Fusion Server 4.2 System Administration Guide

2019-07-26
Table of Contents

Starting and Stopping ................................................................. 1
  Command summary ................................................................. 2
  Define groups of services ....................................................... 3
  Unix ...................................................................................... 4
    Start Fusion ....................................................................... 4
      Start required services ................................................... 4
      Start a group of services .................................................. 4
      Start services individually ............................................... 4
      Run Fusion in the foreground ............................................. 5
      Run Fusion in shell mode ................................................... 5
    Stop Fusion ....................................................................... 6
    Using systemd to manage processes ...................................... 6
    Launching Fusion at system start ....................................... 6
    Starting and stopping Fusion ............................................. 6
    Using Ubuntu Upstart to manage processes .......................... 7
  Windows ........................................................................... 8
    Start Fusion ....................................................................... 8
      Start required services ................................................... 8
      Start services individually ............................................... 8
      Run Fusion in the foreground ............................................. 9
      Stop Fusion ................................................................. 9
      Run Fusion with a service account ................................... 9
    Access Fusion after startup ............................................. 12
  Licensing ........................................................................ 13
    Uploading a license using the UI ....................................... 14
    System requirements ....................................................... 17
    Cluster dashboard ......................................................... 18
    Hosts dashboard ............................................................ 19
    Services dashboard ....................................................... 21
    Datasources dashboard .................................................... 22
    The Log Viewer .............................................................. 23
      Service logs ............................................................... 23
      Request logs .............................................................. 23
    Exporting data from the DevOps Center ............................... 24
    Troubleshooting display problems in the DevOps Center ........ 25
  Information Collected by the Usage Monitor ......................... 26
  How Data is Sent ............................................................... 27
  How to Opt Out ............................................................... 28
  System Dashboard ............................................................. 29
    System Overview ........................................................... 29
    API Insight ................................................................. 29
    Configurations .............................................................. 29
## SSL Security (Unix)

### Required software

- Preliminary steps
- Generate SSL certificate files

### Overview of procedure

1. **Load an SSL certificate into a Fusion keystore**
   - Alternative 1: Self-signed certificate
   - Alternative 2: CA-signed certificate
     - Preliminary steps
     - Generate SSL certificate files
     - Convert the certificate chain and private key files to a PKCS #12 certificate
     - Create the Fusion Proxy service keystore and import the PKCS #12 certificate

2. **Enable HTTPS in the Fusion Proxy service**

3. **Restart Fusion and test access through HTTPS**

4. **Disable HTTP access to the Fusion Proxy service**
   - Alternative 1: Disable HTTP access on the firewall or load balancer
   - Alternative 2: Disable listening for HTTP requests in the Fusion Proxy service

5. **Ensure that the Fusion Agent can do heartbeats against the Proxy service**

### References and tutorials

- Solr Clusters
- Messaging Services
- Threads
- Metrics
- History
- Heartbeat Data
- Other System Menu Items
- Tasks
4. Configure panels ................................................................. 187
5. Configure input panel settings ............................................. 188
6. Configure specific types of panels ..................................... 189
7. Save a dashboard .............................................................. 190
8. Test the dashboard ............................................................ 191

Use Dashboards ...................................................................... 192
  Use built-in dashboards ....................................................... 192
    Service Logs dashboard .................................................... 192
    Access Logs dashboard .................................................... 192
    Combined Logs dashboard ............................................... 193
    Blank dashboard ............................................................ 193
    Fusion Metrics dashboard ............................................... 193
    Search Analytics dashboard ............................................. 194
    Fusion Signals dashboard ................................................ 195
    Default Non-Time-Series dashboard ............................... 195
    Default Time-Series dashboard ...................................... 196
Open a dashboard ............................................................... 196
  Interact with data ............................................................. 197
  Interact with specific types of panels ................................. 198
    Table panels ............................................................... 198
    Full Text Search panels ................................................. 198
    Histogram and Range Facet panels ................................. 198
    Filtering panels .......................................................... 199
    Heat Map panels .......................................................... 199
Filter by Fusion request ID ................................................. 199
  Click a link in the error message or toast ........................... 200
  Filter in a logs dashboard ................................................ 202
  Append a query filter parameter to the dashboard URL ....... 204
Open the Service Logs dashboard from a job history ........... 205
Manage Dashboards ............................................................ 207
Set Permissions ................................................................... 207
  Permissions to Manage Dashboards .................................. 207
  Permissions to Use Dashboards ....................................... 207
  Manage Dashboards ......................................................... 207
Input Panels ......................................................................... 209
Query Syntax ....................................................................... 211
  Rules for the Simplest Cases ......................................... 211
  Examples ........................................................................ 211
Inspect a Panel Query .......................................................... 212
Display Panels ..................................................................... 214
  What Data is Displayed .................................................... 214
  Layout Panels ................................................................. 214
  Textual Information Panels ............................................. 214
  Graphical Visualization Panels ....................................... 215
Starting and Stopping

This topic explains how to start and stop Fusion Server and its services using the scripts in the `bin` directory below the Fusion home directory:

- `/opt/fusion/4.2.x/bin` (Unix)
- `C:\lucidworks\fusion\4.2.x\bin` (Windows)
Command summary

You can control all Fusion services at once under the management of the Fusion agent, or you can control services individually.

To control all services using the Fusion agent:

- **Unix:** /opt/fusion/4.2.x/bin/fusion <command>
- **Windows:** C:\lucidworks\fusion\4.2.x\bin\fusion.cmd <command>

To control individual services:

- **Unix:** /opt/fusion/4.2.x/bin/<servicename> <command>
  
  For example: /opt/fusion/4.2.x/bin/proxy restart

- **Windows:** C:\lucidworks\fusion\4.2.x\bin\<servicename>.cmd <command>
  
  For example: C:\lucidworks\fusion\4.2.x\bin\proxy.cmd restart

<table>
<thead>
<tr>
<th>Tip</th>
<th>When starting services individually, start Zookeeper first.</th>
</tr>
</thead>
</table>

The commands below can be issued to the fusion/fusion.cmd script to issue the command to all services in the correct sequence, or they can be issued to an individual service.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>Start one or all Fusion services.</td>
</tr>
<tr>
<td>status</td>
<td>Display the status of one or all Fusion services.</td>
</tr>
<tr>
<td>restart</td>
<td>Restart one or all Fusion services.</td>
</tr>
<tr>
<td>stop</td>
<td>Stop one or all Fusion services.</td>
</tr>
<tr>
<td>run</td>
<td>Start one or all Fusion services in the foreground.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>run-in-shell (Unix only)</td>
<td>Start an individual service using Bash’s exec function, which allows the service to assume the shell process’s PID. See Run Fusion in shell mode below.</td>
</tr>
</tbody>
</table>
Define groups of services

The `fusion.properties` file includes the property definition `group.default`. This property defines the Fusion services to start and stop by default (if no property is named in the start or stop command).

The default list of services out-of-the-box is also the *minimum* set of services, with the exception of the `log-shipper` service, which you can remove if you don’t use it.

Here is the `group.default` definition in `fusion.properties`:

```
group.default = zookeeper, solr, api, connectors-classic, connectors-rpc, proxy, webapps, admin-ui, log-shipper
```

With the exception of the `log-shipper` service, these are all required services. Even if only using RPC connectors, the `connectors-classic` service is required. The `log-shipper` service is required to use the Log Shipper.

To modify the default list of services

Edit the `group.default` property, for example, to include Spark and SQL related services:

```
group.default = zookeeper, solr, api, connectors-classic, connectors-rpc, proxy, webapps, admin-ui, log-shipper, spark-master, spark-worker, sql
```

To define other lists of services (Unix)

You can define other lists of services by defining other `group` properties. For example, define this group property to start and stop services for Spark and SQL together:

```
group.spark-only = spark-master, spark-worker, sql
```

Define this group property to start and stop services for classic and RPC connectors together:

```
group.connectors = connectors-classic, connectors-rpc
```
Unix

Start and stop Fusion on Unix.

Start Fusion

All Fusion start scripts must be executed by a user who has permissions to read and write to the directories where Fusion is installed. These scripts don’t need to be run as root (or sudo), nor should they be. Use a suitable user, or create a new one, and then ensure that it owns the directory where Fusion resides, (for example, $C:\lucidworks$).

Give the commands that follow from the directory $fusion/4.2.x/bin$.

Start required services

Start the required services that are defined in the $group.default$ property.

To start all required services

`./fusion start`

Tip

This is equivalent to `./fusion start default`. You can omit the group name $default$.

Start a group of services

You can start a group of services together. Reference the property in $fusion.properties$ that defines the group.

Examples of when this is useful are:

- **Spark and SQL** – The `$spark-master`, `$spark-worker`, and `$sql` services are interdependent and should be started and stopped together.

  `./fusion start spark-master spark-worker sql`

- **Classic and RPC connectors** – RPC connectors require both the `$connectors-classic` and `$connectors-rpc` services to be running.

  `./fusion start connectors-classic connectors-rpc`

Start services individually

You can start services individually.

To start services individually

- Fusion UI service:

  `./admin-ui start`
• API services:
  ./api start

• Classic Connectors services:
  .connectors-classic start

• RPC Connectors services:
  .connectors-rpc start

• Log shipper service:
  .log-shipper start

• Proxy:
  .proxy start

• Solr:
  .solr start

• Spark Master:
  .spark-master start

• Spark Worker:
  .spark-worker start

• SQL service:
  .sql start

• Web Apps:
  .webapps start

• ZooKeeper:
  .zookeeper start

For information about default ports, see Default Ports.

Run Fusion in the foreground

To run Fusion or any of its services in the foreground, use the run command-line argument in place of start.

Run Fusion in shell mode

To start any of Fusion's services using Bash's exec function, which allows the service to assume the shell process's PID, use the run-in-shell command-line argument in place of start or run. The run-in-shell argument can only be used to start one service at a time.

Examples
Shell mode is particularly useful in containerized environments, which generally assume that only one process runs per container and that process is "process 0", that is, the initial process invoked within the container, not a separate spawned process.

**Stop Fusion**

To stop Fusion services

To stop Fusion or any of its services, use the command above with the `stop` command-line argument in place of `start`, for example:

```bash
./solr stop
```

**Using systemd to manage processes**

On Red Hat Enterprise Linux, CentOS 7 and newer, and Ubuntu 15.04 LTS and newer, we support using the operating system-provided `systemd` for process management.

**Launching Fusion at system start**

You can configure `systemd` to launch Fusion when your system starts.

To launch Fusion at system start:

1. Change your working directory to Fusion's `systemd` directory, for example:

   ```bash
   cd /opt/fusion/4.2.x/init/systemd
   ```

2. Edit `fusion.service` to provide correct values for the `FUSION_HOME` and `JAVA_HOME` environment variables.
3. Stop Fusion if it is already running:

   ```bash
   /opt/fusion/4.2.x/bin/fusion stop
   ```

4. Create the `systemd` management file, which launches Fusion under `systemd` management:

   ```bash
   sudo bash install.sh
   ```

**Starting and stopping Fusion**

You can use the `systemctl` command to start and stop Fusion:
sudo systemctl start fusion
sudo systemctl stop fusion

Log files for Fusion services are found in directories under fusion/4.2.x/var/log.

Using Ubuntu Upstart to manage processes

Under Ubuntu 12.04 LTS through Ubuntu 14.10, we support using Upstart for process management. This requires Fusion to be installed in the /opt/lucidworks/ directory.

To configure upstart, run the following commands:

```
$ cd /opt/lucidworks/fusion/{version}/init/upstart
$ sudo bash install.sh
```

If this complains with no JAVA_HOME set, replace sudo with sudo -E. Then you can use the service command to control the server:

```
$ sudo service fusion-solr start
$ sudo service fusion-api start
$ sudo service fusion-connectors start
$ sudo service fusion-ui start
```

and similarly use stop and status.

Log files for Fusion services are found in directories under fusion/4.2.x/var/log.
Windows

Start and stop Fusion on Windows.

Start Fusion

All Fusion start scripts must be executed by a user who has permissions to read and write to the directories where Fusion is installed. Ensure that the user owns the directory where Fusion resides (for example, C:\lucidworks).

Give the commands that follow from the directory fusion4.2.x\bin.

Start required services

To start all required Fusion services as Java processes

```bash
fusion.cmd start
```

To start all required Fusion services as Windows services

```bash
start-services.cmd
```

Start services individually

To start specific services as Java processes

- UI service:
  ```bash
  admin-ui.cmd start
  ```

- API services:
  ```bash
  api.cmd start
  ```

- Classic Connectors services:
  ```bash
  connectors-classic.cmd start
  ```

- RPC Connectors services:
  ```bash
  connectors-rpc.cmd start
  ```

- Log shipper service:
  ```bash
  log-shipper.cmd start
  ```

- Proxy:
  ```bash
  proxy.cmd start
  ```

- Solr:
  ```bash
  solr.cmd start
  ```
• Spark Master:
  spark-master.cmd start

• Spark Worker:
  spark-worker.cmd start

• SQL service:
  sql.cmd start

• Web Apps:
  webapps.cmd start

• ZooKeeper:
  zookeeper.cmd start

For information about default ports, see Default Ports.

**Run Fusion in the foreground**

To run Fusion or any of its services in the foreground, use the `run` command-line argument in place of `start`, for example:

connectors.cmd run

**Stop Fusion**

To stop all Fusion services

- `fusion.cmd stop` (Stop all Fusion services, if they are running as Java processes)
- `stop-services.cmd` (Stop all Fusion services, if they are running as Windows services)

To stop a specific service running as a Java process

To stop a specific Fusion service that is running as a Java process, use the command above with the `stop` command-line argument in place of `start`, for example:

connectors.cmd stop

**Run Fusion with a service account**

This example assumes the following:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Account</td>
<td>FUSION_SVC</td>
</tr>
<tr>
<td>Domain</td>
<td>qe</td>
</tr>
<tr>
<td>Installation directory</td>
<td>C:\fusion&lt;version&gt;</td>
</tr>
</tbody>
</table>
1. As an administrator, create the service account, and install it to the server you want to use for Fusion:

```powershell
C:\Users\Administrator> New-ADServiceAccount -Name "FUSION_SVC" -RestrictToSingleComputer
C:\Users\Administrator> Add-ADComputerServiceAccount -Identity EC2AMAZ-79FD9JL -ServiceAccount "FUSION_SVC"
C:\Users\Administrator> Install-ADServiceAccount -Identity "FUSION_SVC"
C:\Users\Administrator> Test-ADServiceAccount "FUSION_SVC"
```

2. Run `install-services.cmd` as a local administrator:

```powershell
C:\Users\Administrator> C:\fusion\4.2.2\bin\install-services.cmd
ECHO is off.
Thank you for choosing
```

```
```

```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
| Note | Only the account name is required. The password is managed by Windows. |
Access Fusion after startup

After Fusion services have started, you can open the Fusion UI in a web browser at http://localhost:8764/ (replace localhost with your server name or IP address if necessary).

The first time you access Fusion, you must set the password for the user admin and agree to the Fusion Licensing Agreement (which contains terms of service). This is followed by an optional registration step. After this, Fusion displays the Fusion launcher (the page from which you can open apps).

If you can't access the system, see the Troubleshooting topic. Checking System State shows how to inspect Fusion services.
Licensing

Fusion Server requires a valid license. Depending on the details of your contract, your license may also enable optional connectors or Fusion AI.

When you download Fusion Server, it comes with a 30-day trial license. Contact Lucidworks to obtain a permanent license.

Fusion Server provides a license management UI and a license API for installing and managing licenses. When you upload a license, Fusion Server stores it in ZooKeeper, so you only need to upload it to one node per cluster.

The Fusion UI notifies you when your trial license is about to expire. When your license has expired, Fusion Server accepts no configuration changes until you upload a valid license.
Uploading a license using the UI

1. Log in to the Fusion UI.
2. In the upper right, open the profile menu and select **License Details**:

The License Details window appears:

- **Environment**: non-production
- **Fusion Connectors**: *
- **Fusion Server**: *
- **Fusion AI**: *
- **Organisation**: Lucidworks
- **User**: license@lucidworks.com

License is valid until Jan 31, 2018

**Upload license**

Choose License: No file chosen

Upload
3. Click **Choose License** and select your license file.

4. Click **Upload**. = The DevOps Center

The DevOps Center is a suite of tools for monitoring, troubleshooting, and incident investigation. It consists of a set of dashboards and an interactive log viewer, providing views into this Fusion cluster's hosts and services using metrics and events.

- Metrics come from the `system_monitor` collection.
  
  See System Metrics for details about this collection.

- Events come from the `system_logs` collection.

You can export all metrics and events for any selected time period.

In any Fusion app, you can open the DevOps Center by navigating to **System > DevOps Center**.
Tip

In the DevOps Center, dates and times are local, as determined by your browser. In the Log Viewer, you can toggle between local time and UTC.
System requirements

The DevOps Center is included as part of Fusion Server 4.2 and above. It requires the following:

- Your Web browser must support HTML5.
- Every node that you want to monitor must be running Fusion Server version 4.2.0 or later.
- Every node that you want to monitor must be running the agent and log-shipper services.

The DevOps Center is enabled by Fusion’s default configuration. If you find that the DevOps Center is not correctly populated with data, see the Troubleshooting section below.
Cluster dashboard

The first screen that displays is a high-level dashboard showing indicators of general health of your Fusion cluster.

The Cluster dashboard includes this data:

- Fusion license information
- The number of hosts, collections, and Solr clusters in this cluster
- The cluster’s service status
- Average and max values during the selected time interval for the following metrics:
  - Query rate
  - Response time
  - Index size
  - Indexing rate
  - Active sessions
Hosts dashboard

Data about individual hosts is available in the Hosts dashboard. The initial view includes all hosts.

From here you can filter the set of hosts displayed:

- Select one or more status codes to display only the hosts with those statuses.
- Select one or more services to display only the hosts on which they are running.

Click on any host’s IP address to drill down to additional visualizations about that host:

In this single-host view, you can see the following:
Host details

- Cores
- IP address
- Operating system
- Uptime
- CPU usage
- CPU load
- Memory total
- Memory free
- Disk total
- Disk free
- ulimit files

Host graphs and tables

- CPU utilization
- CPU load
- Per-service CPU utilization
- Time spent in garbage collection (GC)
- Free memory
- Free disk space
- Overview of services running on the host
Services dashboard

The Services dashboard displays the status of each of the Fusion services: "OK" for services that are running and "Bad" for services that are unreachable. Metrics are also displayed for each service, and these vary depending on the service. A service is marked "Bad" if it does not return metrics after several expected reporting intervals, that is, after a few minutes.
Datasources dashboard

The Datasources dashboard displays the status, number of fetched documents, number of failed documents, and number of indexed documents for each Fusion datasource. By default, the dashboard displays only the datasources for the currently-selected app. Select one or more apps on the left to view all of the datasources in those apps.
The Log Viewer dashboard displays service logs and request logs from Fusion’s `system_logs` collection.

By default, timestamps are displayed in local time, as determined by your browser. To view timestamps in UTC (unless `default.timezone` is set to another time zone in `fusion.properties`), set `LOCAL TIMEZONE` to “Off”.

**Auto-Refresh** is off by default, to display static log data. To display real-time logs, set this to “On”.

### Service logs

Service log files are written to the filesystem by each running service, such as `var/log/api/api.log`, `var/log/proxy/proxy.log`, and so on. Fusion indexes them in the `system_logs` collection with `type=java`.

In the DevOps Center, you can filter service logs by:

- Service
- Log level
- Host

### Request logs

HTTP request log files are written to the filesystem by Jetty, at `var/log/proxy/jetty_request_<date>.log`. Fusion indexes them in the `system_logs` collection with `type=http`.

In the DevOps Center, you can filter request logs by:

- HTTP status code
- HTTP method
- Host
Exporting data from the DevOps Center

The DevOps Center can export metrics and events in CSV format for the currently-selected time period. Exported metrics and events are useful for collaborating with other Fusion administrators, using third-party analysis tools, or sending troubleshooting data to the Lucidworks support team.

How to export data from the DevOps Center

1. In the Fusion UI, navigate to System > DevOps Center > Log Viewer.
2. Select the time period that is relevant to the incident you are interested in.

Log files will be exported according to the tab you are viewing.

Click Export service logs in the Service Logs tab or Export request logs in the Request Logs tab to download a .csv file containing the logs (shown here with example timestamps):

Troubleshooting display problems in the DevOps Center

The DevOps Center is enabled by Fusion's default configuration. If you find that the DevOps Center is not correctly populated with data, check the following:

• Check for browser extensions that block HTML5 autoplay.
• Check that the Fusion log-shipper service is enabled, as it is by default:

