied to one or more endpoints.

Summary of How to Use the HTTPS Client Profile Feature

The HTTPS Client Profile feature is implemented as follows:

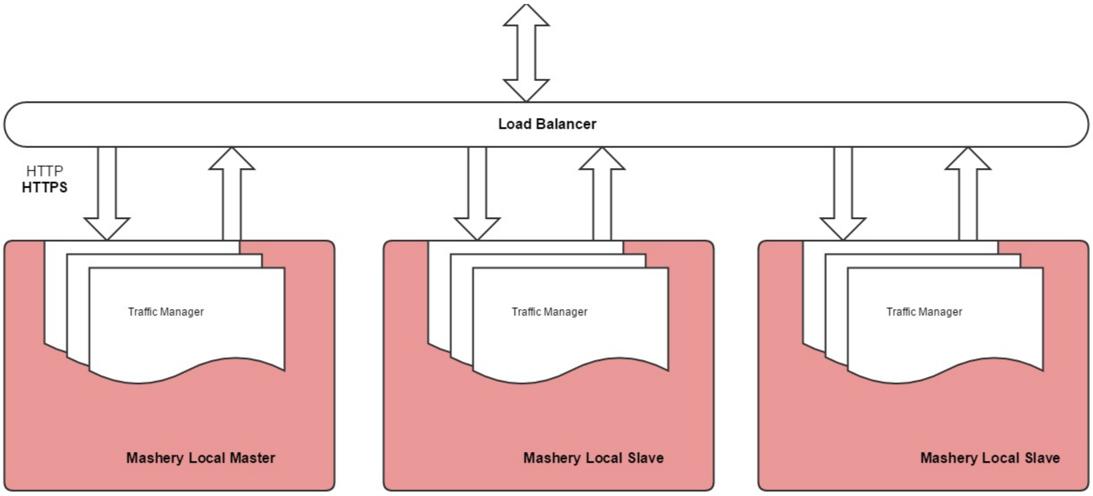
1. Certificates and identities are uploaded or updated in Mashery Local using Mashery Cluster Manager.
2. Metadata of certificates and identity are synchronized to Mashery SaaS (Control Center). Then, they are available for creating HTTPS Client profiles in Mashery SaaS.
3. An HTTPS Client profile is created In Mashery SaaS, and the profile is applied to the endpoint.
4. The HTTPS Client Profile is synchronized to Mashery Local, then the list of certificates and one identity will be used in HTTPS Client connection to backend servers.

For detailed steps on setting up the HTTPS Client feature, please refer to following sections in the Appendix:

- Configuring and using the HTTPS Client Feature without Mutual Authentication
- Configuring and using the HTTPS Client Feature with Mutual Authentication

HTTPS Server Feature Overview

Mashery Local supports HTTPS requests. The following picture illustrates a typical TIBCO Mashery Local deployment, which consists of one master node and two slave nodes. The communication between the Load Balancer and Mashery Local nodes can be either HTTP or HTTPS.



Administrators can configure the certificates and port number for the Mashery Local HTTPS Server. Mashery Local supports using self-signed certificates or customer-provided certificates. The configuration for certificates and port number must be done on a per-node basis using Mashery Cluster Manager.

For detailed steps on setting up an HTTPS Server, please refer to following sections in the Appendix:

- Setting up HTTPS Server using Self-Signed Certificate
- Setting up HTTPS Server using Customer-Provided Certificate

Advanced Configuration and Maintenance

This section describes how you can extend your installation by adding the following capabilities:

- Quota Notifications
- LDAP Configuration
- API and JMX Reporting

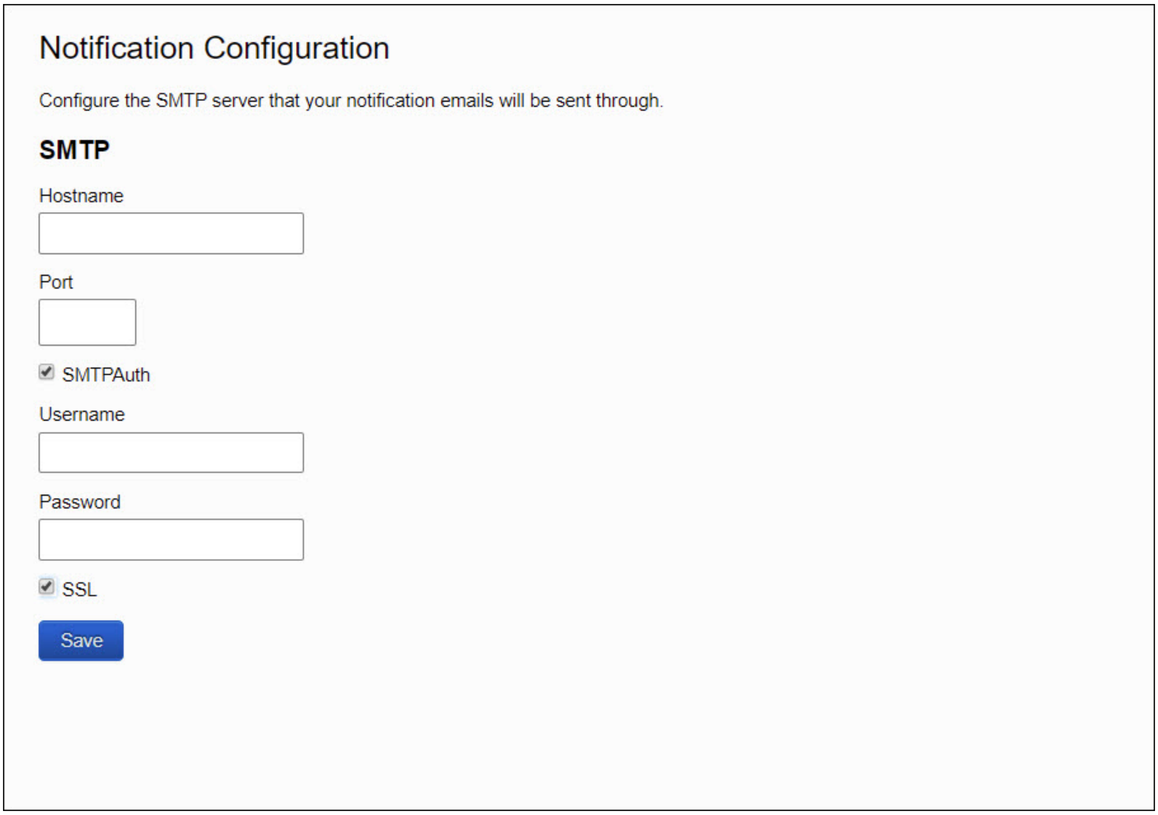
This section also describes how to locate and install updates.

Configuring Quota Notifications

You can configure Mashery Local to send Over Throttle Limit, Over Quota Limit and Near Quota Limit Warning notifications when an API key exceeds or nears its limits.

To configure quota notifications, follow the steps below:

1. Click **Notification Configuration**.



Notification Configuration

Configure the SMTP server that your notification emails will be sent through.

SMTP

2. In the **Notification Configuration** page, configure/enable the following fields for the SMTP server:

    - Hostname
    - Port
    - SMTPAuth
    - Username
    - Password
    - SSL

3. Click **Save**.

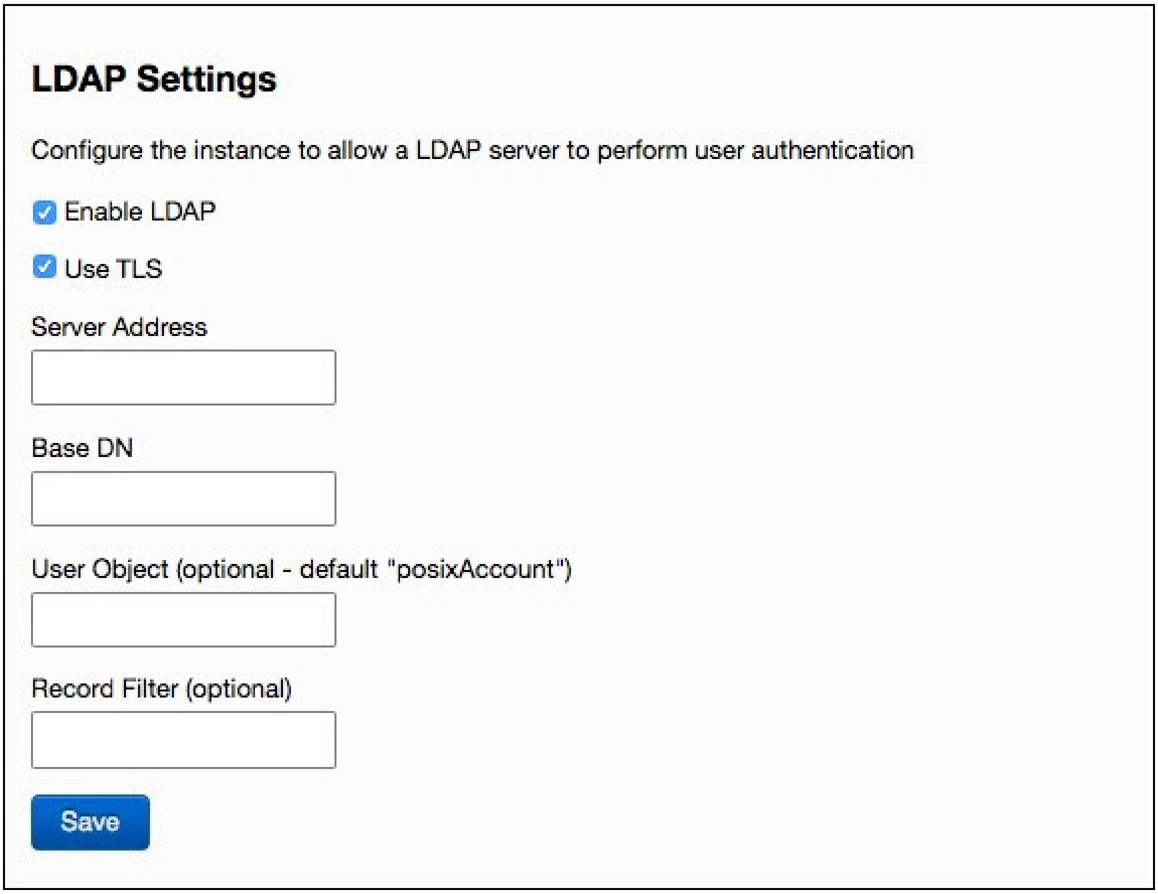
Similar notifications settings are available on the slave instances as well.

Configuring LDAP

You can configure Mashery Local to be authenticated against your existing LDAP server and to include filters and objects.

To configure LDAP, follow the instructions below:

1. Click **Account Settings**.



LDAP Settings

Configure the instance to allow a LDAP server to perform user authentication

☑ Enable LDAP

☑ Use TLS

2. In the **LDAP Settings** section, enable/configure the following fields to supply LDAP information:
   - **Enable LDAP**
   - **Use TLS**
   - **Server Address**
   - **Base DN**
   - The **User Object** field allows you to specify what the user object is called on your LDAP server.
   - The **Record Filter** field accepts a Pluggable Authentication Module (PAM) filter. This allows you to set a filter that checks for a specific attribute of the user object and deny login if the match fails.
3. Click **Save**.

Similar LDAP settings are available on the slave instances as well.

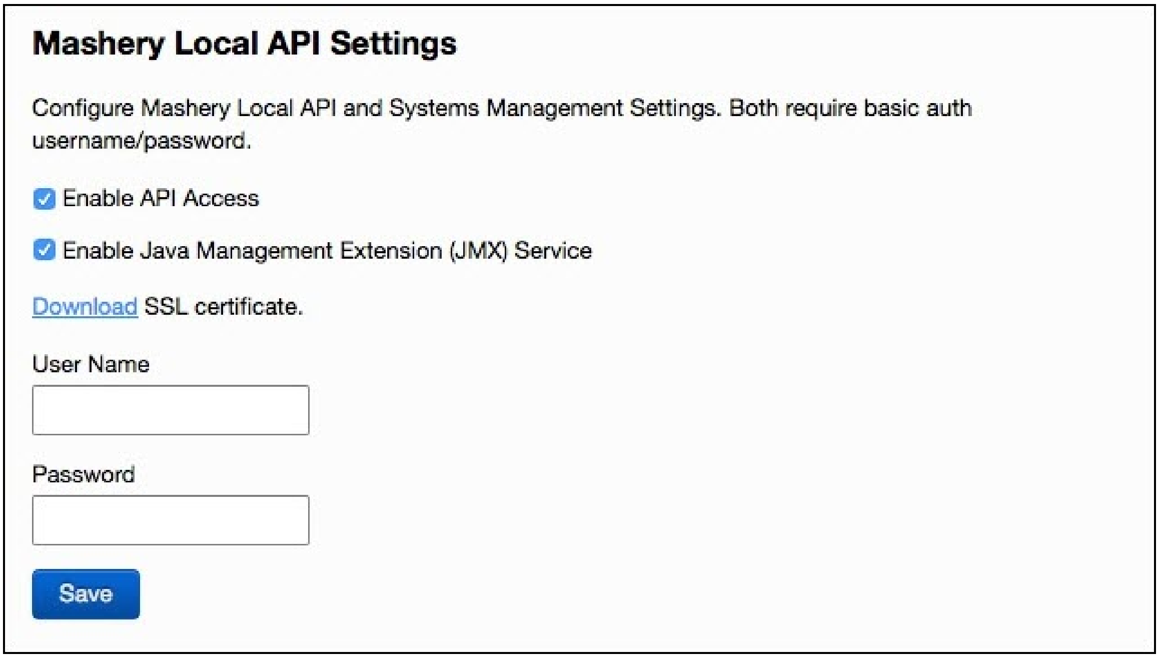
Configuring OAuth 2.0 API Access

Mashery Local provides a local OAuth 2.0 API. The API can be available on any computer within the cluster, regardless if the instance is a Master or Slave.

To configure and use the OAuth 2.0 API, follow the instructions below:

Procedure

1. Click **Account Settings**.



Mashery Local API Settings

Configure Mashery Local API and Systems Management Settings. Both require basic auth username/password.

☑ Enable API Access

☑ Enable Java Management Extension (JMX) Service

Download SSL certificate.

User Name

Password

2. In the **Mashery Local API Settings** section, enable/configure the following fields to enable the API:
   - **Enable API Access**
   - **User Name** and **Password**: You need to enter a username and password to access the API using basic HTTP authentication.
3. Click **Save**.
4. Similar API settings are available on the slave instances as well. On all slave instances, navigate to **Account Settings**, and under **Mashery Local API Settings**, select the **Enable API Access** check box.

Making OAuth 2.0 Calls

The Mashery Local OAuth 2.0 API is available over SSL on port 8083 of your local instance.

Authorization is handled via HTTP authentication using the credentials you specified in the steps for Configuring OAuth 2.0 API Access.

Sample Call

The following is a sample call for OAuth 2.0.

```
curl -v -d '{
"method":"oauth2.fetchApplication",
"id": "2",
"params":{
        "service_key" : "<service_key>",
         "client" : {"client_id":"<api_key>",
                          "client_secret":"<api_secret>"},
                 "response_type" : null,
         "uri" : {"redirect_uri": "http://sometest.test.com/error?key=foo",
                     "state":"bar"}
        }
}' 'https://<masherylocal_host>:8083/v2/json-rpc/' -u '<api_access_user>:
<api_access_password>' -k
```

For host name, the IP Address of the instance where the OAuth API is enabled can be used. If the OAuth API on TIBCO Mashery® Local is to be used in conjunction with traffic outside the firewall, it is recommended that the TIBCO Mashery® Local cluster be fronted with a load balancer with a host name is associated with it (in addition to mapping port 8083 to an acceptable externally accessible port, such as 443).

Understanding the OAuth 2.0 API

The complete technical documentation for the Mashery OAuth 2.0 API is available online at http://support.mashery.com/docs/read/mashery_api/20/OAuth_Supporting_Methods. To see this documentation, request access by contacting your client services contact.

The following table describes the API at a high level:

API Method	Purpose
fetchApplication	Used during the Authorization step when the service provider’s authorization server presents the resource owner with information about the client requesting access to the resource owner’s data. The API calls is used to verify if the client is valid and fetches the client application data (name, attributes, redirection url) which will be used to provide information to the end user.
createAuthorizationCode (Authz Code grant type only)	After the resource owner has successfully authenticated against the service provider’s authorization server and authorized the client, the authz server will make this API call to TIBCO Mashery to generate the authz code which can be subsequently used to obtain an access token. As a part of this API call, the service provider will also supply the user-context (userid) for the authenticated user. The service provider returns the authz code to the client using the redirection url.
createAccessToken	API call used to generate the access token. • For the authz code grant type, a valid authz code must be presented. • For implicit and resource owner grant types, this occurs after the resource owner has been authenticated (user-context should be supplied). Service provider initiates the API call. • For Client Credentials flow, only the client credentials are verified. • When exchanging a refresh token, a valid refresh token must be presented. Both client id and secret must be presented when requesting an access token except in the case of Implicit grant type.

API Method	Purpose
fetchAccessToken	May be used by the service provider to validate access tokens and may be used as an additional layer of security or when certain API calls are sent directly to the provider instead of through TIBCO Mashery.
fetchUserApplications	Used by the service provider to present the resource owner with the client applications that been authorized by that resource owner. This is typically used in the **Account** section of the service provider's site where the resource owner can view the list.
revokeAccessToken	Used by the service provider to allow the resource owner to revoke access to specific client applications that been authorized by that resource owner. This is typically used in the "Account" section of the service provider's site where the resource owner can view the list authorized applications and select which application should no longer be allowed access.
revokeUserApplication	Revokes all tokens for an application for the specified user.

Configuring JMX Reporting Access

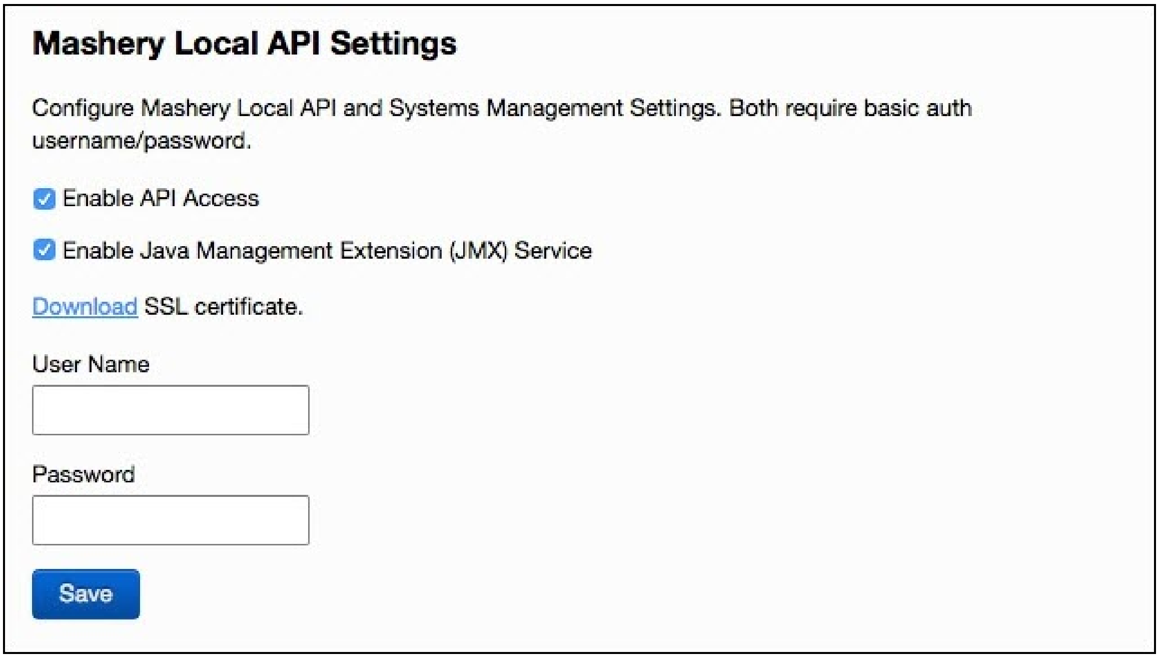
You can enable the JMX service to allow you to monitor Mashery Local using the Java Management Extension. In addition to the standard MBeans exposed by JMX, the list of TIBCO Mashery-specific components and metrics that are exposed are:

- **Jetty**: Metrics exposed are Connections, Open connections, Thread pool low threads, Thread pool max threads, Thread pool min threads, Thread pool spawn or shrink
- **DB Connection Pools**: Metrics exposed are Active Connections, Timeout
- **Memcache Connection Pools**: Metrics exposed are Active connections, Connection Timeout, Max Reconnect Delay
- **HTTP Connection Pools**: Metrics exposed are Number of connections

To enable and configure the JMX Service, follow the instructions below:

Procedure

1. Click **Account Settings**.



Mashery Local API Settings

Configure Mashery Local API and Systems Management Settings. Both require basic auth username/password.

☑ Enable API Access

☑ Enable Java Management Extension (JMX) Service

Download SSL certificate.

User Name

Password

2. In the **Mashery Local API Settings** section, enable/configure the following fields to turn on JMX service:
   - **Enable Java Management Extension (JMX) Service**
   - **User Name** and **Password**: You need to enter a username and password to access the service.
3. Download the server certificate to your local machine using the link provided on the same configuration screen:
4. On your local computer, import the certificate to a new trust store. Use an arbitrary trust store password; for example, `trustpassword`.

```
keytool -import -file ~/Downloads/mashery-proxy.cer -keystore mashery-proxy-
jmxremote.jks
```

5. Launch JConsole with the newly-created trust store.

```
jconsole -J-Djavax.net.ssl.trustStore=mashery-proxy-jmxremote.jks -J-
Djavax.net.ssl.trustStorePassword=trustpassword
```

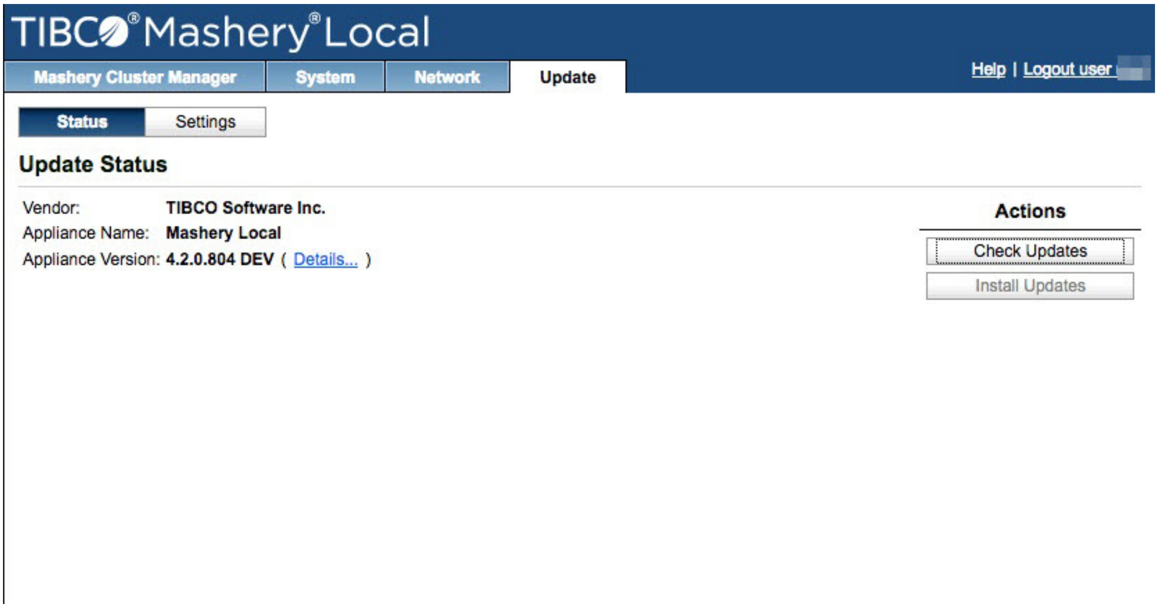
6. In JConsole, connect to the Mashery Local node to monitor by specifying its IP address (the second or private interface) and port 8084. Enter the credentials specified in the Admin console. For example: `192.168.1.110:8084 apiuser/apipassword`.

   You should be successfully connected and able to monitor the node.

Installing Updates

To find and install updates, follow the instructions below.

Prerequisite: The ISO update image should be mounted in the URL.



Update Status

Vendor: TIBCO Software Inc.
Appliance Name: Mashery Local
Appliance Version: 4.2.0.804 DEV (Details...)

Procedure

1. On the **TIBCO Mashery Local** page, click the **Update** tab.
   The **Status** tab is selected by default.
2. Click **Check Updates** to check if the latest version of the appliance is available for installation.

   Install Updates is enabled only when a latest version of the appliance is available.

   To update the Repository Settings, see Updating Repository Settings.
3. Click **Install Updates**.
   The appliance installation starts on the Mashery Local Instance.

   On completion of the process, the `Process Completed` message is displayed.

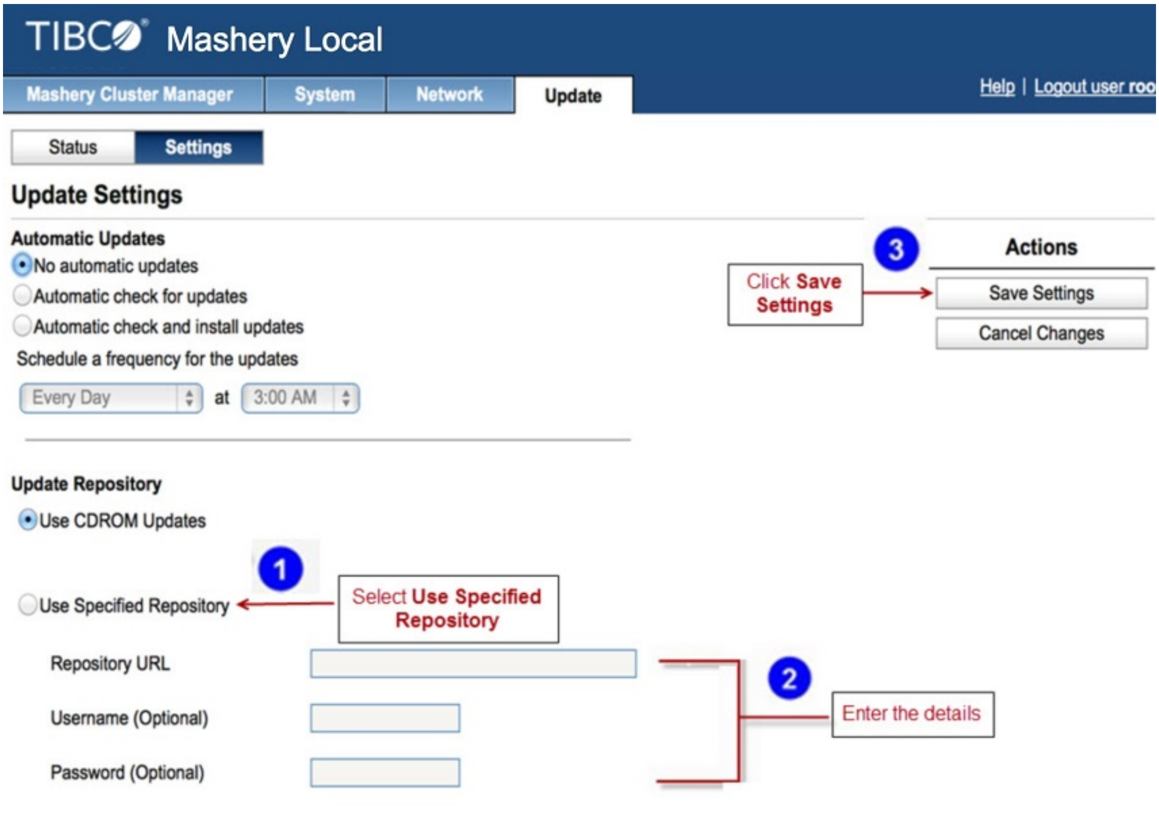
   - If the appliance update is successful, the **Appliance Version** on the **Update Status** section is updated automatically to the latest version.
   - If the appliance update fails, an error message is displayed in red.

Updating Repository Settings

To update the Repository Settings, follow the instructions in the screen shot:

Procedure

1. On the **Update** tab, click **Settings**.
   The **Settings** page is displayed.



Automatic Updates

● No automatic updates

○ Automatic check for updates

○ Automatic check and install updates

Schedule a frequency for the updates

Every Day at 3:00 AM

● Use CDROM Updates

○ Use Specified Repository

2. Click **Use Specified Repository**.
3. Enter the following details:

Field	Description
Repository URL	The URL on which the ISO is mounted in the **Settings** table.
Username	User name.
Password	Password.

4. Click **Save Settings**.
   The URL settings are saved.

Using the Adapter SDK

This section outlines the development process for writing custom adapters using the Adapter SDK for Mashery Local Traffic Manager. This section also provides the list of areas of extension provided in the SDK, along with code samples to illustrate the extension process.

Adapter SDK Package

The Adapter SDK defines the Traffic Manager domain model, tools and APIs and provides extension points to inject custom code in the processing of a call made to the Traffic Manager.

DIY SDK adapters need to be coded and compiled using JDK 1.6 or lower.

The Adapter SDK package contains the following:

- TIBCO Mashery Domain SDK
- TIBCO Mashery Infrastructure SDK

TIBCO Mashery Domain SDK

TIBCO Mashery Domain SDK packaged in com.mashery.trafficmanager.sdk identifies the traffic manager SDK and provides access to the TIBCO Mashery domain model which includes key objects such as Members, Applications, Developer Classes, Keys, Packages.

TIBCO Mashery Infrastructure SDK

TIBCO Mashery Infrastructure SDK provides the ability to handle infrastructure features and contains the following:

- **TIBCO Mashery HTTP Provider**

  The HTTP provider packaged as com.mashery.http provides HTTP Request/Response processing capability and tools to manipulate the HTTP Request, Response, their content and headers.

- **TIBCO Mashery Utility**

  The utility packaged as com.mashery.util provides utility code which handles frequently occurring logic such as string manipulations, caching, specialized collection handling, and logging.

SDK Domain Model

The Traffic Manager domain model defines the elements of the Traffic Manager runtime.