```
fusion/4.2.x/bin/log-shipper status
```

If it is not enabled, make sure it is included in the list of default services in `fusion.properties`, as in this example:

```
group.default = zookeeper, solr, api, connectors-classic, connectors-rpc, proxy, webapps, admin-ui, sql, log-shipper
```

See Configure Fusion logging for additional details.

| Tip | If you are running some services on separate nodes, make sure those nodes are also running Fusion's agent and log-shipper. |

• In `fusion.properties`, check that `collectMetrics` is "true" and `collectMetricsIntervalSecs` is set to a reasonable interval:

```
# capture host and services metrics
# and ship them to system_monitor Solr collection
default.collectMetrics = true
default.collectMetricsIntervalSecs = 30
```

= System Usage Monitor

The System Usage Monitor is a voluntary program to allow users to anonymously send metrics about their system to Lucidworks. We use this information to analyze the types of systems in use by our customers and how they are used so we can improve our product.

At no point does the system collect information that could identify you, your organization, or the specific documents indexed. Only minimal data is sent about the type of content indexed. Our website has more information about our privacy policy.
Information Collected by the Usage Monitor

The System Usage Monitor collects the following types of information:

• The uuid, a randomly generated identifier per Fusion cluster.
• System information, including the operating system, version, and Java version.
• Fusion information, including the version, number of Fusion nodes, and number of CPU cores in use.
• Solr statistics including the number of collections, number, types, and indexing rates of documents, and number, type, and rate of search requests.
• Aggregation information, including number, types, and rates of aggregation runs, and number, types, and rates of signals requests.
• Recommendations information, including number, types, and rates of recommendation requests.

You can see the data that will be sent to Lucidworks with the Usage API and also in the UI by navigating to System > System, then clicking the Heartbeat Data tab. The UI and the REST API report only the data currently scheduled to be sent, so they are not a complete picture of all data collected.
How Data is Sent

Data is sent to Lucidworks once per week, and also whenever Fusion is restarted.

When Fusion is started, the System Usage Monitor transmits data about your system to a server hosted by Lucidworks with two HTTP requests. The first request contains system-level information. If the first request is successful, the second request sends system-specific information.

The information is sent via an encrypted POST request to https://heartbeat.lucidworks.io. Each request includes a unique identifier, which is anonymous and cannot be used to identify the sender. The IP that sent the request is not stored with the request.
How to Opt Out

By default, the usage monitor is enabled in your system. If you would like to opt out of sending this data to Lucidworks, you can disable the usage monitor. There are two ways to enable or disable the usage monitor:

- Go to the Heartbeat Data page in the Fusion UI (System > System > Heartbeat Data), and deselect the "Report Heartbeat" option.
- Use the Configurations API and send a PUT request as follows:

  curl -u user:pass -H 'Content-type: application/json' -X PUT -d "false"
  http://localhost:8764/api/configurations/usageMonitor

= System Administration with the Fusion UI

The Fusion UI provides information for system administration and lets you configure system components.

The REST API provides equivalent functionality. See System Admin APIs.
System Dashboard

To access the system dashboard, click [System > System]. You need to be logged in as admin to access the full feature set.

System Overview

The System Overview tab shows an overview of the collections and their states in the cluster. Click a specific collection to view the core name, collection name, Solr memory, system memory, and swap space usage.

API Insight

The API Insight tab shows the available REST API endpoints in the system.

Configurations

The Configurations tab provides read access to all of the global property settings of the system. The UI allows you to filter the list of configuration items by terms and choose the node of the cluster to examine. If you need to change any configuration item, you should use the Configurations API.

History

The History tab shows the start and stop times of all services.

Heartbeat Data

The Heartbeat Data tab shows you the information that would be sent to Lucidworks as part of the System Usage Monitor, an optional program to report information about your system. You can enable or disable sending the data in this screen, by checking the box next to Report Heartbeat.

Metrics

The Metrics tab provides access to the metrics indexed in the system_metrics collection, if enabled.

| Note |
| In Fusion 4.2 and above, this metrics collection implementation is deprecated in favor of the system_monitor collection and the DevOps Center. |

For details, see System Metrics.

Threads

The Threads tab shows the details of each active thread in the system. This is the same information returned by the threads endpoint of the System API.
Other System Menu Items

In addition to the System Dashboard, the System menu includes several tools for configuring and managing your Fusion deployment. Among them are the Messaging Services and Solr Clusters items.

Log viewer dashboards

Fusion includes some dashboards to view and analyze events in Fusion logs. You can also customize existing dashboards or create new ones. To access these dashboards, click System > Log Viewer. See Dashboards.

Messaging Services

The Messaging Services menu lets you configure system notifications, log messages, and alerts to external applications. Several messaging services are available. See Messaging Services.

Solr Clusters

The Solr Clusters menu provides controls for both reporting on and configuring Solr servers. When using Fusion with an existing Solr cluster, this provides a single point of access for both systems.
Jobs and Schedules

A job is a runnable Fusion object that performs a specific activity. Datasource jobs, Spark jobs, and tasks are all types of jobs. Examples of the activities include:

- Aggregating signal data
- Training a shallow neural model and projecting each document onto the resulting vector embedding space
- Loading data into a data source

A schedule defines when Fusion will perform a job.

You can schedule a job using cron notation, ISO-8601 interval definitions, or triggers that depend on the results of other jobs.

You can also run jobs on demand.

The Jobs API lets you view jobs and their run histories, configure their schedules, and control them directly.
## Job types

Each job type is a type of Fusion object that you can run or schedule to be run.

<table>
<thead>
<tr>
<th>Job Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>datasource</strong></td>
<td>A job to ingest data according to the specified datasource configuration, such as <code>datasource:movie-db</code>. Datasources are created using the Connector Datasources API or the Fusion UI. See Datasource Jobs.</td>
</tr>
<tr>
<td><strong>spark</strong></td>
<td>A Spark job to process data, such as <code>spark:dailyMetricsRollup-counters</code>. Spark jobs are created using the Spark Jobs API or the Fusion UI. See Spark Jobs.</td>
</tr>
<tr>
<td><strong>task</strong></td>
<td>A job to perform an HTTP call or log cleanup, such as <code>task:delete-old-system-logs</code>. Tasks are created using the Tasks API or the Fusion UI. See Tasks.</td>
</tr>
</tbody>
</table>
The Jobs manager

The Jobs manager, available in the Fusion workspace at Collections > Jobs, provides a simple interface for viewing and scheduling jobs. You can also create tasks and Spark jobs here.
You can't create, run, or schedule datasource jobs in the Jobs manager. You must create them on the Datasources page, in the Index Workbench, or in the Quickstart. To run a datasource job, use the scheduler.

To create a new job

1. In the Fusion workspace, navigate to < Jobs.>
2. Click Add and select the job type.

The New Job Configuration panel appears. This panel is different for each job type. See the job types above for details.

3. Configure the new job as needed.
4. Click Save.

To run a job once

1. In the Fusion workspace, navigate to < Jobs.>
2. Select the job from the job list.
3. Click Run.

4. Click Start.

To schedule a job
1. In the Fusion workspace, navigate to Scheduler.

2. Select the job from the job list.

3. Click Run.

4. Click New Schedule.

5. Select a job trigger:
   - **After Another Job Completes**
     
     Enter the job ID and job result that trigger this one.
   
   - **Cron String**
     
     Enter a Quartz cron expression, using UTC time unless default.timezone is set to another time zone in fusion.properties. See the Quartz documentation for additional details.
* Crontab Expression

Enter a start date/time, an interval, and the interval units.

Start + Interval

Enter a start date/time, an interval, and the interval units.
The Scheduler

The Scheduler, available in the Fusion workspace at System > Scheduler, provides an interface for scheduling jobs.
You can’t create jobs in the Scheduler. To create new jobs, use the Jobs manager or the REST API for each job type.
Datasource Jobs

A datasource is a configuration that manages the import and indexing of data into the collection.

Datasources are created using the Index Workbench, or at Search > Datasources, or using the Connector Datasources API.

A datasource is also a job that can run on demand or on a schedule.

Scheduling a datasource job in the Fusion UI

1. Create a datasource.
2. Navigate to System > Scheduler.
3. Select the datasource from the job list.
4. Click New Schedule.
5. Select and configure a trigger:
   - **After Another Job Completes**
     Enter the job ID and job result that trigger this one.
   - **Cron String**
     Enter a Quartz cron expression. See the Quartz documentation for details.
   - **Start + Interval**
     Enter a start date/time, an interval, and the interval units.
Tasks

Tasks are a flexible job type that can be used to clean up old logs or run any REST call.

Task subtypes

The "task" job type has the following subtypes:

- Log Cleanup
  
  Delete old log messages from system logs collection.

- REST Call
  
  A versatile job type that runs an arbitrary REST/HTTP/Solr command.

Task configuration

Tasks can be created and modified using the Fusion UI or the Tasks API. They can be scheduled using the Fusion UI or the Jobs API.

For the complete list of configuration parameters for all task subtypes, see the Jobs Configuration Reference.
Spark Jobs

Apache Spark can power a wide variety of data analysis jobs. In Fusion, Spark jobs are especially useful for generating recommendations.

Spark job subtypes

For the Spark job type, the available subtypes are listed below.

- Aggregation
  
  Define an aggregation job.

- Custom Spark job
  
  Run a custom Spark job.

- Script
  
  Run a custom Scala script as a Fusion job.

| Note | Additional Spark jobs are available with a Fusion AI license. |

Spark job configuration

Spark jobs can be created and modified using the Fusion UI or the Spark Jobs API. They can be scheduled using the Fusion UI or the Jobs API.

For the complete list of configuration parameters for all Spark job subtypes, see the Jobs Configuration Reference.
Security

Important

Lucidworks recommends not virus scanning the fusion/data folder. Virus scanning can cause slow performance, and it can cause downtime if it quarantines an index file identified as a possible virus.

Fusion uses a number of security measures:

- **Authenticating UI users** – Fusion authenticates users when they log in. Logging in creates a new Fusion session. Fusion also authenticates users when the Sessions REST API creates a session.
- **Authorizing UI users** – Fusion authorizes users to use specific parts of the Fusion UI.

  Note

  UI users must also be authorized to make API requests, because the UI makes API requests.

- **Authenticating and authorizing users who make API requests**
- **Using session cookies**
- **Using an external authentication provider** (optional) - A security realm can specify use of an external authentication provider, such as LDAP, JWT, or SAML.
- **Using SSL/TLS to ensure that data in flight between your application and Fusion is not observable** (optional) - See SSL Security (Unix) or SSL Security (Windows).
- **Constraining the documents that are indexed** (optional)

- **Trimming the documents that are returned by queries based on authorization** (optional)
Fusion user login

When logging into the Fusion UI, a user provides a username and password, as well as their assigned security realm. An administrator must specify these in Fusion (using the native security realm) or configure Fusion to use an external authentication provider (for example, LDAP or SAML). See Access control.

Fusion uses roles defined by permissions to authorize Fusion UI access and perform tasks in Fusion, including searching. The recommended method to delegate permissions is as follows:

- Assign each user to a role and create custom roles as needed.
- Assign permissions on a per-app basis.
Manage users with security realms

Fusion uses security realms to authenticate users of the Fusion UI. Each user has an assigned security realm, which the user must select when logging in. If the user selects a different realm, authentication fails.

A security realm also provides a list of roles as follows:

- The list always includes the role(s) that are specified in the security realm.

- (Optional) If an external directory service (such as LDAP) is used for authentication, the list can also contain roles that are mapped from the names of the directory-service groups. That is, you can configure a security realm to return group information for users from *that same* directory service.

- (Optional) The security realm can reference one or more Fusion roles or, when using an external directory service provider, use group membership information from the provider to determine roles for users. Fusion maps the group names to role names and adds these roles to the user’s list of roles.

| Note | Fusion does not use permissions from LDAP to authorize UI access or API requests. It only obtains group names (optionally), which are used as role names or are mapped to role names. If an Active Directory Security Query Trimming Stage is used, then directory-service permissions *are* used for trimming. If a connector supports security trimming, then connector permissions *are* used for trimming. |
Per-Request Authentication

Requests to the Fusion REST API must specify a security realm for per-request authentication, unless a session cookie is used (which contains information about the security realm).

Fusion authorizes requested operations based on API permissions specified for the user and for the user's role(s). Fusion considers the role(s) specified in the user definition and in the security realm. Fusion creates a list of roles when a session is created, that is, when a user logs in or when the Sessions REST API creates a session. Authorization based on permissions and its layering is at request time.

You can define multiple security realms for a Fusion instance. A Fusion instance can manage multiple security realms, which allows users from different domains to have (different levels of) access to specific Fusion collections.
SSL Security (Unix)

Fusion's UI (which is accessed through the Fusion Proxy service) can run over SSL for secure communication with any HTTP client, using the Java Secure Socket Extension (JSSE) framework. You configure Fusion for SSL by configuring Fusion's Proxy and UI services.
Required software

Configuring Fusion for SSL requires the following software:

- **Java Development Kit** – To store certificates, you can use the Java `keytool` Key and Certificate Management utility which is a part of the JDK.

  The JDK is also a requirement for Fusion Server.

- **OpenSSL** – You might need the `openssl` command line tool:
  - If you have the certificate chain and private key as separate files, then you can use the `openssl` command line tool to create a PKCS #12 file.
  - If you have an intermediate CA certificate, then you can use it and `openssl` to generate the certificate chain and private key files.
Overview of procedure

To configure Fusion for SSL:

1. Load an SSL certificate into a Fusion keystore.
2. Enable SSL in the Fusion Proxy service.
3. Restart Fusion and test access through HTTPS.
4. Disable HTTP access to the Fusion Proxy service.
5. Ensure that the Fusion Agent can do heartbeats against the Proxy service.
1. Load an SSL certificate into a Fusion keystore

The SSL protocol is based on public-key cryptography where encryption keys come in public key/private key pairs. An SSL certificate is used to verify the authenticity of a particular server. It contains the web site name, contact email address, company information and the public key used to encrypt the communication which is shared with the entities that communicate with the owner of the public/private key pair.

The server has a locally-protected private key that is accessible via a JSEE keystore.

The keystore maintains both the server certificate and the private key, so that when a server authenticates itself to the client, it uses the private key from its keystore for the initial SSL handshake.

Load the certificate into a Fusion keystore. Perform the tasks in the appropriate section:

- **Self-signed certificate** – If Fusion is behind a firewall, you can use a self-signed certificate for SSL communication with other hosts in your internal network. Create a keystore for the Fusion Proxy service and load the keystore with the self-signed PKCS #12 certificate.

- **Certificate signed by a certificate authority** – In a production environment, SSL certificates typically originate with certificate signing requests (CSRs) and are signed by a trusted third-party Certificate Authority (CA). Create a keystore for the Fusion Proxy service and load the keystore with the PKCS #12 certificate from a CA.

**Alternative 1: Self-signed certificate**

| Note                                      | If you are using a CSR-originated certificate from a trusted certificate authority, proceed to Alternative 2: CA-signed certificate. |

If Fusion is behind a firewall, you can use a self-signed certificate for SSL communication with other hosts in your internal network. Create a keystore for the Fusion Proxy service and load the keystore with the self-signed PKCS #12 certificate.

To store certificates, you can use the Java keytool Key and Certificate Management utility which is a part of the JDK.

To create a keystore and load a self-signed certificate

1. Set environment variables:

   ```
   export JAVA_HOME=JavaHomeDirectory
   export FUSION_HOME=FusionHomeDirectory
   ```

   For example:

2. Create the Fusion Proxy service keystore, generate the key pair and self-signed certificate, and load them into the keystore:

   ```
   "$JAVA_HOME/bin/keytool" -genkeypair -keystore "$FUSION_HOME/apps/jetty/proxy/etc/keystore" -dname "CN=CommonName, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown" -keypass KeyPassword -storepass KeystorePassword -keyalg RSA -alias selfsigned -deststoretype pkcs12 -ext SAN=dns:ServerFqdn,ip:ServerIpAddress
   ```
You must include the qualified domain name and/or the IP address of the Fusion server in the -ext SAN part of the command. Failure to do so results in SSL validation errors.

Example command:

```
```

The resulting certificate enables validated SSL transport to these hosts:

- \https://search.mycorp:ProxyPort
- https://localhost:ProxyPort

Where ProxyPort is the Fusion Proxy port.

**Alternative 2: CA-signed certificate**

If Fusion is behind a firewall and you are using a self-signed certificate, skip this section. Perform the tasks in Alternative 1: Self-signed certificate.

In a production environment, SSL certificates typically originate with certificate signing requests (CSRs) and are signed by a trusted third-party Certificate Authority (CA).

The steps here assume that you are the person who will be obtaining the SSL certificate chain and private key files. If you aren't that person, contact your system administrator.

**Preliminary steps**

1. Obtain a domain from a domain registrar.
2. Change the A record of your domain to the public IP address of your web server instance.

**Generate SSL certificate files**

Use an SSL certificate provider to generate the certificate chain and private key files, or a PKCS #12 certificate, from a trusted CA:

- **Certificate chain and private key files** – In this case, you'll need to convert these files into a single certificate file in PKCS #12 format.

- **A PKCS #12 certificate that contains both the certificate chain and private key** – In this case, no conversion is necessary.

To obtain certificate files
1. In most cases, you will need to temporarily open ports 80 and 443 in your firewall configuration. The SSL certificate provider must be able to make successful HTTP and HTTPS requests to your server through the Domain Name System (DNS).

2. Use an SSL certificate provider to generate the certificate chain (fullchain.pem) and private key (privkey.pem) files, or the PKCS #12 certificate, from a trusted CA. Steps will vary based on the certificate provider. Contact your certificate provider for details.

3. Close ports 80 and 443 in your firewall configuration.

4. Change the A record of your domain to the public domain-name address of your web server instance.

Proceed to the next sections as follows:

- If you have certificate chain and private key files, perform the steps in Convert the certificate chain and private key files to a PKCS #12 certificate and Import the PKCS #12 certificate into the Fusion Proxy service keystore.
- If you have a PKCS #12 certificate, perform the steps in Import the PKCS #12 certificate into the Fusion Proxy service keystore.

**Convert the certificate chain and private key files to a PKCS #12 certificate**

| Note | If you have a PKCS #12 certificate, skip this section and proceed to the section Import the PKCS #12 certificate into the Fusion Proxy service keystore. |

To create a PKCS #12 certificate

With the certificate chain and private key as separate files, use the `openssl` command line tool in OpenSSL to create a PKCS #12 certificate.