The following table highlights some of the key elements:

Element	Description	Usage
User	A user or member subscribing to APIs and accesses the APIs.	`com.mashery.trafficmanager.model.User`
API	An API represents the service definition. A service definition has endpoints defined for it.	`com.mashery.trafficmanager.model.API`

Element	Description	Usage

Endpoint	An Endpoint is a central resource of an API managed within Mashery. It is a collection of configuration options that defines the inbound and outbound URI's, rules, transformations, cache control, security, etc. of a unique pathway of your API.
	An Endpoint is specialized as either an API Endpoint or a Plan Endpoint. This specialization provides context to whether or not the Endpoint is being used as part of a Plan or not.
		• Generic endpoint entity representation:
		`com.mashery.trafficmanager.model.Endpoint`
		• API endpoint entity representation:
		`com.mashery.trafficmanager.model.APIEndpoint`
		• Plan endpoint entity representation:
		`com.mashery.trafficmanager.model.PlanEndpoint`

Method	A method is a function that can be called on an endpoint and represents the method currently being accessed/requested from the API request. A method could have rate and throttle limits specified on it to dictate the volume of calls made using a specific key to that method.
	A Method is specialized as either an API Method or Plan Method. The specialization provides context to whether or not the Method belong to a Plan.
		• Generic method entity representation:
		`com.mashery.trafficmanager.model.Method`
		• API method entity representation:
		`com.mashery.trafficmanager.model.APIMethod`
		• Plan method entity representation:
		`com.mashery.trafficmanager.model.PlanMethod`

Package	A Package is a mechanism to bundle or group API capability allowing the API Manager to then offer these capabilities to customers/users based on various access levels and price points. A Package represents a group of Plans.	`com.mashery.trafficmanager.model.Package`
Plan	A Plan is a collection of API endpoints, methods and response filters to group functionality so that API Product Managers can manage access control and provide access to appropriate Plans to different users.	`com.mashery.trafficmanager.model.Plan`

Element	Description	Usage
API Call	The API Call object is the complete transaction of the incoming request received by the Traffic Manager and the outgoing response as processed by the Traffic Manager. It provides an entry point into all other entities used in the execution of the request.	`com.mashery.trafficmanager.model.core.APICall`

Key	A key is an opaque string allowing a developer to access the API functionality. A key has rate and throttle controls defined on it and dictates the volume of calls that can be made to the API by the caller.
	A Key can be specialized as an API key or Package Key. This specialization provides context to whether the key provides access to an API or a specific Plan in a Package.
		• Generic key entity representation:
		`com.mashery.trafficmanager.model.Key`
		• API key entity representation:
		`com.mashery.trafficmanager.model.APIKey`
		• Package key entity representation:
		`com.mashery.trafficmanager.model.PackageKey`

Application	An application is a developer artifact that is registered by the developer when he subscribes to an API or a Package.	`com.mashery.trafficmanager.model.Application`
Rate Constraint	A Rate Constraint specifies how the amount of traffic is managed by limiting the number of calls per a time period (hours, days, months) that may be received.	`com.mashery.trafficmanager.model.RateConstraint`
Throttle Constraint	A Throttle Constraint specifies how the velocity of traffic is managed by limiting the number of calls per second that may be received.	`com.mashery.trafficmanager.model.ThrottleConstraint`
Customer Site	A customer specific area configured through the developer portal.	`com.mashery.trafficmanager.model.CustomerSite`

Extended Attributes

The traffic manager model allows defining name-value pairs on different levels of the model. The levels are identified here:

- Application
- Customer Site
- Key (both API Key and Package Key)
- Package

- Plan
- User

Pre and Post Processor Extension Points

This version of the SDK allows extensions for Processors only. This means that only pre and post processing of requests prior to invocation of the target host are allowed.

Listener Pattern

The extension API leverages a listener pattern to deliver callbacks to extension points to allow injecting custom logic.

A call made to the traffic manager is an invocation to a series of tasks. Each step in the workflow accomplishes a specific task to fulfill the call. The current API release only allows customization of the tasks prior to invoking the API server (pre-process) and post receipt of the response from the API server (post-process). The callback API handling these extensions is called a Processor.

The pre-process step allows a processor to receive a fully-formed HTTP request targeted to the API server. The processor is allowed to alter the headers or the body of the request prior to the request being made to the server. Upon completion of the request and receiving the response the Traffic Manager allows the processor to alter the response content and headers prior to the response flowing back through a series of exit tasks out to the client.

Event Types and Event

The transition of the call from one task to the next is triggered through ‘events’ and an event is delivered to any subscriber interested in receiving the event. The SDK supports two event-types which are delivered synchronously:

- **Pre-Process Event type**: This event is used to trigger any pre-process task.
- **Post-Process Event type**: This event is used to trigger any post-process task.
- **Authentication Event type**: This event is used to trigger any custom authentication.

The subscribers in this case will be Processors registered in a specific manner with the Traffic Manager API.

Event Listener API

The Traffic Manager SDK provides the following interface and is implemented by custom processors to receive Processor Events.

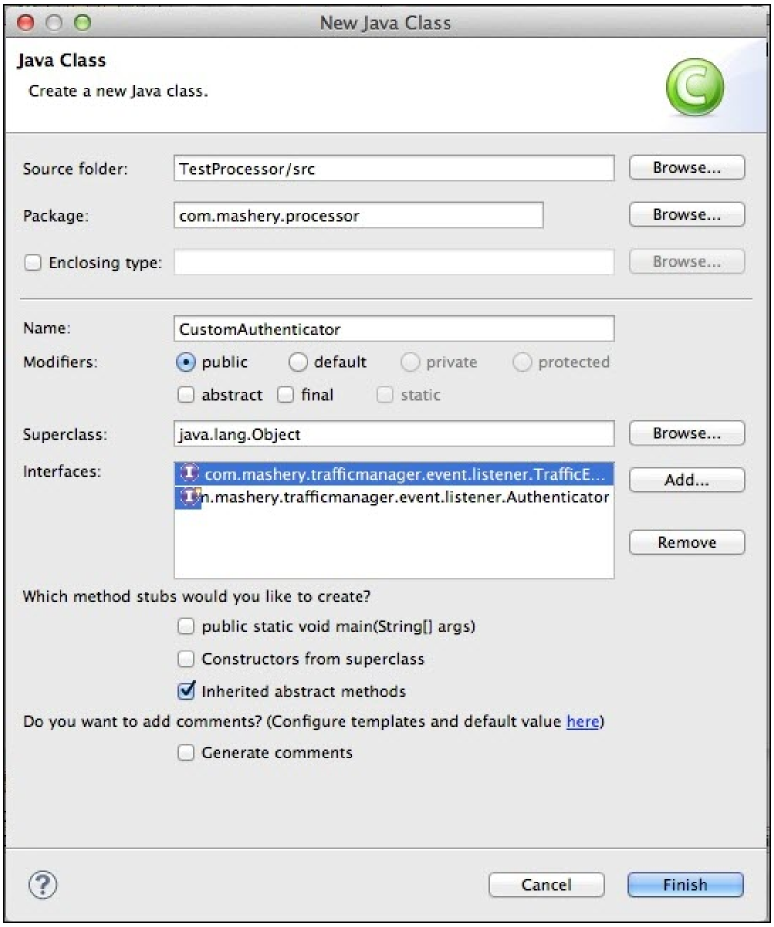
```
package com.mashery.trafficmanager.event.listener;
import com.mashery.trafficmanager.event.model.TrafficEvent;
/*** Event listener interface which is implemented by listeners which wish to
handle Traffic events. Traffic events will be delivered via this callback
synchronously to handlers implementing the interface.
The implementers of this interface subscribe to events via annotations. E.g.
Processor events need to handle events by using annotations in the
com.mashery.proxy.sdk.event.processor.annotation */
public interface TrafficEventListener {
   /*** The event is delivered to this API @param event*/
   void handleEvent(TrafficEvent event);
}
```

Creating a Custom Authenticator

This version of the SDK allows you to create a custom authenticator. This triggers an authentication event that a custom processor can handle for the custom authentication.

To create and use a custom authenticator, follow the steps below:

1. Download the SDK, then add the SDK to the class path for **com.mashery.trafficmanager.event.listener.TrafficEventListener** and **com.mashery.trafficmanager.event.listener.Authenticator**.
2. Create a new class. In Eclipse, select the Java project, right-click and select `New > Class` in the context menu.
3. Set the following fields as shown in the image below, implement the **com.mashery.trafficmanager.event.listener.TrafficEventListener** and **com.mashery.trafficmanager.event.listener.Authenticator** interface, then click **Finish**.



New Java Class

Java Class

Create a new Java class.

Source folder: TestProcessor/src

Package: com.mashery.processor

☐ Enclosing type:

Name: CustomAuthenticator

Modifiers: ● public ○ default ○ private ○ protected

☐ abstract ☐ final ☐ static

Superclass: java.lang.Object

Interfaces:
com.mashery.trafficmanager.event.listener.TrafficE...
n.mashery.trafficmanager.event.listener.Authenticator

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

☐ Generate comments

4. The new class should contain the following content:

```
package com.mashery.processor;
import com.mashery.trafficmanager.event.listener.Authenticator;
import com.mashery.trafficmanager.event.listener.TrafficEventListener;
import com.mashery.trafficmanager.event.model.TrafficEvent;

public class CustomAuthenticator implements TrafficEventListener, Authenticator {
    @Override
    public void handleEvent(TrafficEvent arg0) {
        // TODO Auto-generated method stub
    }
}
```

5. Add the following class annotation, as shown:

```
package com.mashery.processor;
import com.mashery.trafficmanager.event.listener.Authenticator;
import com.mashery.trafficmanager.event.listener.TrafficEventListener;
import com.mashery.trafficmanager.event.model.TrafficEvent;
import com.mashery.trafficmanager.processor.ProcessorBean;
```

```
@ProcessorBean(enabled = true, name = "CustomAuthProcessor", immediate = true)
public class CustomAuthenticator implements TrafficEventListener, Authenticator {
    @Override
    public void handleEvent(TrafficEvent arg0) {
        // TODO Auto-generated method stub
    }
}
```

The "name" attribute on the ProcessorBean annotation, `CustomAuthProcessor`, is what will be used to reference this class from the endpoint definition.

6. Update the **CustomAuthenticator** class with the following code snippet:

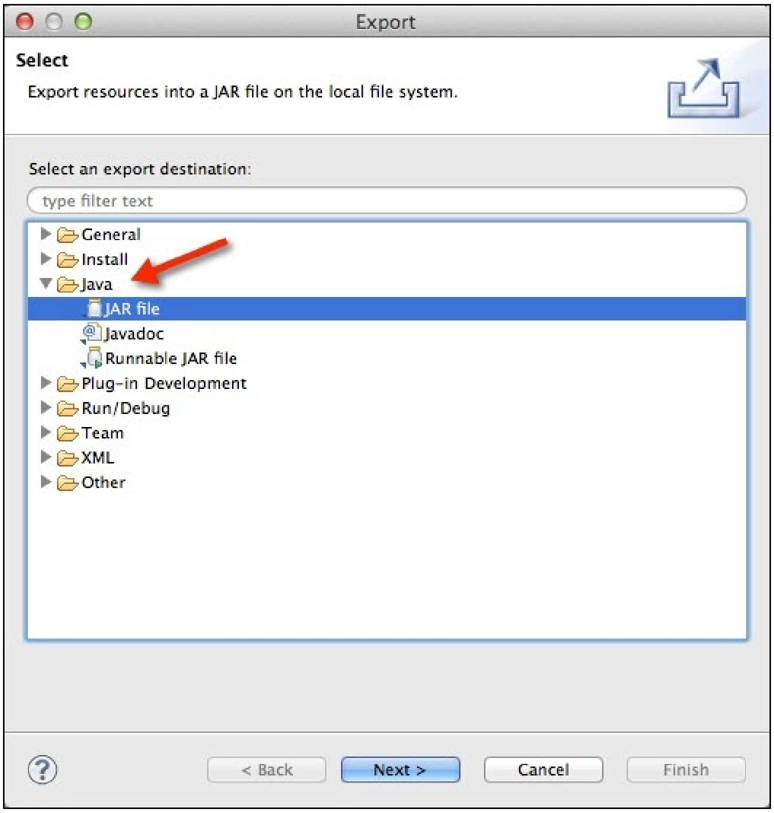
```
package com.mashery.processor;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.mashery.http.server.HTTPServerRequest;
import com.mashery.trafficmanager.event.listener.Authenticator;
import com.mashery.trafficmanager.event.listener.TrafficEventListener;
import com.mashery.trafficmanager.event.model.TrafficEvent;
import com.mashery.trafficmanager.event.processor.model.AuthenticationEvent;
import com.mashery.trafficmanager.event.processor.model.PostProcessEvent;
import com.mashery.trafficmanager.event.processor.model.PreProcessEvent;
import com.mashery.trafficmanager.processor.ProcessorBean;
import com.mashery.trafficmanager.processor.ProcessorException;

@ProcessorBean(enabled = true, name = "CustomAuthProcessor", immediate = true)
public class CustomAuthenticator implements TrafficEventListener, Authenticator {

    private final Logger log =
LoggerFactory.getLogger(CustomAuthenticator.class);
    @Override
    public void handleEvent(TrafficEvent event) {
        try {
            if (event instanceof PreProcessEvent) {
                //preProcess((PreProcessEvent) event);
            } else if(event instanceof PostProcessEvent){
                //postProcess((PostProcessEvent) event);
            } else if(event instanceof AuthenticationEvent){
                authCallprocess((AuthenticationEvent)event);
            }
        } catch (ProcessorException e) {
            log.error("Exception occurred when handling processor event");
        }
    }

    private void authCallprocess(AuthenticationEvent event) throws
ProcessorException{
        HTTPServerRequest httpRequest = event.getServerRequest();
        //add code to perform authentication of the request.
    }

}
```

Use **HttpServerRequest** to perform any authentication activity.

All the headers, status code and status messages set in the custom authentication would not be returned as part of response in case of Authentication Event handling (Custom authenticator). If you want to fail authentication request from custom authenticator, then you need to terminate the call in order to throw "ERR_403_NOT_AUTHORIZED" for a request. Refer to the example in Terminating a Call During Processing of an Event for more information.

7. Ensure there are no compilation errors and export the JAR file. In Eclipse, select the Java project, right click and select **Export** from the context menu.

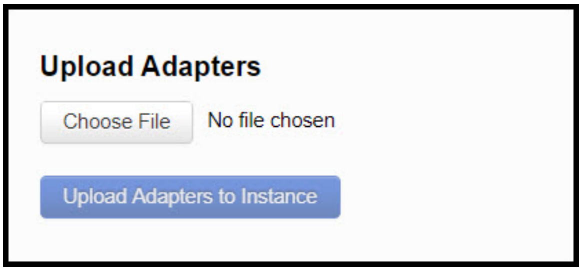
8. In the Export wizard, select and export destination of type `Java > Jar` file.



9. Provide the JAR file name and click **Finish**.

   You can ignore all warnings during the export process.

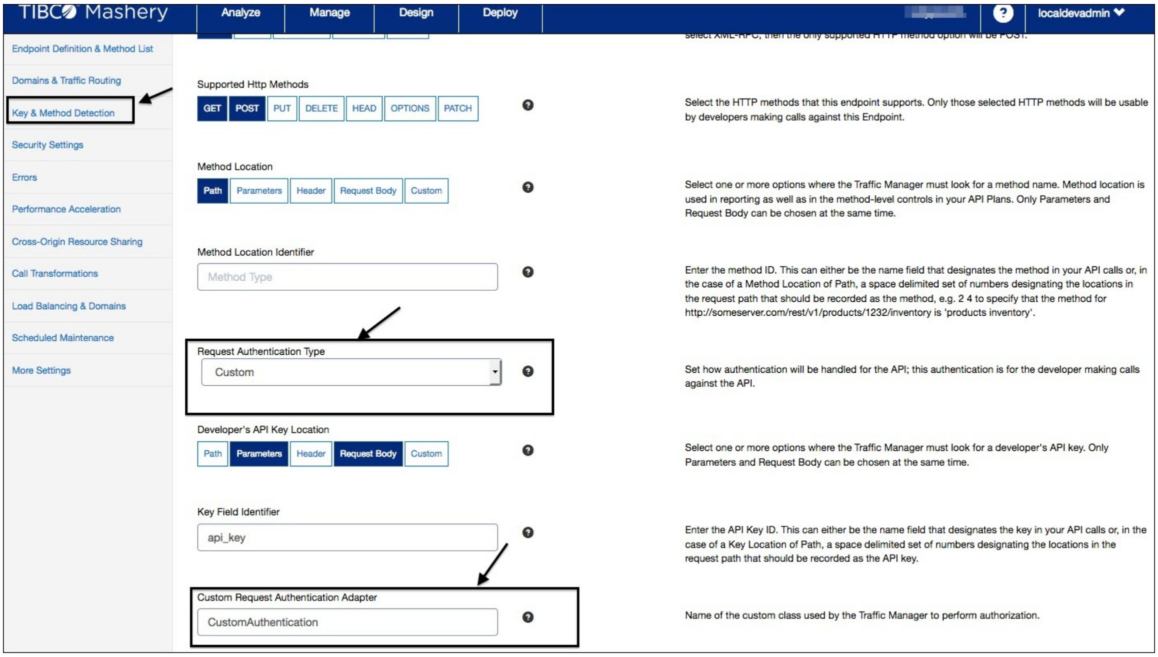
10. Create a zip archive, adding the exported JAR file created in the previous step.
11. Sign into the Mashery Local Admin UI and select **Adapter SDK** from the left menu.
12. In the **Upload Adapters** section, click **Choose File** and select the zip file created in Step 9, then click **Upload Adapters to Instance**.



If the upload is successful, a message appears that the adapters were updated successfully.

13. Update the endpoint's authentication configuration to use the custom authenticator. Log into TIBCO Mashery SaaS (Control Center), navigate to `Design > API > Select Endpoint > Key & Method Detection`, select **Request Authentication Type** as **Custom**, then enter the name of the custom authenticator in the **Custom Request Authentication Adapter** field.

    The name used for the **Custom Request Authentication Adapter** field is the `ProcessorBean` name used in Step 4.



select XML-RPC, then the only supported HTTP method option will be POST.

Supported Http Methods

GET **POST** PUT DELETE HEAD OPTIONS PATCH

Select the HTTP methods that this endpoint supports. Only those selected HTTP methods will be usable by developers making calls against this Endpoint.

Method Location

Path Parameters Header Request Body Custom

Select one or more options where the Traffic Manager must look for a method name. Method location is used in reporting as well as in the method-level controls in your API Plans. Only Parameters and Request Body can be chosen at the same time.

Method Location Identifier

Method Type

Enter the method ID. This can either be the name field that designates the method in your API calls or, in the case of a Method Location of Path, a space delimited set of numbers designating the locations in the request path that should be recorded as the method, e.g. 2 4 to specify that the method for http://someserver.com/rest/v1/products/1232/inventory is 'products inventory'.

Request Authentication Type

Custom

Set how authentication will be handled for the API; this authentication is for the developer making calls against the API.

Developer's API Key Location

Path **Parameters** Header **Request Body** Custom

Select one or more options where the Traffic Manager must look for a developer's API key. Only Parameters and Request Body can be chosen at the same time.

Key Field Identifier

api_key

Enter the API Key ID. This can either be the name field that designates the key in your API calls or, in the case of a Key Location of Path, a space delimited set of numbers designating the locations in the request path that should be recorded as the API key.

Custom Request Authentication Adapter

CustomAuthentication

Name of the custom class used by the Traffic Manager to perform authorization.

14. Save the configuration in the portal dashboard.

Implementing and Registering Processors

Writing custom processors involves the following general steps:

- Downloading the SDK
- Implementing the event listener
- Implementing lifecycle callback handling
- Adding libraries to the classpath

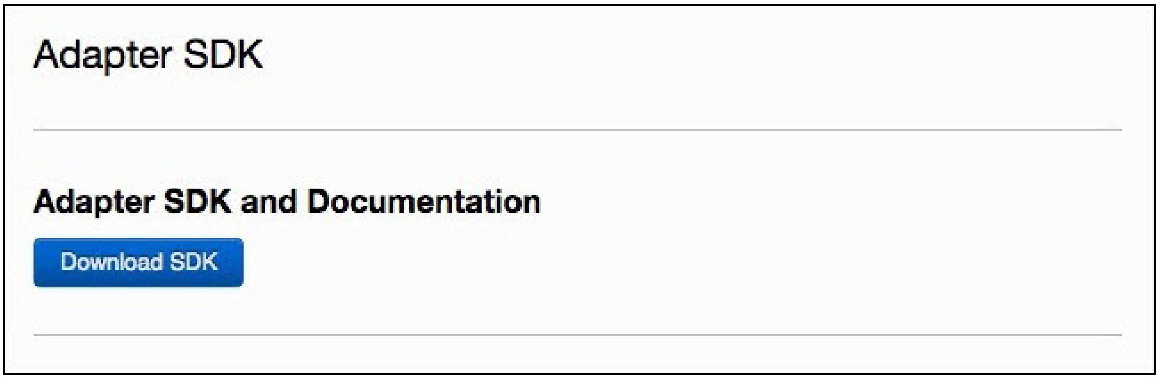
The following sections describe these steps in more detail.

Downloading the SDK

To download the SDK:

Procedure

1. Click **Adapter SDK**.
2. Click **Download SDK**, as shown in the screenshot:



3. Use your favorite IDE and put the SDK jars in your classpath.
4. Create a project and a new java class. The details of that process are skipped here and assumed that the developer will use the relevant IDE documentation to accomplish this.

Implementing the Event Listener

To implement the event listener:

Procedure

1. Employ the Traffic Event Listener interface (introduced in Event Listener API) as shown in the following example:

```
package com.company.extension;
public class CustomProcessor implements TrafficEventListener{
     public void handleEvent(TrafficEvent event){
        //write your custom code here
      }
}
```

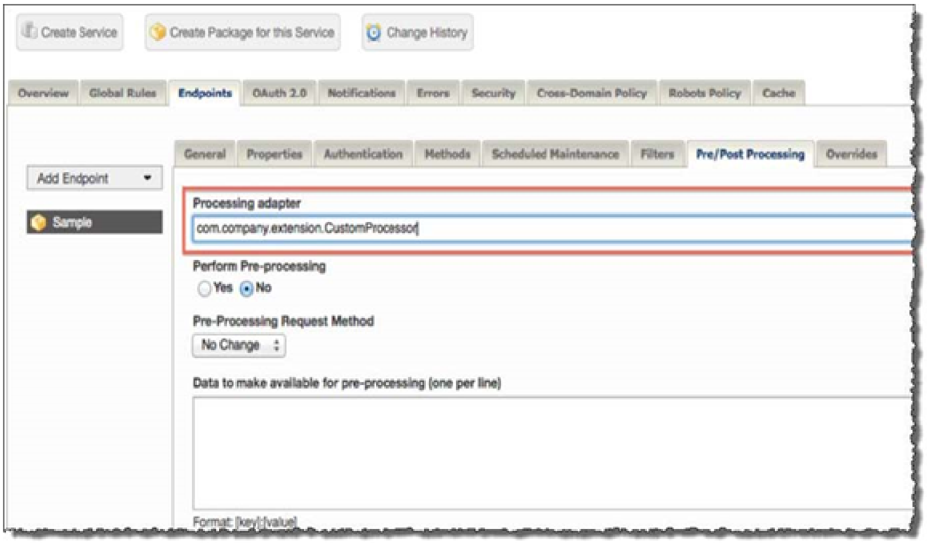
2. Annotate your code to ensure that the processor is identified correctly for callbacks on events related to the specific endpoints it is written to handle:

```
@ProcessorBean(enabled=true, name=”com.company.extension.CustomProcessor”,
immediate=true)
public class CustomProcessor implements TrafficEventListener{
     public void handleEvent(TrafficEvent event){
        //write your custom code here
    }
}
```

The annotation identifies the following properties:

- **enabled**: Identifies if the processor is to be enabled.
- **name**: Identifies the unique name of the processor as configured in API Settings (see marked area in ‘red’ in the following screenshot).
- **immediate**: Identifies if the processor is enabled immediately.

The name used in the annotation for the Processor MUST be the same as configured on the portal for the Endpoint>Pre/Post Processing, as shown in the following screenshot:



Processing adapter
com.company.extension.CustomProcessor

Perform Pre-processing
○ Yes ● No

Pre-Processing Request Method
No Change

Data to make available for pre-processing (one per line)

Format: [key]:[value]

Implementing Lifecycle Callback Handling

If you wish to have some initialization work done once and only once for each of the processors, then implement the following interface:

`package com.mashery.trafficmanager.event.listener`:

```
/*** The lifecycle callback which gets called when the processor gets loaded when
installed and released*/
public interface ListenerLifeCycle {
  /*** The method is called once in the life-cycle of the processor before the
processor is deemed ready to handle requests. If the processor throws an exception,
the activation is assumed to be a failure and the processor will not receive any
requests @throws ListenerLifeCycleException*/
  public void onLoad(LifeCycleContext ctx) throws ListenerLifeCycleException;

 /*** The method is called once in the life-cycle of the processor before the
processor is removed due. The processor will not receive any requests upon
inactivation.*/
  public void onUnLoad(LifeCycleContext ctx);
}
```

The **onLoad** call is made once prior to the processor handling any requests and **onUnLoad** call is made before the processor is decommissioned and no more requests are routed to it.

The lifecycle listener can be implemented on the Processor class or on a separate class. The annotation needs to add a reference to the lifecycle-class if the interface is implemented (see highlighted property in **bold**).

```
package com.company.extension;
@ProcessorBean(enabled=true, name=”com.company.extension.CustomProcessor”,
immediate=true, lifeCycleClass=”com.company.extension.CustomProcessor”)
public class CustomProcessor implements TrafficEventListener, ListenerLifeCycle{
  public void handleEvent(TrafficEvent event){
    //write your custom code here
  }
  public void onLoad(LifeCycleContext ctx) throws ListenerLifeCycleException{
  }
  public void onUnLoad(LifeCycleContext ctx){
  }
}
```

The `lifeCycleClass` property should point to the class implementing the Listener LifeCycle interface. This also allows having a separate lifecycle listener interface as follows (note the different lifeCycleClass name).

The following example shows a different class implementing the LifeCycle callback:

```
package com.company.extension;
@ProcessorBean(enabled=true, name=”com.company.extension.CustomProcessor”,
immediate=true, lifeCycleClass=”com.company.extension.CustomProcessorLifeCycle”)
public class CustomProcessor implements TrafficEventListener {
   public void handleEvent(TrafficEvent event){
     //write your custom code here
   }
   public void onLoad(LifeCycleContext ctx) throws ListenerLifeCycleException{
   }
   public void onUnLoad(LifeCycleContext ctx){
   }
}
public class CustomProcessorLifeCycle implements ListenerLifeCycle{
   public void onLoad(LifeCycleContext ctx) throws ListenerLifeCycleException{
   }
   public void onUnLoad(LifeCycleContext ctx){
   }
}
```

Adding Libraries to Classpath

If the processor needs third-party libraries, those can be used in development and packaged with the processors, as described in Deploying Processors to Runtime.

Deploying Processors to Runtime

Deploying a custom processor involves the following general steps:

- Packaging the custom processor
- Uploading the custom processor
- Enabling Debugging

The following sections describe these steps in more detail.

Packaging the Custom Processor

To package your custom processor once the processor code is written, perform the following steps:

1. Compile the classes and create a JAR file with all the classes.
2. Specify third party libraries used in the `MANIFEST.MF` of the JAR containing the processor classes. and should be introduced as follows:
    - Third party libraries should be listed as values of the property **Class-Path** in the `MANIFEST.MF`.
    - Only third party library names are used, do not include paths to libraries.
    - Third party library names should be separated by spaces.

    Example: In `META-INF/MANIFEST.MF`:

    ```
    Class-Path: org.apache.commons.codec_1.4.0.v201209201156.jar
    com.amazonaws_1.1.9.jar
    ```
3. Create a folder "`lib`" alongside the JAR file, then copy all third party libraries to the "`lib`" folder.
4. Zip the JAR file and all contents of the "`lib`" folder to a ZIP package.

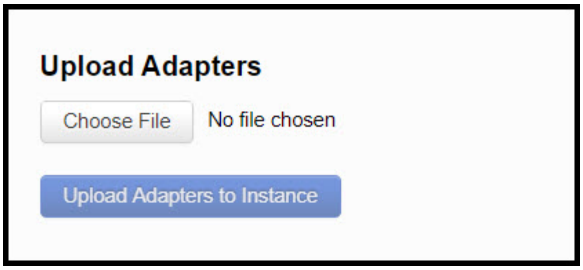
For more information, please refer to the Adapter SDK Usage and Examples section in the Appendix.

Uploading the Custom Processor

To upload the custom processor:

Procedure

1. In the **Upload Adapters** section, click **Choose file** to navigate and select the zip file containing the JAR file.
2. Click **Upload Adapters to Instance** to upload the package .ZIP file to the Mashery Local instance.



Upload Adapters

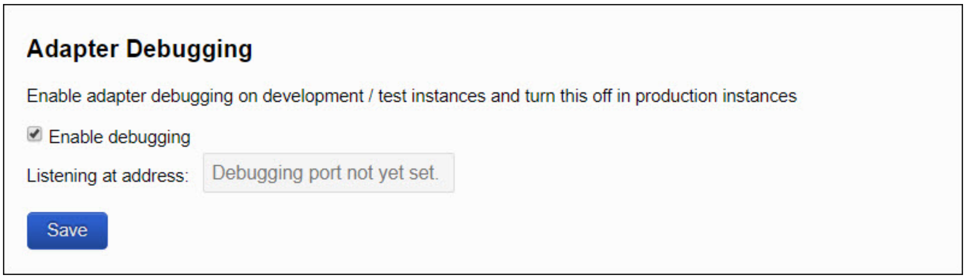
No file chosen

If the upload is successful, a message appears that the adapters were uploaded successfully.

Enabling Debugging

During development, it is sometimes necessary to enable debugging on the Mashery Local instance.

To enable debugging, click the **Enable debugging** check box, indicate the port number to which you will connect your debugger, and then click **Save**:



Adapter Debugging

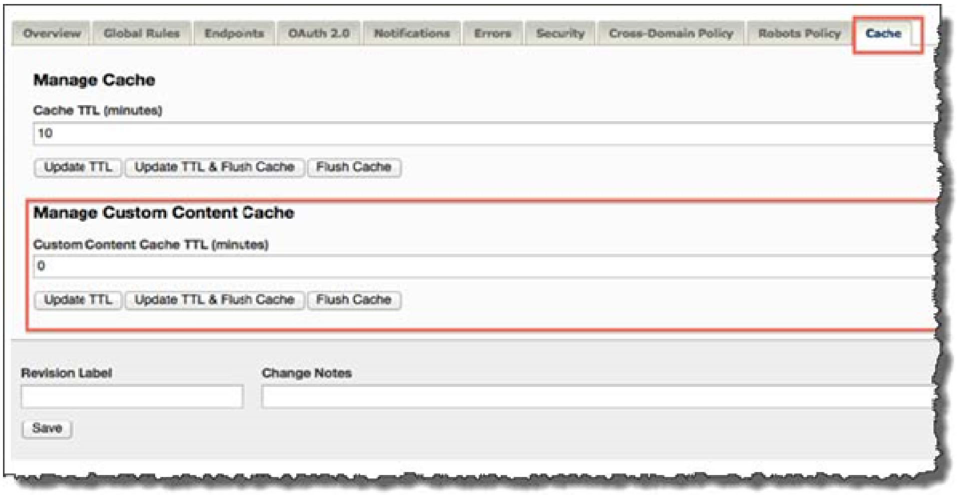
Enable adapter debugging on development / test instances and turn this off in production instances

☑ Enable debugging

Listening at address: Debugging port not yet set.

Caching Content

The custom endpoints can cache content during the call handling. The cache configuration is found in the **Manage Custom Content Cache** section on the **API Settings** page.



Manage Cache

Cache TTL (minutes): 10

Manage Custom Content Cache

Custom Content Cache TTL (minutes): 0

Revision Label

Change Notes

Manage Custom Content Cache provides the following options:

- **Custom TTL**: A default TTL provided for the cache.
- **Update TTL**: Provides ability to save any TTL changes.
- **Update TTL & Flush Cache**: Updates the database with the updated TTL and flushes the cache contents.
- **Flush Cache**: Allows the cache contents to be flushed.

The SDK provides references to a Cache where all this data is stored. The cache interface provided in the callback to the `TrafficEventListener` is:

```
package com.mashery.trafficmanager.cache;
/*** Cache API which allows extensions to store and retrieve data from cache*/
public interface Cache {
  /**
  * Retrieves the value from the cache for the given key
  * @param key
  * @return
  * @throws CacheException
  */
  Object get(String key) throws CacheException;
  /**
  * Puts the value against the key in the cache for a given ttl
  * @param key
  * @param value
  * @param ttl
  * @throws CacheException
  */
  void put(String key, Object value, int ttl) throws CacheException;
}
```

A reference to the cache can be found on the `ProcessorEvent` which is reported on the callback. Here is an example of how to access cache on callback:

```
package com.company.extension;
@ProcessorBean(enabled=true, name=”com.company.extension.CustomProcessor”,
immediate=true
public class CustomProcessor implements TrafficEventListener, ListenerLifeCycle{
  public void handleEvent(TrafficEvent event){
       ProcessorEvent processorEvent = (ProcessorEvent) event;
       Cache cacheReference = processorEvent.getCache();
       //Add data to cache
       try{
           cacheReference.put(“testkey”, “testValue”, 10)
       }catch(CacheException e){
        //load data or load default data
       }
    //write your custom processor code here
    }
}
```

A reference to cache is also available on the lifecycle callback:

```
package com.company.extension;
public class CustomProcessorLifeCycle implements ListenerLifeCycle{
   public void onLoad(LifeCycleContext ctx) throws ListenerLifeCycleException{
      Cache cache = ctx.getCache();
      // perform cache operations
   }
   public void onUnLoad(LifeCycleContext ctx){
   }
}
```

Terminating a Call During Processing of an Event

This version of the SDK allows a user to terminate a call during pre or post processing, or in authentication event handling. For example, if the request does not have a required URL parameter, Mashery Local can be configured to terminate the call in the pre-processing.

All the headers, status code and status messages set in the custom processing is returned to the client as part of the response in pre processing and post processing.

All the headers, status code and status messages set in the custom authentication would not be returned as part of response in case of Authentication Event handling (Custom authenticator). If you want to fail authentication request from the custom authenticator, then you need to terminate the call in order to throw "ERR_403_NOT_AUTHORIZED" for a request.

For example, if you want to terminate the call in authenticator, if request doesn't contain the authorization header, then the call can be terminated by marking the response as complete as shown in the following example:

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.mashery.http.HTTPHeaders;
import com.mashery.trafficmanager.debug.DebugContext;
import com.mashery.trafficmanager.event.listener.Authenticator;
import com.mashery.trafficmanager.event.listener.TrafficEventListener;
import com.mashery.trafficmanager.event.model.TrafficEvent;
import com.mashery.trafficmanager.event.processor.model.AuthenticationEvent;
import com.mashery.trafficmanager.event.processor.model.PostProcessEvent;
import com.mashery.trafficmanager.event.processor.model.PreProcessEvent;
import com.mashery.trafficmanager.processor.ProcessorBean;
import com.mashery.trafficmanager.processor.ProcessorException;

@ProcessorBean(enabled = true, name = "CustomAuthentication", immediate = true)

public class CustomAuthentication implements TrafficEventListener,Authenticator {

    @Override
    public void handleEvent(TrafficEvent event) {
        try {
            if (event instanceof AuthenticationEvent) {
                authenticate((AuthenticationEvent) event);
            }
        } catch (ProcessorException e) {
        }
    }

    private void authenticate(AuthenticationEvent event)
            throws ProcessorException {
        //For example request doesn't contain the authorization header then user
can terminate the call by marking response as complete
        // in order to thrown 403 ERR_403_NOT_AUTHORIZED for the incoming request.
        if (headers != null) {
            String authorization = headers.get(HEADER_AUTHORIZATION);
            if ((null == authorization || authorization == "")
                    || !authorization.startsWith(AUTH_BASIC)) {
                debugContext.logEntry("Final Value", "DIY-CUSTOM-AUTH-HEADER-
FAILIURE");
                event.getCallContext().getResponse().setComplete();
            }
    }
}
```

If you want to terminate the call in pre or post processing, refer to the following example:

```
package com.mashery.processor;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.mashery.trafficmanager.event.listener.TrafficEventListener;
import com.mashery.trafficmanager.event.listener.Authenticator;
import com.mashery.trafficmanager.event.model.TrafficEvent;
import com.mashery.trafficmanager.event.processor.model.PostProcessEvent;
import com.mashery.trafficmanager.event.processor.model.PreProcessEvent;
import com.mashery.trafficmanager.model.core.ExtendedAttributes;
import com.mashery.trafficmanager.processor.ProcessorBean;
import com.mashery.trafficmanager.processor.ProcessorException;

@ProcessorBean(enabled = true, name = "PrePostProcessing", immediate = true)

public class PrePostProcessing implements TrafficEventListener{

    private final Logger log = LoggerFactory.getLogger(PrePostProcessing.class);

    @Override
    public void handleEvent(TrafficEvent event) {
        try {
            if (event instanceof PreProcessEvent) {
                preProcess((PreProcessEvent) event);
            } else if (event instanceof PostProcessEvent) {
                postProcess((PostProcessEvent) event);
            }
        } catch (ProcessorException e) {
            log.error("Exception occurred when handling processor event");
        }
    }

   //In the below example we checking the query parameter's value to decide whether
to terminate the call or not.
    private void preProcess(PreProcessEvent event) throws ProcessorException {
        String complete =
event.getCallContext().getRequest().getQueryData().get("preComplete");
        if (complete != null) {
            event.getCallContext().getResponse().getHTTPResponse().setBody(new
StringContentProducer("{\"response\": \"Terminated the call in pre-processing\"}"));
            event.getCallContext().getResponse().setComplete();
        }
    }
   //In the below example we checking the query parameter's value to decide whether
to terminate the call or not.
    private void postProcess(PostProcessEvent event) throws ProcessorException {
       String complete =
event.getCallContext().getRequest().getQueryData().get("postComplete");
        if (complete != null) {
            event.getCallContext().getResponse().getHTTPResponse().setBody(new
StringContentProducer("{\"response\": \"Terminated the call in post-processing
\"}"));
            event.getCallContext().getResponse().setComplete();
        }
    }
}
```

Accessing Package Key EAVs in the Custom Processor

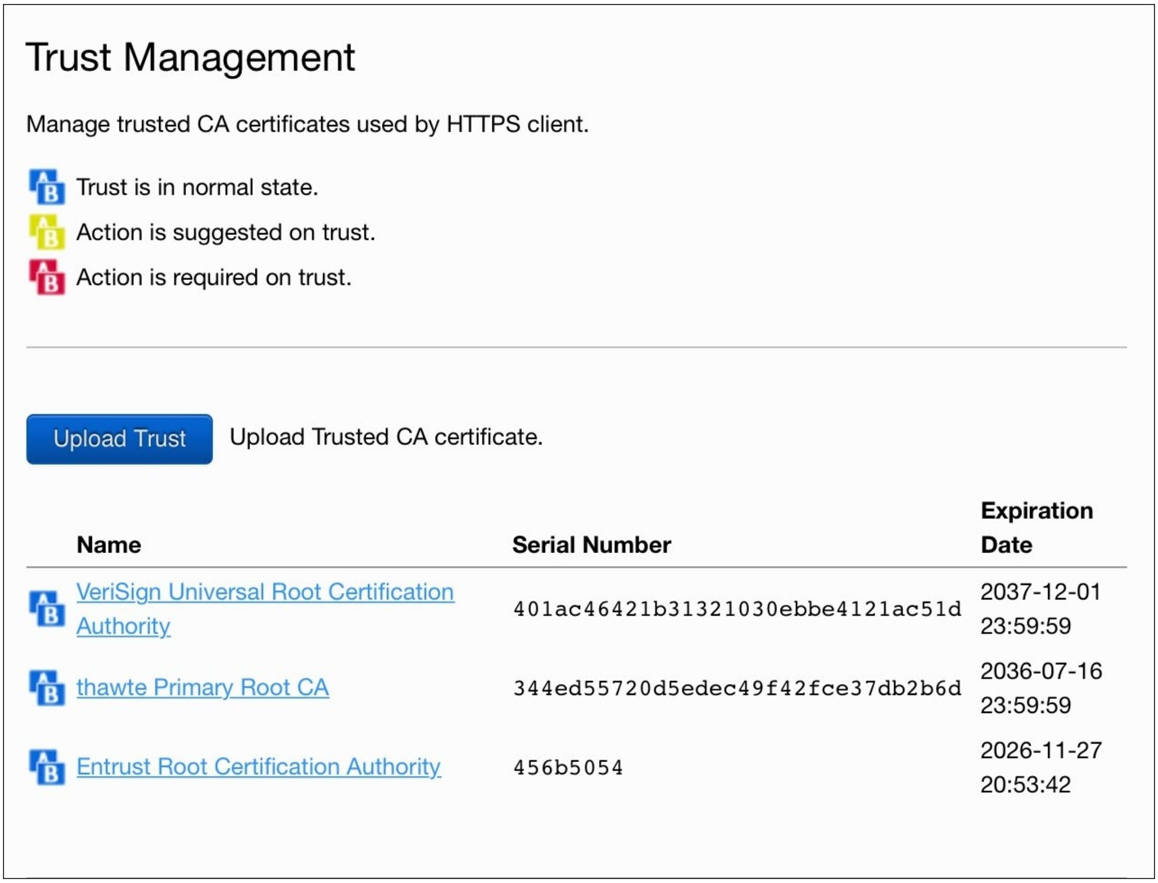
This version of the SDK includes an example custom processor that can access package EAVs (Extended Attribute Values).

For example, this sample processor looks for "user_defined_key" EAVs for a package key:

```
package com.mashery.processor;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import com.mashery.trafficmanager.event.listener.TrafficEventListener;
import com.mashery.trafficmanager.event.listener.Authenticator;
import com.mashery.trafficmanager.event.model.TrafficEvent;
import com.mashery.trafficmanager.event.processor.model.PostProcessEvent;
import com.mashery.trafficmanager.event.processor.model.PreProcessEvent;
import com.mashery.trafficmanager.model.core.ExtendedAttributes;
import com.mashery.trafficmanager.processor.ProcessorBean;
import com.mashery.trafficmanager.processor.ProcessorException;

@ProcessorBean(enabled = true, name = "PrePostProcessing", immediate = true)

public class PrePostProcessing implements TrafficEventListener,Authenticator {

    private final Logger log = LoggerFactory.getLogger(PrePostProcessing.class);

    @Override
    public void handleEvent(TrafficEvent event) {
        try {
            if (event instanceof PreProcessEvent) {
                preProcess((PreProcessEvent) event);
            } else if (event instanceof PostProcessEvent) {
                postProcess((PostProcessEvent) event);
            } else if (event instanceof AuthenticationEvent) {
               authenticate((AuthenticationEvent) event)
            }
        } catch (ProcessorException e) {
            log.error("Exception occurred when handling processor event");
        }
    }

    //In the below snippet we are extracting the package key and checking its value.

    private void preProcess(PreProcessEvent event) throws ProcessorException {
        ExtendedAttributes attrs = (event).getKey().getExtendedAttributes();
        String strAllowed = attrs.getValue("user_defined_key");
    }
    private void postProcess(PostProcessEvent event) throws ProcessorException {
        ExtendedAttributes attrs = (event).getKey().getExtendedAttributes();
        String strAllowed = attrs.getValue("user_defined_key");
    }
    private void authenticate(AuthenticationEvent event)
            throws ProcessorException {
        ExtendedAttributes attrs = (event).getKey().getExtendedAttributes();
        String strAllowed = attrs.getValue("user_defined_key");
    }
}
```

Configuring Trust Management

The **Trust Management** page allows the administrator to add or update certificates used by the HTTPS client. The HTTPS client profile references these certificates.



Trust Management

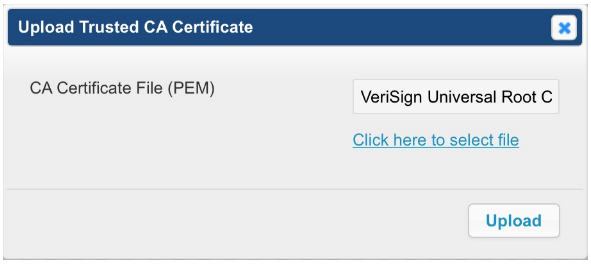
Manage trusted CA certificates used by HTTPS client.

🟦 Trust is in normal state.
🟨 Action is suggested on trust.
🟥 Action is required on trust.

Upload Trust — Upload Trusted CA certificate.

Name	Serial Number	Expiration Date
🟦 VeriSign Universal Root Certification Authority	401ac46421b31321030ebbe4121ac51d	2037-12-01 23:59:59
🟦 thawte Primary Root CA	344ed55720d5edec49f42fce37db2b6d	2036-07-16 23:59:59
🟦 Entrust Root Certification Authority	456b5054	2026-11-27 20:53:42

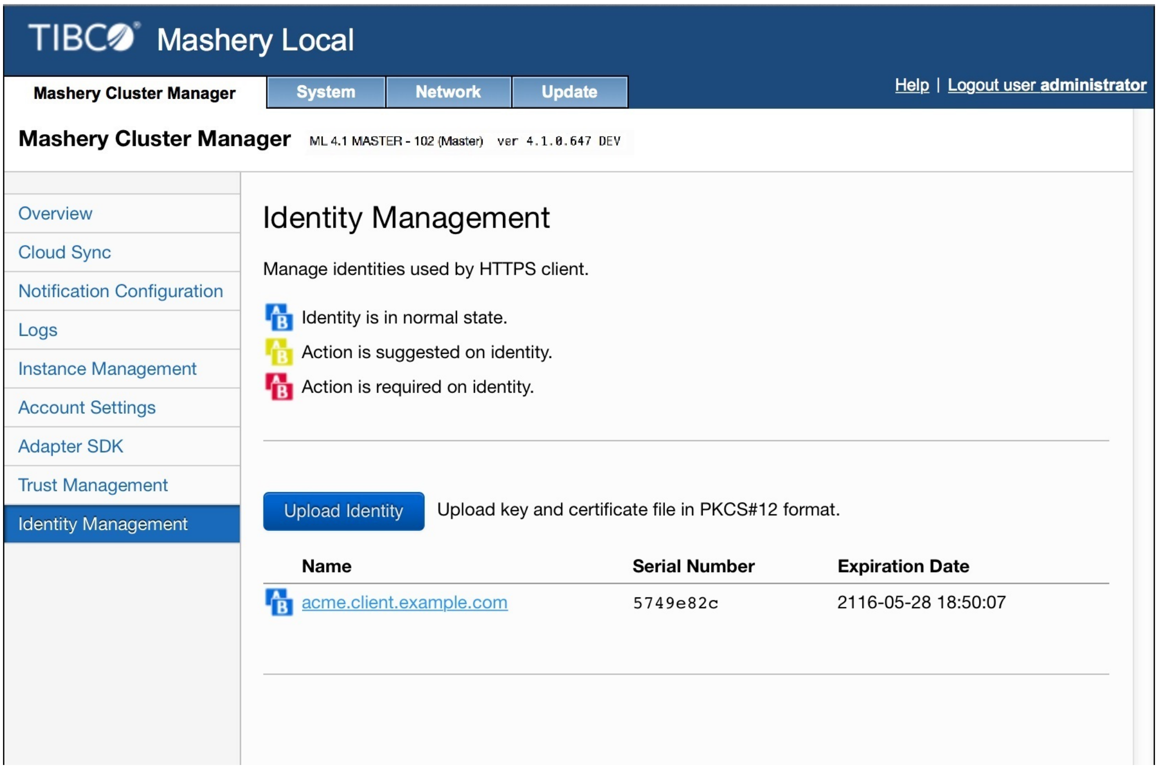
The following table describes the fields in the **Trust Management** page.

Field or button	Description
Upload Trust	Opens the **Upload a Trusted CA Certificate** window.  Upload Trusted CA Certificate CA Certificate File (PEM): VeriSign Universal Root C Click here to select file Upload To upload a certificate, click **Click here to select file**, then click **Upload**.

Field or button	Description
Name	The name of the certificate.
Serial Number	The serial number of the certificate.
Expiration Date	The date and time the certificate expires.
State	Identifies the following information: • The state of the certificate: – Certificate manifest will be synchronized with TIBCO Mashery SaaS – Certificate manifest has been synchronized with TIBCO Mashery SaaS – Certificate is about to expire - The expiration warning is shown one month before expiration date. – Certificate expired. – Certificate manifest update will be synchronized with TIBCO Mashery SaaS – Certifcate in TIBCO Mashery Local is outdated for TIBCO Mashery SaaS – Certificate is not present in TIBCO Mashery Local • The number of profile(s) using the certificate. • The number of endpoint(s) using the certificate. • The available action suggested or required for the certificate.

Configuring Identity Management

The **Identity Management** page allows the administrator to add or update identities used by the HTTPS client. The HTTPS client profile references these identities.



Mashery Cluster Manager ML 4.1 MASTER - 102 (Master) ver 4.1.0.647 DEV

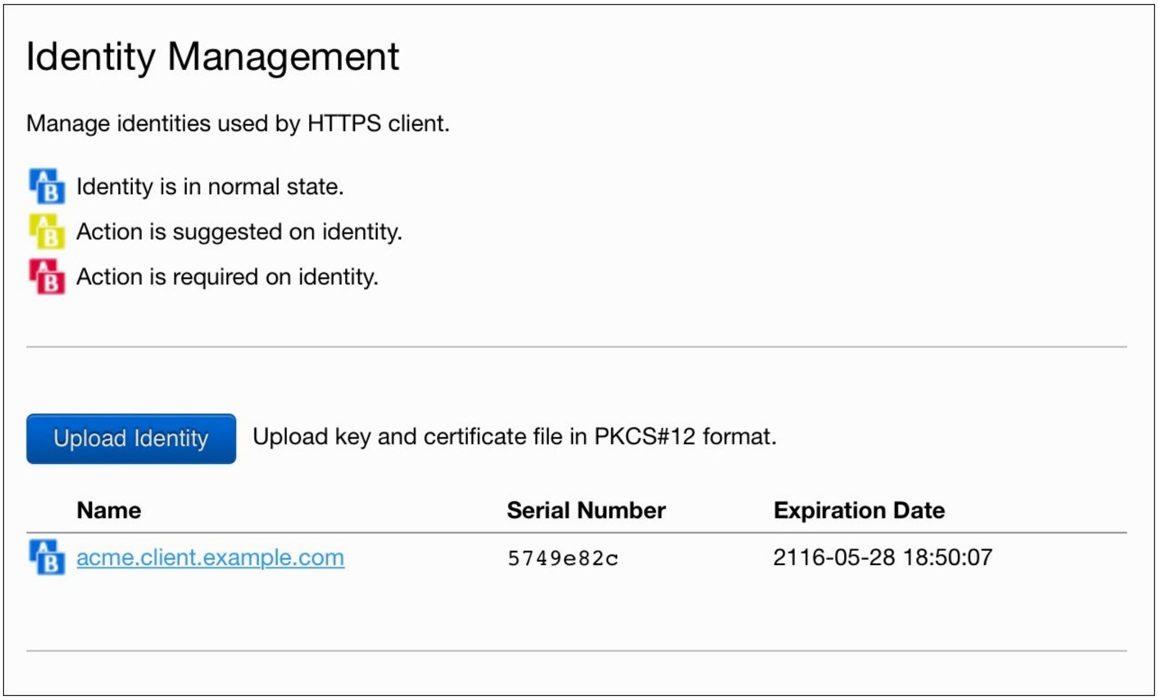
Identity Management

Manage identities used by HTTPS client.

🟦 Identity is in normal state.
🟨 Action is suggested on identity.
🟥 Action is required on identity.

Upload key and certificate file in PKCS#12 format.

	Name	Serial Number	Expiration Date
🟦	acme.client.example.com	5749e82c	2116-05-28 18:50:07



Identity Management

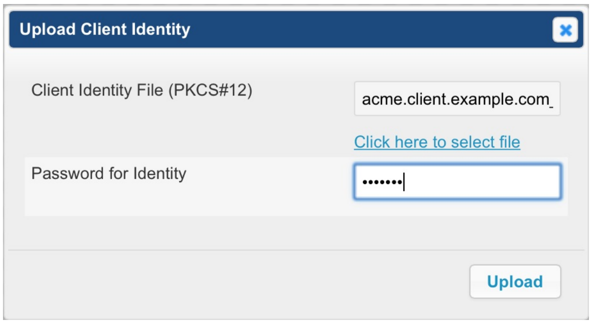
Manage identities used by HTTPS client.

🟦 Identity is in normal state.
🟨 Action is suggested on identity.
🟥 Action is required on identity.

Upload key and certificate file in PKCS#12 format.

	Name	Serial Number	Expiration Date
🟦	acme.client.example.com	5749e82c	2116-05-28 18:50:07

The following table describes the fields in the **Identity Management** page.

Field or button	Description
Upload Identity	Opens the **Upload a Trusted CA Certificate** window.  Client Identity File (PKCS#12): acme.client.example.com_ Password for Identity: ••••••• To upload an identity, click **Click here to select file**, enter the **Password**, then click **Upload**.
Name	The name of the identity.
Serial Number	The serial number of the identity.
Expiration Date	The date and time the identity expires.
State	Identifies the following information: • The state of the identity: – Certificate manifest will be synchronized with TIBCO Mashery SaaS – Certificate manifest has been synchronized with TIBCO Mashery SaaS – Certificate is about to expire - The expiration warning is shown one month before expiration date. – Certificate expired. – Certificate manifest update will be synchronized with TIBCO Mashery SaaS – Certifcate in TIBCO Mashery Local is outdated for TIBCO Mashery SaaS – Certificate is not present in TIBCO Mashery Local • The number of profile(s) using the identity. • The number of endpoint(s) using the identity. • The available action suggested or required for the identity.

Testing the New Instance

You should test a new instance after installing and creating it.

Testing a New Instance

One approach to test a new instance is:

Procedure

1. Find the API to test in the **API Settings** area of the TIBCO Mashery Admin Dashboard and identify an associated endpoint that is ready for testing.
2. Create a test API key for the API identified in the previous step. You accomplish this in the **Users** area accessed by clicking the **Users** tab of the TIBCO Mashery Admin Dashboard.
3. Perform a manual sync of the Services and Developers in the **Cloud Sync** page of the Mashery Local Cluster Manager, as described in step 7 of Configuring Slaves to the Master.
4. Construct a test API call for the API you wish to test.
5. Execute the API call against the instance. Unless you have set up a domain name for the instance, your API call will need to be made against the IP address of the instance directly. You can find the IP address as described in step 12 of Deploying the Mashery Local OVF Template.

   > Should you use a hostname or IP in your test call? When a service is setup in the dashboard, the hostnames (IP addresses as well could be used) that will consume the service are defined. When a call is made to the proxy, the hostname used for the call must match one of the hostnames setup in the dashboard for the service, otherwise the call will fail. If you make a call directly to one of the instances using its IP address and that IP address was not configured in the service definition, then the proxy returns a 596 error.

   If you receive the expected response from the API, then your instance is working properly.

Tracking the Database Restore and Replication Status

Mashery Local slave node registration process provides a status endpoint that helps to track the asynchronous steps of database restore and replication status. However, these steps are processed in the background, so there is no active feedback on completion or failure of processes.

> The status endpoint (registration_status.py) is an experimental API and is subject to change in later implementations.

There is no need to Enable API Access on the node for this endpoint to function. See Configuring OAuth 2.0 API Access.

To obtain the database restore and replication status, type the URL of the endpoint in your browser. For example: `https://<Mashery_Local_Slave_IP>:5480/service/mashery/cgi/replication_status.py`

The endpoint returns the following JSON:

```
{
  "replication_status":
  {
    "restore": {
               "error": false,
               "errors": "",
               "log": "1. transfer backup from master\nMon Aug 22 18:38:04
UTC
                       2016\n\n2. unzip backup\nMon Aug 22 18:38:41 UTC 2016\n\n3.
                       stop slave\nSlave stopped\nMon Aug 22 18:38:41 UTC
                       2016\n\n4. restore from backup\nMon Aug 22 20:36:18 UTC
                       2016\n\n5. start slave\nSlave started\nMon Aug 22 20:36:18
                       UTC 2016\n\n6. done\nMon Aug 22 20:36:18 UTC 2016\n\n",
               "complete": true
               },
  "replication": {
                 "last_error": "",
                 "seconds_behind_master": "250832\n",
                 "slave_io_running": "Yes\n",
                 "slave_sql_running": "Yes\n"
                 }
  },
  "error": null
}
```

The following table provides details about the JSON that is returned by the status endpoint:

JSON Node	Value in JSON	Description
Database restore log		
`replication_status.restore.log`		Provides database restore log.
`replication_status.restore.complete`	true	Implies that the database restore step is done.
	false	Implies that the database restore step is not complete.
`replication_status.restore.error`	true	Implies that there are errors during the process. Refer to the `replication_status.restore.errors` node to get more details for the errors.
	false	Implies that there are errors during the process.
Database replication status		
`replication_status.replication`		Provides replication-specific information from “show slave status”.
`replication_status.replication.last_error`	<error>	Provides the details for the errors (if any).
`replication_status.replication.seconds_behind_master`	<time in milliseconds>	Provides an estimate of how long it takes the slave node to catch up to the master.

JSON Node	Value in JSON	Description
`replication_status.replication.slave_io_running` or `replication_status.replication.slave_sql_running`	No Yes	Replication is not running and `last_error` provides the details for the errors (if any). `Replication starts`, `seconds_behind_master` provides an estimate of how long it takes the slave node to catch up to the master.

When the restore step is in process, replication is disabled. Therefore, the value of `slave_io_running` and `slave_sql_running` is `No`.

Troubleshooting

Mashery Local provides a set of tools that help an administrator to debug issues with API calls as they flow through TIBCO Mashery, troubleshoot networking issues with the system, identify issues with cloud synchronization, and collect system logs to facilitate Operations and Support staff to identify root-cause faster. This section outlines the tools available and their usage scenarios.

Verbose Logs

The Mashery Local administrator can troubleshoot issues related to call payloads or identify any inconsistencies as API call data flows through TIBCO Mashery by enabling verbose logs on API calls. This feature is not enabled as an “always on” feature as producing these verbose logs may have some impact on API call performance. Instead, options are provided on the **Cluster Manager** UI to enable and disable verbose logs.

Using the Verbose Logs Feature

To use the Verbose Log feature:

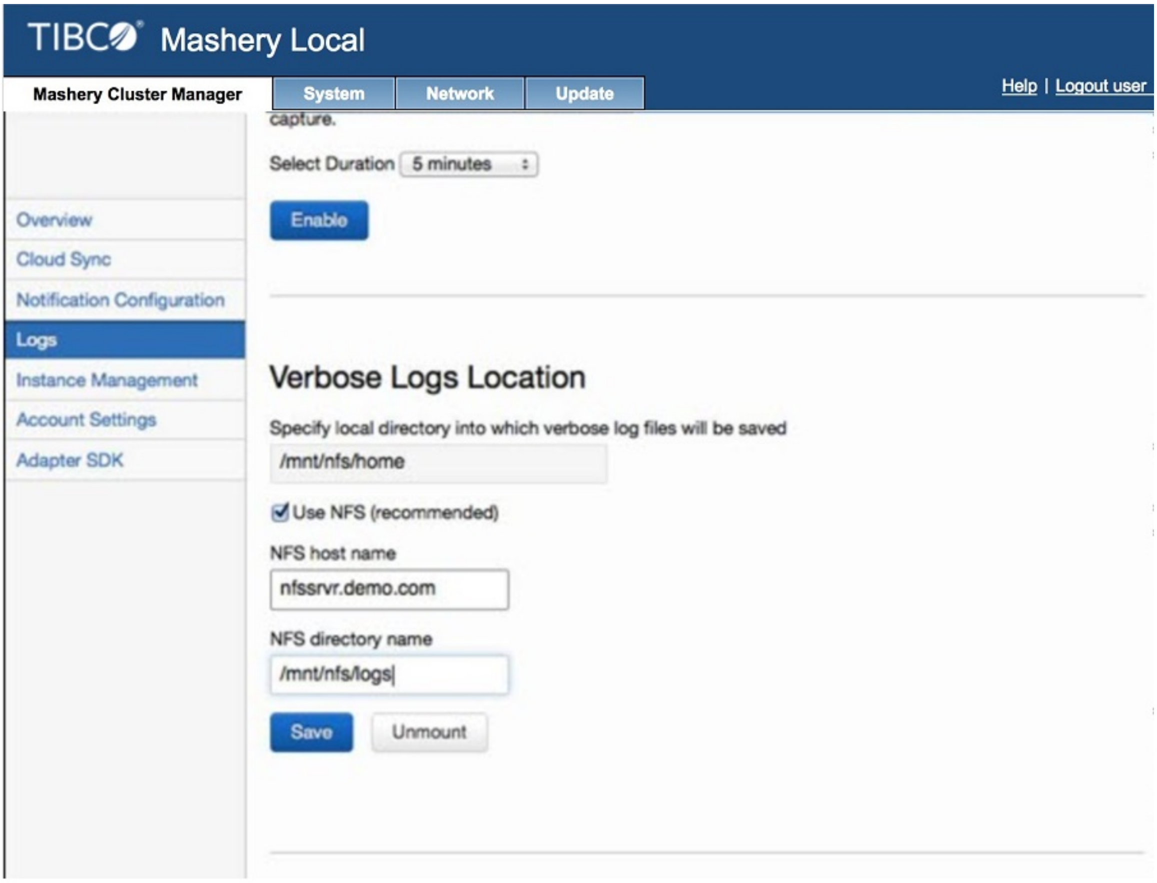
Procedure

1. Specify the Verbose Logs location.
    a) Select **Use NFS**.

       TIBCO Mashery highly recommends using NFS location for verbose logs so that local system usage is not impacted. In addition, all the nodes in a cluster (the Master and Slaves) can write to the same centralized location for easier, further analysis.

        1. Enter the **NFS host name**.
        2. Enter the **NFS directory name**.
        3. Click **Save**.

       The specified NFS directory is mounted onto `/mnt/nfs/home` local directory location.



capture.

Select Duration 5 minutes

Verbose Logs Location

Specify local directory into which verbose log files will be saved

/mnt/nfs/home

☑ Use NFS (recommended)

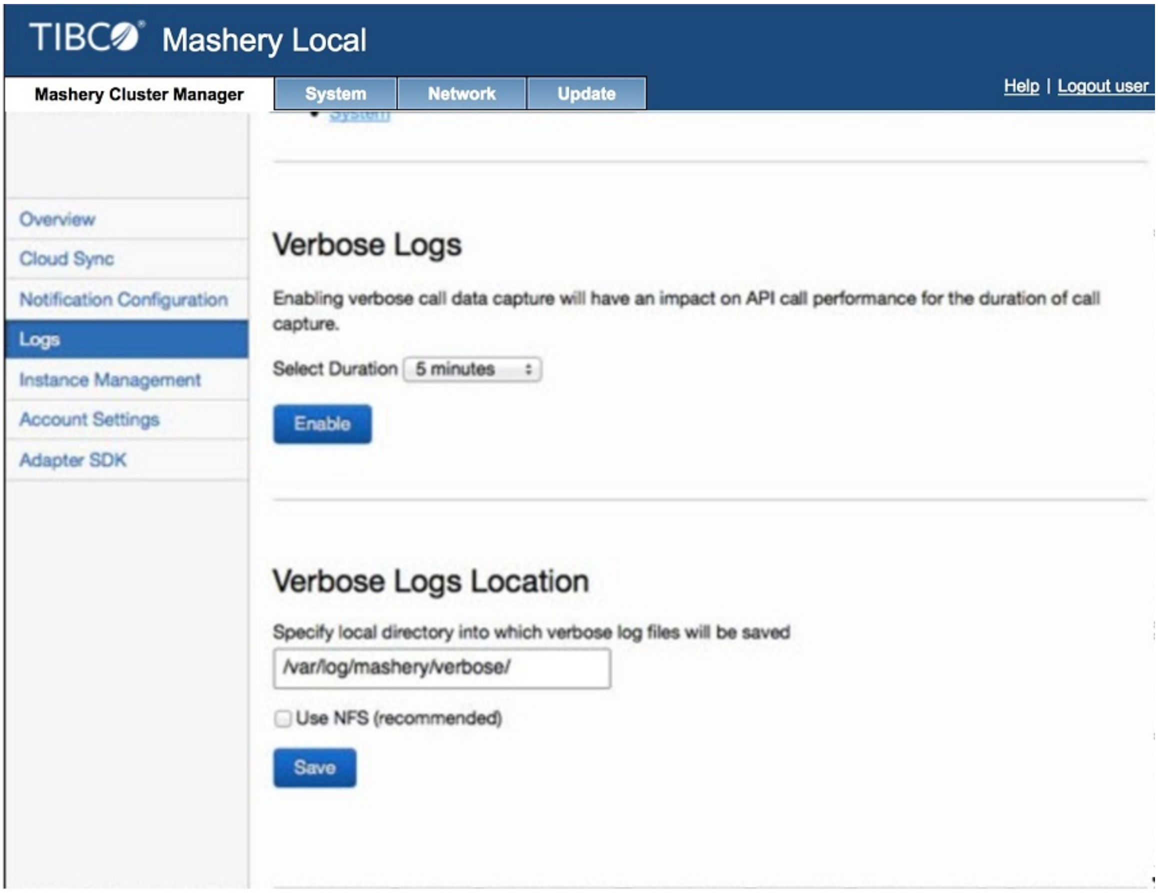
NFS host name

nfssrvr.demo.com

NFS directory name

/mnt/nfs/logs

b) If you do not select **Use NFS** (not recommended), verbose logs will be saved to the local directory. The default directory `</var/log/mashery/verbose>` can be changed.



Verbose Logs

Enabling verbose call data capture will have an impact on API call performance for the duration of call capture.

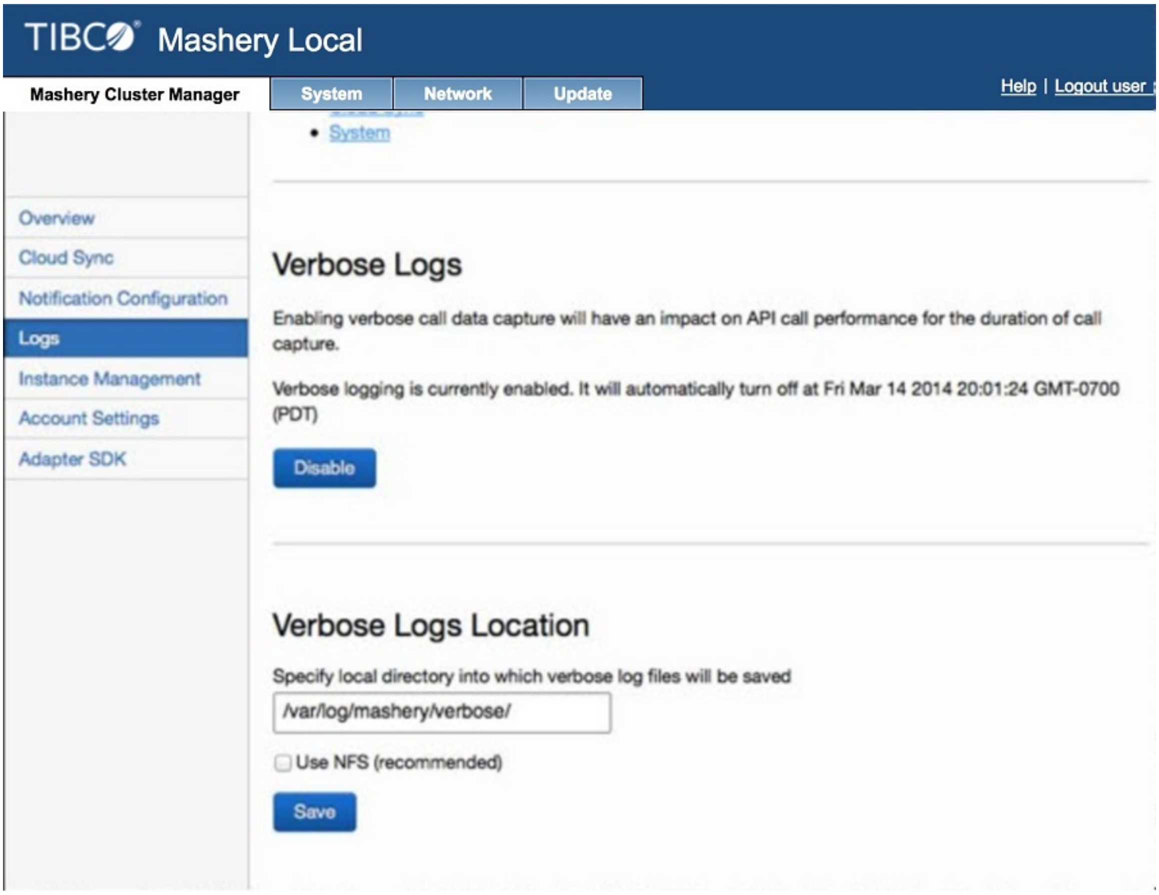
Select Duration: 5 minutes

Verbose Logs Location

Specify local directory into which verbose log files will be saved: /var/log/mashery/verbose/

☐ Use NFS (recommended)

2. Enable Verbose Logs.
   a) Select duration for capturing the logs (05, 10, 15, or 30 minutes).
   b) Click **Enable**.



Verbose Logs

Enabling verbose call data capture will have an impact on API call performance for the duration of call capture.

Verbose logging is currently enabled. It will automatically turn off at Fri Mar 14 2014 20:01:24 GMT-0700 (PDT)

Verbose Logs Location

Specify local directory into which verbose log files will be saved

/var/log/mashery/verbose/

After you enable verbose logs, Mashery Local writes the call data logs that include inbound request data, inbound processed data, outbound response data, and outbound processed data. Verbose logs (call data capturing) is disabled after the selected time duration expires.

You must set the Verbose Logs Location on each node in the cluster including Master and all Slaves. Enabling or disabling verbose logs can only occur on the Master node. The Slave nodes just inherit the current verbose log enablement status from the Master.

Working with Verbose Logs

A directory is created every minute with the name format as YYYY-MM-DD-HH-MM. All the calls that are logged in a minute become part of one directory and so on. For each call, a sub-directory is created using the name <timestamp>-<Mashery Message ID>.

Mashery Message ID is a globally unique ID generated for every API call that is processed by TIBCO Mashery. The Mashery Message ID provides a possible mechanism for administrators to create a golden thread for debugging issues between TIBCO Mashery, your partners and your backend system. To be able to include this GUID in request and response headers, you can toggle on **Include X-Mashery-Message-ID in Request** and **Include X-Mashery-Message-ID in Response** properties on in the **Services>Endpoint>Properties** page in the TIBCO Mashery Administration Dashboard.

Within each sub-directory, four log files are `InboundRequest.log`, `InboundProcessed.log`, `OutboundResponse.log`, `OutboundProcessed.log`.

- `InboundRequest` contains the request data on the API call as it is originally received by Mashery Local from the client application.
- `InboundProcessed` contains the Mashery processed version of the inbound request as sent to API server (or to cache if enabled).

- `OutboundResponse` contains the response data as it is originally received by Mashery from the API server (or from cache if enabled).
- `OutboundProcessed` contains the Mashery processed version of the outbound response as sent to the client application.

Each of the four files contain some important metadata written as key-value pairs with 1 pair on one line. After the metadata, a new delimiter line is written followed by the actual message.

The metadata included are:

- Key: Key a developer uses to get access to a Package Plan or API.
- Service ID: TIBCO Mashery generated unique ID to identify an API.
- Endpoint ID: TIBCO Mashery generated unique ID to identify an Endpoint.
- Site ID: TIBCO Mashery generated unique ID to identify your Site within the TIBCO Mashery Network.
- IP address: IP address of the client application invoking the API call.
- Method (if available): Method that was being accessed in the API call (available if appropriate **Method Configuration** settings are specified in **Services**>**End-points**>**Properties** tab in the TIBCO Mashery Administration dashboard).
- Cache hit: 1 if cache is enabled and response is met from cache, 0 otherwise.
- Error message (if any): TIBCO Mashery generated error message on that API call (if any).

Mapping Endpoint IDs

TIBCO Mashery Local provides a script that allows fetching a list of endpoints with details such as the Endpoint ID and the Endpoint name. The Endpoints associated with a service are displayed. The Service ID is the parameter used to fetch the Endpoint details.

```
//Request searching with a particular service id
python getEndpointNames.py --service 95bpf2jv3f8p5x3xqsu657x5

//Response in json formatter
{
   "services":[
      {
         "endpoints":[
            {
               "id":"7xwgjatahmuwgrz79cgw286a",
               "name":"CORS-disabled"
            },
            {
               "id":"2m4zz8nw4n9w36uau7j2bnqb",
               "name":"Custom CORS(custom rest as the API key source)"
            },
            {
               "id":"g2qx6vhxubu4d4w66egqnxsh",
               "name":"CORS-enabled- EIN-112-dontallow"
            },
            {
               "id":"uavv2nm6yy7j94nhp8zp5kjf",
               "name":"CORS-enabled-EIN-112"
            },
            {
               "id":"pgbrzzu89dtyvumqht4ncnt4",
               "name":"preflight-requestmultipledomainno"
            },
            {
               "id":"7qcpe6dsss4kxp4u8c6fy5pr",
               "name":"EIN-222"
            }
         ],
         "id":"95bpf2jv3f8p5x3xqsu657x5"
      }
   ]
}
```

Debugging Utility

A debug utility is provided that can be used to capture information about system health and configuration, connectivity to the cloud for synchronization, fix Slave registration issues, and resolve any replication issues between Master and Slave. A command line console is available to run the various options available. This utility is useful for gathering information to assist with trouble-shooting common system configuration errors with TIBCO Mashery Local.

Running the Debug Utility

Execute the following command to run the debug utility:

```
$ python /opt/mashery/utilities/debug_util.py
```

The following options are available. Some options are available to be run only on Master and some only on Slave.

```
Select from the following:
1: Collect Logs
2: Test connectivity to Cloud Sync
3: Show Slave Status
4: Check IP address
5: Update record or Master IP address in Master (Master IP address has changed and
registration of new Slave with cluster fails)
6: Fix slave corruption (Restart slave at last valid read position)
7: Update record of Master IP address in old Slave node (Master IP address has
changed and cluster is not updated)
8: System manager (Remove non-functional or unused slaves form Master)
9: Collect system state (Disk health, process health, time setting, network
settings)
menu: Show this menu
exit: Quit
```

For option **9: Collect system state**, the resulting files for this option are created in the home directory, depending on the login users (root/administrator).

Collect Logs

This tool produces a `tar.gz` file that collects Traffic Manager component logs, sync status with the cloud, the Slave and Master IP address checks, logs required to check replication issues between Master and Slave and verbose logs for the day (if any).

This option can be run on Master and Slave nodes.

Test Connectivity to Cloud Sync

This tool helps to determine if there are any errors connecting to the TIBCO Mashery Cloud system for synchronization.

This option can be run on Master and Slaves.

Show Slave Status

This option displays whether a Slave is functioning correctly, including its status, the Master systems IP address and any replication errors that are present between Master and Slave.

This option can be run on a Slave node.

Check IP Address

This option allows you to check the current IP address of the Master.

This option can be run on a Master node.

Update Record of Master IP Address in Master

Sometimes if the IP address of a Master node changes, new Slave registration with the Master fails. Running this option fixes the record of the Master IP address in the Master node for successful Slave registration.

This option can be run on a Master node.

Fix Slave Corruption

This option allows you to resolve Master Slave replication issues.

This option can be run on a Slave node.

Update Record of Master IP Address in Old Slave Node

This option updates the record of the Master IP address in the Slave nodes and is useful for resolving Master-Slave replication issues.

System Manager (Remove Non-functional or Unused Slaves from Master)

Sometimes Slave nodes are decommissioned and new Slave nodes are created. This option on Master system can be used to remove unused slaves.

System Level Troubleshooting

TIBCO Mashery Local administrators have the ability to run the following select commands to investigate and troubleshoot the network or system level issues.

- ping
- ping6
- tracepath
- tracepath6
- tcpdump
- traceroute
- arping
- tshark
- route
- ifconfig
- iptables
- dhclient

sudo edit for the following files:

- /etc/resolv.conf
- /etc/sysconfig/network-scripts/ifcfg-eth0
- /etc/sysconfig/network-scripts/ifcfg-eth1
- /etc/rc.local
- /etc/hosts

- /etc/securitylimits.conf
- /etc/nssswitch.conf

General Troubleshooting

The following table provides information for troubleshooting general Mashery Local (for Appliance and Docker) issues.

Form Factor	Issue	Notes
All	API call returns a 596 error	**Possible Cause** API is configured with specific supported HTTP Methods, and the HTTP Method used for this call is not allowed. **Diagnostic Steps** 1. Test the API call using the SaaS domain (`<customer>`.api.mashery.com) 2. If the call returns a 596 error, review the **Key & Method Detection** settings for this endpoint and confirm that the HTTP Method used in the API call is allowed. **Resolution** 1. If the HTTP method used in this call is not configured on the endpoint, update the supported HTTP Methods to include the HTTP method. 2. Run a manual Mashery Local sync to update the configuration in the on-prem traffic manager.

Form Factor	Issue	Notes

Appliance	API call returns a 596 error	**Possible Cause**
		Memcached is not running
		Diagnostic Steps
		Check that the API configuration is loaded into memcache:
		1. SSH into the Mashery Local Instance, for example: `ssh root@<IP ADDRESS OF THE INSTANCE>`
		2. TELNET to the Memcache port: **telnet localhost <port>**. Here are the port numbers for the various memcache pools:
		1. "memservicePool": 11214
		2. "memcachePool": 11211
		3. "memcachePackaged": 11215
		4. "contentCachePool": 11213
		5. "memcountPool": 11212
		3. Run the **stats items** command.
		4. Identify the item number with more than 1 record.
		5. Run the command:
		`stats cachedump <ITEM NUMBER> <NUMBER OF RECORDS>`
		Resolution
		If the response is coming from the master and the settings are not in memcache, you likely have a synchronization issue.
		If the response is coming from the slave and the settings are not in memcache, you like have a replication issue. Force a memcache load of the service definitions:
		`/opt/javaproxy/proxy/memcacheloader --env production --verbose --service`

Form Factor	Issue	Notes

Docker	API call returns a 596 Error	**Possible Cause**
		Memcached container is not running.
		Diagnostic Steps
		1. Check the proxy.log for Memcached errors.
		2. Check whether the memcached is running:
		`docker exec -it ml-mem ps -ef`
		and look for the memcached process.
		Resolution
		1. If memcache is not running, ssh into the ml-mem container to start it and see whether there's any error:
		`docker exec -it ml-mem /bin/bash`
		then:
		`service memcached start`
		2. If it's caused by running out of file limit, increase the **ulimit** setting by editing the docker-compose.yml file, and if you are using docker-compose, do a docker-compose down followed by a docker-compose up to restart the containers.
		Please see https://docs.docker.com/compose/compose-file/#/ulimits.
		Add those to the docker-compose.yml file under the container section, most likelyml_memwould need this(if memcached failed to start). For example:

```
ulimits:
  nproc: 65535
  nofile:
    soft: 20000
    hard: 40000
```

Watch out for the leading spaces. They must align with others using the correct indentation.

Form Factor	Issue	Notes
All	API call returns intermittent 596 error on a previously working slave.	**Possible Cause** Sync between master and one or more slaves is not working. **Diagnostic Steps** Errors are intermittent indicating that there is a problem with one slave. Use the following command: `python /opt/mashery/utilities/debug_util.py` Select Option 3 (Show Slave Status). This option displays whether a Slave is functioning correctly, including its status, the Master systems IP address and any replication errors that are present between Master and Slave. **Resolution** If errors are present, recreate the Slave instance.

All	API call returns 596 error on a new slave.	**Possible Cause**
		Sync between master and slave is not working.
		Diagnostic Steps
		When connecting a new slave to a Master, the customer sees this error:
		`Registering as Slave ERROR: Failed to configure the node as slave.`
		Resolution
		This can happen if the IP Address of the Master was changed after the initial installation of the Master. The built in Debug Utility (debug_util.py) should be run on the Master in order to fix this.
		Have the customer run the debug_util.py on the "Master", using the following command:
		`python /opt/mashery/utilities/debug_util.py`
		Select Option 5. (Update record or Master IP address in Master. (Master IP address has changed and registration of new Slave with cluster fails)).
		The customer should then be able to register the new Slave to the Master node.

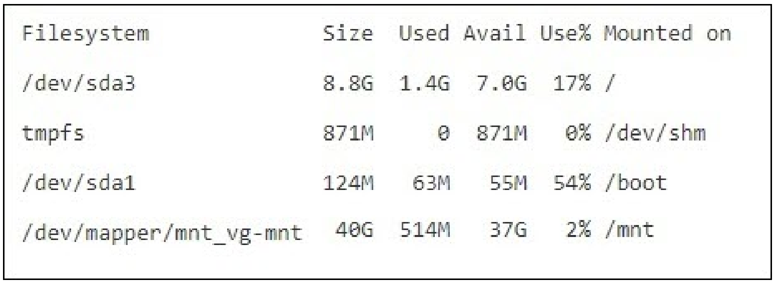
<table>
<tr><th>Form Factor</th><th>Issue</th><th>Notes</th></tr>
<tr><td>All</td><td>Mashery Local Web Console is blank.</td><td>

Possible Cause

Disk is full.

Diagnostic Steps

Review disk space using the "df -h" command. This will give you a percentage usage of both disks (there are usually 2 disks, 1 "system" and the other "mnt" (mnt contains the logs and the mysql database, the rest is on system)



```
Filesystem               Size  Used Avail Use% Mounted on
/dev/sda3                8.8G  1.4G  7.0G  17% /
tmpfs                    871M     0  871M   0% /dev/shm
/dev/sda1                124M   63M   55M  54% /boot
/dev/mapper/mnt_vg-mnt    40G  514M   37G   2% /mnt
```

Resolution

If disk space utilization is over 90% for either disk, customer should ask their System Administrator to increase the size of the respective disk.

</td></tr>
<tr><td>All</td><td>Mashery Local Web Console is blank.</td><td>

Possible Cause

Available memory is low.

Diagnostic Steps

Review free memory using the "free -h" command.

Resolution

If available memory is low or the system is using swap, customer should ask their System Administrator to increase the size of memory on this instance or add more nodes to the cluster so that this instance is not at capacity.

</td></tr>
</table>

Form Factor	Issue	Notes
Appliance	Mashery Local Web Console is blank.	**Possible Cause** Basic processes are not running. **Diagnostic Steps** Review basic processes using the "ps aux | more" command. Check for: • memcached • javaproxy • mysqld • vami-sfcbd • lighttpd **Resolution** If any of these processes are not running, reboot Mashery Local instance.

All	Cannot synchronize API Settings.	**Possible Cause**
		Connection to Mashery On-Prem Manager (MOM) is not present.
		Diagnostic Steps
		Run the following command:
		`dig api-mom.mashery.com`
		If you get a response, then try:
		`curl -k https://api-mom.mashery.com/ping`
		Resolution
		If you get a response, then you do have a good connection to MOM.
		If you do not get a response, check your network configuration to ensure outbound HTTPS / 443 access is allowed.

Form Factor	Issue	Notes
All	Mashery Local returns a 503 Service Unavailable error.	**Possible Cause** Failsafe is being triggered for the endpoint in question. **Diagnostic Steps** Confirm that the error message of `503_Service_Unavailable_Proxy` is being returned. **Resolution** This means Mashery's failsafe has been triggered due to excessive 504 responses from the API over a short period of time. It could be that the customer's origin servers are now taking longer than the configured connection or response TTLs set on the endpoint. If those values are low, then the customer should increase the values. If they are already high, then the customer needs to improve performance on their origin server to alleviate the issue.

Docker	Docker Instance cannot be reached.	**Possible Cause**
		Docker containers need to be returned to a clean state.
		Diagnostic Steps
		Error checking TLS connection: Something went wrong running an SSH command!
		error getting ip address: host is not running
		Docker-Machine instances in Timeout state
		Resolution
		If you are connected to the VPN, disconnect VPN
		• Stop All containers
		`docker stop $(docker ps -a -q)`
		• Delete all containers
		`docker rm $(docker ps -a -q)`
		• Delete all images
		`docker rmi $(docker images -q)`
		• If using Virtualbox, remove host adapter -
		Open Virtualbox, click `File -> Preferences -> Network -> Host-only Network`, remove Vboxnet#
		• Unsetting DOCKER variables
		`unset ${!DOCKER*}`
		Restart Docker Terminal and start creating new instance.

Appendix

This appendix describes some configurations for using Mashery Local features.

Adapter SDK Usage and Examples

The following sections describe a typical setup of a development environment for the Adapter SDK, as well as examples of Adapter SDK usage in various projects.

Adapter SDK Development Environment Example Setup

The following is an example setup used for the Adapter SDK development environment. The environment details are listed below for reference.

- **Operating System**: CentOS Linux release 7.3.1611
- **JDK**: OpenJDK version "1.8.0_121"
- **Eclipse**: Eclipse Java EE IDE for Web Developers
  - **Version**: Mars.2 Release (4.5.2)
  - **Build id**: 20160218-0600
- **Ant**: Apache Ant™ version 1.9.2 compiled on June 10 2014

  ```
  yum install ant
  ```

- **Maven**: Apache Maven 3.0.5 (Red Hat 3.0.5-17)

  ```
  yum install maven
  ```

Setting up the Adapter SDK for Maven

To set up the Adapter SDK for Maven, follow the steps below:

Procedure

1. Download the Adapter SDK from Mashery Local Cluster Manager.
2. Copy the Adapter SDK to your development environment, for example, `/home/beta/Documents/sdk.zip`.
3. Unzip the Adapter SDK:

   ```
   unzip sdk.zip -d sdk
   ```

4. Install Adapter SDK JARs in Maven local repository:

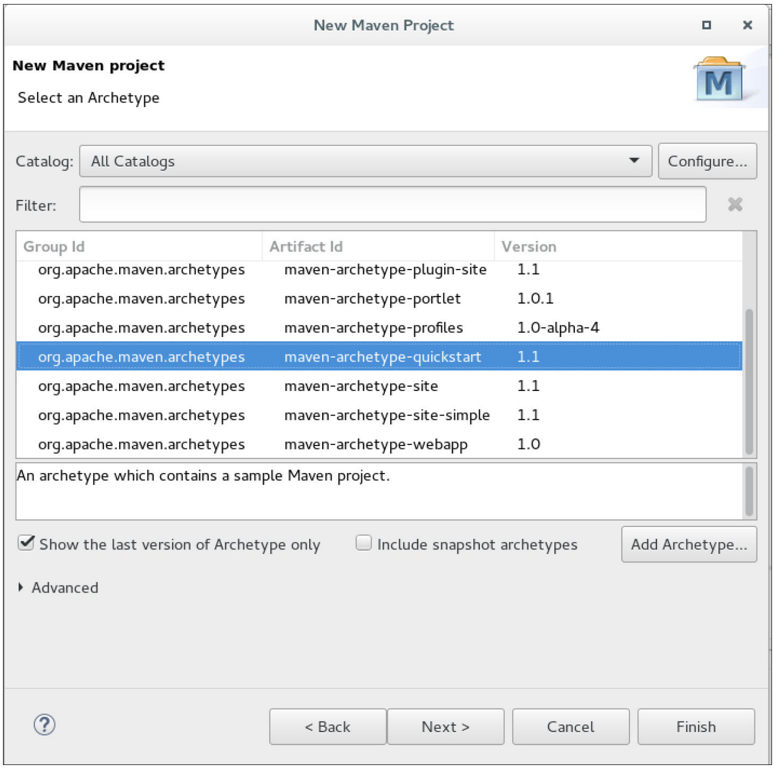
   ```
   cd /home/beta/Documents/sdk
   mvn install:install-file -Dfile=com.mashery.http_1.0.0.v20130130-0044.jar -
   DgroupId=com.mashery \
   -DartifactId=http -Dversion=1.0.0.v20130130-0044 -Dpackaging=jar -
   DgeneratePom=true
   mvn install:install-file -
   Dfile=com.mashery.trafficmanager.sdk_1.1.0.v20130214-0043.jar -
   DgroupId=com.mashery \
   -DartifactId=trafficmanager -Dversion=1.1.0.v20130214-0043 -Dpackaging=jar -
   DgeneratePom=true
   mvn install:install-file -Dfile=com.mashery.util_1.0.0.v20130214-0015.jar -
   DgroupId=com.mashery \
   -DartifactId=util -Dversion=1.0.0.v20130214-0015 -Dpackaging=jar -
   DgeneratePom=true
   ```

Using the Adapter SDK in Mashery Local with Single Processor

To use the Adapter SDK in Mashery Local with a single processor, follow the steps below:

Procedure

1. Create a new Maven Project in Eclipse. Go to `File > New > Maven Project`.

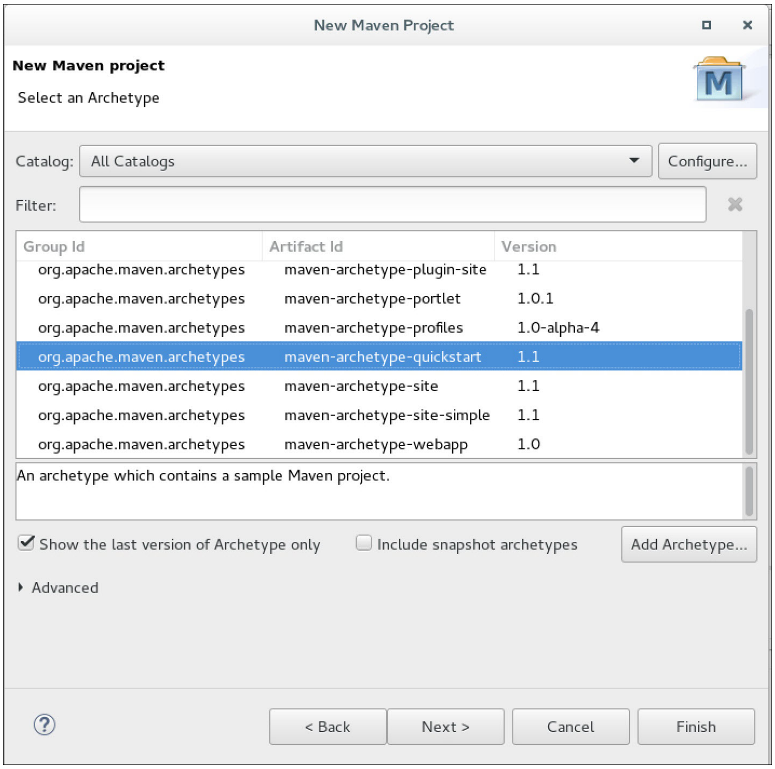


Catalog: All Catalogs

Group Id	Artifact Id	Version
org.apache.maven.archetypes	maven-archetype-plugin-site	1.1
org.apache.maven.archetypes	maven-archetype-portlet	1.0.1
org.apache.maven.archetypes	maven-archetype-profiles	1.0-alpha-4
org.apache.maven.archetypes	maven-archetype-quickstart	1.1
org.apache.maven.archetypes	maven-archetype-site	1.1
org.apache.maven.archetypes	maven-archetype-site-simple	1.1
org.apache.maven.archetypes	maven-archetype-webapp	1.0

An archetype which contains a sample Maven project.

☑ Show the last version of Archetype only ☐ Include snapshot archetypes Add Archetype...



Catalog: All Catalogs

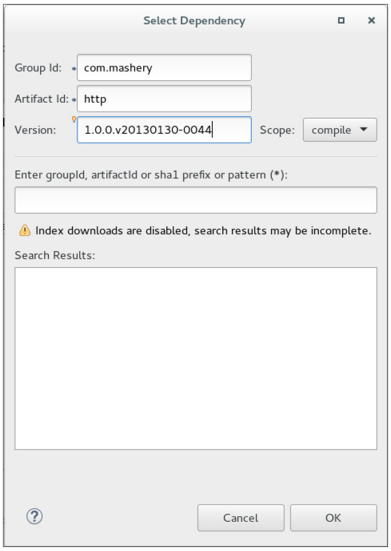
Filter:

Group Id	Artifact Id	Version
org.apache.maven.archetypes	maven-archetype-plugin-site	1.1
org.apache.maven.archetypes	maven-archetype-portlet	1.0.1
org.apache.maven.archetypes	maven-archetype-profiles	1.0-alpha-4
org.apache.maven.archetypes	maven-archetype-quickstart	1.1
org.apache.maven.archetypes	maven-archetype-site	1.1
org.apache.maven.archetypes	maven-archetype-site-simple	1.1
org.apache.maven.archetypes	maven-archetype-webapp	1.0

An archetype which contains a sample Maven project.

☑ Show the last version of Archetype only ☐ Include snapshot archetypes

2. Add dependencies to the project. In Eclipse, go to `Project Explorer > pom.xml > Dependencies`.



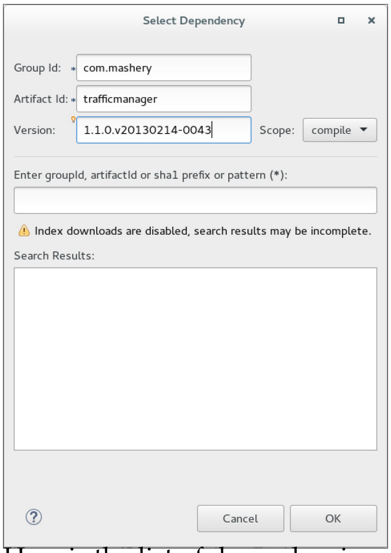
Group Id: * com.mashery

Artifact Id: * http

Version: 1.0.0.v20130130-0044 Scope: compile

Enter groupId, artifactId or sha1 prefix or pattern (*):

Index downloads are disabled, search results may be incomplete.



Group Id: com.mashery

Artifact Id: trafficmanager

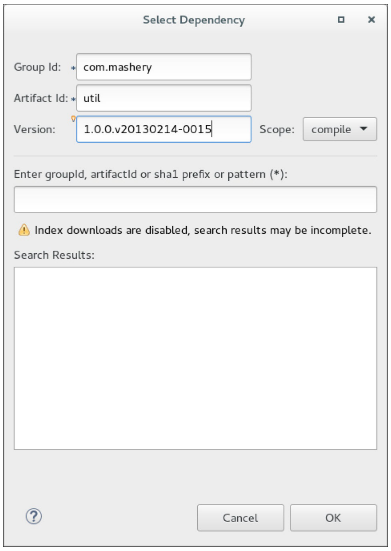
Version: 1.1.0.v20130214-0043

Scope: compile

Enter groupId, artifactId or sha1 prefix or pattern (*):

Index downloads are disabled, search results may be incomplete.

Search Results:



Group Id: com.mashery

Artifact Id: util

Version: 1.0.0.v20130214-0015

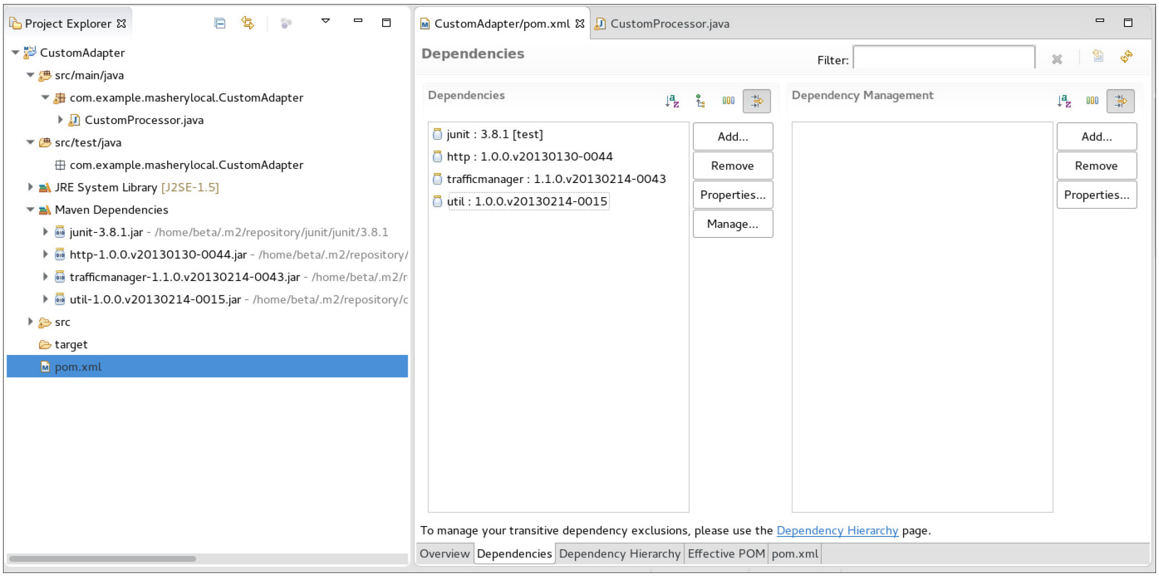
Scope: compile

Enter groupId, artifactId or sha1 prefix or pattern (*):

Index downloads are disabled, search results may be incomplete.

Search Results:

Here is the list of dependencies:



- CustomAdapter
  - src/main/java
    - com.example.masherylocal.CustomAdapter
      - CustomProcessor.java
  - src/test/java
    - com.example.masherylocal.CustomAdapter
  - JRE System Library [J2SE-1.5]
  - Maven Dependencies
    - junit-3.8.1.jar - /home/beta/.m2/repository/junit/junit/3.8.1
    - http-1.0.0.v20130130-0044.jar - /home/beta/.m2/repository/
    - trafficmanager-1.1.0.v20130214-0043.jar - /home/beta/.m2/r
    - util-1.0.0.v20130214-0015.jar - /home/beta/.m2/repository/c
  - src
  - target
  - pom.xml

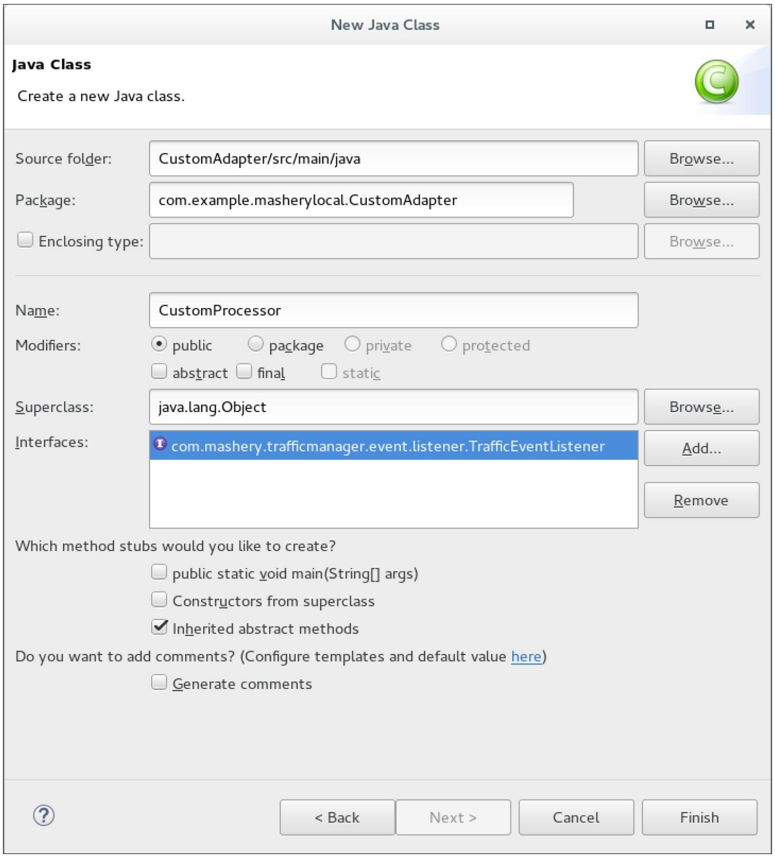
Dependencies:

- junit : 3.8.1 [test]
- http : 1.0.0.v20130130-0044
- trafficmanager : 1.1.0.v20130214-0043
- util : 1.0.0.v20130214-0015

To manage your transitive dependency exclusions, please use the Dependency Hierarchy page.

3. Create new Java Class. In Eclipse, go to `Project > New > Class`.

   In the package `com.example.masherylocal.CustomAdapter`, create the following class **`CustomProcessor`**.



New Java Class

Java Class

Create a new Java class.

Source folder: CustomAdapter/src/main/java

Package: com.example.masherylocal.CustomAdapter

☐ Enclosing type:

Name: CustomProcessor

Modifiers: ● public ○ package ○ private ○ protected

☐ abstract ☐ final ☐ static

Superclass: java.lang.Object

Interfaces: com.mashery.trafficmanager.event.listener.TrafficEventListener

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

☐ Generate comments

This is the reference code:

```
package com.example.masherylocal.CustomAdapter;

import com.mashery.http.server.HTTPServerRequest;
import com.mashery.http.server.HTTPServerResponse;
import com.mashery.trafficmanager.event.listener.TrafficEventListener;
import com.mashery.trafficmanager.event.model.TrafficEvent;
import com.mashery.trafficmanager.event.model.TrafficEventCategory;
import com.mashery.trafficmanager.event.model.TrafficEventType;
import com.mashery.trafficmanager.event.processor.model.ProcessorEvent;
import com.mashery.trafficmanager.model.core.APICall;
import com.mashery.trafficmanager.model.core.ApplicationRequest;
import com.mashery.trafficmanager.model.core.TrafficManagerResponse;
import com.mashery.trafficmanager.processor.ProcessorBean;

@ProcessorBean(enabled=true,
name="com.example.masherylocal.CustomAdapter.CustomProcessor", immediate=true)
public class CustomProcessor implements TrafficEventListener {

    public void handleEvent(TrafficEvent event) {
        TrafficEventType eventType = event.getType();
        if (eventType.getCategory() != TrafficEventCategory.PROCESSOR)
            return;
        ProcessorEvent processorEvent = (ProcessorEvent) event;
        APICall apiCall = processorEvent.getCallContext();

        if (eventType.getName().contentEquals("Pre-Process Event")) {
            ApplicationRequest appRequest = apiCall.getRequest();
            HTTPServerRequest httpRequest = appRequest.getHTTPRequest();
        } else if (eventType.getName().contentEquals("Post-Process Event")) {
            TrafficManagerResponse tmResp = apiCall.getResponse();
            HTTPServerResponse httpResp = tmResp.getHTTPResponse();

httpResp.getHeaders().add("CustomAdapter.CustomProcessor::handleEvent", "Post-
Process Event");
        }
    }
}
```

4. Build the project in the folder, for example, `/home/beta/work_diysdk/CustomAdapter/`:

```
mvn package
```

5. Use a simple zip command to package the project in the folder, for example, `/home/beta/work_diysdk/CustomAdapter/target/`:

```
zip CustomAdapter-1.0.zip CustomAdapter-1.0.jar
```

6. Unload the zipped package using Mashery Local Cluster Manager.
7. Apply the Custom Processor on the Endpoint using Mashery SaaS Control Center.

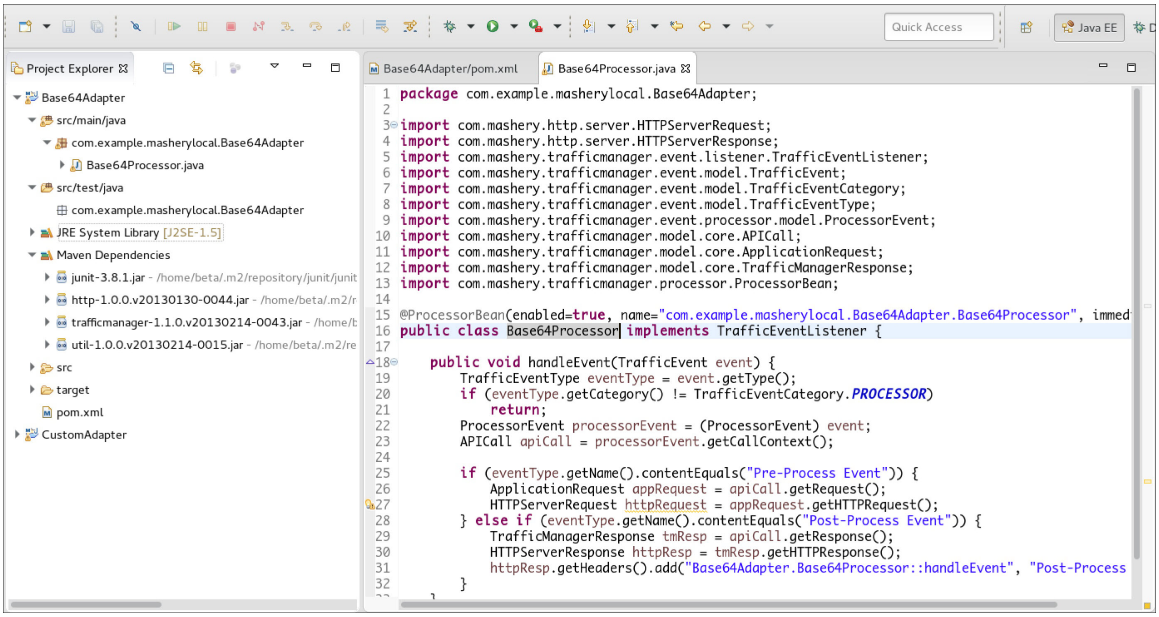
Using the Adapter SDK in Mashery Local with Third-Party Libraries

To use the Adapter SDK in Mashery Local with third-party libraries, follow the steps below:

Procedure

1. Create a new Maven Project "Base64Adapter" in Eclipse, then add dependencies and class, as described in Using the Adapter SDK in Mashery Local with a Single Processor.

   The project layout should look like the following:

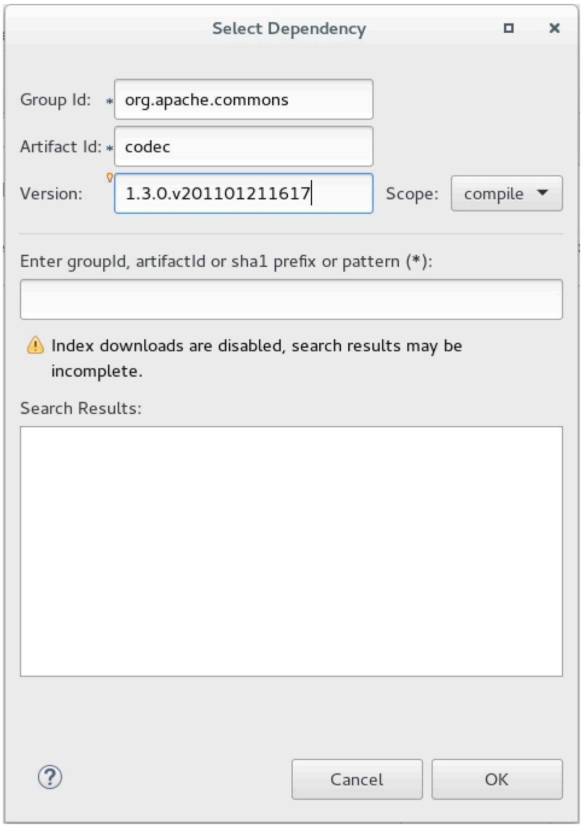

Quick Access
Java EE
Project Explorer
Base64Adapter
src/main/java
com.example.masherylocal.Base64Adapter
Base64Processor.java
src/test/java
com.example.masherylocal.Base64Adapter
JRE System Library [J2SE-1.5]
Maven Dependencies
junit-3.8.1.jar - /home/beta/.m2/repository/junit/junit
http-1.0.0.v20130130-0044.jar - /home/beta/.m2/r
trafficmanager-1.1.0.v20130214-0043.jar - /home/b
util-1.0.0.v20130214-0015.jar - /home/beta/.m2/re
src
target
pom.xml
CustomAdapter
Base64Adapter/pom.xml
Base64Processor.java

```
package com.example.masherylocal.Base64Adapter;

import com.mashery.http.server.HTTPServerRequest;
import com.mashery.http.server.HTTPServerResponse;
import com.mashery.trafficmanager.event.listener.TrafficEventListener;
import com.mashery.trafficmanager.event.model.TrafficEvent;
import com.mashery.trafficmanager.event.model.TrafficEventCategory;
import com.mashery.trafficmanager.event.model.TrafficEventType;
import com.mashery.trafficmanager.event.processor.model.ProcessorEvent;
import com.mashery.trafficmanager.model.core.APICall;
import com.mashery.trafficmanager.model.core.ApplicationRequest;
import com.mashery.trafficmanager.model.core.TrafficManagerResponse;
import com.mashery.trafficmanager.processor.ProcessorBean;

@ProcessorBean(enabled=true, name="com.example.masherylocal.Base64Adapter.Base64Processor", immed
public class Base64Processor implements TrafficEventListener {

    public void handleEvent(TrafficEvent event) {
        TrafficEventType eventType = event.getType();
        if (eventType.getCategory() != TrafficEventCategory.PROCESSOR)
            return;
        ProcessorEvent processorEvent = (ProcessorEvent) event;
        APICall apiCall = processorEvent.getCallContext();

        if (eventType.getName().contentEquals("Pre-Process Event")) {
            ApplicationRequest appRequest = apiCall.getRequest();
            HTTPServerRequest httpRequest = appRequest.getHTTPRequest();
        } else if (eventType.getName().contentEquals("Post-Process Event")) {
            TrafficManagerResponse tmResp = apiCall.getResponse();
            HTTPServerResponse httpResp = tmResp.getHTTPResponse();
            httpResp.getHeaders().add("Base64Adapter.Base64Processor::handleEvent", "Post-Process
        }
```

2. Install the third-party library in Maven local repository, for example:

```
mvn install:install-file -Dfile=org.apache.commons.codec_1.3.0.v201101211617.jar
-DgroupId=org.apache.commons \
    -DartifactId=codec -Dversion=1.3.0.v201101211617 -Dpackaging=jar -
DgeneratePom=true
```

3. Add dependency on third-party library in the project:



Group Id: * org.apache.commons

Artifact Id: * codec

Version: 1.3.0.v201101211617

Scope: compile

Enter groupId, artifactId or sha1 prefix or pattern (*):

Index downloads are disabled, search results may be incomplete.

Search Results:

4. Use third-party library in the project, such as this example, which uses `org.apache.commons.codec_1.3.0.v201101211617.jar`:

```
package com.example.masherylocal.Base64Adapter;

import java.io.UnsupportedEncodingException;

import org.apache.commons.codec.binary.Base64;

import com.mashery.http.server.HTTPServerRequest;
import com.mashery.http.server.HTTPServerResponse;
import com.mashery.trafficmanager.event.listener.TrafficEventListener;
import com.mashery.trafficmanager.event.model.TrafficEvent;
import com.mashery.trafficmanager.event.model.TrafficEventCategory;
import com.mashery.trafficmanager.event.model.TrafficEventType;
import com.mashery.trafficmanager.event.processor.model.ProcessorEvent;
import com.mashery.trafficmanager.model.core.APICall;
import com.mashery.trafficmanager.model.core.ApplicationRequest;
import com.mashery.trafficmanager.model.core.TrafficManagerResponse;
import com.mashery.trafficmanager.processor.ProcessorBean;

@ProcessorBean(enabled=true,
name="com.example.masherylocal.Base64Adapter.Base64Processor", immediate=true)
public class Base64Processor implements TrafficEventListener {

    public void handleEvent(TrafficEvent event) {
        TrafficEventType eventType = event.getType();
        if (eventType.getCategory() != TrafficEventCategory.PROCESSOR)
            return;
        ProcessorEvent processorEvent = (ProcessorEvent) event;
        APICall apiCall = processorEvent.getCallContext();

        if (eventType.getName().contentEquals("Pre-Process Event")) {
            ApplicationRequest appRequest = apiCall.getRequest();
            HTTPServerRequest httpRequest = appRequest.getHTTPRequest();
        } else if (eventType.getName().contentEquals("Post-Process Event")) {
            TrafficManagerResponse tmResp = apiCall.getResponse();
            HTTPServerResponse httpResp = tmResp.getHTTPResponse();
            try {
                byte[] base64bytes = Base64.encodeBase64("Post-Process
Event".getBytes("UTF-8"));
                String base64str = new String(base64bytes, "UTF-8");

httpResp.getHeaders().add("Base64Adapter.Base64Processor_handleEvent",
base64str);
            } catch (UnsupportedEncodingException e) {
                // TODO Add logging
            }
        }
    }
}
```

5. Update build script to include Class-Path in MANIFEST.MF, as in this example Maven Project, this is done by adding `org.apache.maven.plugins.maven-jar-plugin` to pom.xml:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/
2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.example.masherylocal</groupId>
  <artifactId>Base64Adapter</artifactId>
  <version>1.0</version>
  <packaging>jar</packaging>

  <name>Base64Adapter</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-jar-plugin</artifactId>
        <version>3.0.2</version>
        <configuration>
          <archive>
            <index>true</index>
            <manifest>
              <addClasspath>true</addClasspath>
            </manifest>
          </archive>
        </configuration>
      </plugin>
    </plugins>
  </build>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>com.mashery</groupId>
        <artifactId>http</artifactId>
        <version>1.0.0.v20130130-0044</version>
    </dependency>
    <dependency>
        <groupId>com.mashery</groupId>
        <artifactId>trafficmanager</artifactId>
        <version>1.1.0.v20130214-0043</version>
    </dependency>
    <dependency>
```

```
        <groupId>com.mashery</groupId>
        <artifactId>util</artifactId>
        <version>1.0.0.v20130214-0015</version>
    </dependency>
    <dependency>
        <groupId>org.apache.commons</groupId>
        <artifactId>codec</artifactId>
        <version>1.3.0.v201101211617</version>
    </dependency>
  </dependencies>
</project>
```

Here is an example of MANIFEST.MF:

```
Manifest-Version: 1.0
Built-By: beta
Class-Path: http-1.0.0.v20130130-0044.jar trafficmanager-1.1.0.v201302
 14-0043.jar util-1.0.0.v20130214-0015.jar codec-1.3.0.v201101211617.j
 ar
Created-By: Apache Maven 3.0.5
Build-Jdk: 1.8.0_121
```

6. Build the project in the folder, for example, `/home/beta/work_diysdk/Base64Adapter/`:

   ```
   mvn package
   ```

7. Use a simple zip command to package the project in the folder, for example, `/home/beta/work_diysdk/Base64Adapter/target/`:

   ```
   cd /home/beta/work_diysdk/Base64Adapter/target/
   mkdir lib
   cp /home/beta/.m2/repository/org/apache/commons/codec/1.3.0.v201101211617/
   codec-1.3.0.v201101211617.jar lib/
   zip Base64Adapter-1.0.zip Base64Adapter-1.0.jar lib/codec-1.3.0.v201101211617.jar
   ```

   Here is an example of the zipped contents:

   ```
   Base64Adapter-1.0.jar
   ▼ lib
       codec-1.3.0.v201101211617.jar
   ```

8. Unload the zipped package using Mashery Local Cluster Manager.
9. Apply the Custom Processor on the Endpoint using Mashery SaaS Control Center.

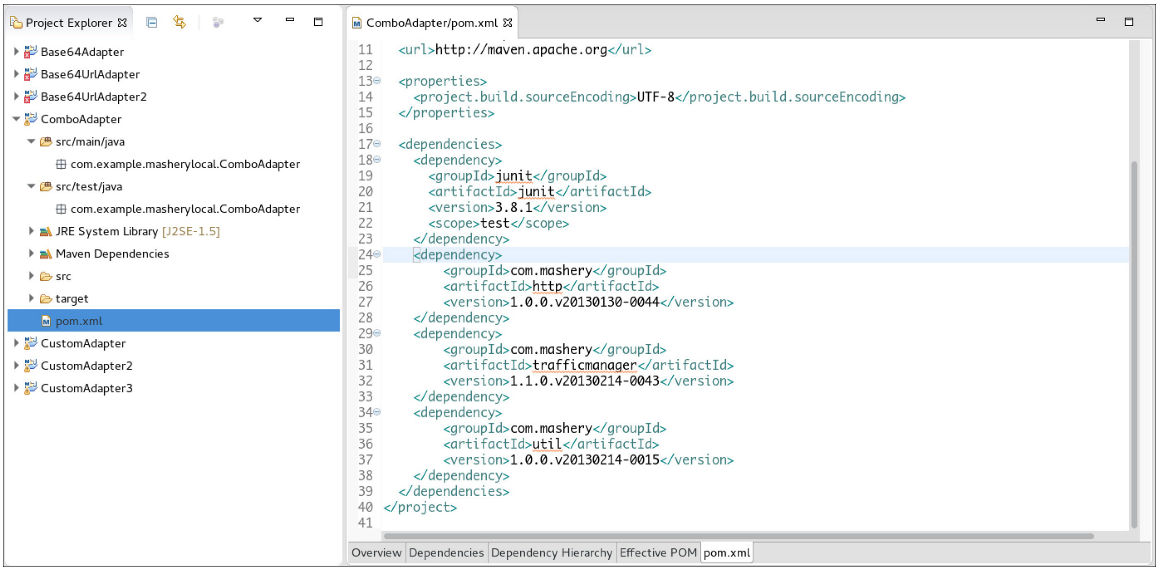
Using the Adapter SDK in Mashery Local with Multiple Processors in One Eclipse Project

To use the Adapter SDK in Mashery Local with multiple processors in one Eclipse project, follow the steps below:

Procedure

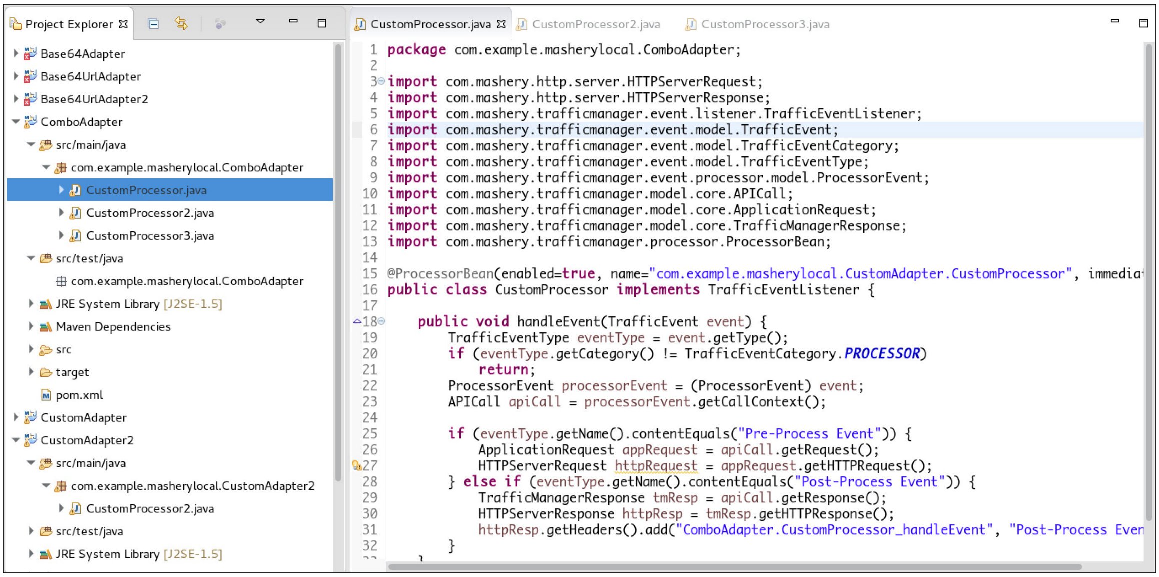
1. Create a new Maven Project "ComboAdapter" in Eclipse, then add dependencies, as described in Using the Adapter SDK in Mashery Local with a Single Processor.

   The project layout should look like the following:



```xml
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>com.mashery</groupId>
        <artifactId>http</artifactId>
        <version>1.0.0.v20130130-0044</version>
    </dependency>
    <dependency>
        <groupId>com.mashery</groupId>
        <artifactId>trafficmanager</artifactId>
        <version>1.1.0.v20130214-0043</version>
    </dependency>
    <dependency>
        <groupId>com.mashery</groupId>
        <artifactId>util</artifactId>
        <version>1.0.0.v20130214-0015</version>
    </dependency>
  </dependencies>
</project>
```

2. Add the classes for three custom processors as shown below:



```java
package com.example.masherylocal.ComboAdapter;

import com.mashery.http.server.HTTPServerRequest;
import com.mashery.http.server.HTTPServerResponse;
import com.mashery.trafficmanager.event.listener.TrafficEventListener;
import com.mashery.trafficmanager.event.model.TrafficEvent;
import com.mashery.trafficmanager.event.model.TrafficEventCategory;
import com.mashery.trafficmanager.event.model.TrafficEventType;
import com.mashery.trafficmanager.event.processor.model.ProcessorEvent;
import com.mashery.trafficmanager.model.core.APICall;
import com.mashery.trafficmanager.model.core.ApplicationRequest;
import com.mashery.trafficmanager.model.core.TrafficManagerResponse;
import com.mashery.trafficmanager.processor.ProcessorBean;

@ProcessorBean(enabled=true, name="com.example.masherylocal.CustomAdapter.CustomProcessor", immedia
public class CustomProcessor implements TrafficEventListener {

    public void handleEvent(TrafficEvent event) {
        TrafficEventType eventType = event.getType();
        if (eventType.getCategory() != TrafficEventCategory.PROCESSOR)
            return;
        ProcessorEvent processorEvent = (ProcessorEvent) event;
        APICall apiCall = processorEvent.getCallContext();

        if (eventType.getName().contentEquals("Pre-Process Event")) {
            ApplicationRequest appRequest = apiCall.getRequest();
            HTTPServerRequest httpRequest = appRequest.getHTTPRequest();
        } else if (eventType.getName().contentEquals("Post-Process Event")) {
            TrafficManagerResponse tmResp = apiCall.getResponse();
            HTTPServerResponse httpResp = tmResp.getHTTPResponse();
            httpResp.getHeaders().add("ComboAdapter.CustomProcessor_handleEvent", "Post-Process Even
        }
```

3. Build the project in the folder, for example:

```
cd /home/beta/work_diysdk/ComboAdapter
mvn package
```

4. Use a simple zip command to package the project in the folder, for example:

```
/home/beta/work_diysdk/ComboAdapter/target
zip ComboAdapter-1.0.zip ComboAdapter-1.0.jar
```

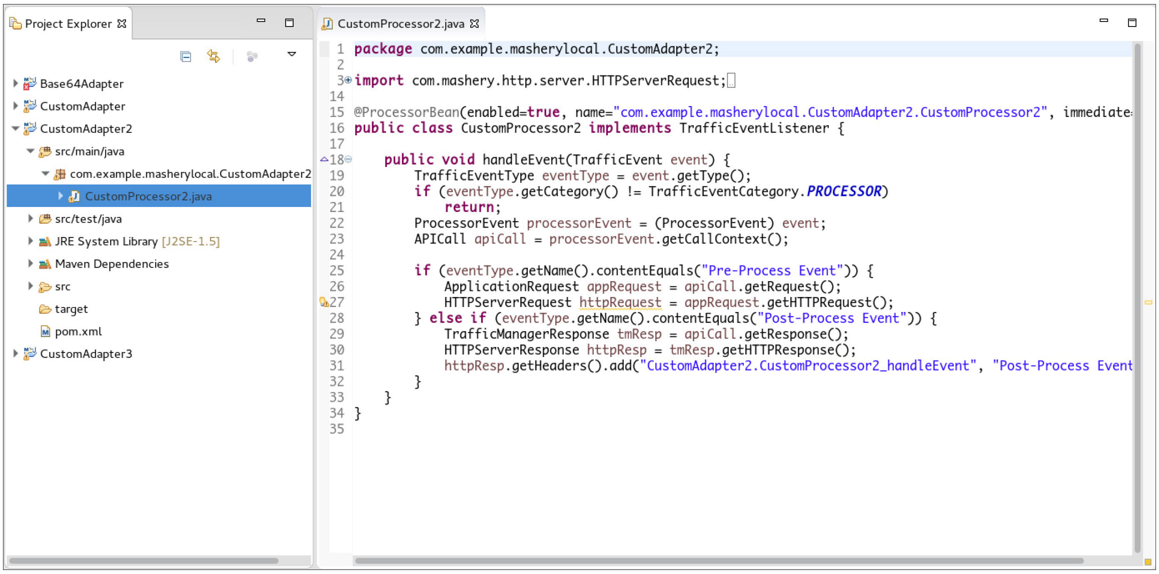
5. Unload the zipped package using Mashery Local Cluster Manager.
6. Apply the Custom Processor on the Endpoint using Mashery SaaS Control Center.