```
openssl pkcs12 -export -out /path/to/keystore.p12 -in /path/to/fullchain.pem -inkey /path/to/privkey.pem
```

**Important**

Don't enter a blank password.

**Create the Fusion Proxy service keystore and import the PKCS #12 certificate**

Use the Java `keytool` Key and Certificate Management utility to create a keystore for the Fusion Proxy service (`$FUSION_HOME/apps/jetty/proxy/etc/keystore`) and import the PKCS #12 certificate file. Fusion uses this certificate to perform SSL.

1. To create the keystore and import the PKCS #12 certificate:

2. Use the `keytool import` command to create a JSSE keystore.

   ```
   keytool -importkeystore -srckeystore /path/to/keystore.p12 -srcstoretype PKCS12 -destkeystore "$FUSION_HOME/apps/jetty/proxy/etc/keystore" -deststoretype PKCS12
   ```

3. (Optional) If desired, delete the PKCS #12 certificate file that resides outside of the Fusion Proxy service keystore (the one you created from the certificate chain and private key files, or obtained from a trusted CA).
rm /path/to/keystore.p12
2. Enable HTTPS in the Fusion Proxy service

Before beginning these steps, load an SSL certificate into a Fusion keystore.

To enable HTTPS in the Fusion Proxy service:

1. **(Only for Fusion Server 4.0.x and 4.1.0)** Prevent the `start.jar` program from downloading a default keystore file, which isn’t needed. Edit `$FUSION_HOME/apps/jetty/home/modules/ssl.mod`. Comment out the indicated line. Change:

```plaintext
[files]
https://raw.githubusercontent.com/eclipse/jetty.project/master/jetty-server/src/test/config/etc/keystore?id=${jetty.tag.version}|etc/keystore
```

To:

```plaintext
[files]
#https://raw.githubusercontent.com/eclipse/jetty.project/master/jetty-server/src/test/config/etc/keystore?id=${jetty.tag.version}|etc/keystore
```

2. Set environment variables:

```plaintext
export JAVA_HOME=JavaHomeDirectory
export FUSION_HOME=FusionHomeDirectory
```

For example:

3. Add HTTPS protocol support to the Jetty TLS (SSL) connector:

```plaintext
cd "$FUSION_HOME/apps/jetty/proxy/
java -jar "$FUSION_HOME/apps/jetty/home/start.jar" --add-to-start=https
```

Example output:

```plaintext
INFO: ssl initialised (transitively) in ${jetty.base}/start.ini
INFO: https initialised in ${jetty.base}/start.ini
INFO: Base directory was modified
```

4. Get the obfuscated version of your keystore password:

For Fusion 4.1.1 and later:

```plaintext
java -cp "$FUSION_HOME/apps/jetty/home/lib/jetty-util-9.4.12.v20180830.jar"
org.eclipse.jetty.util.security.Password password
```

For Fusion 4.1.0:
java -cp "$FUSION_HOME/apps/jetty/home/lib/jetty-util-9.3.8.v20160314.jar"
org.eclipse.jetty.util.security.Password password

Replace `password` with the password you used for the keystore. If the password contains special characters, URL encode them.

Example output:

```
2018-05-15 12:32:48.988:INFO::main: Logging initialized @133ms
password345XYZ
OBF:1vn2luguisaj1v9i1v941sar1ugwlvo0
MD5:b91cd1a54781790beaa2baf741fa6789
```

5. Edit the file `$FUSION_HOME/apps/jetty/proxy/start.ini`:

   a. Include obfuscated passwords by adding these properties to the end of the file:
      
      ```
      ▪ jetty.sslContext.keyStorePassword
      ▪ jetty.sslContext.keyManagerPassword
      ▪ jetty.sslContext.trustStorePassword
      ```

   b. Use the OBF-encrypted password from step 4 (including the `OBF:` string) as the value for all three of the properties.

      For example:

      ```
      ## Keystore password
      jetty.sslContext.keyStorePassword=OBF:2uha1vgt1jg01a4b1a4j1jda1vg11ugg
      ...
      
      ## KeyManager password
      jetty.sslContext.keyManagerPassword=OBF:2uha1vgt1jg01a4b1a4j1jda1vg11ugg
      
      ## Truststore password
      jetty.sslContext.trustStorePassword=OBF:2uha1vgt1jg01a4b1a4j1jda1vg11ugg
      ```

   c. Set the local SSL port by adding the `jetty.ssl.port` property to the end of the file, and providing the port number. For example:

      ```
      ## Connector port to listen on
      jetty.ssl.port=8443
      ```

   d. Save the file `$FUSION_HOME/apps/jetty/proxy/start.ini`. 
3. Restart Fusion and test access through HTTPS

1. Restart all Fusion services:

```
./bin/fusion restart
```

HTTPS should now be enabled in the Fusion Proxy service.

2. Sign in to the Fusion UI. Specify the HTTPS URL scheme and SSL port, for example, https://search.mycorp:8443.
4. Disable HTTP access to the Fusion Proxy service

Disable HTTP access. You have a choice. Perform the tasks in the appropriate section:

- Disable HTTP access on the firewall or load balancer – *This is the preferred approach.*
- Disable listening for HTTP requests in the Fusion Proxy service

**Alternative 1: Disable HTTP access on the firewall or load balancer**

Disable HTTP access to the Fusion Proxy service on the firewall or load balancer:

1. Disallow all requests for port 8764 from the outside world. Only `localhost` should be able to communicate with Fusion on the non-SSL port 8764. Block all other requestors.
2. If you are using a firewall or load balancer in front of Fusion, use it to redirect all HTTP requests to use HTTPS instead. For example, Apache would redirect all incoming HTTP traffic to HTTPS.

**Alternative 2: Disable listening for HTTP requests in the Fusion Proxy service**

<table>
<thead>
<tr>
<th>Important</th>
<th>Ideally, you should disable HTTP access using the firewall or load balancer. Follow the steps in this section only if disabling HTTP access on the firewall or load balancer isn’t feasible.</th>
</tr>
</thead>
</table>

You can only use this alternative if your SSL certificate covers a hostname that can be accessed from the local host. For example, if your certificate only covers `https://fusion.com`, then your local machine must be able to access Fusion from that exact host. If necessary, change the `hosts` file so that this can work.

To disable HTTP

1. Edit `/opt/lucidworks/fusion/4.2.x/apps/jetty/proxy/start.d/http.ini`.
   a. Change this line:

   ```
   --module=http
   ```

   To:

   ```
   #--module=http
   ```

   b. Save the file.

2. Edit the Fusion configuration file, `/opt/lucidworks/fusion/4.2.x/conf/fusion.properties`.
   a. Ensure that the Agent JVM uses the Fusion Proxy service’s keystore by adding this to the end of the file:
Replace `password` with your Fusion keystore password.

b. Uncomment the `default.address` and change it to the hostname of the server that is validated by your SSL certificate.

If the hostname saved in `default.address` is not validated by your SSL certificate, then the Fusion Proxy service won't start, because the agent's liveness detector won't be able to access the HTTPS port to determine whether Fusion is running.

<table>
<thead>
<tr>
<th>Important</th>
<th>If you self-signed the certificate, then the <code>default.address</code> must match the hostname you specified while signing the certificate. Failure to do this will result in the Fusion Proxy service not starting after you have disabled HTTP.</th>
</tr>
</thead>
</table>

For example, if your SSL certificate's validated hostname is `search.mycorp`, then change:

```java
#default.address = 127.0.0.1
```

to:

```java
default.address = search.mycorp
```

c. Change the `proxy.port` to the SSL port you chose.

d. Uncomment `proxy.ssl` and change its value to `true`. Change:

```java
# proxy.ssl=false
```

to:

```java
proxy.ssl=true
```

Stopping and starting Fusion is needed for these changes to take effect. But there is a bit more to do concerning the Fusion Agent before restarting Fusion.
5. Ensure that the Fusion Agent can do heartbeats against the Proxy service

By default, the Fusion Agent makes HTTP requests to the Fusion Proxy service to ask whether the Proxy service is running (heartbeat checks). So, when you disable HTTP access to the Proxy service, the Fusion Agent needs an alternative way to do the heartbeat checks.

1. Add an entry to the `/etc/hosts` file for the Fusion node so that the DNS routes to the correct IP address:

   192.168.1.6   search.mycorp

   Or using a local IP address:

   127.0.0.1   search.mycorp

2. Add `proxy.address` and specify the hostname of the server on which the Proxy service runs, for example:

   proxy.address=mydomain

3. Stop and start all Fusion services (don't just perform a restart):

   cd "$FUSION_HOME/bin"
   ./fusion stop
   ./fusion start

   HTTPS should now be disabled in the Fusion Proxy service.

4. Verify that you can't sign in to the Fusion UI using the HTTP URL scheme and non-SSL port, for example, `http://search.mycorp:8764`. 
References and tutorials

• Transport Layer Security (Wikipedia)
• Public Key Certificate (Wikipedia)
• OpenSSL Cookbook (free ebook)
• OpenSSL Command Line Utilities (OpenSSL wiki)
• Java Tutorials: Generating and Verifying Certificates
• IBM developerWorks: What is the JSSE all about?
SSL Security (Windows)

Fusion’s UI (which is accessed through the Fusion Proxy service) can run over SSL for secure communication with any HTTP client, using the Java Secure Socket Extension (JSSE) framework. You configure Fusion for SSL by configuring Fusion’s Proxy and UI services.

Required software

Configuring Fusion for SSL requires the following software:

- **Java Development Kit** – To store certificates, you can use the Java `keytool` Key and Certificate Management utility which is a part of the JDK.

  The JDK is also a requirement for Fusion Server.

- **OpenSSL** – You might need the `openssl` command line tool:

  ◦ If you have the certificate chain and private key as separate files, then you can use the `openssl` command line tool to create a PKCS #12 file.

  ◦ If you have an intermediate CA certificate, then you can use it and `openssl` to generate the certificate chain and private key files.

Overview of procedure

To configure Fusion for SSL:

1. Load an SSL certificate into a Fusion keystore.
2. Enable SSL in the Fusion Proxy service.
3. Restart Fusion and test access through HTTPS.
4. Disable HTTP access to the Fusion Proxy service.
5. Ensure that the Fusion Agent can do heartbeats against the Proxy service.

1. Load an SSL certificate into a Fusion keystore

The SSL protocol is based on public-key cryptography where encryption keys come in public key/private key pairs. An SSL certificate is used to verify the authenticity of a particular server. It contains the web site name, contact email address, company information and the public key used to encrypt the communication which is shared with the entities that communicate with the owner of the public/private key pair.

The server has a locally-protected private key that is accessible via a **JSEE keystore**.

The keystore maintains both the server certificate and the private key, so that when a server authenticates itself to the client, it uses the private key from its keystore for the initial SSL handshake.

Load the certificate into a Fusion keystore. Perform the tasks in the appropriate section:

- **Self-signed certificate** – If Fusion is behind a firewall, you can use a self-signed certificate for SSL communication with other hosts in your internal network. Create a keystore for the Fusion Proxy service and load the keystore with the self-signed PKCS #12 certificate.
• **Certificate signed by a certificate authority** – In a production environment, SSL certificates typically originate with certificate signing requests (CSRs) and are signed by a trusted third-party Certificate Authority (CA). Create a keystore for the Fusion Proxy service and load the keystore with the PKCS #12 certificate from a CA.

**Alternative 1: Self-signed certificate**

| Note | If you are using a CSR-originated certificate from a trusted certificate authority, proceed to Alternative 2: CA-signed certificate. |

If Fusion is behind a firewall, you can use a self-signed certificate for SSL communication with other hosts in your internal network. Create a keystore for the Fusion Proxy service and load the keystore with the self-signed PKCS #12 certificate.

To store certificates, you can use the Java keytool Key and Certificate Management utility which is a part of the JDK.

To create a keystore and load a self-signed certificate

1. Set environment variables:

   ```
   set JAVA_HOME=JavaHomeDirectory
   set FUSION_HOME=FusionHomeDirectory
   ```

   For example:

2. Create the Fusion Proxy service keystore, generate the key pair and self-signed certificate, and load them into the keystore:

   ```
   "%JAVA_HOME%\bin\keytool.exe" -genkeypair -keystore "%FUSION_HOME%/apps/jetty/proxy/etc/keystore" -dname "CN=CommonName, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown" -keypass KeyPassword -storepass KeystorePassword -keyalg RSA -alias selfsigned -deststoretype pkcs12 -ext SAN=dns:ServerFqdn,ip:ServerIpAddress
   ```

   **Important**

   You must include the qualified domain name and/or the IP address of the Fusion server in the -ext SAN part of the command. Failure to do so results in SSL validation errors.

   **Example command:**

   ```
   ```

   The resulting certificate enables validated SSL transport to these hosts:
Where `ProxyPort` is the Fusion Proxy port.

**Alternative 2: CA-signed certificate**

<table>
<thead>
<tr>
<th>Note</th>
<th>If Fusion is behind a firewall and you are using a self-signed certificate, skip this section. Perform the tasks in Alternative 1: Self-signed certificate.</th>
</tr>
</thead>
</table>

In a production environment, SSL certificates typically originate with certificate signing requests (CSRs) and are signed by a trusted third-party [Certificate Authority (CA)](https://search.mycorp:ProxyPort).

The steps here assume that you are the person who will be obtaining the SSL certificate chain and private key files. If you aren’t that person, contact your system administrator.

**Preliminary steps**

1. Obtain a domain from a domain registrar.
2. Change the A record of your domain to the public IP address of your web server instance.

**Generate SSL certificate files**

Use an SSL certificate provider to generate the certificate chain and private key files, or a PKCS #12 certificate, from a trusted CA:

- **Certificate chain and private key files** – In this case, you’ll need to convert these files into a single certificate file in PKCS #12 format.
- **A PKCS #12 certificate that contains both the certificate chain and private key** – In this case, no conversion is necessary.

To obtain certificate files:

1. In most cases, you will need to temporarily open ports 80 and 443 in your firewall configuration. The SSL certificate provider must be able to make successful HTTP and HTTPS requests to your server through the Domain Name System (DNS).
2. Use an SSL certificate provider to generate the certificate chain (`fullchain.pem`) and private key (`privkey.pem`) files, or the PKCS #12 certificate, from a trusted CA. Steps will vary based on the certificate provider. Contact your certificate provider for details.
3. Close ports 80 and 443 in your firewall configuration.
4. Change the A record of your domain to the public domain-name address of your web server instance.

Proceed to the next sections as follows:

- If you have certificate chain and private key files, perform the steps in Convert the certificate chain and private key files to a PKCS #12 certificate and Import the PKCS #12 certificate into the Fusion Proxy service keystore.
- If you have a PKCS #12 certificate, perform the steps in Import the PKCS #12 certificate into the Fusion Proxy service keystore.
Convert the certificate chain and private key files to a PKCS #12 certificate

| Note | If you have a PKCS #12 certificate, skip this section and proceed to the section Import the PKCS #12 certificate into the Fusion Proxy service keystore. |

To create a PKCS #12 certificate

With the certificate chain and private key as separate files, use the `openssl` command line tool in OpenSSL to create a PKCS #12 certificate.

```
openssl pkcs12 -export -out \path\to\keystore.p12 -in \path\to\fullchain.pem -inkey \path\to\privkey.pem
```

Important

Don't enter a blank password.

Create the Fusion Proxy service keystore and import the PKCS #12 certificate

Use the Java `keytool` Key and Certificate Management utility to create a keystore for the Fusion Proxy service (`%FUSION_HOME%\apps\jetty\proxy\etc\keystore`) and import the PKCS #12 certificate file. Fusion uses this certificate to perform SSL.

| Note | If you are starting with a certificate file in PFX format (file extension `.pfx`), that format is now identical to PKCS #12 format. |

1. To create the keystore and import the PKCS #12 certificate:

2. Use the `keytool import` command to create a JSSE keystore.

```
keytool -importkeystore -srckeystore \path\to\keystore.p12 -srcstoretype PKCS12 -destkeystore "%FUSION_HOME%\apps\jetty\proxy\etc\keystore" -deststoretype PKCS12
```

3. (Optional) If desired, delete the PKCS #12 certificate file that resides outside of the Fusion Proxy service keystore (the one you created from the certificate chain and private key files, or obtained from a trusted CA.