Using the Adapter SDK in Mashery Local with Multiple Processors in One Zip Package

To use the Adapter SDK in Mashery Local with multiple processors in one zip package, follow the steps below:

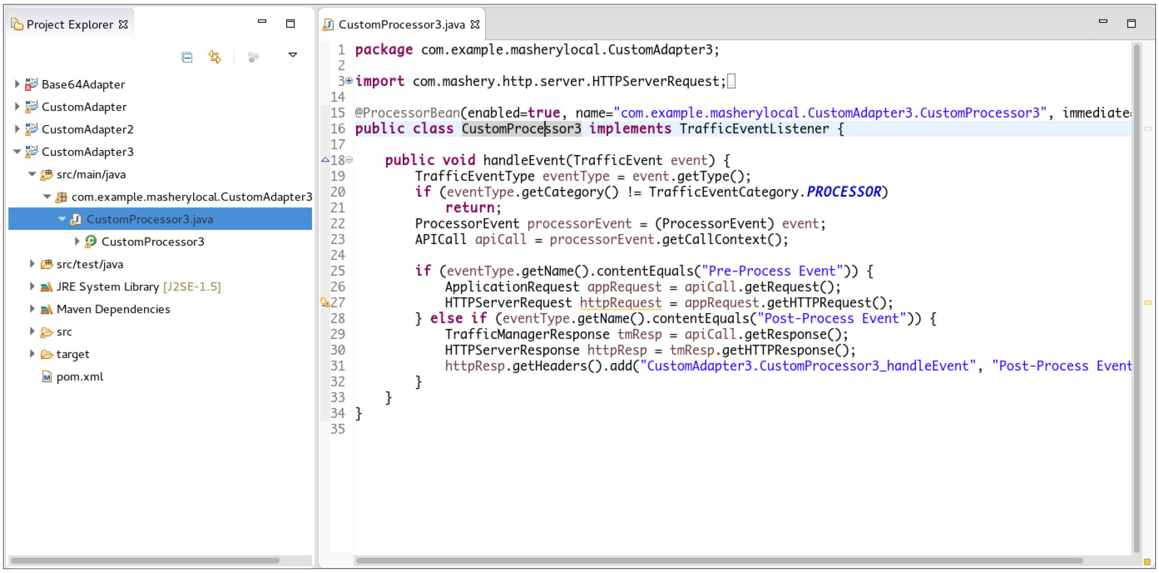
Procedure

1. Create a new Maven project "CustomAdapter2" in Eclipse, then add dependencies and class, as described in Using the Adapter SDK in Mashery Local with a Single Processor.
   The project layout should look like the following:



```java
package com.example.masherylocal.CustomAdapter2;

import com.mashery.http.server.HTTPServerRequest;

@ProcessorBean(enabled=true, name="com.example.masherylocal.CustomAdapter2.CustomProcessor2", immediate
public class CustomProcessor2 implements TrafficEventListener {

    public void handleEvent(TrafficEvent event) {
        TrafficEventType eventType = event.getType();
        if (eventType.getCategory() != TrafficEventCategory.PROCESSOR)
            return;
        ProcessorEvent processorEvent = (ProcessorEvent) event;
        APICall apiCall = processorEvent.getCallContext();

        if (eventType.getName().contentEquals("Pre-Process Event")) {
            ApplicationRequest appRequest = apiCall.getRequest();
            HTTPServerRequest httpRequest = appRequest.getHTTPRequest();
        } else if (eventType.getName().contentEquals("Post-Process Event")) {
            TrafficManagerResponse tmResp = apiCall.getResponse();
            HTTPServerResponse httpResp = tmResp.getHTTPResponse();
            httpResp.getHeaders().add("CustomAdapter2.CustomProcessor2_handleEvent", "Post-Process Event
        }
    }
}

```

2. Create a new Maven Project "CustomAdapter3" in Eclipse, then add dependencies and class, as described in Using the Adapter SDK in Mashery Local with a Single Processor.
   The project layout should look like the following:



```java
package com.example.masherylocal.CustomAdapter3;

import com.mashery.http.server.HTTPServerRequest;

@ProcessorBean(enabled=true, name="com.example.masherylocal.CustomAdapter3.CustomProcessor3", immediate
public class CustomProcessor3 implements TrafficEventListener {

    public void handleEvent(TrafficEvent event) {
        TrafficEventType eventType = event.getType();
        if (eventType.getCategory() != TrafficEventCategory.PROCESSOR)
            return;
        ProcessorEvent processorEvent = (ProcessorEvent) event;
        APICall apiCall = processorEvent.getCallContext();

        if (eventType.getName().contentEquals("Pre-Process Event")) {
            ApplicationRequest appRequest = apiCall.getRequest();
            HTTPServerRequest httpRequest = appRequest.getHTTPRequest();
        } else if (eventType.getName().contentEquals("Post-Process Event")) {
            TrafficManagerResponse tmResp = apiCall.getResponse();
            HTTPServerResponse httpResp = tmResp.getHTTPResponse();
            httpResp.getHeaders().add("CustomAdapter3.CustomProcessor3_handleEvent", "Post-Process Event
        }
    }
}

```

3. Build the projects in the folders, for example, `/home/beta/work_diysdk/CustomAdapter2/` and `/home/beta/work_diysdk/CustomAdapter3/`:

```
cd /home/beta/work_diysdk/CustomAdapter2/
mvn package


cd /home/beta/work_diysdk/CustomAdapter3/
mvn package
```

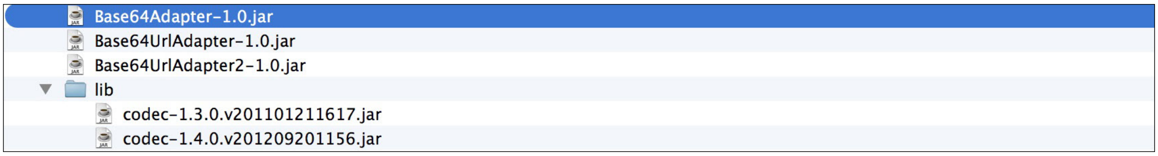
4. Use a simple zip command to package the project in the folder, for example, `/home/beta/work_diysdk/CustomAdapter/target/`:

```
cd /home/beta/work_diysdk/CustomAdapter/target
cp /home/beta/work_diysdk/CustomAdapter2/target/CustomAdapter2-1.0.jar .
cp /home/beta/work_diysdk/CustomAdapter3/target/CustomAdapter3-1.0.jar .
zip CustomAdapters-1.0.zip CustomAdapter-1.0.jar CustomAdapter2-1.0.jar
CustomAdapter3-1.0.jar
```

5. Unload the zipped package using Mashery Local Cluster Manager.
6. Apply the Custom Processor on the Endpoint using Mashery SaaS Control Center.

Using the Adapter SDK in Mashery Local with Multiple Processors in One Package and Third Party Libraries

To use the Adapter SDK in Mashery Local with multiple processors in one package and third party libraries, note that the following package contains three custom processors: `Base64Adapter-1.0.jar references codec-1.3.0.v201101211617.jar`, `Base64UrlAdapter-1.0.jar` and `Base64UrlAdapter2-1.0.jar reference codec-1.4.0.v201209201156.jar`.



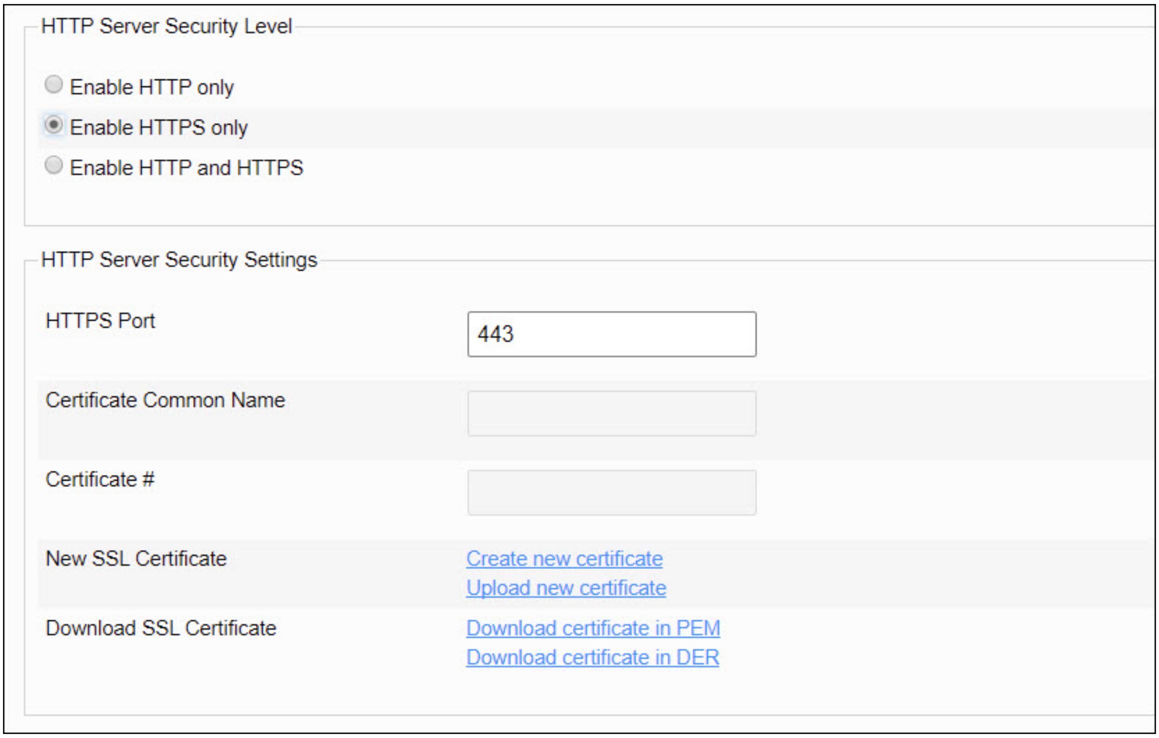
Base64Adapter-1.0.jar
Base64UrlAdapter-1.0.jar
Base64UrlAdapter2-1.0.jar
lib
    codec-1.3.0.v201101211617.jar
    codec-1.4.0.v201209201156.jar

Setting up HTTPS Server using Self-Signed Certificate

To set up HTTPS Server to use a self-signed certificate, follow the steps below:

Procedure

1. On the **Mashery Cluster Manager** tab, click **Instance Management**.
2. Scroll down to **HTTP Server Security Level** section and select **Enable HTTPS only**.



HTTP Server Security Level

○ Enable HTTP only
● Enable HTTPS only
○ Enable HTTP and HTTPS

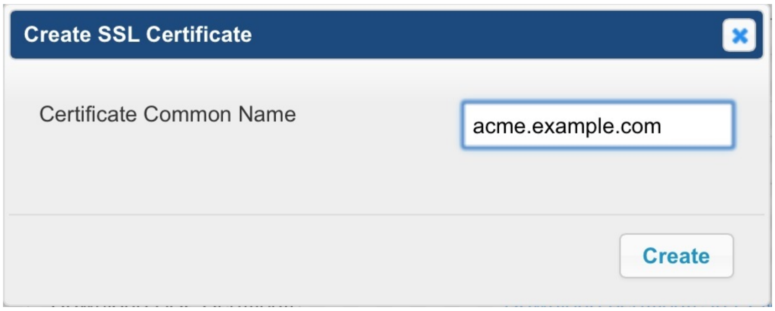
HTTP Server Security Settings

HTTPS Port	443
Certificate Common Name	
Certificate #	
New SSL Certificate	Create new certificate Upload new certificate
Download SSL Certificate	Download certificate in PEM Download certificate in DER

3. In the **HTTP Server Security Settings** section, specify the **HTTPS Port** number (default is 443).

   > Administrators can change the port number to another number, such as `8443`. For Linux installations, it is not advised to choose a port number below 1000. In addition, the following reserved port numbers are used by TIBCO Mashery Local: 3306, 8081, 8082, 8083, 5489, 11211, 11212, 11213, 11214, 11215, and 11216.

4. Use your server self-signed certificate. Click **Create new certificate**.
   The **Create SSL Certificate** window is displayed.



Create SSL Certificate

Certificate Common Name: acme.example.com

5. Enter a name in the **Certificate Common Name** field, for example, `acme.example.com`, then click **Create**.
6. Click **Save** to save all changes. It will take a few minutes for Mashery Local service to restart.

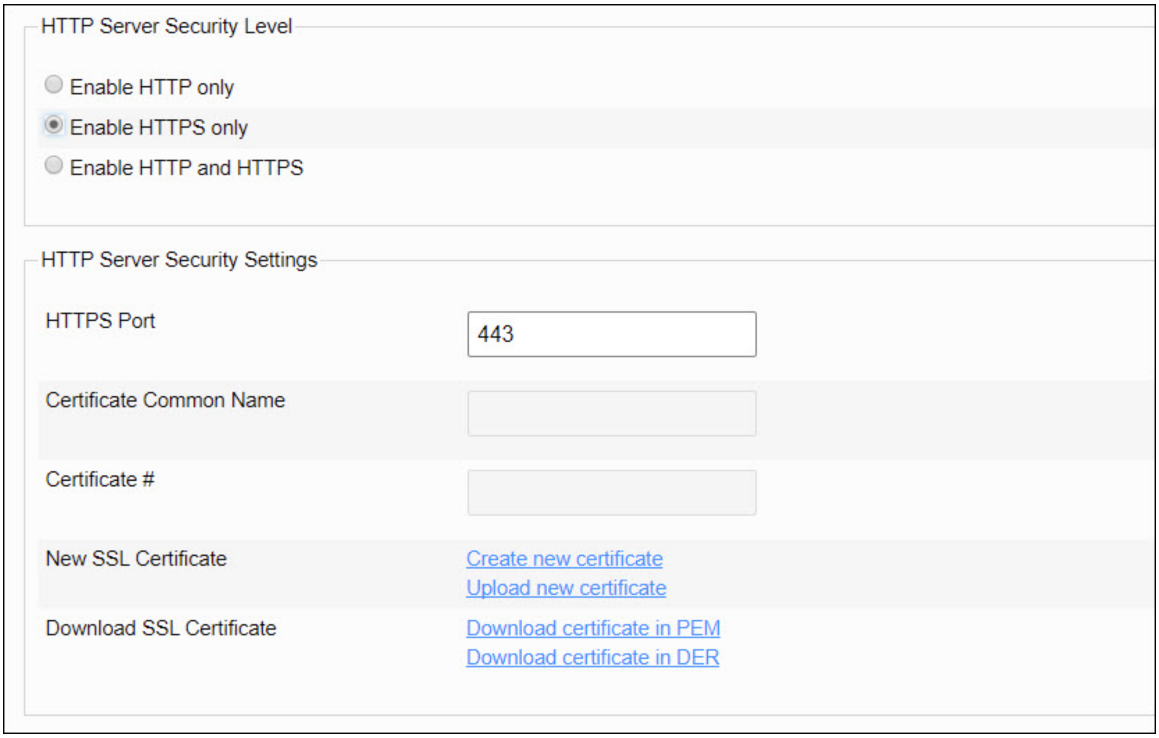
Setting up HTTPS Server using Customer-Provided Certificate

To set up HTTPS Server to use a self-signed certificate, follow the steps below:

Procedure

1. On the **Mashery Cluster Manager** tab, click **Instance Management**.

2. Scroll down to **HTTP Server Security Level** section and select **Enable HTTPS only**.



HTTP Server Security Level

○ Enable HTTP only
● Enable HTTPS only
○ Enable HTTP and HTTPS

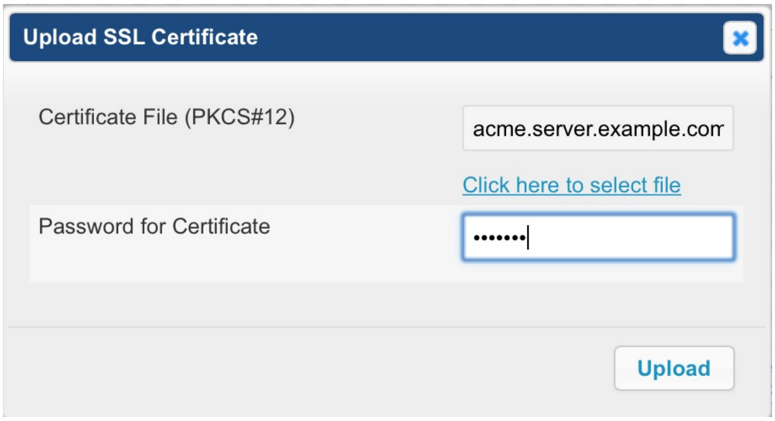
HTTP Server Security Settings

HTTPS Port	443
Certificate Common Name	
Certificate #	
New SSL Certificate	Create new certificate Upload new certificate
Download SSL Certificate	Download certificate in PEM Download certificate in DER

3. In the **HTTP Server Security Settings** section, specify the **HTTPS Port** number (default is 443).

> Administrators can change the port number to another number, such as `8443`. For Linux installations, it is not advised to choose a port number below 1000. In addition, the following reserved port numbers are used by TIBCO Mashery Local: 3306, 8081, 8082, 8083, 5489, 11211, 11212, 11213, 11214, 11215, and 11216.

4. Use your CA certificate. Click **Upload new certificate**.
   The **Upload SSL Certificate** window is displayed.



Upload SSL Certificate

Certificate File (PKCS#12)	acme.server.example.com Click here to select file
Password for Certificate	•••••••

5. In the **Upload SSL Certificate** window, click the **Click here to select file** link, browse to the CA certificate file, then click **Upload**.

> The Certificate File should be in PKCS#12 format.

6. Click **Save** to save all changes. It will take a few minutes for Mashery Local service to restart.

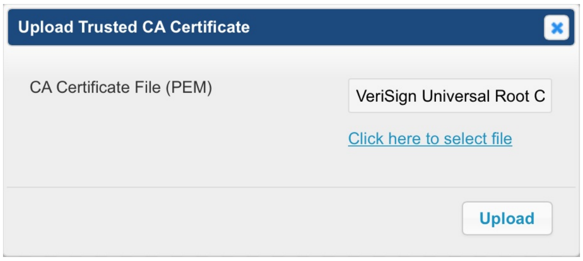
Configuring and Using the HTTPS Client Feature without Mutual Authentication

To use the HTTPS Client Feature without Mutual Authentication, you will need to configure **Trust Management** settings in Mashery Cluster Manager and configure an HTTPS Client Profile in Mashery SaaS (Control Center).

For Mashery Local:

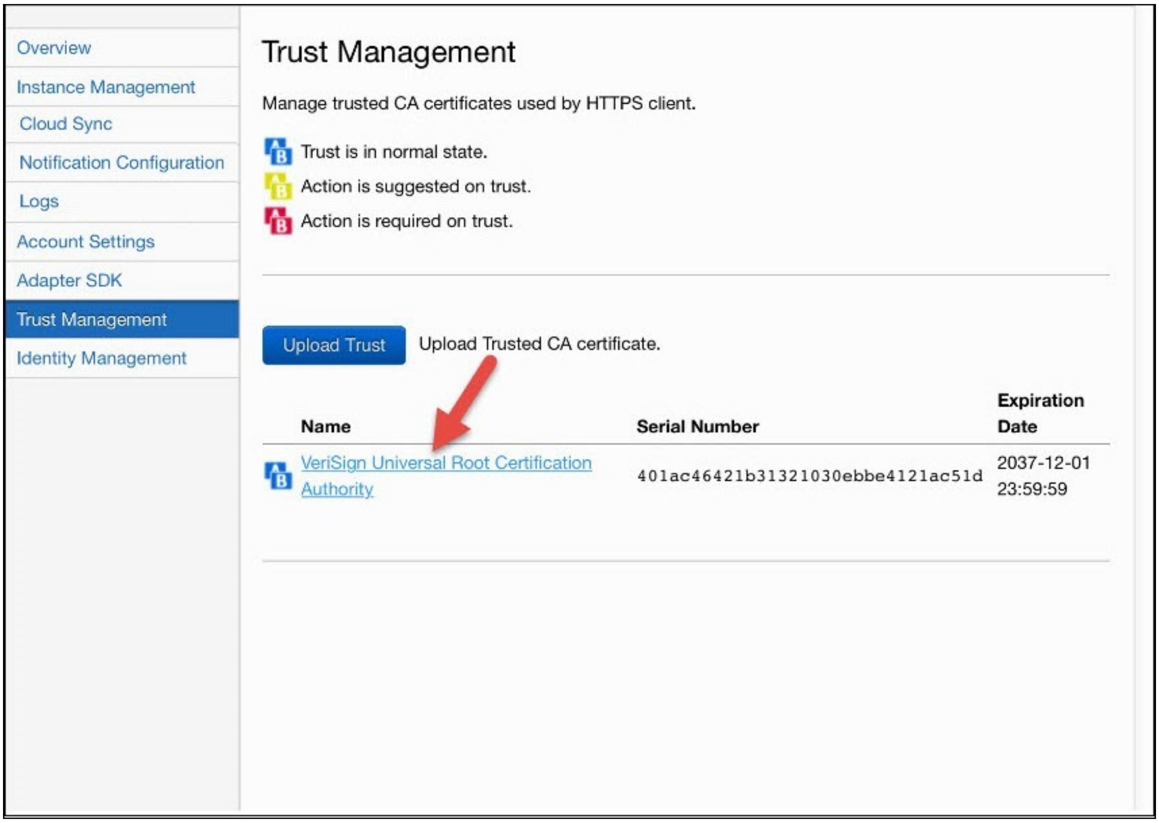
Procedure

1. On the **Mashery Cluster Manager** tab, click **Trust Management**.
2. Click **Upload Trust**. The **Upload Trusted CA Certificate** window is displayed.


Upload Trusted CA Certificate

CA Certificate File (PEM): VeriSign Universal Root C

3. In the **Upload Trusted CA Certificate** window, click the **Click here to select file** link, browse to the CA certificate, then click **Upload**.
4. The CA Certificate is now added as a trusted certificate. Click the link next to the certificate name to view the state.



Trust Management

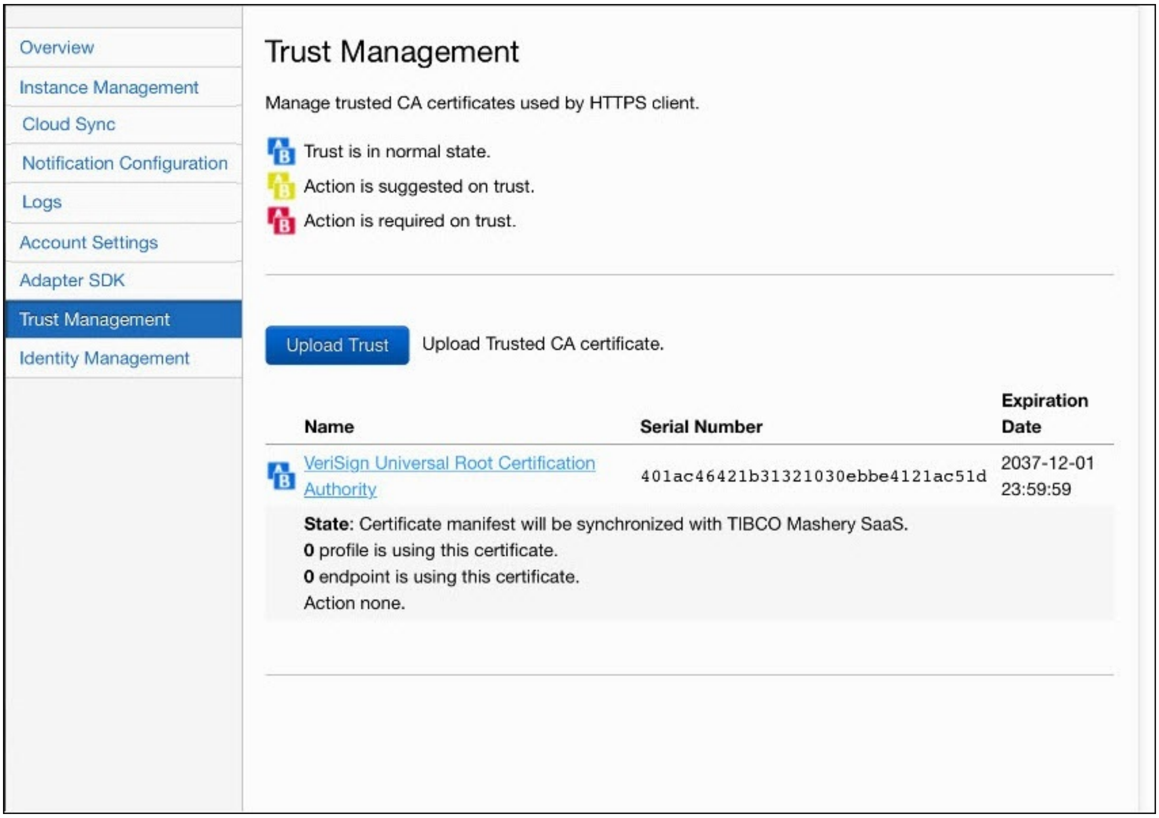
Manage trusted CA certificates used by HTTPS client.

- Trust is in normal state.
- Action is suggested on trust.
- Action is required on trust.

Upload Trusted CA certificate.

Name	Serial Number	Expiration Date
VeriSign Universal Root Certification Authority	401ac46421b31321030ebbe4121ac51d	2037-12-01 23:59:59

5. In this example, the State is `Certificate manifest will be synchronized with TIBCO Mashery SaaS`. Mashery Local will synchronize automatically, or you can manually trigger a sync in **Cloud Sync** settings.



Trust Management

Manage trusted CA certificates used by HTTPS client.

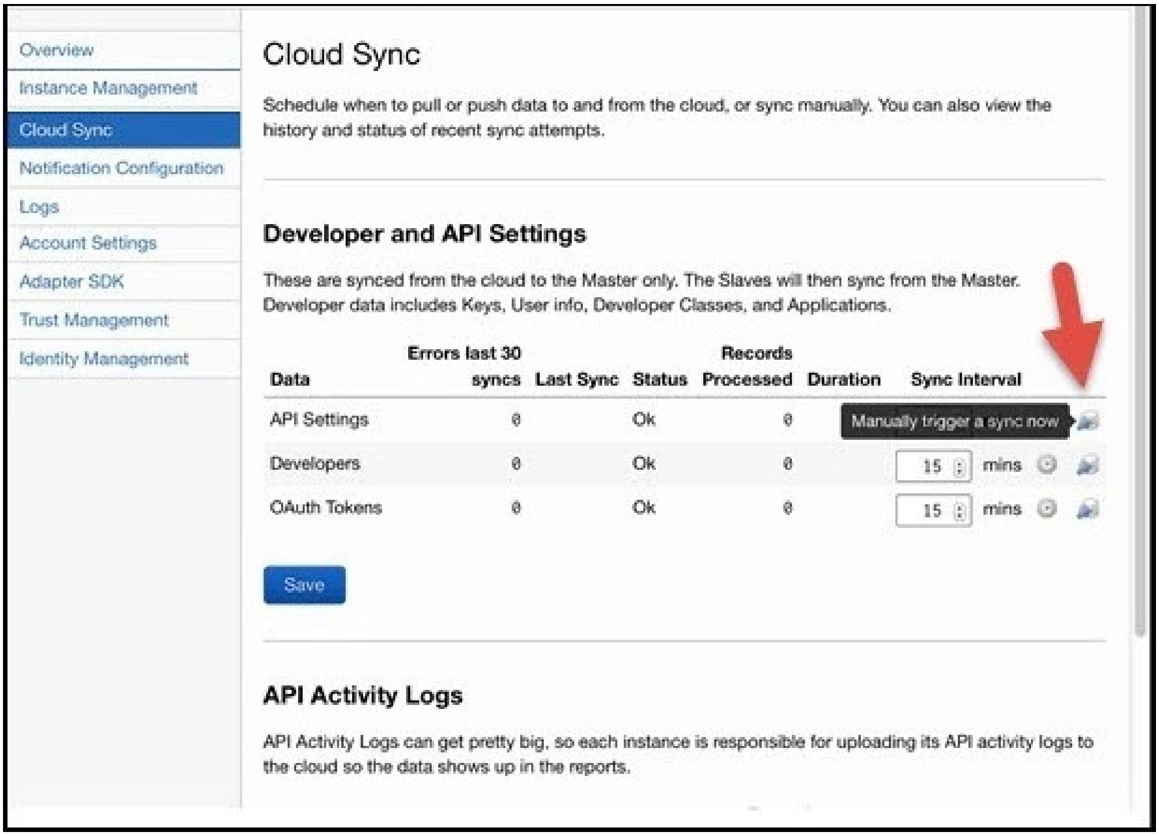
Trust is in normal state.
Action is suggested on trust.
Action is required on trust.

Upload Trusted CA certificate.

Name	Serial Number	Expiration Date
VeriSign Universal Root Certification Authority	401ac46421b31321030ebbe4121ac51d	2037-12-01 23:59:59

State: Certificate manifest will be synchronized with TIBCO Mashery SaaS.
0 profile is using this certificate.
0 endpoint is using this certificate.
Action none.

6. To manually trigger a sync, on the **Mashery Cluster Manager** tab, click **Cloud Sync**.



Cloud Sync

Schedule when to pull or push data to and from the cloud, or sync manually. You can also view the history and status of recent sync attempts.

Developer and API Settings

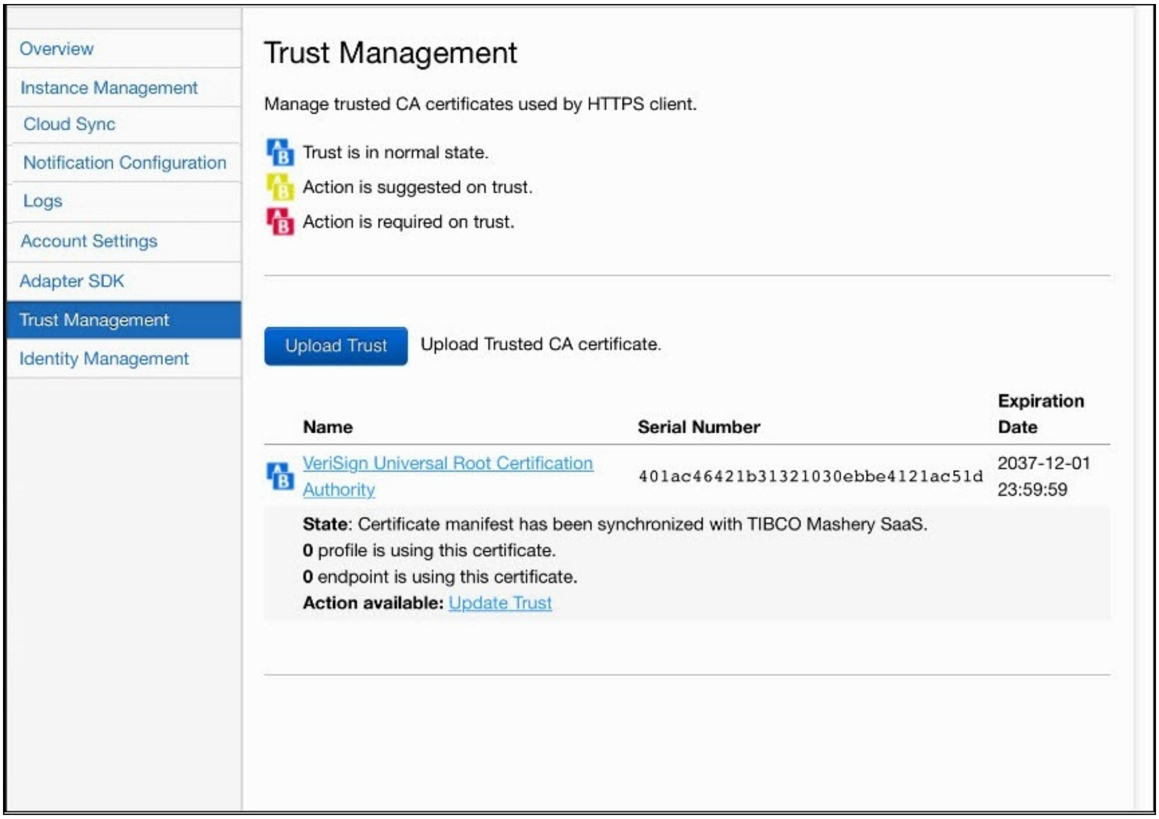
These are synced from the cloud to the Master only. The Slaves will then sync from the Master. Developer data includes Keys, User info, Developer Classes, and Applications.

Data	Errors last 30 syncs	Last Sync	Status	Records Processed	Duration	Sync Interval
API Settings	0		Ok	0		
Developers	0		Ok	0		15 mins
OAuth Tokens	0		Ok	0		15 mins

API Activity Logs

API Activity Logs can get pretty big, so each instance is responsible for uploading its API activity logs to the cloud so the data shows up in the reports.

7. In the **Developer and API Settings** section, for API Settings, click the **Sync** button to manually trigger a sync. This will make the certificate metadata available instantly in Mashery SaaS; otherwise, the sync will occur according to the minutes defined for the **Sync Interval** setting.
8. After the certificate becomes available in SaaS, the State changes to `Certificate manifest has been synchronized with TIBCO Mashery SaaS`.



Trust Management

Manage trusted CA certificates used by HTTPS client.

Trust is in normal state.

Action is suggested on trust.

Action is required on trust.

Upload Trusted CA certificate.

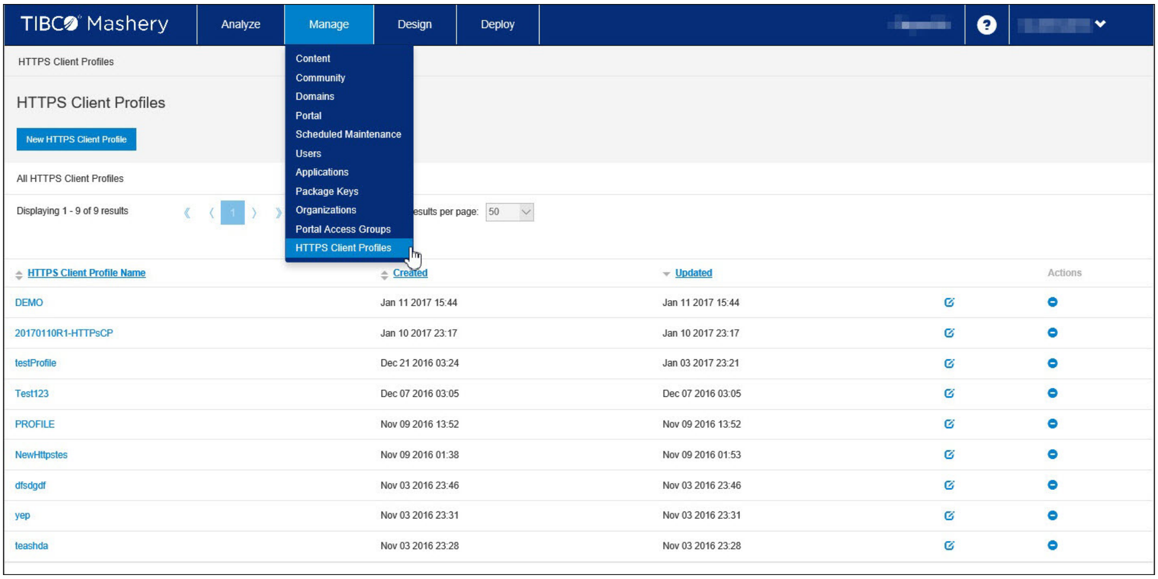
Name	Serial Number	Expiration Date
VeriSign Universal Root Certification Authority	401ac46421b31321030ebbe4121ac51d	2037-12-01 23:59:59

State: Certificate manifest has been synchronized with TIBCO Mashery SaaS.
0 profile is using this certificate.
0 endpoint is using this certificate.
Action available: Update Trust

What to do next

To create an HTTPS Client Profile in TIBCO Mashery SaaS, follow the steps below:

1. Click `Manage > HTTPS Client Profiles`. The **HTTPS Client Profiles** window is displayed.



HTTPS Client Profiles

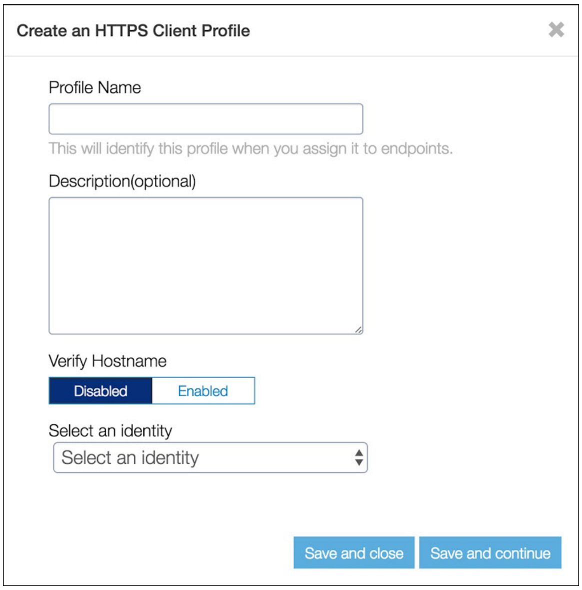
HTTPS Client Profiles

All HTTPS Client Profiles

Displaying 1 - 9 of 9 results

HTTPS Client Profile Name	Created	Updated		Actions
DEMO	Jan 11 2017 15:44	Jan 11 2017 15:44	✎	⊖
20170110R1-HTTPsCP	Jan 10 2017 23:17	Jan 10 2017 23:17	✎	⊖
testProfile	Dec 21 2016 03:24	Jan 03 2017 23:21	✎	⊖
Test123	Dec 07 2016 03:05	Dec 07 2016 03:05	✎	⊖
PROFILE	Nov 09 2016 13:52	Nov 09 2016 13:52	✎	⊖
NewHttpstes	Nov 09 2016 01:38	Nov 09 2016 01:53	✎	⊖
dfsdgdf	Nov 03 2016 23:46	Nov 03 2016 23:46	✎	⊖
yep	Nov 03 2016 23:31	Nov 03 2016 23:31	✎	⊖
teashda	Nov 03 2016 23:28	Nov 03 2016 23:28	✎	⊖

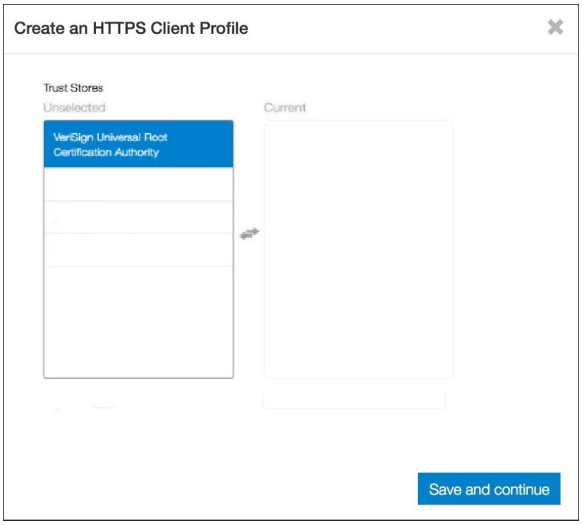
2. Click the **New HTTPS Client Profile** button. The **Create an HTTPS Client Profile** window is displayed.



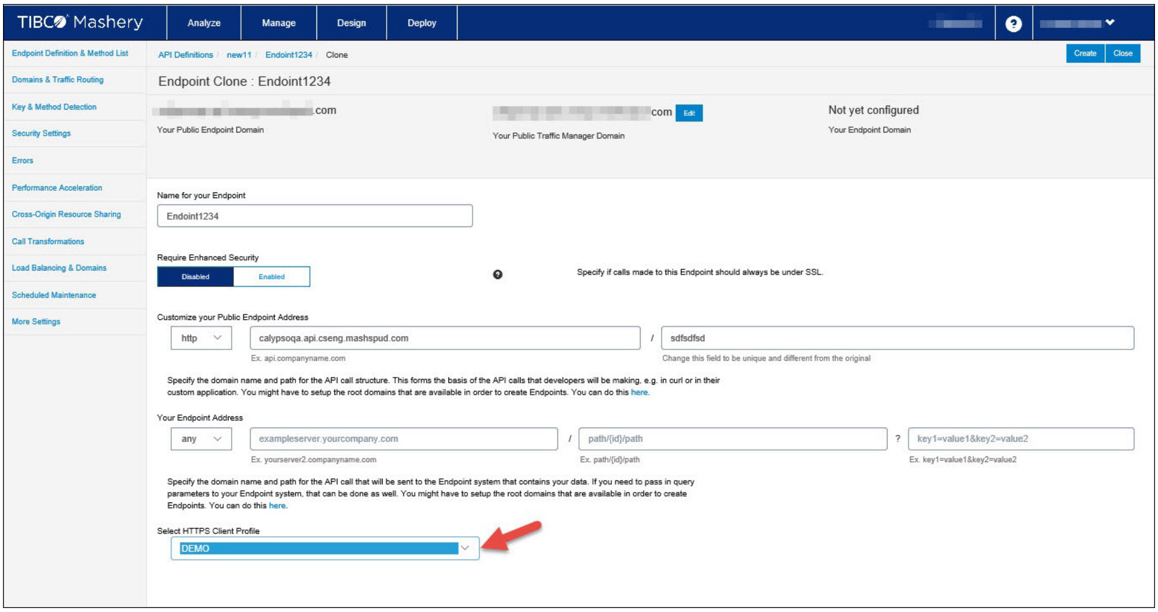
3. On the **Create an HTTPS Client Profiles** window, enter information in the following fields:

Field	Description
Profile name	Enter a name for the HTTPS client profile.
Description	(Optional) Enter a description for the HTTPS client profile.
Verify Hostname	Select one of the following: • **Disabled**: Click to not have the hostname verified. • **Enabled**: Click to have the hostname verified.
Select an identity	Select an identity for the HTTPS client profile.

4. Click **Save and Continue**.
5. A second **Create an HTTPS Client Profile** page displays.


Create an HTTPS Client Profile
Trust Stores
Unselected
Current
VeriSign Universal Root Certification Authority
Save and continue

6. In the **Unselected** list of the **Trust stores** section, click the Trusted CA Certificate you uploaded in Mashery Cluster Manager to move it to the **Current** list, then click **Save and continue** to finish creating the HTTPS Client Profile. Once the HTTPS Client Profile is created, you can then select the profile when creating an endpoint on the **Endpoint Create: New Endpoint Definition** page.
7. To assign the HTTPS Client Profile to an endpoint, click `Design > API Definitions > Domains & Traffic Routing`.
8. On the **Domains & Traffic Routing** page for the existing endpoint of your API definition (or, **Endpoint Create: New Endpoint Definition** page for a new endpoint), use the **Select HTTP Client Profile** field to select the HTTPS Client Profile you just created, then click **Save** (or **Create**). The endpoint is now associated with the HTTPS Client Profile.



[redacted] [redacted]

Endpoint Clone : Endoint1234

[redacted].com
Your Public Endpoint Domain

[redacted].com
Your Public Traffic Manager Domain

Not yet configured
Your Endpoint Domain

Name for your Endpoint: Endoint1234

Require Enhanced Security: Disabled | Enabled

Specify if calls made to this Endpoint should always be under SSL.

Customize your Public Endpoint Address: http calypsoqa.api.cseng.mashspud.com / sdfsdfsd

Ex. api.companyname.com

Change this field to be unique and different from the original

Specify the domain name and path for the API call structure. This forms the basis of the API calls that developers will be making, e.g. in curl or in their custom application. You might have to setup the root domains that are available in order to create Endpoints. You can do this here.

Your Endpoint Address: any exampleserver.yourcompany.com / path/{id}/path ? key1=value1&key2=value2

Ex. yourserver2.companyname.com

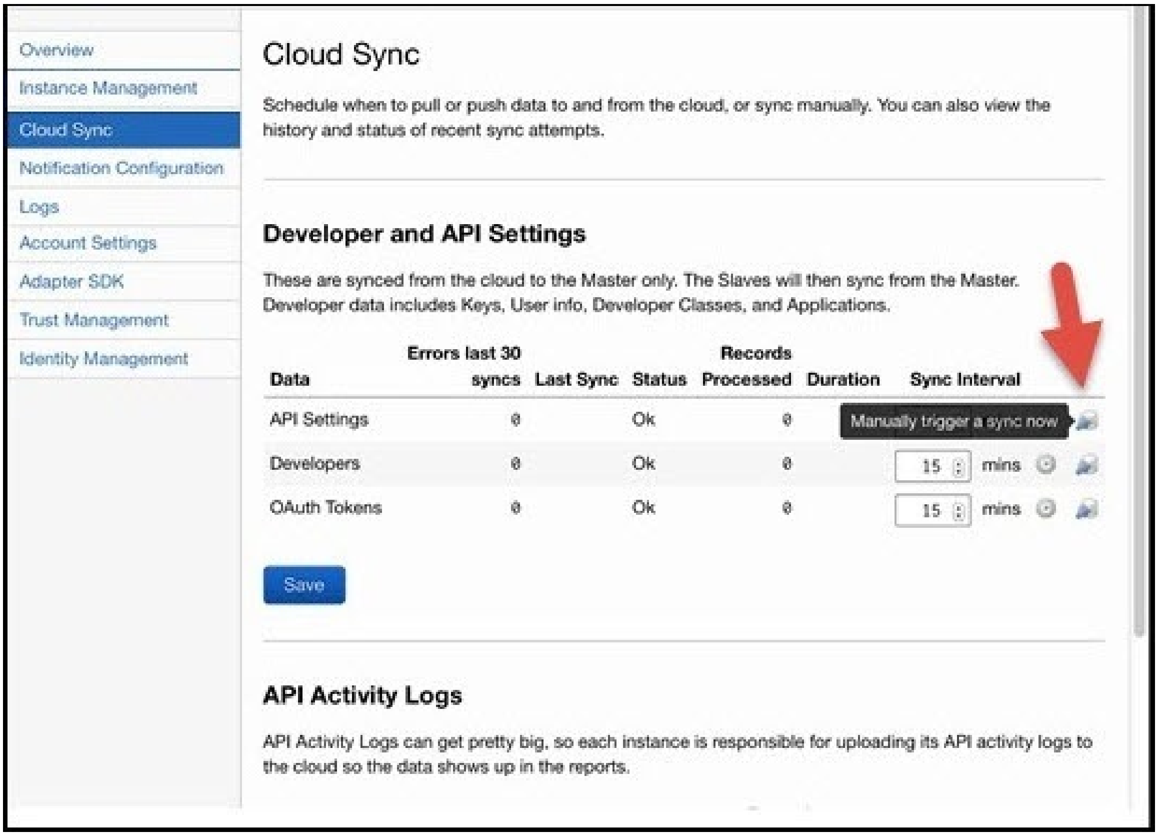
Ex. path/{id}/path

Ex. key1=value1&key2=value2

Specify the domain name and path for the API call that will be sent to the Endpoint system that contains your data. If you need to pass in query parameters to your Endpoint system, that can be done as well. You might have to setup the root domains that are available in order to create Endpoints. You can do this here.

Select HTTPS Client Profile: DEMO

9. Log back into Mashery Cluster Manager, go to the **Cloud Sync** tab, and click the manual sync button. This syncs the HTTPS Client Profile and Endpoint configuration updates to Mashery Local, and the HTTP Client Profile is now in use for the customer.



Cloud Sync

Schedule when to pull or push data to and from the cloud, or sync manually. You can also view the history and status of recent sync attempts.

Developer and API Settings

These are synced from the cloud to the Master only. The Slaves will then sync from the Master. Developer data includes Keys, User info, Developer Classes, and Applications.

Data	Errors last 30 syncs	Last Sync	Status	Records Processed	Duration	Sync Interval
API Settings	0		Ok	0		
Developers	0		Ok	0		15 mins
OAuth Tokens	0		Ok	0		15 mins

API Activity Logs

API Activity Logs can get pretty big, so each instance is responsible for uploading its API activity logs to the cloud so the data shows up in the reports.

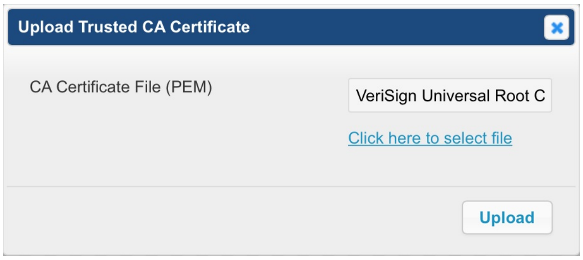
Configuring and Using the HTTPS Client Feature with Mutual Authentication

To use the HTTPS Client Feature without Mutual Authentication, you will need to configure **Trust Management** and **Identity Management** settings in Mashery Cluster Manager and configure an HTTPS Client Profile in Mashery SaaS (Control Center).

For Mashery Local:

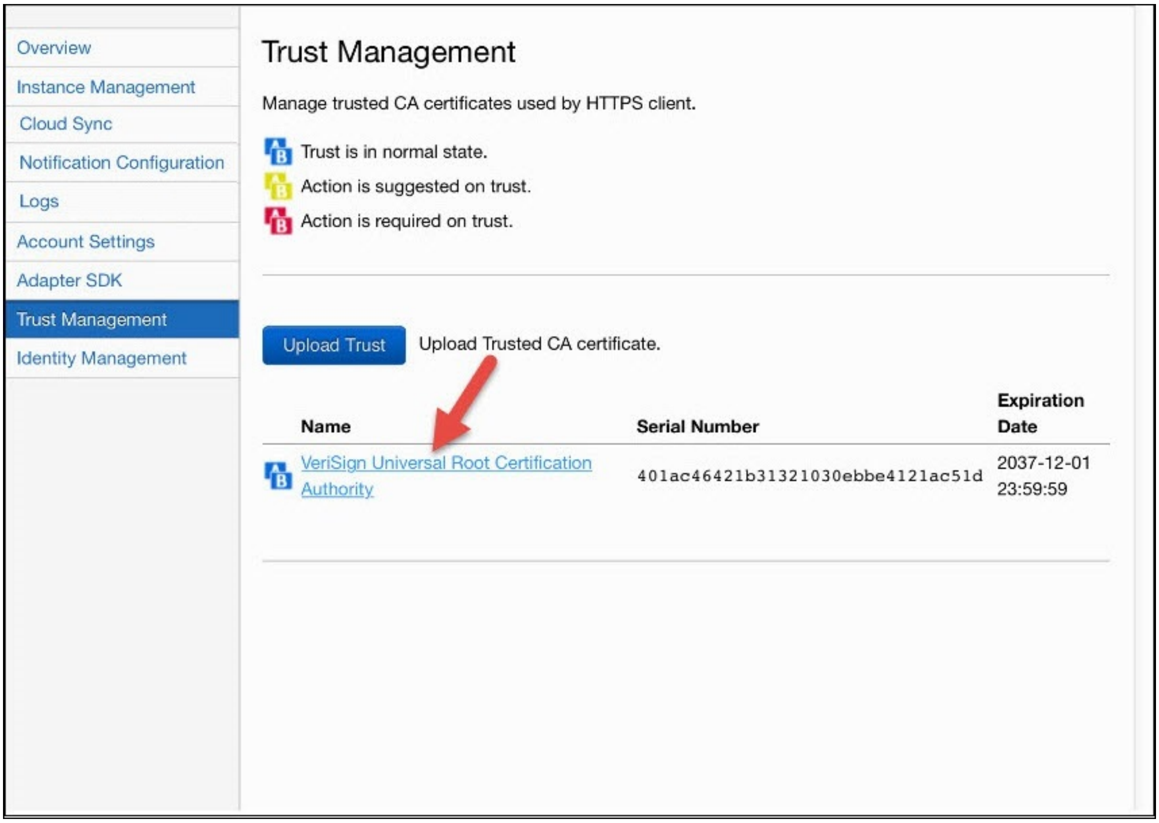
Procedure

1. On the **Mashery Cluster Manager** tab, click **Trust Management**.
2. Click **Upload Trust**. The **Upload Trusted CA Certificate** window is displayed.



CA Certificate File (PEM): VeriSign Universal Root C

3. In the **Upload Trusted CA Certificate** window, click the **Click here to select file** link, browse to the CA certificate, then click **Upload**.
4. The CA Certificate is now added as a trusted certificate. Click the link next to the certificate name to view the state.



Trust Management

Manage trusted CA certificates used by HTTPS client.

Trust is in normal state.

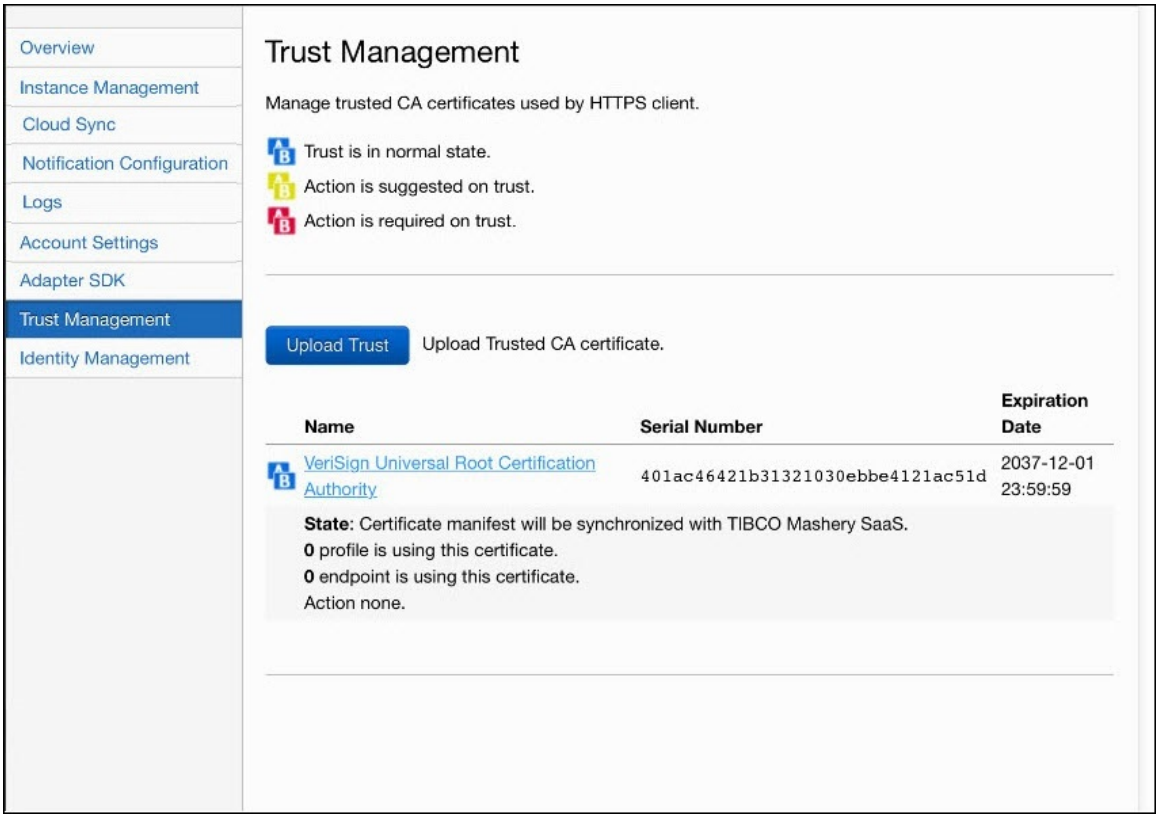
Action is suggested on trust.

Action is required on trust.

Upload Trusted CA certificate.

Name	Serial Number	Expiration Date
VeriSign Universal Root Certification Authority	401ac46421b31321030ebbe4121ac51d	2037-12-01 23:59:59

5. In this example, the State is `Certificate manifest will be synchronized with TIBCO Mashery SaaS`. Mashery Local will synchronize automatically, or you can manually trigger a sync in **Cloud Sync** settings.



Trust Management

Manage trusted CA certificates used by HTTPS client.

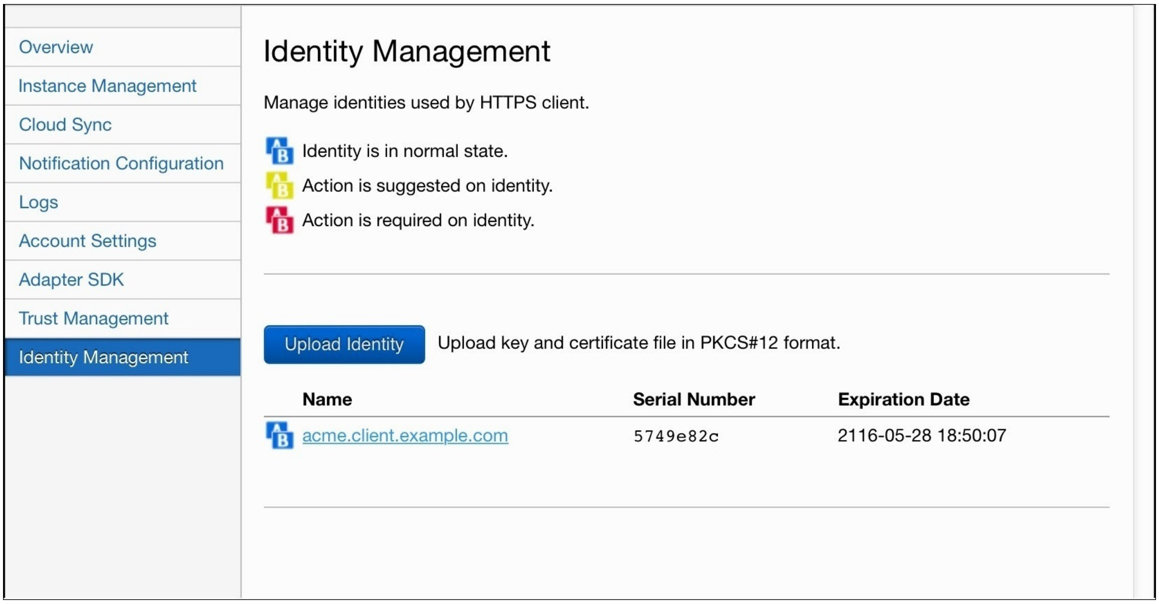
Trust is in normal state.
Action is suggested on trust.
Action is required on trust.

Upload Trusted CA certificate.

Name	Serial Number	Expiration Date
VeriSign Universal Root Certification Authority	401ac46421b31321030ebbe4121ac51d	2037-12-01 23:59:59

State: Certificate manifest will be synchronized with TIBCO Mashery SaaS.
0 profile is using this certificate.
0 endpoint is using this certificate.
Action none.

6. On the **Mashery Cluster Manager** tab, click **Identity Management**.



Identity Management

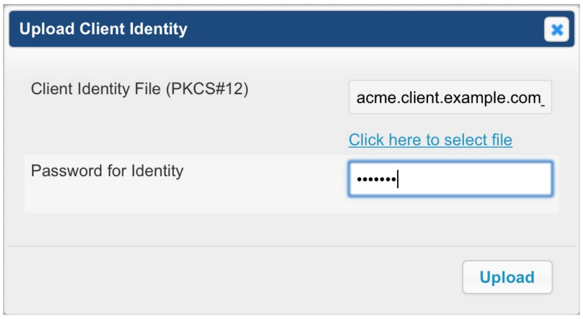
Manage identities used by HTTPS client.

Identity is in normal state.
Action is suggested on identity.
Action is required on identity.

Upload key and certificate file in PKCS#12 format.

Name	Serial Number	Expiration Date
acme.client.example.com	5749e82c	2116-05-28 18:50:07

7. Click **Upload Identity**. The **Upload Client Identity** window is displayed.

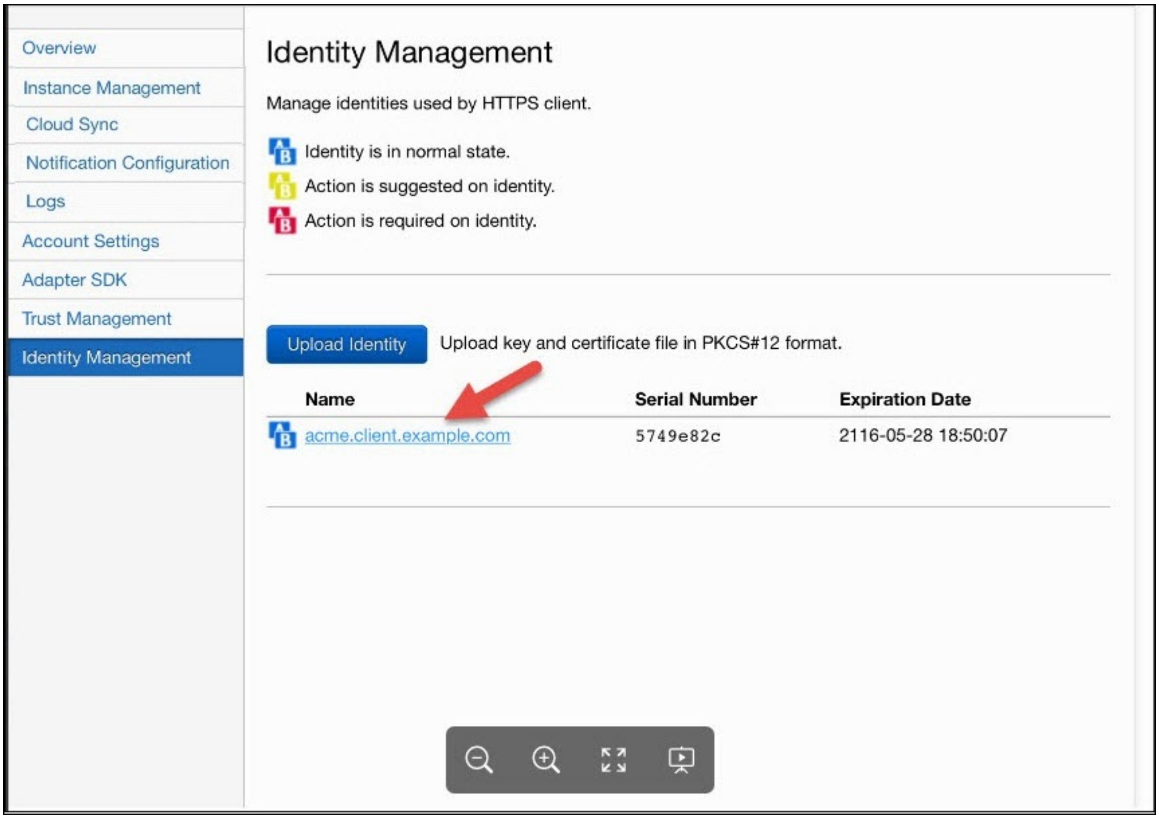


Upload Client Identity

Client Identity File (PKCS#12): acme.client.example.com_

Password for Identity: •••••••

8. In the **Upload Client Identity** window, click the **Click here to select file** link, browse to the identity file (PKCS#12 format), enter the **Password for Identity**, then click **Upload**.
9. The Client Identity is now added as an Identity. Click the link next to the Identity name to view the state.



Identity Management

Manage identities used by HTTPS client.

Identity is in normal state.

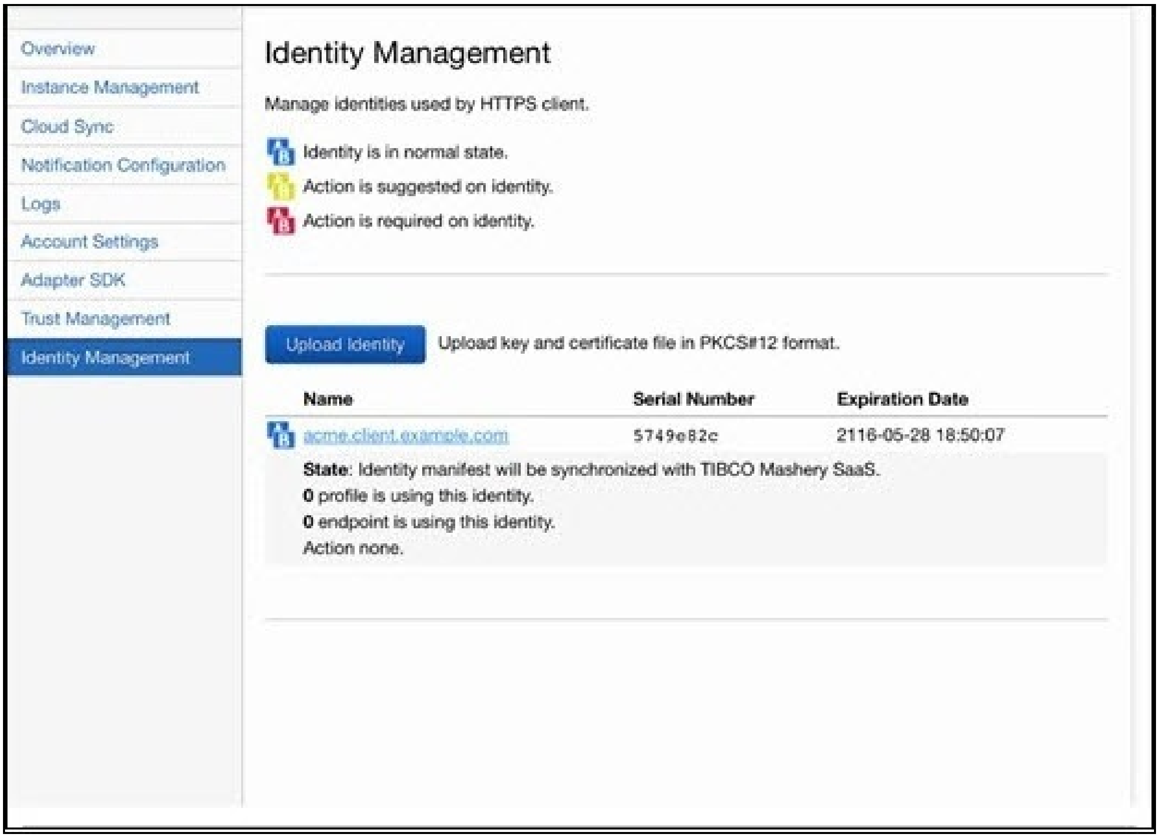
Action is suggested on identity.

Action is required on identity.

Upload key and certificate file in PKCS#12 format.

Name	Serial Number	Expiration Date
acme.client.example.com	5749e82c	2116-05-28 18:50:07

10. In this example, the State is `Identity manifest will be synchronized with TIBCO Mashery SaaS`. Mashery Local will synchronize automatically, or you can manually trigger the sync in **Cloud Sync** settings.



Identity Management

Manage identities used by HTTPS client.

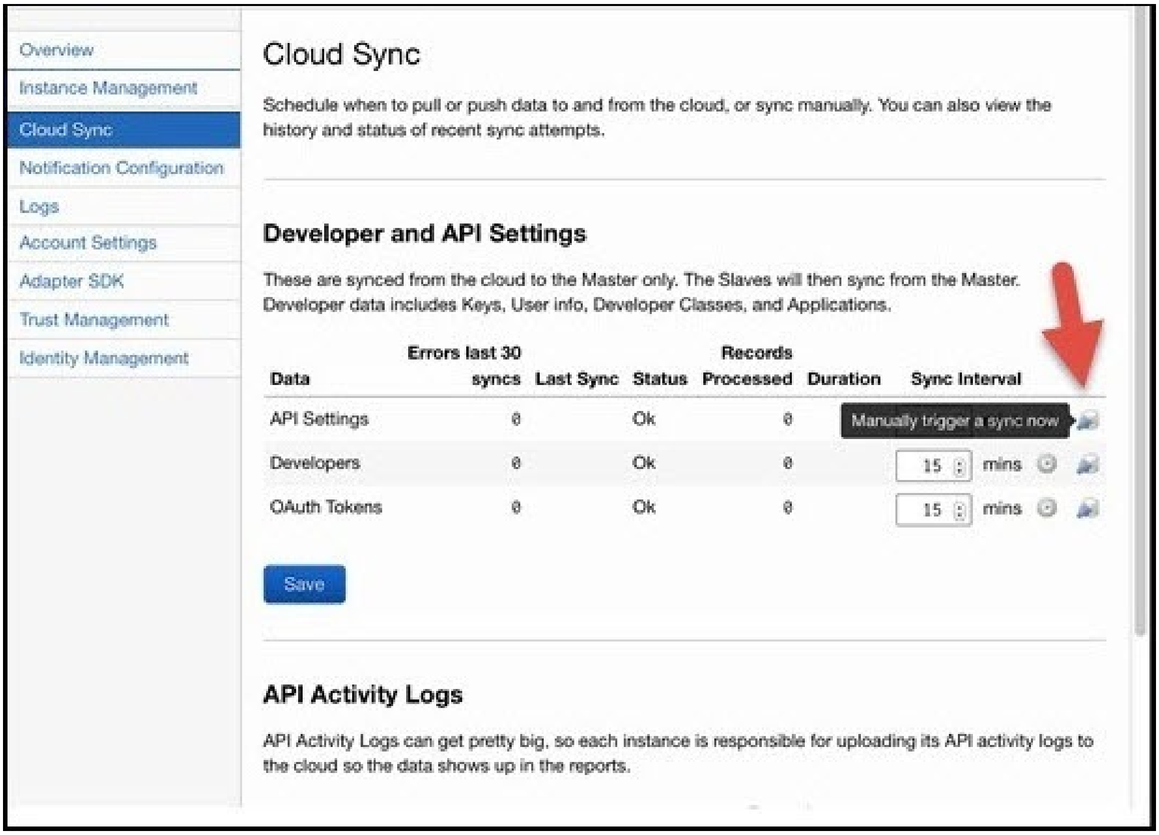
Identity is in normal state.
Action is suggested on identity.
Action is required on identity.

Upload key and certificate file in PKCS#12 format.

Name	Serial Number	Expiration Date
acme.client.example.com	5749e82c	2116-05-28 18:50:07

State: Identity manifest will be synchronized with TIBCO Mashery SaaS.
0 profile is using this identity.
0 endpoint is using this identity.
Action none.

11. To manually trigger a sync, on the **Mashery Cluster Manager** tab, click **Cloud Sync**.



Cloud Sync

Schedule when to pull or push data to and from the cloud, or sync manually. You can also view the history and status of recent sync attempts.

Developer and API Settings

These are synced from the cloud to the Master only. The Slaves will then sync from the Master. Developer data includes Keys, User info, Developer Classes, and Applications.

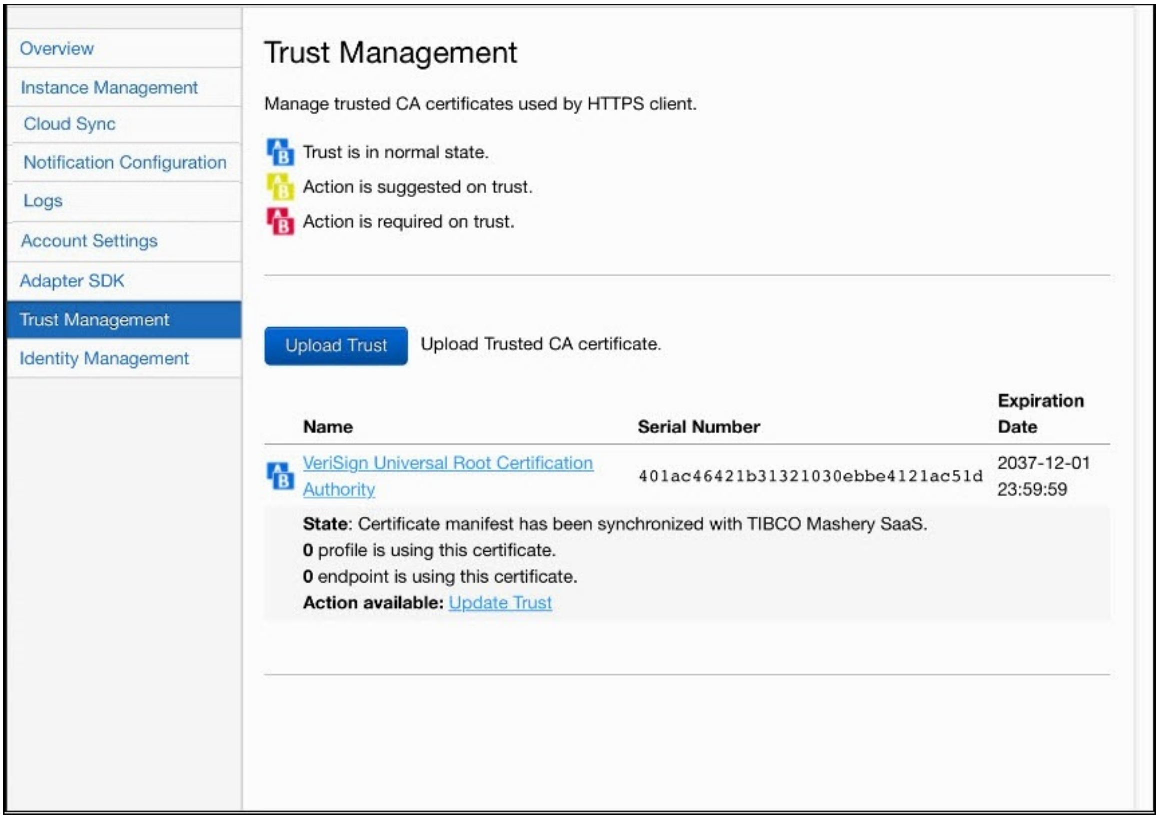
Data	Errors last 30 syncs	Last Sync	Status	Records Processed	Duration	Sync Interval
API Settings	0		Ok	0		
Developers	0		Ok	0		15 mins
OAuth Tokens	0		Ok	0		15 mins

API Activity Logs

API Activity Logs can get pretty big, so each instance is responsible for uploading its API activity logs to the cloud so the data shows up in the reports.

12. In the **Developer and API Settings** section, for API Settings, click the **Sync** button to manually trigger a sync. This will make the certificate and identity metadata available instantly in Mashery SaaS; otherwise, the sync will occur according to the minutes defined for the **Sync Interval** setting.

13. On the **Trust Management** tab, after the certificate becomes available in SaaS, the State changes to `Certificate manifest has been synchronized with TIBCO Mashery SaaS`.



Trust Management

Manage trusted CA certificates used by HTTPS client.

Trust is in normal state.

Action is suggested on trust.

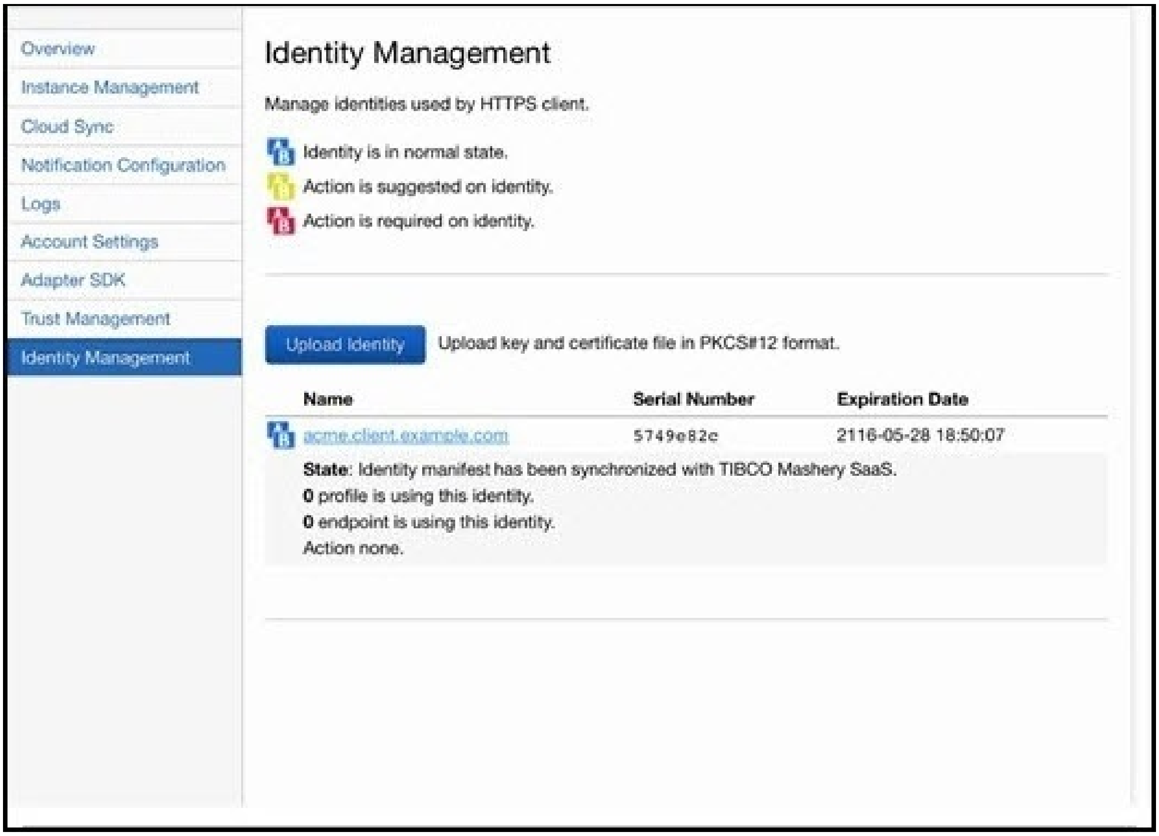
Action is required on trust.

Upload Trusted CA certificate.

Name	Serial Number	Expiration Date
VeriSign Universal Root Certification Authority	401ac46421b31321030ebbe4121ac51d	2037-12-01 23:59:59

State: Certificate manifest has been synchronized with TIBCO Mashery SaaS.
0 profile is using this certificate.
0 endpoint is using this certificate.
Action available: Update Trust

14. On the **Identity Management** tab, after the Identity becomes available in SaaS, the State changes to: `Identity manifest has been synchronized with TIBCO Mashery SaaS`.



Identity Management

Manage identities used by HTTPS client.

Identity is in normal state.

Action is suggested on identity.

Action is required on identity.

Upload key and certificate file in PKCS#12 format.

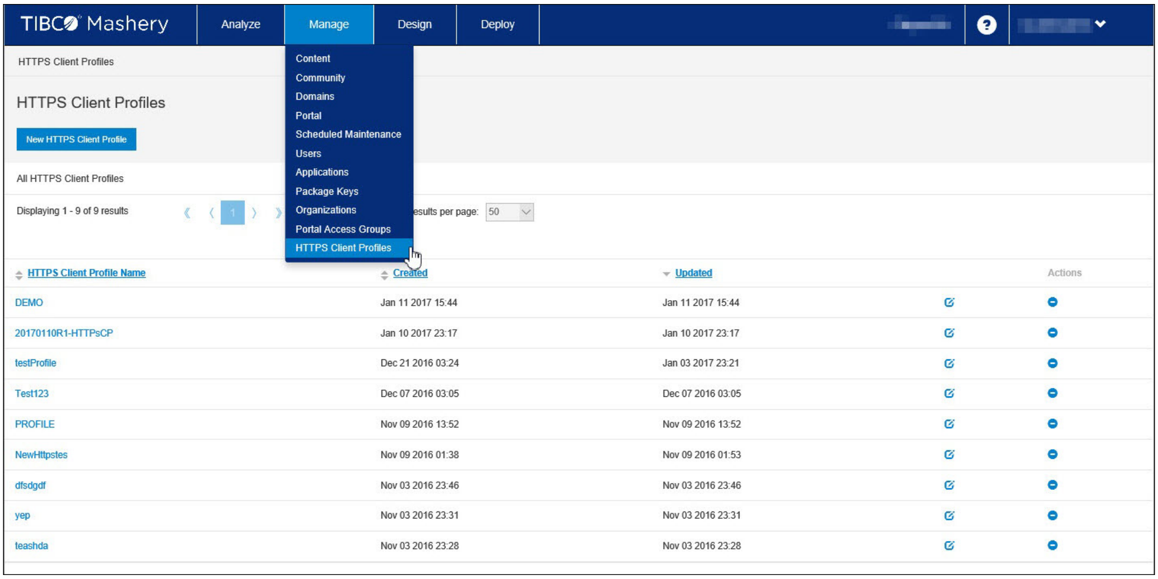
Name	Serial Number	Expiration Date
acme.client.example.com	5749e82c	2116-05-28 18:50:07

State: Identity manifest has been synchronized with TIBCO Mashery SaaS.
0 profile is using this identity.
0 endpoint is using this identity.
Action none.

What to do next

To create an HTTPS Client Profile in TIBCO Mashery SaaS, follow the steps below:

1. Click `Manage > HTTPS Client Profiles`. The **HTTPS Client Profiles** window is displayed.

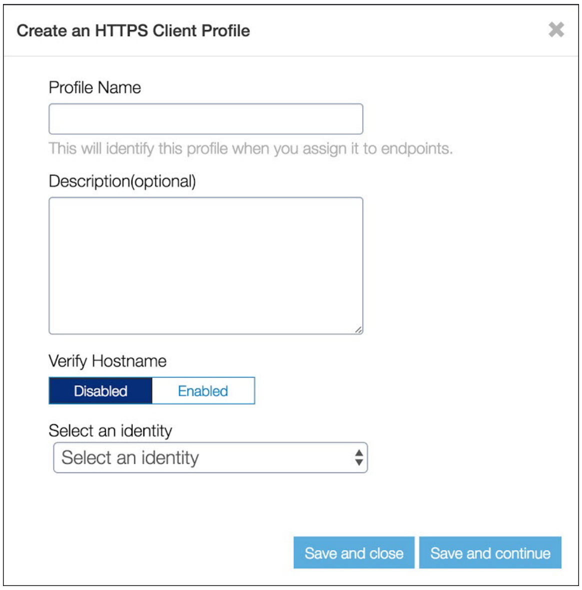


HTTPS Client Profiles

Displaying 1 - 9 of 9 results

HTTPS Client Profile Name	Created	Updated	Actions	
DEMO	Jan 11 2017 15:44	Jan 11 2017 15:44	✎	⊖
20170110R1-HTTPsCP	Jan 10 2017 23:17	Jan 10 2017 23:17	✎	⊖
testProfile	Dec 21 2016 03:24	Jan 03 2017 23:21	✎	⊖
Test123	Dec 07 2016 03:05	Dec 07 2016 03:05	✎	⊖
PROFILE	Nov 09 2016 13:52	Nov 09 2016 13:52	✎	⊖
NewHttpstes	Nov 09 2016 01:38	Nov 09 2016 01:53	✎	⊖
dfsdgdf	Nov 03 2016 23:46	Nov 03 2016 23:46	✎	⊖
yep	Nov 03 2016 23:31	Nov 03 2016 23:31	✎	⊖
teashda	Nov 03 2016 23:28	Nov 03 2016 23:28	✎	⊖

2. Click the **New HTTPS Client Profile** button. The **Create an HTTPS Client Profile** window is displayed.



Create an HTTPS Client Profile

Profile Name

This will identify this profile when you assign it to endpoints.

Description(optional)

Verify Hostname

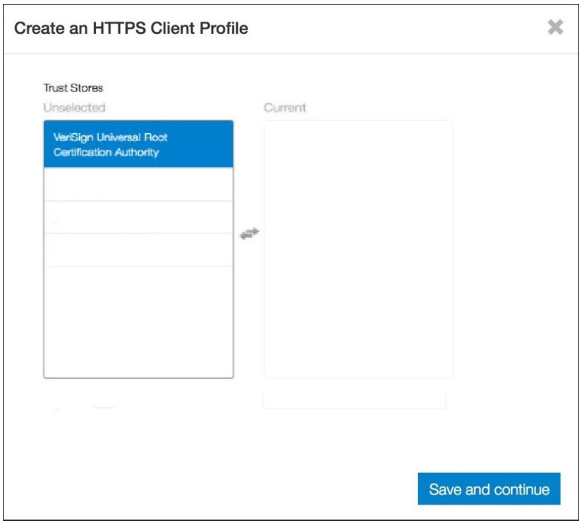
Select an identity

Select an identity

3. On the **Create an HTTPS Client Profiles** window, enter information in the following fields:

Field	Description
Profile name	Enter a name for the HTTPS client profile.
Description	(Optional) Enter a description for the HTTPS client profile.
Verify Hostname	Select one of the following: • **Disabled**: Click to not have the hostname verified. • **Enabled**: Click to have the hostname verified.
Select an identity	Select an identity for the HTTPS client profile.

4. Click **Save and Continue**.
5. A second **Create an HTTPS Client Profile** page displays.



Create an HTTPS Client Profile

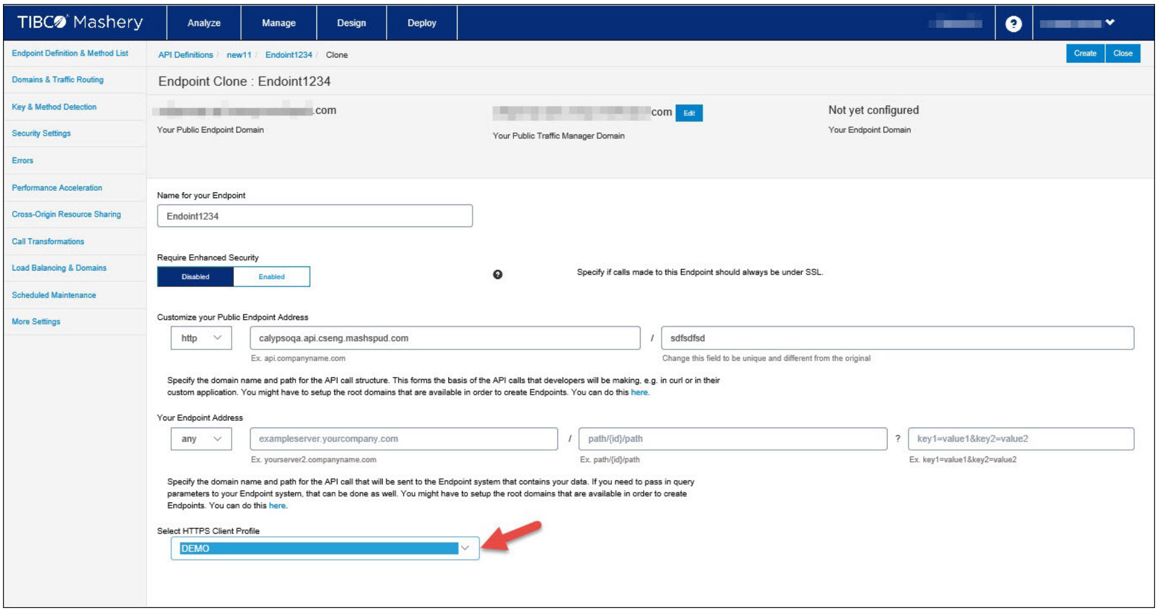
Trust Stores

Unselected

VeriSign Universal Root Certification Authority

Current

6. In the **Unselected** list of the **Trust stores** section, click the Trusted CA Certificate you uploaded in Mashery Cluster Manager to move it to the **Current** list, then click **Save and continue** to finish creating the HTTPS Client Profile. Once the HTTPS Client Profile is created, you can then select the profile when creating an endpoint on the **Endpoint Create: New Endpoint Definition** page.
7. To assign the HTTPS Client Profile to an endpoint, click `Design > API Definitions > Domains & Traffic Routing`.
8. On the **Domains & Traffic Routing** page for the existing endpoint of your API definition (or, **Endpoint Create: New Endpoint Definition** page for a new endpoint), use the **Select HTTP Client Profile** field to select the HTTPS Client Profile you just created, then click **Save** (or **Create**). The endpoint is now associated with the HTTPS Client Profile.



Endpoint Clone : Endoint1234

[redacted].com
Your Public Endpoint Domain

[redacted]com
Your Public Traffic Manager Domain

Not yet configured
Your Endpoint Domain

Name for your Endpoint: Endoint1234

Require Enhanced Security: Disabled

Specify if calls made to this Endpoint should always be under SSL.

Customize your Public Endpoint Address: http calypsoqa.api.cseng.mashspud.com / sdfsdfsd

Ex. api.companyname.com

Change this field to be unique and different from the original

Specify the domain name and path for the API call structure. This forms the basis of the API calls that developers will be making, e.g. in curl or in their custom application. You might have to setup the root domains that are available in order to create Endpoints. You can do this here.

Your Endpoint Address: any exampleserver.yourcompany.com / path/{id}/path ? key1=value1&key2=value2

Ex. yourserver2.companyname.com

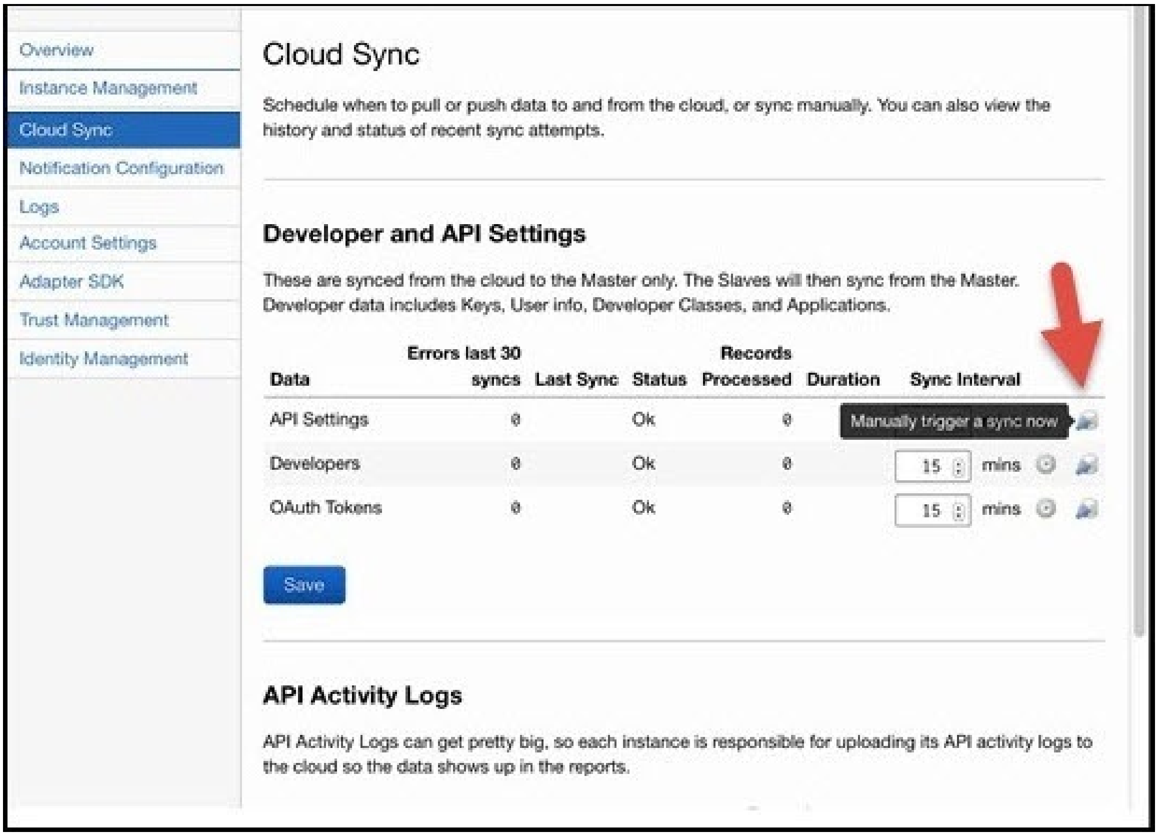
Ex. path/{id}/path

Ex. key1=value1&key2=value2

Specify the domain name and path for the API call that will be sent to the Endpoint system that contains your data. If you need to pass in query parameters to your Endpoint system, that can be done as well. You might have to setup the root domains that are available in order to create Endpoints. You can do this here.

Select HTTPS Client Profile: DEMO

9. Log back into Mashery Cluster Manager, go to the **Cloud Sync** tab, and click the manual sync button. This syncs the HTTPS Client Profile and Endpoint configuration updates to Mashery Local, and the HTTP Client Profile is now in use for the customer.



Cloud Sync

Schedule when to pull or push data to and from the cloud, or sync manually. You can also view the history and status of recent sync attempts.

Developer and API Settings

These are synced from the cloud to the Master only. The Slaves will then sync from the Master. Developer data includes Keys, User info, Developer Classes, and Applications.

Data	Errors last 30 syncs	Last Sync	Status	Records Processed	Duration	Sync Interval
API Settings	0		Ok	0		
Developers	0		Ok	0		15 mins
OAuth Tokens	0		Ok	0		15 mins

API Activity Logs

API Activity Logs can get pretty big, so each instance is responsible for uploading its API activity logs to the cloud so the data shows up in the reports.

Enabling Java SSL Debug Logging

To enable Java SSL debug logging for Mashery Local, follow the steps below:

Procedure

1. Add the following setting in "/opt/javaproxy/proxy/proxy.ini" (root privilege is needed via TIBCO Support):

   ```
   -Djavax.net.debug=all
   ```

2. Restart javaproxy (administrator):

   ```
    sudo service javaproxy restart
   ```

3. Send requests to Mashery Local, watch log in "/var/log/javaproxy-runtime.log". For example:

   ```
   tail -f /var/log/javaproxy-runtime.log
   ```