```
del \path\to\keystore.p12
```

2. Enable HTTPS in the Fusion Proxy service

Before beginning these steps, load an SSL certificate into a Fusion keystore.

To enable HTTPS in the Fusion Proxy service:

1. (Only for Fusion Server 4.0.x and 4.1.0) Prevent the `start.jar` program from downloading a default keystore file, which isn’t needed. Edit `%FUSION_HOME%\apps\jetty\home\modules\ssl.mod`. Comment out the indicated line. Change:
2. Set environment variables:

```bash
set JAVA_HOME=JavaHomeDirectory
set FUSION_HOME=FusionHomeDirectory
```

For example:

3. Add HTTPS protocol support to the Jetty TLS (SSL) connector:

```bash
cd "%FUSION_HOME%/apps/jetty/proxy/"
java -jar "%FUSION_HOME%/apps/jetty/home/start.jar" --add-to-start=https
```

Example output:

```
INFO: ssl initialised (transitively) in ${jetty.base}/start.ini
INFO: https initialised in ${jetty.base}/start.ini
INFO: Base directory was modified
```

4. Get the obfuscated version of your keystore password:

For Fusion 4.1.1 and later:

```bash
java -cp "%FUSION_HOME%/apps/jetty/home/lib/jetty-util-9.4.12.v20180830.jar"
org.eclipse.jetty.util.security.Password password
```

For Fusion 4.1.0:

```bash
java -cp "%FUSION_HOME%/apps/jetty/home/lib/jetty-util-9.3.8.v20160314.jar"
org.eclipse.jetty.util.security.Password password
```

Replace `password` with the password you used for the keystore. If the password contains special characters, [URL encode them](https://en.wikipedia.org/wiki/URL_encoding).

Example output:
5. Edit the file `%FUSION_HOME%/apps/jetty/proxy/start.ini`:

   a. Include obfuscated passwords by adding these properties to the end of the file:
      
      - `jetty.sslContext.keyStorePassword`
      - `jetty.sslContext.keyManagerPassword`
      - `jetty.sslContext.trustStorePassword`

   b. Use the OBF-encrypted password from step 4 (including the `OBF:` string) as the value for all three of the properties.

   For example:

   ```
   ## Keystore password
   jetty.sslContext.keyStorePassword=OBF:2uha1vgt1jg01a4b1a4j1jda1vg11ugg
   ...
   
   ## KeyManager password
   jetty.sslContext.keyManagerPassword=OBF:2uha1vgt1jg01a4b1a4j1jda1vg11ugg
   
   ## Truststore password
   jetty.sslContext.trustStorePassword=OBF:2uha1vgt1jg01a4b1a4j1jda1vg11ugg
   ```

   c. Set the local SSL port by adding the `jetty.ssl.port` property to the end of the file, and providing the port number. For example:

   ```
   ## Connector port to listen on
   jetty.ssl.port=8443
   ```

   d. Save the file `%FUSION_HOME%/apps/jetty/proxy/start.ini`.

3. Restart Fusion and test access through HTTPS

   1. Restart all Fusion services:

   ```
   bin\fusion.cmd restart
   ```

   HTTPS should now be enabled in the Fusion Proxy service.

   2. Sign in to the Fusion UI. Specify the HTTPS URL scheme and SSL port, for example, `https://search.mycorp:8443`.

4. Disable HTTP access to the Fusion Proxy service

Enable HTTP access. You have a choice. Perform the tasks in the appropriate section:
• Disable HTTP access on the firewall or load balancer – This is the preferred approach.
• Disable listening for HTTP requests in the Fusion Proxy service

**Alternative 1: Disable HTTP access on the firewall or load balancer**

Disable HTTP access to the Fusion Proxy service on the firewall or load balancer:

1. Disallow all requests for port 8764 from the outside world. Only `localhost` should be able to communicate with Fusion on the non-SSL port 8764. Block all other requestors.

2. If you are using a firewall or load balancer in front of Fusion, use it to redirect all HTTP requests to use HTTPS instead. For example, Apache would redirect all incoming HTTP traffic to HTTPS.

**Alternative 2: Disable listening for HTTP requests in the Fusion Proxy service**

| Important | Ideally, you should disable HTTP access using the firewall or load balancer. Follow the steps in this section only if disabling HTTP access on the firewall or load balancer isn’t feasible. |

You can only use this alternative if your SSL certificate covers a hostname that can be accessed from the local host. For example, if your certificate only covers `https://fusion.com`, then your local machine must be able to access Fusion from that exact host. If necessary, change the `hosts` file so that this can work.

To disable HTTP

1. Edit `\lucidworks\fusion\4.2.x\apps\jetty\proxy\start.d\http.ini`.
   a. Change this line:

   ```
   --module=http
   ```

   To:

   ```
   #--module=http
   ```

   b. Save the file.

2. Edit the Fusion configuration file, `\lucidworks\fusion\4.2.x\conf\fusion.properties`.
   a. Ensure that the Agent JVM uses the Fusion Proxy service’s keystore by adding this to the end of the file:

   ```
   agent.jvmOptions=-Djavax.net.ssl.trustStore=%{FUSION_HOME}\apps\jetty\proxy\etc\keystore
   -Djavax.net.ssl.trustStorePassword=password
   -Djavax.net.ssl.keyStore=%{FUSION_HOME}\apps\jetty\proxy\etc\keystore
   -Djavax.net.ssl.keyStorePassword=password
   ```

   Replace `password` with your Fusion keystore password.

   b. Uncomment the `default.address` and change it to the hostname of the server that is validated by your SSL certificate.
If the hostname saved in `default.address` is not validated by your SSL certificate, then the Fusion Proxy service won’t start, because the agent’s liveness detector won’t be able to access the HTTPS port to determine whether Fusion is running.

| Important | If you self-signed the certificate, then the `default.address` must match the hostname you specified while signing the certificate. Failure to do this will result in the Fusion Proxy service not starting after you have disabled HTTP. |

For example, if your SSL certificate’s validated hostname is `search.mycorp`, then change:

```bash
#default.address = 127.0.0.1

to:

default.address = search.mycorp
```

c. Change the `proxy.port` to the SSL port you chose. For example, change:

```bash
proxy.port = 8764

to:

proxy.port = 8443
```

d. Uncomment `proxy.ssl` and change its value to `true`. Change:

```bash
# proxy.ssl=false

to:

proxy.ssl=true
```

Stopping and starting Fusion is needed for these changes to take effect. But there is a bit more to do concerning the Fusion Agent before restarting Fusion.

### 5. Ensure that the Fusion Agent can do heartbeats against the Proxy service

By default, the Fusion Agent makes HTTP requests to the Fusion Proxy service to ask whether the Proxy service is running (heartbeat checks). So, when you disable HTTP access to the Proxy service, the Fusion Agent needs an alternative way to do the heartbeat checks.

1. Add an entry to the `/etc/hosts` file for the Fusion node so that the DNS routes the qualified domain name to the
correct IP address, for example:

192.168.1.6   search.mycorp

Or use a local IP address:

127.0.0.1   search.mycorp

2. Add **proxy.address** and specify the hostname of the server on which the Proxy service runs, for example:

proxy.address=search.mycorp

3. Stop and start all Fusion services (don't just perform a restart):

```bash
cd "%FUSION_HOME%\bin"
fusion.cmd stop
fusion.cmd start
```

HTTPS should now be disabled in the Fusion Proxy service.

4. Verify that you can't sign in to the Fusion UI using the HTTP URL scheme and non-SSL port, for example, http://search.mycorp:8764.

References and tutorials

- Transport Layer Security (Wikipedia)
- Public Key Certificate (Wikipedia)
- OpenSSL Cookbook (free ebook)
- OpenSSL Command Line Utilities (OpenSSL wiki)
- Java Tutorials: Generating and Verifying Certificates
- IBM developerWorks: What is the JSSE all about?
Changing a Password

Change your Fusion Server password

While you are logged in, from anywhere in the Fusion UI:

1. Click Account.
2. Click My Profile.
3. Follow the UI instructions to update your password.

Change a user’s Fusion Server password

If you are an admin, you can change the password for any of your users. While you are logged in, from any app in the Fusion UI:

1. Click System > Access Control.
2. Make sure the Users tab is selected.
3. Click the user’s name.
4. Check the box under Change password.
5. Enter and confirm a new password, and click Update.

Reset a lost admin password

If you have lost the password for the admin account, contact Lucidworks Support.
Access Control

These additional topics explain how to configure the supported authentication methods:

- Configuring Fusion for JWT
- Configuring Fusion for Kerberos
- Configuring Fusion for LDAP
- Configuring Fusion for SAML
- Configuring Fusion for SSO

User Authentication and Authorization

Fusion provides application security by restricting access to known users via a two-stage process consisting of:

- Authentication - users must sign on using a username and password.
- Authorization - each username is associated with one or more permissions which specify the Fusion UI components and REST API requests that user has access to. Permissions can be restricted to specific endpoints and path parameters. Roles are named sets of permissions which provide access to a specific function.

The access control component runs in the same process as the Fusion UI. It referred to as the "auth proxy" because it handles authentication and authorization for all requests to the Fusion REST API services.

All requests to Fusion must be authenticated, as described in section User Access Request Params.

User Account Administration

A Fusion Security Realm encapsulates a user database together with specific authentication and authorization mechanisms. This information is stored in ZooKeeper so that it is always available to all Fusion components across the deployment.

Fusion's native security realm manages both authentication and authorization directly. All user information is stored in ZooKeeper: usernames, hashes of passwords, roles, and permissions. Passwords are hashed using bcrypt. Authentication compares a hash of the entered login password with the stored password hash. The native realm is the home of the Fusion admin user and is the default realm type.

Fusion can be configured to use the host domain’s security mechanism for user administration. The following configurations are possible:

- LDAP - Fusion stores a local user record in ZooKeeper. Authentication is performed by the LDAP server. LDAP group membership can be used to assign Fusion permissions.
- Kerberos - Fusion stores a local user record in ZooKeeper. SPNEGO is used for authentication via Kerberos.
- Kerberos authentication, LDAP authorization - Fusion stores a local user record in ZooKeeper. SPNEGO is used for authentication via Kerberos. LDAP group membership can be used to assign Fusion permissions.
- SAML - Fusion stores a local user record in ZooKeeper. The SAML 2.0 protocol is used to provide web-browser single sign-on.
- JWT - JSON Web token.
Cross-origin resource sharing (CORS)

As a security measure, CORS is disallowed by default in Fusion 4.2.0 and above. You can enable it, if needed, by editing the `proxy.corsAllowOrigin` property in `conf/fusion.properties`. The default value is a regular expression that matches nothing:

```
proxy.corsAllowOrigin = (?!)  
```

You can change this to a regular expression that matches the specific hosts or domains that you trust. Setting this to allow all domains (`proxy.corsAllowOrigin = .*`) is not recommended.

Video tutorial

Security Realms

Fusion uses security realms to authenticate users of the Fusion UI. Each user has an assigned security realm, which the user must choose when logging in. Choosing a different realm results in an authentication failure.

A security realm also provides a list of roles:

- The list always includes the role(s) that are specified in the security realm.
- (Optional) The security realm can reference one or more Fusion roles and/or get groups to which the user belongs from an external directory service that is the authentication provider. Fusion maps the group names to role names and adds these roles to the user’s list of roles.

| Note | Fusion does not use permissions from the LDAP to authorize UI access or API requests. It only obtains group names (optionally), which are used as role names or are mapped to role names. If an Active Directory Security Query Trimming Stage is used, then directory-service permissions are used for trimming. If a connector supports security trimming, then connector permissions are used for trimming. |

Requests to the Fusion REST API must specify a security realm for per-request authentication, unless a session cookie is used (which contains information about the security realm).

Fusion authorizes requested operations based on API permissions specified for the user and for the user’s role(s). Fusion considers the role(s) specified in the user definition and in the security realm. Fusion creates a list of roles when a session is created, that is, when a user logs in or when the Sessions REST API creates a session. Authorization based on permissions is at request time.

You can define multiple security realms for a Fusion instance. This lets you give different sets of users different levels of access to specific Fusion collections.

| Note | The directory service that you use to return group information must be the same one that is used for authentication. |
Security Realm Types

When you create a security realm, you can choose among the following security realm types:

**Native**

Fusion has a single preconfigured security realm named `native`. The admin user is in the native realm. The native realm also provides a fallback mechanism in case of LDAP server or communication failure.

This realm is required to bootstrap Fusion. Because all requests to Fusion require authentication and authorization, on initial startup you must access the Fusion UI to set the admin password. After Fusion has a valid admin password, it creates the admin account in the Fusion native realm.

For the native realm, Fusion manages all authentication and permissions information directly.

You can create Fusion user accounts and manage them using either the Fusion UI or the User API.

Passwords are not stored, but are hashed using **bcrypt**.

**SSO Trusted HTTP**

The “SSO Trusted HTTP” realm type (`trusted-http` in the REST API) is useful in single sign-on (SSO) environments.

If the SSO environment contains groups that make sense regarding partitioning Fusion functionality for Fusion users (that is, giving Fusion users different UI and API permissions based on the SSO groups), then you can configure an SSO Trusted HTTP security realm to return a list of group names, and then map the groups to Fusion roles in the security realm definition.

See Configuring Fusion for SSO.

**JWT**

The JSON Web Token (JWT) realm uses an “Authorization” header in the request to authenticate the user and the data inside the JWT token for the authorization. You can configure Fusion to use a shared secret key to encrypt the JWT payload.

For Fusion’s JWT realm, first create a JWT token, with the tool you use to validate users. Then create a Fusion realm.

See Configuring Fusion for JWT.

**Kerberos**

In the case where a host domain uses Kerberos for authentication and LDAP for authorization, Fusion can be configured to do the same, by configuring a realm of type “LDAP” and then specifying Kerberos as the authentication mechanism.

Fusion stores a local user record in ZooKeeper and a mapping to the Kerberos **principal**.

**SPNEGO** is used for authentication via Kerberos.

See Configuring Fusion for Kerberos.

**LDAP**

You can use LDAP as an authentication provider for Fusion. If the LDAP contains groups that make sense regarding
partitioning Fusion functionality for Fusion users (that is, giving Fusion users different UI and API permissions based on the LDAP-group memberships of LDAP users), then you can configure an LDAP security realm to search for LDAP groups and to map the LDAP groups to Fusion roles.

Fusion stores a local user record in ZooKeeper, and authentication is performed by the LDAP server. User accounts can be managed by Fusion, or created automatically, in which case the Fusion user ID maps directly to the LDAP Distinguished Name (DN). Fusion permissions can be assigned automatically based on LDAP group membership.

See Configuring Fusion for LDAP.

**SAML**

Fusion stores a local user record in ZooKeeper and the URL and information about the SAML Identity Provider. The SAML 2.0 protocol is used to provide web browser single sign-on.

See Configuring Fusion for SAML.

**Manage Security Realms**

Only Fusion users with admin privileges can manage security realms. There are two ways to manage security realms:

In the Fusion UI

Navigate to System > Access Control > Security Realms.

Using the Realms API

Use the `http://localhost:8764/api/realm-configs/` endpoint to manage security realms. See the Realms API reference for details. In production environments, use port 8765.

**Configuring Fusion for JWT**

You can configure Fusion Server to use JSON Web Tokens (JWTs) for user authentication and authorization. Fusion can also use a shared secret key between the issuer and Fusion to encrypt the JWT payload.

<table>
<thead>
<tr>
<th>Warning</th>
<th>Fusion 4.1.0 and 4.1.1 have known issues with JWT signature validations. If you are using either of these versions and want to utilize JWT, consider upgrading to the latest version of Fusion.</th>
</tr>
</thead>
</table>

**How the JWT realm works**

A JWT is comprised of three distinct parts: the header, payload, and signing key. Each of these parts are separately encoded using Base64url encoding.

- **The header** identifies the algorithm used to generate the token.

<table>
<thead>
<tr>
<th>Note</th>
<th>Fusion uses only the HS256 signature algorithm.</th>
</tr>
</thead>
</table>

- **The payload** consists of data that will be passed with the token.

- **The signing key** validates the integrity of the token by using a "secret" to ensure the header and payload being
Submitted in the token match the header and payload stored in the signing key.

Within Fusion, the JWT realm uses an authorization header in the request to authenticate the user and the data inside the JWT token for the authorization. This authorization header uses the following format:

```
Bearer <jwt-token>
```

Upon receiving the authorization header, Fusion authenticates the token and emits a response accordingly.

**Configure JWT for Fusion**

**Create a JWT token**

Using the tool you use to validate users, create a JWT token. The token should have following properties:

- **iss**: Issuer value. If the issuer value does not match the value configured in Fusion, the user will be denied access.
- **iat**: A JSON numeric date value. This value is calculated by counting the number of seconds between 1970-01-01T00:00:00Z UTC and the specified UTC date/time, ignoring leap seconds.
- **sub**: Subject. The name/id of the user. The user is logged in by this name.
- **groups**: The groups from the group-role mappings that this user belongs to. The groups key should match the one you specify while creating the JWT realm.

Example data inside token:

```
{
  :iss "fusion-enterprise-app"
  :iat 1562633069
  :sub "username"
  :groups ["group-1" "group-2"]
}
```

**Create the Fusion JWT realm**

To create a Fusion realm, in the Fusion UI:

1. Click **System > Access Control**.
2. Click the **Security Realms** tab, then **Add Security Realm**.
3. Enter a realm name. Under **type**, select **jwt**.
4. Create the security realm with the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Realm type</td>
<td>jwt</td>
</tr>
<tr>
<td>JWT Issuer</td>
<td>A unique value that is used in the JWT authorization header. This value should match the value of the iss in the token, for example fusion-enterprise-app.</td>
</tr>
<tr>
<td>Roles</td>
<td>Check the roles that this realm provides after successfully authenticating a user by default.</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Groups Key</td>
<td>The JWT token value that contains the list of groups the user is in. Default value: GRPS.</td>
</tr>
</tbody>
</table>
| Group mappings        | A list of pairs which associate groups to roles. For example: `{<name of group>, <role assigned to group>}`.

1. Click **Add new mapping**. Two rows appear.
2. In the first row, add a group. The JWT token contains the groups for a particular user.
3. In the second row, add the role or roles (separated by spaces) for that group.

| Signing Key           | A string of secret characters that will be used to encrypt the JWT token. The key must be a shared secret key. Leave empty if no signing key is used. |

5. Click **Save**.

**Validate the new realm**

When you send a request to Fusion, you should receive a response. The request to Fusion looks like this:

```
curl http://127.0.0.1:8764/api/users -H 'authorization: Bearer <token-header>.<token-payload>.<token-signing-key>'
```

**Note**

Bearer is case sensitive.

If no signing key is used, truncate the JWT to read ... Bearer <token-header>.<token-payload>..

**Enable diagnostic logs for debugging**

1. Edit the file `conf/proxy-log4j2.xml`.
2. In the `<Loggers>` element, add the following:

```
<logger name="com.lucidworks.apollo-admin.models.realm-config" level="DEBUG" />
<logger name="com.lucidworks.apollo-admin.middleware.authc.jwt" level="DEBUG" />
```
3. Restart proxy for this change to take effect: `bin/proxy restart`

Now `var/log/proxy/proxy.log` will show some diagnostic logs to help you troubleshoot your issues.

**Resources**

JWT.io is an excellent resource for learning about JWT, creating tokens, debugging tokens, and more.

**Example of generating a JWT using PyJWT**

Typically, you will generate your JWT token from the application you are integrating with Fusion APIs.

This example, however, uses Python to create a JWT token with the PyJWT Python egg.

1. Install the PyJWT egg:

```bash
pip install pyjwt
```

2. Inside a Python console, run the following command using your realm configurations:

```python
import jwt
from datetime import datetime
key = '<signing key>'
jwt_issuer = '<JWT issuer>'
username = '<username>'
groups = ['<groups assigned to user>']
payload = {'iss': jwt_issuer, 'iat': datetime.utcnow(), 'sub': username, 'groups': groups}
print jwt.encode(payload, key, algorithm='HS256')
```

A Bearer token will be printed for use in API requests:

```
eyhb6ci0i1JUz1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJmdXNpb24tYXBwIiwiaWF0IjoxNTYyNjMzMDY5LCJzdWIiOiJmY2Jib0tLbSIsImlhdCI6MTY4NjE5NjcyMCwiZXhwIjoxNjQ2NTgwOTkifQ._ACGk4q3Y5gBQvLrUUQ1cMN2ynACypcpei2JmYF1zc
```

3. You can now use this as a bearer token in an authorization header to authenticate to Fusion.

Test your Fusion API web service call with cURL. For example, if your role is an administrator, call:

```bash
curl -H "Authorization: Bearer eyhb6ci0i1JUz1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJmdXNpb24tYXBwIiwiaWF0IjoxNTYyNjMzMDY5LCJzdWIiOiJmY2Jib0tLbSIsImlhdCI6MTY4NjE5NjcyMCwiZXhwIjoxNjQ2NTgwOTkifQ._ACGk4q3Y5gBQvLrUUQ1cMN2ynACypcpei2JmYF1zc" http://localhost:8764/api/roles
```

**Configuring Fusion for Kerberos in Unix**

To configure the Fusion UI service to use Kerberos for user authentication, you must create a Kerberos security realm.

Kerberos is a system that provides authenticated access for users and services on a network. Instead of sending passwords in plaintext over the network, encrypted passwords are used to generate time-sensitive tickets that are used
for authentication. SPNEGO provides a mechanism for extending Kerberos to Web applications through the standard HTTP protocol.

Kerberos uses symmetric-key cryptography and a trusted third party called a Key Distribution Center (KDC) to authenticate users to a suite of network services. (By users we mean both end users and client programs). The computers managed by that KDC and any secondary KDCs constitute a realm. When a user authenticates to the KDC, the KDC sends a set of credentials (a ticket) specific to that session back to the user’s machine. Kerberos-aware services use the ticket on the user’s machine for authentication instead of requiring sign-on with a password. Because tickets are used rather than passwords, this provides the convenience of Single Sign-On (SSO) in addition to security.

A Kerberized process is one that has been configured so it can get tickets from a KDC and negotiate with Kerberos-aware services. When a user sends an HTTP request, Fusion tries to authenticate using the Kerberos/SPNEGO protocol. If the request was sent from a browser, Fusion does not display the initial sign-on panel; instead on login, the user sees the main Fusion collections panel.

| Important | This article focuses on configuring Fusion for Kerberos in Unix. To learn how to configure Fusion for Windows, see Configuring Fusion for Kerberos in Windows. |

To Kerberize Fusion, you must:

- Configure the Kerberos client on the server that the Fusion UI service will be running on so that it can talk to the KDC (section Configuring the Kerberos client below).
- Configure the security realm of the Fusion UI (section Configuring Fusion Authentication for Kerberos Realm below).
- Depending on the encryption used by the KDC, you may also need to install additional Java security libraries into the Fusion distribution. These are freely available from Oracle, see download and installation instructions below.

To do this, you need following information, which you can get from your sysadmin:

- Kerberos realm name - in most cases, this is your domain name in upper case.
- KDC name - usually "kerberos." + your domain name.
- Kerberos principal name and password. A principal is a unique identity to which Kerberos can assign tickets. When the entity is a client program, this is called the Service Principal name.
- A keytab file which holds the encrypted credentials.

The usual scenario in an enterprise organization is to have a Kerberos admin create a service principal with a random key password. Then, the admin generates a keytab, which is then used for Fusion service principal authentication.

The Kerberos commands needed for configuration and testing are:

- `kinit` - obtain and cache a ticket from the KDC (i.e., domain login)
- `kdestroy` - destroys credentials (i.e., domain logout)
- `klist` - lists cached credentials
- `ktutil` - create or add credentials to a keytab file

If your browser is not already configured to use the Kerberos/SPNEGO, you need to do so in order to test the Fusion configuration.
Configuring the Kerberos client

Step 1: Edit the Kerberos configuration file.

To configure your local Kerberos client so that it can talk to the Kerberized server, you must edit Kerberos configuration file named `krb5.conf`. On most Unix systems, this file is located at `/etc/krb5.conf`.

This file contains Kerberos configuration information, including the locations of KDCs and admin servers for the Kerberos realms of interest, defaults for the current realm and for Kerberos applications, and mappings of hostnames onto Kerberos realms.

If your organization realm name is "MYORG.ORG", and your KDC server is named "kerberos.myorg.org", then you edit two entries. The first entry is `libdefaults`. Set MYORG.ORG as the default realm:

```
[libdefaults]
default_realm = MYORG.ORG
```

The second entry is `realms`. Add MYORG.ORG as a realm:

```
[realms]
MYORG.ORG = {
    kdc = kerberos.myrealm.com
}
```

For example, for realm LUCIDWORKS.IO, the krb5.conf file is just like the above example, except that instead of "myorg.org" we specify "lucidworks.io".

Step 2: Authenticate to Kerberos

The command `kinit` is the Kerberos authentication command. To get started, you authenticate to Kerberos using the Kerberos principal name and password (which you may need to obtain from your sysadmin). For this example, the principal name is "prince".

```
> kinit prince
```

The `kinit` command prompts for a password. Successful authentication is silent. Unsuccessful authentication results in an error message.

The command `klist` shows all cached Kerberos credentials. To check that you’ve successfully authenticated, run this command:

```
> klist
```

Output should be in this form, but with your data:
The Service Principal Keytab file

The usual scenario in an enterprise organization is to have a Kerberos admin create a service principal with a random key password. Then, the admin generates a keytab, which is then used for Fusion service principal authentication. If you are your own Kerberos admin, then you will need to create this file for yourself.

Step 3. Create a Keytab file

The command **ktutil** creates the service principal keytab file which holds the encrypted credentials that the Fusion UI Proxy will use for Kerberos authentication. In order to generate the keytab file, you must have a set of cached credentials, therefore, first run the **kinit** command (step 2).

From the command line, run the command **ktutil**. You must enter your password twice.

```bash
> ktutil -k http-myrealm.org.keytab add -p HTTP/myrealm.org@MYORG.ORG -e aes256-cts-hmac-sha1-96 -V 0
```

The `-k` argument specifies the name of the keytab file which will be created or updated. The command "add" takes the following argument flags:

- `-p` : service principal, format `<service>/<fully.qualified.domain>@REALM`
- `-e` : encryption type. Depending on the encryption type, you may need to download additional Java security libraries for strong encryption.
- `-V` : key version number (kvno). Key version numbers are used in the Kerberos V5 protocol to distinguish between different keys in the same domain.

If successful, this command creates a keytab file called "http-myrealm.org.keytab". Note the directory you're in - you'll need this full path when creating the Proxy realm-config later.

Step 4. Test the Keytab file

The location of this keytab file will be used to configure UI Proxy configuration. Before configuring the Fusion UI Proxy, you should check that the keytab file is valid. Testing the keytab requires the following sequence of steps:

- Clear any existing credentials via command **kdestroy**.
- Log in using the keytab as an argument to the command **kinit -kt <keytab file> <principal>**, where `<principal>` is the name of a principal within the keytab file.
- Examine your credentials via command **klist**.
- Clear credentials via command **kdestroy**, which removes any existing credentials, effectively logging you out of Kerberos.

To remove cached credentials, use the **kdestroy** command. This command succeeds silently. To check that credentials have been removed, re-run the **klist** command:
Use the keytab to login as the service principal, without being prompted for a password:

```bash
> kinit -t http-myrealm.org.keytab HTTP/myrealm.org
```

Examine your credentials via the command `klist`. The output should be similar to this:

```
Credentials cache: API:51D488FF-5CD9-4E16-98FA-B47743F5B4ED
    Principal: HTTP/myrealm.org@MYORG.ORG

    |       |       |                  |
    |-------|-------|------------------|
    | Issued| Expires| Principal        |
    | Apr  1| 09:15:02| 2015  | Apr  1| 19:13:42| 2015 | krbtgt/MYORG.ORG@MYORG.ORG
```

Logout again with `kdestroy`.

**Configuring Fusion Authentication for Kerberos Realm**

Once you have tested both the user and service principal logins, you must create the service principal realm-config in the Fusion Authentication Proxy. This allows the Proxy to authenticate to Kerberos as the service principal, without a password.

**Step 5. Configure the Fusion Realm**

Fusion security realms can be configured either via the Fusion UI Admin tool or the Fusion REST API. The advantage of using the Fusion UI Admin Tool is that a single panel's worth of configuration requires a series of calls to the REST API. It is important to understand the set of configuration properties collected by the Fusion UI and how they are used by the REST API.

A security realm for Kerberos has the following properties:

- **name**: unique string identifier
- **realmType**: "kerberos"
- **enabled**: whether or not the realm is available for users to use with system authentication
- **config**: this property is required for the realm type "kerberos". It takes two key-value pairs:
  - **principal**: the principal service name
  - **keytab**: this must be the full path to the keytab file.

To configure Fusion via the Fusion UI:

1. Log in to Fusion as "admin" or as a user who has super-admin privileges.
2. In the left-hand side navigation bar, click **System > Access control**.
3. Click the **Security Realms** tab, then **Add Security Realm**.
4. On the New Security Realm form, choose type "kerberos" from the pulldown menu so that there are input boxes for the Kerberos realm properties "Service Principal" and "Keytab path".
5. Choose a realm name, and make sure the "enabled" box is checked.

6. If the **auto-created** box is unchecked, a user with admin privileges will need to create Fusion user accounts.

7. Select default roles. The default roles are assigned to a user the first time they access Fusion via this realm, using the Kerberos/SPNEGO protocol. For example, once you have defined a Kerberos realm “my-kerberos-realm” for domain “MYORG.ORG”, when user “any.user” in domain “MYORG.ORG” authenticates to Fusion for the first time via this realm, they are added to the set of Fusion users as username "any.user@MYORG.ORG" and they have all default roles.

   It is prudent to allow the minimum set of default roles, as all users will have these permissions. Some users will require admin privileges and a few users will require super-admin privileges. There should always be a user with super-admin privileges that can authenticate to Fusion using the native security realm and can then grant permissions to individual users as needed.

   | Note | The Fusion proxy Kerberos realm works only with one Fusion host. For a multi-node Fusion cluster where each node resides on a different host, configure a separate realm for each host with a dedicated service/principal.

8. Enter the service principal and the full path to the keytab file.

9. Click **Save**. Fusion displays a confirmation message.

**Kerberos/SPNEGO HTTP Authentication**

**SPNEGO** provides a mechanism for extending Kerberos to applications that use the HTTP protocol including web browsers and the **curl** command-line utility.

**Step 6. Configure the HTTP client**

When a user sends a request to the Kerberized Fusion UI, a SPNEGO request (http[s]) is made. If the user is not already authenticated, the Fusion authentication proxy will yield a 401 status code and a Negotiate header. This status/header response triggers compatible clients to fetch a local ticket from their Kerberos "ticket tray". This ticket is then encoded and sent back to the Fusion. The Fusion authentication proxy will then decode the ticket, and perform a SPN.doAs(user) authentication request to the KDC/Authentication Service. Depending on the results, the proxy then successfully executes the original request (along with a session cookie) or a 401 (without the Negotiate). Clients can either choose to use the session cookie or continue authenticating on every request.

**Configuring Web Browsers and **curl** for SPNEGO**

The **--negotiate** option enables SPNEGO in **curl**.

IE and Safari require no additional configuration to use SPNEGO.

To configure Firefox, access the low level configuration page by loading the about:config page. Then go to the network.negotiate-auth.trusted-uris preference and add the hostname or the domain of the web server that is HTTP Kerberos SPNEGO protected (if using multiple domains and hostname use comma to separate them).

The Chrome browser must be launched from the command line with several added parameters.

To run Chrome on linux:
To run Chrome on a Mac:

```bash
> open 'Google Chrome.app' --args
--auth-server-whitelist="*ROGUECLOUD.COM"
--auth-negotiate-delegate-whitelist="*KERBEROS_DOMAIN"
--auth-schemes="basic,digest,ntlm,negotiate"
```

To run Chrome on Windows:

```bash
chrome.exe --auth-server-whitelist="*KERBEROS_DOMAIN"
--auth-negotiate-delegate-whitelist="*ROGUECLOUD.COM"
--auth-schemes="basic,digest,ntlm,negotiate"
```

For more information, see Using a Web Browser to Access an URL Protected by Kerberos HTTP SPNEGO and http://www.roguelynn.com/words/apache-kerberos-for-django/.

**Session cookies**

A successful Kerberos/SPNEGO login will yield a session cookie, this cookie is identical to the cookie yielded by the Fusion authentication proxy's current POST-login-mechanism.

The expiration policy on the cookie is currently fixed at 8 hours. But has a 1 hour "idle" max, which means if you don’t make a request for 1 hour, the cookie is invalidated. Otherwise the lifetime is pushed ahead until the 8 hour max is met.

The name of the cookie is "id" and the value is a UUID. This UUID is a key that maps to an in-memory value containing the real user ID.

**Testing and Troubleshooting**

Once you have configured the Kerberos security realm, you can test it by logging out of Fusion and shutting down the browser.

Check that you have a valid Kerberos authentication ticket via the klist command.

Now open a new browser session and access the Fusion installation, `<domain>:<UI port>`.

This should take you directly to the main Fusion panel, bypassing the the Fusion "Welcome" login panel. To view your login profile, click on the profile icon at the top right. Your user name should be your login name + "@" + your domain name.

If instead, you see the Fusion login panel, your browser is not configured for SPNEGO.

If the Fusion display consists only of an empty top nav bar, this indicates an authentication failure. Check that the path to your keytab file is correct. Then check the Fusion logs.

If your KDC uses "AES256 CTS mode with HMAC SHA1-96" for key encryption, the proxy will log this error when
attempting to authenticate:

GSSException: Failure unspecified at GSS-API level (Mechanism level: Encryption type AES256 CTS mode with HMAC SHA1-96 is not supported/enabled)

To get around this, the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy File will need to be downloaded and installed. It can be downloaded from:


Place the jars in your JAVA_HOME/jre/lib/security/ directory, then restart Fusion.

Clicking on the "logout" icon on the top nav bar (rightmost icon or a padlock) takes you back to the main Fusion panel. If you destroy your Kerberos credentials cache via the "kdestroy" command, the next time you logout of Fusion, you will be logged out and the browser will display the Fusion login panel.

References and Tutorials

https://en.wikipedia.org/wiki/Kerberos_%28protocol%29
http://www.roguelynn.com/words/explain-like-im-5-kerberos/
https://en.wikipedia.org/wiki/SPNEGO
http://www.oracle.com/technetwork/articles/idm/weblogic-sso-kerberos-1619890.html - section "Install Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files"

Configuring Fusion for LDAP

You can create security realms that use external LDAP servers for authentication. Optionally, Fusion can search in the LDAP for groups to which a user belongs, and then map those groups to Fusion roles. Fusion performs authorization using permissions stored in Fusion users and Fusion roles.

Note: Fusion doesn't use permissions from the LDAP for authorization of UI access or API requests. It only obtains group names (optionally), which are mapped to role names. If an Active Directory Security Query Trimming Stage is used, then directory-service permissions are used for trimming. If a connector supports security trimming, then connector permissions are used for trimming.

To configure Fusion to use an external LDAP as an authentication provider, you'll need to get information about the LDAP server(s) running on your system, either from your system or your sysadmin.

1. Add an LDAP Security Realm

1. Log in to the Fusion UI as the user admin, or as a different user with corresponding permissions.
3. Click Security Realms.
5. Specify info for the new realm:
   - **name** – Name of the security realm. It must be unique. It should be descriptive but short.
   - **type** – Select **ldap** from the pulldown menu.

   When you select **ldap**, Fusion displays additional, LDAP-specific configuration options.

   - **enabled** checkbox – Whether Fusion allows user logins for this security realm. The default is yes (checked).
   - **auto-create users** checkbox – Whether a user account is created automatically upon initial authentication. The default is yes (checked). If the checkbox is unchecked, then a Fusion user with admin permissions must create Fusion users.

6. Scroll down and specify additional options as explained in detail below.

2. **Specify Static Roles (Optional)**

Specify one or more Fusion roles for the security realm. These roles are always considered. They don't depend on searching for LDAP groups and mapping group names to Fusion role names.

In a security realm, you can specify these static roles, add to the list of roles dynamically through an LDAP search, or both. If you do neither, Fusion uses only the role(s) and permissions defined for the user.
3. Specify LDAP Connection Details

Specify the hostname and port of the LDAP server. Check the checkbox if the server is running over SSL.

4. Specify the Authentication Method

Specify the authentication method:

- **Bind** - LDAP authentication is carried out via a single “Bind” operation. See Bind below.
- **Search** - LDAP authentication is carried out indirectly via a Search operation followed by a Bind operation. See Search below.
- **Kerberos** - Kerberos authenticates Fusion and an LDAP Search operation is carried out to find group-level authorizations. See Kerberos below.

**Bind**

Use the Bind authentication method when the Fusion login username matches a part of the LDAP distinguished name.
(DN). Specify the remainder of the LDAP DN in the "DN Template" configuration entry, which uses a single pair of curly braces (\{\}) as a placeholder for the value of the Fusion username.

Search

Use the Search authentication method when the username used for Fusion login doesn’t match a part of the LDAP DN. The search request returns a valid user DN, which is then used together with the user password for authentication via a Bind request.

1. Construct a search request.

   The Search authentication method is generally required when working with Microsoft Active Directory servers. In this case, you need to know the username and password of some user who has sufficient privileges to query the LDAP server for user and group memberships; this user doesn’t have to be the superuser.

   In addition to a privileged user DN and password, the Search authentication method requires constructing a search request. There are two parts to the request. The first part is the base DN of the LDAP directory tree that contains user account objects. The second part of the request is a Search Filter object that restricts the results to a matching subset of the information.

2. Provide the administrator bind DN:
Kerberos

Use the Kerberos authentication method when Kerberos is the authentication provider.

5. Search for LDAP Groups (Optional)

A Fusion role is a bundle of permissions tailored to the access needs of different kinds of users. Access to services and data for LDAP-managed users is controlled by mappings from LDAP users and groups to Fusion roles.

Roles can be assigned globally or restricted to specific LDAP groups. The security realm configuration panel contains a list of all Fusion roles with a checkbox for each, used to assign that role to all users in that realm. LDAP group names can be mapped directly to specific Fusion roles and LDAP group search and filter queries can also be used to map kinds of LDAP users to specific Fusion roles.
6. Map LDAP Groups to Fusion Roles (Optional)

If LDAP group names returned by the search for groups match Fusion role names, you don’t need to map the group names to role names. You must map any LDAP group names that don’t match to Fusion role names (if you don’t, they won’t be used).

7. Save the Security Realm Configuration

Click Save.

Fusion reports whether or not authentication was successful:
8. Basic LDAP Concepts and Terminology

The LDAP protocol is used to share information about users, systems, networks, and services between servers on the internet. LDAP servers are used as a central store for usernames, passwords, and user and group permissions. Applications and services use the LDAP protocol to send user login and password information to the LDAP server. The server performs name lookup and password validation. LDAP servers also store Access Control Lists (ACLs) for file and directory objects which specify the users and groups and kinds of access allowed for those objects.

LDAP is an open standard protocol and there are many commercial and open-source LDAP servers available. Microsoft environments generally use Active Directory. Unix servers use AD or other LDAP systems such as OpenLDAP, although many Unix systems don't use LDAP at all. To configure Fusion for LDAP, you'll need to get information about the LDAP server(s) running on your system either from your sysadmin or via system utilities.

Directories and Distinguished Names

An LDAP information store is a Directory Information Tree (DIT). The tree is composed of entry nodes; each node has a single parent and zero or more child nodes. Every node must have at least one attribute which uniquely distinguishes it from its siblings which is used as the node's Relative Distinguished Name (RDN). A node’s Distinguished Name (DN) is a globally unique identifier.

The string representation of a DN is specified in RFC 4514. It consists of the node’s RDN followed by a comma, followed by the parent node’s DN. The string representation of the RDN is the attribute-value pair name, connected by an equals (“=” ) sign. This recursive definition means that the DN of a node is composed by working from the node back through its parent and ancestor nodes up to the root node.

Here is a small example of a DIT:
The person entry in this tree has the DN: "uid=babs, ou=people, dc=example, dc=com".

Attribute names include many short strings based on English words and abbreviations, e.g.:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cn</td>
<td>commonName</td>
</tr>
<tr>
<td>dc</td>
<td>domainComponent</td>
</tr>
<tr>
<td>mail</td>
<td>email address</td>
</tr>
<tr>
<td>ou</td>
<td>organizationalUnitName</td>
</tr>
<tr>
<td>sn</td>
<td>surname</td>
</tr>
<tr>
<td>uid</td>
<td>userId</td>
</tr>
</tbody>
</table>

LDAP entry attributes can refer to other LDAP entries by using the DN of the entry as value of that attribute. The following example of a directory which contains user and groups information shows how this works:
This tree contains two organizational units: "ou=people" and "ou=groups". The children of the "group" organizational unit are specific named groups, just as the child nodes of organization unit "people" are specific users. There are three user entries with RDNs "uid=bob", "uid=alice", "uid=bill" and two groups with RDNs "cn=user" and "cn=admin". The dotted lines and group labels around the person nodes indicates group membership. This relationship is declared on the groups nodes by adding an attributes named "member" whose value is a users DN. In the **LDAP data interchange format (LDIF)**, this is written:

\[
\begin{align*}
\text{cn=人都,ou=groups,dc=acme,dc=org} \\
\text{member: uid=bob,ou=people,dc=acme,dc=org} \\
\text{member: uid=alice,ou=people,dc=acme,dc=org} \\
\text{cn=admin,ou=groups,dc=acme,dc=org} \\
\text{member: uid=bill,ou=people,dc=acme,dc=org}
\end{align*}
\]

See the [Wikipedia’s LDAP entry](https://en.wikipedia.org/wiki/LDAP) for details.

**LDAP Protocol Operations**

For authentication purposes, Fusion sends Bind operation requests to the LDAP server. The Bind operation authenticates clients (and the users or applications behind them) to the directory server, establishes authorization identity used for subsequent operations on that connection, and specifies the LDAP protocol version that the client will use.

Depending on the way that the host system uses LDAP to store login information about users and groups, it may be necessary to send Search operation requests to the LDAP server as well. The Search operation retrieves partial or complete copies of entries matching a given set of criteria.

**LDAP filters** specify which entries should be returned. These are specified using prefix notation. Boolean operators are "&" for logical AND, "|" for logical OR, e.g., "A AND B" is written "(&(A(B))". To tune and test search filters for a Unix-based LDAP system, see the [ldapsearch command line utility](https://www.openldap.org/doc/admin22/ldapsearch.html) documentation. For Active Directory systems, see AD...
Configuring Fusion for SAML

**SAML 2.0** is a standard for exchanging authentication and authorization data between security domains. The SAML protocol allows web-browser single sign-on (SSO) through a sequence of messages sent to and from the browser, which is the intermediary between Fusion and the SAML authority acting as the Identity Provider (IDP).

To configure Fusion to use SAML 2.0 for user authentication and authorization you must create a SAML security realm. In addition to configuring the Fusion security realm, you must configure the SAML identity provider to recognize the Fusion application.

Once Fusion is configured for a SAML realm, this realm is added to the list of available realms on the initial Fusion sign-on panel. When the SAML realm is chosen from the list of available realms, the browser then redirects to the IDP which handles user authentication. Upon successful authentication, the IDP sends a response back to the browser which contains authentication and authorization information as well as the URL of the Fusion application. The browser redirects back to the Fusion URL, passing along the SAML message with the user authentication and authorization information. Fusion then issues as session cookie which is used for subsequent user access.

**Fusion Configuration for a SAML Realm**

You must get the following information about the SAML Identity Provider either from your sys admin or from the IDP directly:

- **Identity Provider URL** - the URL used by the SAML authority for single sign-on. For example: "https://www.my-idp.com/<my-app-path>/sso/saml"

  | Note | The URL format may differ depending on the SAML Identity Provider.

- **Issuer** - SAML Issuer Id. A unique ID for that authority, such as "http://www.my-idp.com/exk686w2xi5KTuSXz0h7".

- **Certificate Fingerprint** - the contents of the SAML authority certificate, *without the certificate header and footer*. You must get this certificate from the SAML Identity Provider. The certificate is a text file which has a pair of header and footer lines which say "BEGIN CERTIFICATE" and "END CERTIFICATE", respectively. The fingerprint consists of the lines between the header and the footer. You can cut and paste this information into the text box on the Fusion UI.

- **User ID Attribute** - an optional attribute. The Identity Provider contains the user database. By default, the Fusion username is the same as the login name known to the Identity Provider. When another field or attribute in the user record stored by the IDP should be used as the Fusion username, that attribute name is the value of the User ID Attribute.

All Fusion security realms require the following information:

- **name** - must be unique, should be descriptive yet short.
- **type** - value is "SAML" (one of the choices on the Fusion UI Security Realms config panel).
- **"enabled"** - default value is true. The "enabled" setting controls whether or not Fusion allows user logins for this security realm.
- **"auto-create users"** - default is true. This controls whether or not a user account is created automatically upon initial authentication. If false, new user accounts can only be created by a Fusion user with admin privileges.
SAML Authority Identify Provider Configuration for Fusion

The Fusion application must be registered with the SAML Identity Provider. The amount of information varies depending on the SAML authority.

All systems will require the Fusion URL to redirect to upon successful login; this is the protocol, server, and port for the Fusion application, and path "api/saml", such as "https://www.my-fusion-app.com:8764/api/saml". If the Fusion application is running behind a load-balancer, then this URL is the load-balancer URL plus path "api/saml". Note that the load-balancer should be session-sticky in order for the sequence of messages that comprise the SAML protocol to run to completion successfully.

Some authorities may require additional information. In particular the SAML 2.0 "AudienceRestriction" tag may be part of the SAML message. This tag specifies the domain for which the SAML trust conditions are valid, which is usually the domain in which the Fusion app is running, such as "https://www.my-fusion-app".

Example SAML Realm Configuration

The Fusion endpoint "api/realms-config" returns a JSON list of all the configuration objects for all realms. After configuring a SAML realm named "saml-test" using the okta.com developer preview tool, the configuration object for this realm is:

```json
{
    "name":"saml-test",
    "realmType":"saml",
    "enabled":true,
    "config":{
        "autoCreateUsers":true,
        "idpUrl":"https://dev-417804.oktapreview.com/app/dev417804_1/exk686w2xix5KtSXz0h7/ssa/saml",
        "issuer":"http://www.okta.com/exk686w2xix5KtSXz0h7",
        "certificateFingerprint":"MIIDpDCCaygAwIBAgIGAVr4A4JMA0GCSqGSIb3DQEBBQUAMIGSMQswCQYDVQQGEwJVUzETMBEGA1UECAwK Condensed output
```

References

http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf
Configuring Fusion for SSO

The "SSO Trusted HTTP" realm type (trusted-http in the REST API) is useful in single sign-on (SSO) environments.

If SSO is already set up in your environment, user identities and group information can be sent to Fusion through HTTP headers (REMOTE_USER, for example). The SSO Trusted HTTP realm type provides the configuration options for integrating this into Fusion's authentication systems. It also supports allowing access to only a set of known client IPs, and mapping groups to Fusion roles.

Use the Realms API to configure this realm type:

curl -u user:pass -H 'content-type:application/json' -X POST :3000/api/realm-configs -d @./realm-config.json

Below is a sample configuration:

```json
{"id":"test-id",
 "enabled":true,
 "name":"sso-test",
 "realmType":"trusted-http",
 "config":{"identityKey":"REMOTE_USER",
             "groups": {"key":"GROUPS",
                         "delimiter":"|",
                         "roleMapping": [["a","admin"], ["b","foo"]],
                         "allowedIps": ["127.0.0.1", "0:0:0:0:0:0:0:1", "localhost"]}
}
```

| identityKey | The name of an HTTP header. If this key is found in the request headers, its value is used as the identity of the client (username, for example). |
| groups      | Configuration keys for auth groups: |
|            |   • key + The name of an HTTP header, used as the source of group names. |
|            |   • delimiter + The character used to split the value (defaults to comma). |
|            |   • roleMapping + A set of 2-tuples, used for mapping the external group values to Fusion Roles. |
allowedIps

Allow access to only a set of known client IPs. When this property is defined and the client IP is not included in it, the realm logic return a 401.

The `X-FORWARDED-FOR` header is inspected for this client IP first; the value is split on comma, and the first entry is taken. This would normally be used in cases where the client was forwarded to Fusion through one or more external proxy servers. If the `X-FORWARDED-FOR` header is not present in the request, the `REMOTE-ADDR` header value is used instead.

### Permissions

Permissions determine what a user can do in Fusion. There are two kinds of permissions:

- **UI permissions** – Control which parts of the Fusion UI a user can access. These parts show up in menus and the user can view them. But the ability to use the functionality depends on API permissions.
- **API permissions** – Control which requests a user can submit to which REST API endpoints.

Permissions can be defined by either a role or a user, or both. Fusion combines permissions for authorization as follows:

- UI permissions are positive (permission needs to be given) and additive (the user has the sum of all specified permissions. This is true of roles specified in a user definition, roles specified in a security realm, and roles determined dynamically based on groups in an LDAP authentication provider.

- API permissions specified in roles are positive (permission needs to be given) and additive (the user has the sum of all specified permissions; that is, for a specific endpoint, the most permissive permissions are used). This is true of roles specified in a user definition, roles specified in a security realm, and roles determined dynamically based on groups in an LDAP authentication provider.

- API permissions specified in the role(s) but not in the user definition are used.

- If an API permission for a specific endpoint is specified in both a role and in the user definition, then the permissions in the user definition are used, **overriding** the permissions in the role(s). Use permissions in user definitions to give specific users permissions that are less permissive than the permissions for their role(s).

  For example, say role A allows GET and POST access to a specific endpoint. User X is a member of role A and also has a user definition that allows only GET access to that endpoint. In this case, user X has only GET access to that specific endpoint.

  Alternatively, you could define less permissive roles.

#### Specify UI Permissions

Specify UI permissions in roles.

#### Specify API Permissions

A Fusion API permission denotes an allowed request to a Fusion REST API endpoint or endpoints. A permissions
specification consists of:

- HTTP request method or methods allowed. Multiple HTTP methods are separated by commas.
- REST API services endpoint, which can contain wildcards or named variables. All calls to the REST API start with "api/", followed by the service name and any methods and parameters. The permissions specification includes everything following api/. The endpoint can include wildcards.

Wildcards make it easy to grant broad access to Fusion services. The wildcard symbol * matches all possible values for a single path segment. Two wildcards match all possible values for any number of path segments. Granting access to a subset of Fusion functionality can require a list of narrowly defined permissions. Carefully defining each Fusion app to address a specific use case can simplify permission sets, by letting you grant access liberally within a single app.

A path segment can be a named variable enclosed in curly braces: {variable-name}. Variables are used when a wildcard would be too permissive and a single path segment too restrictive.

- Optionally, the allowed values for any named variables in the endpoint. The variable specification component specifies the restricted value or values for all named variables in the path. Each specification consists of the variable name, followed by "=" (the equals sign), followed by one or more values separated by commas. If the endpoint specification has multiple variable, the semi-colon character ";" is used as the separator between parameter specifications.

A permission specification has three elements: methods, path, and parameters. A permissions specification is written as a string using the colon character ":" as the separator between the elements. The lists of permission specifications are stored in the ZooKeeper User and Roles entries in JSON format.

### Example Permission Sets

**Admin permissions**

The permissions for the admin user can be written in a single line:

```
GET, POST, PUT, DELETE, PATCH, HEAD:/**
```

This permission specification uses two wildcards. One wildcard specifies any path segment, and the other specifies any value.

**Best practice: Delegate access to a Fusion app**

When possible, create a Fusion app for a specific use case and assign permissions on a per-app basis. For example, to configure a Fusion account to be able to run queries against only a certain collection:

1. Create a Fusion app.
2. Create a query profile by defining a collection. This lets you change the underlying collection without having to redo permissions.
3. Assign permissions as follows:

```
GET:/apps/<myapp>/query/<myprofile>/*
```
The above permission set allows a user to run queries against the collection specified by `<myprofile>` under the Fusion app `<myapp>`.

**Roles**

Roles are named sets of permissions that encapsulate the permissions needed for different kinds of users. Permissions grant users access to subsets of Fusion functionality. A role can specify UI permissions, API permissions, or both:

- **UI permissions** grant users access to parts of the Fusion UI
- **API permissions** grant users access to specific API commands for specific REST API endpoints.

See Permissions for information about how permissions supplied by multiple roles and by user definitions combine.

**Where You Specify Roles**

You can specify which roles to apply for a user in one or more of these places:

- **Security realm (directly)** – Under the heading **Roles**, specify the roles to always apply to all users in the security realm.
- **Security realm (from a group/role mapping)** – Security realms of types `ldap` and `trusted-http` can provide a list of groups to which the user belongs. The security realm can map the group names to role names.
- **User definition** – A user definition can specify roles for the user. These roles don’t override the other roles. They are added to the other roles.

**Default Roles**

At initial startup, Fusion creates a set of default roles for common types of users.

**admin**

The admin role is the equivalent to the Unix `root` or superuser. It allows full access to all Fusion services:

```
GET, POST, PUT, DELETE, PATCH, HEAD:/**
```

**developer**

The developer role has all the read/write permissions required for building and running applications.
GET, POST, PUT:/system/**
GET, POST, PUT, DELETE, HEAD:/stopwords/**
GET, POST, PUT:/usage/**
GET:/features/**
GET, POST, PUT, DELETE, HEAD:/blobs/**
GET, POST, PUT, DELETE, HEAD:/scheduler/**
GET, POST, PUT, DELETE, HEAD:/experiments
GET:/introspect/**
PUT:/usage/**
GET, POST, PUT, DELETE, HEAD:/index-stages/**
GET, POST, PUT, DELETE, HEAD:/messaging/**
GET, POST, PUT, DELETE, HEAD:/catalog
GET, POST, PUT, DELETE, HEAD:/parsers/**
GET, POST, PUT:/appkit/**
GET, POST, PUT, DELETE, HEAD:/index-profiles/**
GET, POST, PUT:/recommend/**
GET, POST, PUT, DELETE, HEAD:/history/**
GET, POST, PUT, DELETE, HEAD:/apps/**
GET, POST:/dynamicSchema/**
GET, POST, PUT, DELETE, HEAD:/solr/**
GET, POST:/query/**
GET, POST, PUT,/signals/**
GET, POST, PUT:/searchLogs/**
GET, POST, PUT, DELETE, HEAD:/query-pipelines/**
GET, POST, PUT:/configurations/**
GET:/suggestions/**
GET, POST, PUT, DELETE, HEAD:/searchCluster/**
GET, POST, PUT, DELETE, HEAD:/index-pipelines/**
GET:/license
GET, POST, PUT, DELETE, HEAD:/spark/**
GET, POST, PUT, DELETE, HEAD:/query-stages/**
GET, POST, PUT, DELETE, HEAD:/prefs/apps/search/**
GET:/nodes/**
GET, POST, PUT, DELETE, HEAD:/solrAdmin/**
GET, POST, PUT:/synonyms/**
GET, POST, PUT, DELETE, HEAD:/jobs/**
GET, POST, PUT, DELETE, HEAD, OPTIONS:/collections/**
GET, POST, PUT, DELETE, HEAD:/connectors/**
GET, POST, PUT, DELETE, HEAD:/groups/**
GET, POST, PUT, DELETE, HEAD:/query-profiles/**
GET, POST, PUT:/templates/**
GET, POST, PUT, DELETE, HEAD:/tasks/**
GET, POST, PUT, DELETE, HEAD:/links/**
PATCH:/users/{id}:id=#ID
GET, POST, PUT:/registration/**
POST:/index/**
GET, POST, PUT:/objects/**

Note

The permission PATCH:/users/{id}:id=#ID uses the variable value #ID as a placeholder for the currently logged-in user ID. It is included so the Fusion UI "change password" feature is available to native realm users.

rules

The rules role provides query rewriting API access to all Fusion apps.
**search**

The search role has read-only query and write-only signal API access to the Fusion “default” collection. These permissions are required for search applications, for example, for App Studio.

**webapps-role**

The webapps role can list and download Fusion apps.

**Role Information**

Fusion stores role information in Apache ZooKeeper. Each role in a ZooKeeper entry contains the following:

- **id**—ID string, created by Fusion
- **name**—Role name string
- **desc**—Text description; optional
- **permissions**—A list of Fusion permission specifications
- **ui-permissions**—A list of names of Fusion UI components
- **created-at**—Timestamp; created by Fusion
- **updated-at**—Timestamp for last edit; created by Fusion

**Manage Roles**

Only Fusion users with admin privileges can manage roles.

Restricting access to a subset of Fusion’s functionality requires several narrowly defined permissions. Path variables can be used to designate specific collections. As an example, it’s possible to define a role which allows read-only access
to Fusion dashboards for a specific collection:

- GET:/solr/{id}/*:id=test – Read-only access to the collection "test"
- GET:/solr/{id}/admin/luke:id=test – Also read-only access
- GET:/solr/system_banana/* – Read-only access to dashboards
- GET:/collections/system_banana – Read-only access to the collection where dashboard definitions are stored

**Manage Roles in the Fusion UI**

Manage roles in the Fusion UI. Click **System > Access Control > Roles**.

To create a new role from the Fusion UI, first choose a unique role name, then edit the set of permissions. Specify API permissions one per line in the Permissions input box. There is a separate list of checkboxes that allow access to the Fusion UI components. If users who are assigned this role require access to the Fusion UI, then you must specify UI permissions in addition to REST API permissions.

**Manage Roles via HTTP Requests to the Roles API**

See page Roles API.

**Secure Communication with a SolrCloud Cluster**

You can configure Fusion and an external SolrCloud cluster so that communication between Fusion and the SolrCloud cluster is secured. Use either Kerberos or basic authentication. You can secure communication for both the default search cluster and for other SolrCloud clusters.

**Note**: Securing communication between Fusion and a *bundled* default search cluster is not supported.

Prerequisite: Secure the default search cluster. Use either Solr's Basic Authentication Plugin or Kerberos Authentication Plugin.

The required steps differ. These are the high-level steps. Detailed steps follow.

- **Default search cluster** – Define configuration parameters for bootstrapping Fusion, and then bootstrap Fusion.
- **Other SolrCloud clusters** – In the Fusion UI, add the external SolrCloud cluster.

**Default Search Cluster**

If your default search cluster is in an *external* SolrCloud cluster, then you can secure the cluster with Kerberos or basic authentication, and then configure Fusion to communicate securely with the cluster.

**Prerequisite**

Secure the default search cluster. Use either Solr's Basic Authentication Plugin or Kerberos Authentication Plugin.

Don’t start Fusion yet. Below, you will define bootstrap properties, and then bootstrap Fusion.

**Configure and Bootstrap Fusion**

1. Create a .properties file for the initial bootstrap of Fusion. Place the file outside of the Fusion installation, for example, in /tmp. You will delete the file at the end of this procedure:
$ touch /tmp/fusion-bootstrap.properties

2. Edit the `fusion-bootstrap.properties` file to define Fusion initial-bootstrap configuration properties. Example strings are in bold italics. Replace those with your own values.

<table>
<thead>
<tr>
<th>Caution</th>
<th>Consult with your Kerberos administrator about the correct configuration properties.</th>
</tr>
</thead>
</table>

- **Kerberos authentication** – Specify the authentication type (**kerberos**), the Kerberos principal, and the Kerberos keytab file:

  ```
  default-search-cluster.auth-type=kerberos
  default-search-cluster.auth-principal=fusion@MYORG.ORG
  default-search-cluster.auth-keytab=/path-to-file/keytab.kt
  ```

  Here is information about how to get a service principal and generate a keytab file.

- **Basic authentication**: Specify the authentication type (**basic**), the username of the Solr user to use for authentication, and the password of that user:

  ```
  default-search-cluster.auth-type=basic
  default-search-cluster.auth-user=admin
  default-search-cluster.auth-password=admin-password
  ```

  The Solr user must be the admin user or a different user with full administrative privileges.

<table>
<thead>
<tr>
<th>Note</th>
<th>Fusion doesn’t support Solr authorization plugins.</th>
</tr>
</thead>
</table>

3. Edit the `fusion.properties` file:

a. Uncomment and change the value of this property to point to an external ZooKeeper:

  ```
  # default.zk.connect = localhost:9983
  ```

b. Uncomment and change the value of this property to use an external SolrCloud cluster:

  ```
  # default.solrZk.connect = localhost:2181/solr-zk-namespace
  ```

c. Remove `zookeeper` and `solr` from the `group.default` property:

  ```
  group.default = api, connectors, ui
  ```

d. Add a configuration property for the path to the initial-bootstrap properties file:

  ```
  initial-bootstrap-properties-path = /tmp/fusion-bootstrap.properties
  ```
4. Change your working directory to the directory that contains the Fusion binaries:

```bash
$ cd ~/path/to/fusion/4.2.x/bin
```

5. Bootstrap Fusion:

```bash
$ ./fusion start
```

6. After Fusion starts:
   a. Delete the initial-bootstrap properties file:

```bash
$ rm /tmp/fusion-bootstrap.properties
```

   b. Edit the `fusion.properties` file to remove the entry for the initial-bootstrap properties file:

```
initial-bootstrap-properties-path = /tmp/fusion-bootstrap.properties
```

**Other SolrCloud Cluster**

You can secure an external SolrCloud cluster with Kerberos or basic authentication, and then configure Fusion to communicate securely with the cluster.

**Prerequisite**


**Add the secure SolrCloud cluster in the Fusion UI (Basic Auth)**

1. log in to the Fusion UI as the user `admin`.

2. Click **System > Home > System > Solr Clusters > New Solr Cluster**.

3. Click **Advanced**.

4. Specify the required values ID and Connect String. Under Solr Cluster Authentication, check **include**. Choose Authentication Type **basic**, and specify a username and password for authentication.

5. Click **Save new**.

**Add the secure SolrCloud cluster in the Fusion UI (Kerberos)**

1. log in to the Fusion UI as the user `admin`.

2. Click **System > Home > System > Solr Clusters > New Solr Cluster**.

3. Click **Advanced**.

4. Specify the required values ID and Connect String. Under Solr Cluster Authentication, check **include**. Choose Authentication Type **kerberos**, and specify a Kerberos keytab file and Kerberos principal for authentication.

5. Click **Save new**.
User Access Request Params

Fusion requests must come from a known user, i.e., a user with a unique user id (UUID). Fusion’s ZooKeeper registry tracks all users across all realms. Usernames must be unique within a realm. Fusion creates a globally unique user ID for all users based on the combination of username and realm.

All requests to the Fusion REST API require either a username, password, and security realm name, or the session cookie which contains the unique user ID.

Per-Request Authentication

To pass authentication information with each request, the realmName is specified as a query parameter on the request itself:

```
```

The default realmName parameter is "native", so for native authentication, this parameter can be omitted.

Session Cookies

The Fusion UI service endpoint "api/session" can be used to generate a session cookie which contains the unique user id via a POST request whose body consists of a JSON object which contains the username, password information. For users belonging to a realm other than the native realm, the request parameter "realmName" must be specified. The command to generate a session cookie for the admin user with password "password123" is:

```
curl \
   -c cookie -i -X POST -H "Content-type:application/json" -d @- \
   http://localhost:8764/api/session?realmName=native \
<<EOF
   { "username" : "admin" , "password" : "password123" }
EOF
```

The curl command takes any number of specialized arguments, followed by the URL of the request endpoint. The arguments used here are:

- `-c`: filename of cookies file. If it exists, cookies are added to it. You can use `-c -` which writes to the terminal window (std out).
- `-i`: include the HTTP-header in the output. Used here to see the cookie returned with the response.
- `-X`: request method, in this case POST
- `-H`: request header. The api/session endpoint requires Content-type:application/json.
- `-d`: Pass POST body as part of the command-line request. To get ready the body from a file, use the syntax `-d @<filename>`. The argument `-d @-` reads the data from stdin.

The header output shows the cookie information:
Once the session cookie file has been created, it can be sent along in all subsequent requests to the REST API. For the

curl command-line client, the \texttt{-b} flag is used to send the contents of the cookie file to the server along with the request.

The following command sends a GET request to the Fusion REST API Collections service to check the status of the
\texttt{system_logs} collection. The \texttt{-b} flag sends in a freshly generated session cookie.

\begin{verbatim}
> curl -b cookie -i http://localhost:8764/api/collections/system_logs
\end{verbatim}

\begin{verbatim}
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Content-Encoding: gzip
Vary: Accept-Encoding, User-Agent
Content-Length: 278
Server: Jetty(9.2.11.v20150529)

{
  "id" : "system_logs",
  "createdAt" : "2016-03-04T23:29:47.779Z",
  "searchClusterId" : "default",
  "commitWithin" : 10000,
  "solrParams" : {
    "name" : "system_logs",
    "numShards" : 1,
    "replicationFactor" : 1
  },
  "type" : "METRICS",
  "metadata" : { }
}
\end{verbatim}

If the session cookie has expired, the system returns a 401 Unauthorized code:

\begin{verbatim}
> curl -b cookie -i http://localhost:8764/api/collections/system_logs
\end{verbatim}

\begin{verbatim}
HTTP/1.1 401 Unauthorized
Content-Type: application/json; charset=utf-8
Content-Length: 31
Server: Jetty(9.2.11.v20150529)

{"code":"session-idle-timeout"}
\end{verbatim}

\section*{Users}

All Fusion requests must come from a registered user.

\subsection*{Add Users}

The first user who logs in becomes the user \texttt{admin}. 
There are two approaches for adding users:

- **Manual** – Add users manually to a security realm that doesn’t auto-create users.
- **Automatic** – For a security realm that uses an external authentication provider, Fusion can add users automatically. When creating the security realm, check **auto-create users**. Fusion creates a user the first time someone logs into Fusion.

When you add a new user manually, you must provide a unique username and valid password. All other information is optional. However, unless either roles or permissions are specified (or both), this user won’t be able to do anything in Fusion.

If you specify API permissions in a user definition, those permissions override corresponding permissions defined in the user’s roles. See Permissions for more information about how permissions supplied by multiple roles and by user definitions combine.

### Manage Users in the Fusion UI

Only Fusion users with administrative privileges (for example, those who are assigned the built-in role **admin**) can manage users.

Manage users in the Fusion UI. Click **System > Access Control > Users**.

### Manage Users via HTTP Requests to the Users API

See page Users API.

### User Information

Fusion stores user information in **Apache ZooKeeper**.

Each User entry in ZooKeeper contains the following:

- **id**– A globally unique user ID (UUID), created by Fusion based on username, realm-name
- **realm-name**– The Fusion security realm name; the default is “native”.
- **username**– The username string, which is unique within the specified security realm
- **permissions**– List of permissions that have been explicitly assigned to the user in the Fusion UI (in **System > Access Control**)  
- **role-names**– List of roles assigned to the user in the Fusion UI (in **System > Access Control**)  
- **created-at**– Timestamp; created by Fusion  
- **updated-at**– Timestamp for the last edit; created by Fusion

The following JSON shows the ZooKeeper record for the Fusion admin user:
The following JSON shows the ZooKeeper record for a user entry managed by Fusion:

```
{
  "id": "57f539d2-3f53-4011-ad6f-257a3f00fc6b",
  "username": "admin",
  "realm-name": "native"
  "password-hash": "$2a$08$3I82umlXLPSshQ1W6ngj.Or06D0VgDLGoh6mC9G0yRtvy5Nfkn6",
  "permissions": [],
  "role-names": ["admin"],
  "created-at": "2016-01-28T00:00:18Z"
}
```

```
{
  "id": "ae9b345a-79e2-4e6d-8620-e6ed4ed2cc16",
  "username": "firstname.lastname",
  "realm-name": "lwLDAP",
  "permissions": [{"path": "collections/**","methods": ["GET"]}],
  "role-names": [],
  "created-at": "2016-04-01T21:17:36Z",
  "updated-at": "2016-04-01T21:42:15Z"
}
```
Fusion SQL service

Most organizations that deploy Fusion also have SQL-compliant business intelligence (BI) or dashboarding tools to facilitate self-service analytics.

The Fusion SQL service:

• Lets organizations leverage their investments in BI tools by using JDBC and SQL to analyze data managed by Fusion. For example, Tableau is a popular data analytics tool that connects to Fusion SQL using JDBC to enable self-service analytics.

• Helps business users access important data sets in Fusion without having to know how to query Solr.

Important

In addition to the specified System Requirements, Fusion on Windows requires **Visual C++ Redistributable for Visual Studio 2015** to start the SQL service successfully.
Fusion SQL architecture

The following diagram depicts a common Fusion SQL service deployment scenario using the Kerberos network authentication protocol for single sign-on. Integration with Kerberos is optional. By default, the Fusion SQL service uses Fusion security for authentication and authorization.

The numbered steps in the diagram are:

1. The JDBC/ODBC client application (for example, TIBCO Spotfire or Tableau) uses Kerberos to authenticate a Fusion data analyst.
2. After authentication, the JDBC/ODBC client application sends the user’s SQL query to the Fusion SQL Thrift Server over HTTP.
3. The SQL Thrift Server uses the keytab of the Kerberos service principal to validate the incoming user identity.

The Fusion SQL Thrift Server is a Spark application with a specific number of CPU cores and memory allocated from the pool of Spark resources. You can scale out the number of Spark worker nodes to increase available memory and CPU resources to the Fusion SQL service.

4. The Thrift Server sends the query to Spark to be parsed into a logical plan.
5. During the query planning stage, Spark sends the logical plan to Fusion’s pushdown strategy component.
6. During pushdown analysis, Fusion calls out to the registered AuthZ FilterProvider implementation to get a filter query to perform row-level filtering for the Kerberos-authenticated user.

By default, there is no row-level security provider but users can install their own implementation using the Fusion SQL service API.

7. Spark executes a distributed Solr query to return documents that satisfy the SQL query criteria and row-level security filter. To leverage the distributed nature of Spark and Solr, Fusion SQL sends a query to all replicas for each shard in a Solr collection. Consequently, you can scale out SQL query performance by adding more Spark and/or Solr resources to your cluster.
Fusion pushdown strategy

The pushdown strategy analyzes the query plan to determine if there is an optimal Solr query or streaming expression that can push down aggregations into Solr to improve performance and scalability. For example, the following SQL query can be translated into a Solr facet query by the Fusion pushdown strategy:

```
select count(1) as the_count, movie_id from ratings group by movie_id
```

The basic idea behind Fusion’s pushdown strategy is it is much faster to let Solr facets perform basic aggregations than it is to export raw documents from Solr and have Spark perform the aggregation. If an optimal pushdown query is not possible, then Spark pulls raw documents from Solr, and then performs any joins or aggregations needed in Spark. Put simply, the Fusion SQL service tries to translate SQL queries into optimized Solr queries. But failing that, the service simply reads all matching documents for a query into Spark, and then performs the SQL execution logic across the Spark cluster.
Starting, stopping, and status

Starting the Fusion SQL service

The Fusion SQL service is an optional service that must be started manually.

Give these commands from the `bin` directory below the Fusion home directory, for example, `/opt/fusion/4.2.x` (on Unix) or `C:\lucidworks\fusion\4.2.x` (on Windows).

On Unix:

When starting the Fusion SQL service, the best practice is to also start the Spark master and Spark worker services:

```
./fusion start spark-master spark-worker sql
```

On Windows:

When starting the Fusion SQL service, the best practice is to also start the Spark master and Spark worker services:

```
spark-master.cmd start
spark-worker.cmd start
sql.cmd start
```

Stopping the Fusion SQL service

The Fusion SQL service is an optional service that must be stopped manually.

Give these commands from the `bin` directory below the Fusion home directory, for example, `/opt/fusion/4.2.x` (on Unix) or `C:\lucidworks\fusion\4.2.x` (on Windows).

On Unix:

When stopping the Fusion SQL service, the best practice is to also stop the Spark master and Spark worker services:

```
./fusion stop sql spark-worker spark-master
```

On Windows:

When stopping the Fusion SQL service, the best practice is to also stop the Spark master and Spark worker services:

```
sql.cmd stop
spark-worker.cmd stop
spark-master.cmd stop
```

Updating the group.default definition

If you plan to run the Fusion SQL service for production, we recommend updating the `group.default` definition in the
file `conf/fusion.properties` (on Unix) or `conf\fusion.properties` (on Windows) to include the `spark-master, spark-worker, and sql` services:

```
group.default = zookeeper, solr, api, connectors-classic, connectors-rpc, proxy, webapps, admin-ui, log-shipper, spark-master, spark-worker, sql
```

**Verifying that the Fusion SQL service started**

Verify the Fusion SQL service application started. Give these commands from the `bin` directory below the Fusion home directory.

On Unix:

```
./sql status
```

On Windows:

```
sql.cmd status
```

Alternatively, check the Spark UI, for example:

```
<table>
<thead>
<tr>
<th>Application ID</th>
<th>Name</th>
<th>Cores</th>
<th>Memory per Node</th>
<th>Submitted Time</th>
<th>User</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>app-20161007115919-0007</td>
<td>FusionSQLEngine-20161007175808</td>
<td>4</td>
<td>2.0 GB</td>
<td>2016/10/07 11:59:10</td>
<td>timpoter</td>
<td>RUNNING</td>
</tr>
</tbody>
</table>
```
Which collections are registered

By default, all Fusion collections except system collections are registered in the Fusion SQL service so you can query them without any additional setup. However, empty collections cannot be queried, or even described from SQL, so empty collections won’t show up in the Fusion SQL service until they have data. In addition, any fields with a dot in the name are ignored when tables are auto-registered. You can use the Catalog API to alias fields with dots in the names to include these fields.

If you add data to a previously empty collection, then you can execute either of the following SQL commands to ensure that the data gets added as a table:

```
show tables
show tables in 'default'
```

The Fusion SQL service checks previously empty collections every minute and automatically registers recently populated collections as a table.

You can describe any table using:

```
describe table-name
```

See the movielens lab in the Fusion Spark Bootcamp for a complete example of working with the Fusion Catalog API and Fusion SQL service. Also read about the Catalog API.
Troubleshooting SQL queries

If you encounter an issue with a SQL query, the first place to look for more information about the issue is the `var/log/sql/sql.log` file. If you need more verbose log information, change the level to DEBUG for the following loggers in the file `conf/sql-log4j2.xml` (on Unix) or `conf\sql-log4j2.xml` (on Windows):

```xml
<logger name="com.lucidworks.spark" level="DEBUG"/>
<logger name="com.lucidworks.spark.sql" level="DEBUG"/>
```

After making changes, you must restart the Fusion SQL service. Give these commands from the `bin` directory below the Fusion home directory, for example, `/opt/fusion/4.2.x` (on Unix) or `C:\lucidworks\fusion\4.2.x` (on Windows).

On Unix:

```
./sql restart
```

On Windows:

```
sql.cmd restart
```
Connecting to the Fusion SQL service from JDBC

The default JDBC properties for connecting to the Fusion SQL service are:

**Driver:**

org.apache.hive.jdbc.HiveDriver

**Driver JAR:**

Unix:

/opt/fusion/4.2.x/apps/libs/hive-jdbc-shaded-2.1.1.jar

Windows:

C:\lucidworks\fusion\4.2.x\apps\libs\hive-jdbc-shaded-2.1.1.jar

**JDBC URL:**

jdbc:hive2://localhost:8768/default;transportMode=http;httpPath=fusion

The username and password are the same as the ones you use to authenticate to Fusion.
Increasing resource allocations

So as to not conflict with the CPU and memory settings used for Fusion driver applications (default & script), the Fusion SQL service uses a unique set of configuration properties for granting CPU and memory for executing SQL queries.

You can use the Configurations API to override the default values shown here.

<table>
<thead>
<tr>
<th>Configuration Property and Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fusion.sql.cores</code> 1</td>
<td>Sets the max number of cores to use across the entire cluster to execute SQL queries. Give as many as possible while still leaving CPU available for other Fusion jobs.</td>
</tr>
<tr>
<td><code>fusion.sql.executor.cores</code> 1</td>
<td>Number of cores to use per executor</td>
</tr>
<tr>
<td><code>fusion.sql.memory</code> 1g</td>
<td>Memory per executor to use for executing SQL queries</td>
</tr>
<tr>
<td><code>fusion.sql.default.shuffle.partitions</code> 20</td>
<td>Default number of partitions when performing a distributed group-by-type operation, such as a JOIN</td>
</tr>
<tr>
<td><code>fusion.sql.bucket_size_limit.threshold</code> 30,000,000</td>
<td>Threshold that determines when to use Solr streaming rollup instead of facet when computing aggregations; rollup can handle high cardinality dimensions but is much slower than using facets to compute aggregate measures.</td>
</tr>
<tr>
<td><code>fusion.sql.max.no.limit.threshold</code> 10,000</td>
<td>Sets a limit for SQL queries that select all fields and all rows, that is, <code>select * from table-name</code>.</td>
</tr>
<tr>
<td><code>fusion.sql.max.cache.rows</code> 5,000,000</td>
<td>Don’t cache tables bigger than this threshold. If a user sends the cache-table command for large collections with row counts that exceed this value, then the cache operations will fail.</td>
</tr>
<tr>
<td><code>fusion.sql.max.scan.rows</code> 2,000,000</td>
<td>Safeguard mechanism to prevent queries that request too many rows from large tables. Queries that read more than this many rows from Solr will fail; increase this threshold for larger Solr clusters that can handle streaming more rows concurrently.</td>
</tr>
</tbody>
</table>

Tip

The Fusion SQL service is designed for executing analytics-style queries over large data sets. You need to provide ample CPU and memory so that queries execute efficiently and can leverage Spark’s in-memory caching for joins and aggregations.

Here's an example of increasing the resources for the Fusion SQL service:
If you change any of these settings, you must restart the Fusion SQL service with `./sql restart` (on Unix) or `sql.cmd restart` (on Windows).

The Fusion SQL service is a long-running Spark application and, as such, it holds on to the resources (CPU and memory) allocated to it using the aforementioned settings. Consequently, you might need to reconfigure the CPU and memory allocation for other Fusion Spark jobs to account for the resources given to the Fusion SQL service. In other words, any resources you give to the Fusion SQL service are no longer available for running other Fusion Spark jobs. For more information on adjusting the CPU and memory settings for Fusion Spark jobs, see the Spark configuration settings.
Hive configuration

Behind the scenes, the Fusion SQL service is based on Hive. Use the hive-site.xml file in /opt/fusion/4.2.x/conf/ (on Unix) or C:\lucidworks\fusion\4.2.x\conf\ (on Windows) to configure Hive settings.

If you change hive-site.xml, you must restart the Fusion SQL service with ./sql restart (on Unix) or sql.cmd restart (on Windows).
Using virtual tables with a common join key

With Solr, you can index different document types into the same shard using the composite ID router based on a common route key field. For example, a customer 360 application can index different customer-related document types (contacts, apps, support requests, and so forth) into the same collection, each with a common customer_id field. This lets Solr perform optimized joins between the document types using the route key field. This configuration uses Solr’s composite ID routing, which ensures that all documents with the same join key field end up in the same shard. See Document Routing.

Providing a compositeIdSpec for the Fusion collection

Before indexing, you need to provide a compositeIdSpec for the Fusion collection. For example:

```bash
curl -u $FUSION_USER:$FUSION_PASS -X POST -H "Content-type:application/json" \
   -d '{"id":"customer","solrParams":{"replicationFactor":1,"numShards":1,"maxShardsPerNode":10},"type":"DATA","compositeIdSpec":{"routeKey1Field":"customer_id_s"}}' \
   "$FUSION_API/apps/$MYAPP/collections?defaultFeatures=false"
```

In the example request above, we create a collection named customer with the route key field set to customer_id_s. When documents are indexed through Fusion, the Solr Index pipeline stage uses the compositeIdSpec to create a composite document ID, so documents get routed to the correct shard.

Exposing document types as virtual tables

If you configure your Fusion collection to use a route key field to route different document types to the same shard, then the Fusion SQL service can expose each document type as a virtual table and perform optimized joins between these virtual tables using the route key. To create virtual tables, you simply need to use the Fusion Catalog API on the data asset for the main collection to set the name of the field that determines the document type. For example, if you have a collection named customer that contains different document types (contacts, support tickets, sales contracts, and so forth), then you would set up virtual tables using the following Catalog API update request:

```bash
   "projectId" : "fusion",
   "name" : "customer",
   "assetType" : "table",
   "description" : "Fusion collection customer",
   "format" : "solr",
   "options" : [
      "collection -> customer",
      "exclude_fields -> _lw_*,*_\d_coordinate,_raw_content_",
      "solr.params -> sort=id asc"
   ],
   "cacheOnLoad" : false,
   "id" : "fusion.customer",
   "additionalSettings": {
      "virtualTableField":"doc_type_s"
   }
}
```

In the example above, we set the virtualTableField to doc_type_s. Fusion sends a facet request to the customer collection to get the unique values of the doc_type_s field and creates a data asset for each unique value. Each virtual table is
registered in the Fusion SQL service as a table.

**Performing optimized joins in SQL**

After you have virtual tables configured and documents routed to the same shard using a `compositeIdSpec`, you can perform optimized joins in SQL that take advantage of Solr's domain-join facet feature. For example, the following SQL statement results in a JSON facet request to Solr to perform the aggregation:

```sql
select count(1) num_support_requests,
       c.industry as industry,
       a.app_id as app_id,
       a.feature_id as feature_id
from customer c
join support s on c.customer_id = s.customer_id
join apps a on s.customer_id = a.customer_id
where c.region='US-East' AND s.support_type='Enhancement' AND a.app_type='Search'
group by industry, app_id, feature_id
```

In the example above, we compute the number of feature enhancement requests for Search applications from customers in the US-East region by performing a 3-way join between the `customer`, `support`, and `apps` virtual tables using the `customer_id` join key. Behind the scenes, Fusion SQL performs a JSON facet query that exploits all documents with the same `customer_id` value being in the same shard. This lets Solr compute the `count` for the `industry`, `app_id`, `feature_id` group by key more efficiently than is possible using table scans in Spark.
Using Kerberos for JDBC authentication

Use the following steps to configure the Fusion SQL service to use Kerberos for authentication.

1. Create a service principal and keytab; your Active Directory or Kerberos administrator will know how to do this. At a minimum, enable the AES 128-bit encryption. You can use 256, but you’ll have to install the JCE extensions.

   This is an example command to create a keytab file for the service account:

   ```
   ktpass /out c:\fusion.service.keytab /princ fusion/sawsserver@FUSIONSQL.LOCAL /rndpass /ptype
   KR85_NT_PRINCIPAL /mapUser fusion@FUSIONSQL.LOCAL -mapOp set -crypto AES128-SHA1
   ```

2. Copy the keytab file to the Fusion `conf` directory.

3. Update the file `conf/hive-site.xml` (on Unix) or `conf\hive-site.xml` (on Windows) to use Kerberos authentication and the correct principal and keytab file installed in step 2.

   On Unix:

   ```
   <property>
     <name>hive.server2.authentication</name>
     <value>Kerberos</value>
   </property>
   <property>
     <name>hive.server2.authentication.Kerberos.principal</name>
     <value>fusion/sawsserver@FUSIONSQL.LOCAL</value>
   </property>
   <property>
     <name>hive.server2.authentication.Kerberos.keytab</name>
     <value>./conf/fusion.service.keytab</value>
   </property>
   ```

   On Windows:

   ```
   <property>
     <name>hive.server2.authentication</name>
     <value>Kerberos</value>
   </property>
   <property>
     <name>hive.server2.authentication.Kerberos.principal</name>
     <value>fusion/sawsserver@FUSIONSQL.LOCAL</value>
   </property>
   <property>
     <name>hive.server2.authentication.Kerberos.keytab</name>
     <value>conf\fusion.service.keytab</value>
   </property>
   ```

4. Install the file that contains information about your Kerberos realm on the Fusion server.

   On Unix:

   Place the file `krb5.conf` in the etc directory.
On Windows:

Place the file `krb5.ini` in the `C:\Windows` directory.

5. Update the file `conf/fusion.properties` (on Unix) or `conf\fusion.properties` (on Windows) to point to the file `krb5.conf` (on Windows) or `krb5.ini` (on Windows) installed in step 4.

On Unix:

```shell
sql.jvmOptions = -Xmx1g -Djava.security.krb5.conf=/etc/krb5.conf
```

On Windows:

```shell
sql.jvmOptions = -Xmx1g -Djava.security.krb5.conf=C:\Windows\krb5.ini
```
Key features

Searching and Sorting

Scoring

The WHERE and ORDER BY clauses can be used to search and sort results using the underlying search engine. The score (lower case) keyword can be used to sort by the relevance score of a full text query.

An example of a query that uses the score keyword is below:

```
select id, title, score from books where abstract = 'hello world' order by score desc
```

Searching

Search predicates are specified in the WHERE clause. Search predicates on text fields will perform full text searches. Search predicates on string fields will perform exact matches unless the LIKE expression is used.

By default all multi-term predicates are sent to the search engine as phrase queries. In the example above 'hello world' is searched as a phrase query.

To stop the auto-phrasing of multi-term predicates wrap parenthesis around the terms. For example:

```
select id, title, score from books where abstract = '(hello world)' order by score desc
```

In the example above the '(hello world)' search predicate will be sent to the search engine without phrasing and perform the query hello OR world

When parenthesis are used the search expression is sent to Solr unchanged. This allows for richer search predicates such as proximity search.

The example below performs a proximity search:

```
select id, title, score from books where abstract = '("hello world"~4)' order by score desc
```

Lucene/Solr wildcards can be sent to the search engine directly using this syntax:

```
select id, title from books where abstract = '(he?lo)'
```

The LIKE clause can be used to perform wildcard searches with either the Solr wildcard symbol * or the SQL wildcard symbol %.

When using the traditional SQL % wildcard only leading and trailing wildcards are supported. Use Lucene/Solr wildcards as described above for more complex wildcards.

The example below shows a LIKE query with a trailing % wildcard.
select id, title from books where abstract like 'worl%'

The following operators are supported for numeric and datetime predicates: <, >, >=, <=, =, !=.

Both the IN and BETWEEN clauses can be used to specify predicates.

Boolean predicates can be used and are translated to boolean search queries.

The example below specifies a boolean query:

```sql
select id from products where prod_desc = 'bike' and price < 125 order by price asc
```

**Sorting**

Numeric, datetime and string fields can be sorted on using the ORDER BY clause. The sort is pushed down to the search engine for optimal performance. Multiple sorts can be specified using the standard SQL syntax.

The example below sorts on a numeric field:

```sql
select id, prod_name, price_f from products where prod_desc = 'bike' order by price_f desc
```

**Single and Multi-dimension SQL aggregations**

SQL aggregations are translated to Solr facet queries to take advantage of Solr’s distributed aggregation capabilities. This allows for interactive data analysis over large data sets.

Single and multi-dimension aggregation using supported aggregation functions operate over the entire query result and are designed to return accurate results. The supported aggregation functions that are fully pushed down to the search engine are: count(*), count(distinct), sum, avg, min, max.

An example of a SQL aggregation that is translated to a Solr facet query is below:

```sql
select company_name, count(*) as cnt from orders group by company_name order by cnt desc
```

**Having Clause**

A HAVING clause can also be applied to single and multi-dimension aggregations.

**Time series aggregations**

Fusion SQL provides a powerful and flexible time series aggregation query through the use of the `date_format` function. Aggregations that group by a `date_format` are translated to a Solr range facet query. This allows for fast, interactive time series reporting over large data sets.

An example of a time series aggregation is shown below:
The `date_format` function is used to specify both the output format and the time interval in one compact pattern as specified by the Java `SimpleDateFormat` class.

The example above is performing a monthly time series aggregation over the `rec_time` field which is a `datetime` field.

To switch to a daily time series aggregation all that is needed is to change the date pattern:

```
select date_format(rec_time, 'yyyy-MM-dd') as day, count(*) as cnt
from logrecords where rec_time > '2000-01-01' and rec_time < '2000-12-31'
group by day
```

### Date math predicates

Fusion SQL also supports date math predicates through the `date_add`, `date_sub`, and `current_date` functions.

Below is an example of the use of date math predicates.

```
select date_format(rec_time, 'yyyy-MM-dd') as day, count(*) as cnt
from logrecords where rec_time > date_sub(current_date(), 30)
group by day
```

### Auto-filling of time intervals

Fusion SQL automatically fills any time interval that does not contain data with with zeroes. This ensures that the full time range is included in the output which makes the time series results easy to visualize in charts.

### Sort Order

Time series aggregations are sorted by default in time ascending order. The `ORDER BY` clause can be used to sort time series aggregation results in a different order.

### Having Clause

A `HAVING` clause can also be applied to a time series query to limit the results to rows that meet specific criteria.

### Sampling and Statistics

Sampling is often used in statistical analysis to gain an understanding of the distribution, shape and dispersion of a variable or the relationship between variables.

Fusion SQL returns a random sample for all basic selects that do not contain an ORDER BY clause. The random sample is designed to return a uniform distribution of samples that match a query. The sample can be used to infer statistical information about the larger result set.

The example below returns a random sample of single field:
select filesize_d from logs where year_i = 2019

If no limit is specified the sample size will be 25000. To increase the sample size add a limit larger then 25000.

select filesize_d from logs where year_i = 2019 limit 50000

The ability to subset the data with a query and then sample from that subset is called **Stratified Random Sampling**. Stratified Random Sampling is an important statistical technique used to better understand sub-populations of a larger data set.

**Descriptive Statistics**

Sub-queries can be used to return random samples for fast, often sub-second, statistical analysis. For example:

```
select count(*) as samplesize,
    mean(filesize_d) as mean,
    min(filesize_d) as min,
    max(filesize_d) as max,
    approx_percentile(filesize_d, .50) as median,
    variance(filesize_d) as variance,
    std(filesize_d) as standard_dev,
    skewness(filesize_d) as skewness,
    kurtosis(filesize_d) as kurtosis,
    sum(filesize_d) as sum
from (select filesize_d from logs where year_i = 2019 limit 50000)
```

In the example above the sub-query is returning a random sample of 50000 results which is operated on by the main statistical query. The statistical query returns aggregations which describe the distribution, shape and dispersion of the sample set.

**Correlation and Covariance**

Sub-queries can be used to provide random samples for correlation and covariance:

```
select corr(filesize_d, response_d) as correlation,
    covar_samp(filesize_d, response_d) as covariance
from (select filesize_d, response_d from logs limit 50000)
```

In the example above the random sample returns two fields to the corr and covar_samp functions in the main query. Correlation and covariance are used to show the strength of the linear relationship between two variables.

**Numeric Histograms**

Sub-queries can be used to provide random samples as input for numeric histograms:

```
select histogram_numeric(filesize_d, 12) as hist
from (select filesize_d from testapp limit 50000)
```

In the example above the random sample is operated on by the histogram_numeric function which is returning a
histogram with 12 bins. Histograms are used to visualize the shape of a distribution.

The histogram_numeric function returns an array containing a struct for each bin. For visualization tools to display the histogram it will often need to be exploded into a result table. The `explode` function can be combined with the LATERAL_VIEW clause to return the histograms as a table.

```sql
SELECT CAST(hist.x as double) as bin_center,
       CAST(hist.y as double) as bin_height
FROM (select histogram_numeric(filesize_d, 12) as response_hist from (select filesize_d from testapp limit 50000) a
     LATERAL VIEW explode(response_hist) exploded_table as hist
```

### Pushed Down Statistical Queries

A narrower set of statistical aggregations can be pushed down to the search engine and operate over entire result sets. These functions are: count(*), count(distinct), sum, min, max, avg and approx_percentile.

Below is an example of a fully pushed down statistical query:

```sql
select count(*) as cnt, avg(filesize_d) as avg, approx_percentile(filesize_d, .50) as median from logs where year_i = 2018
```

Statistical queries that contain a mix of the queries above and non-pushdown such as skewness or kurtosis will be operate over a random sample that matches the query.

Below is an example of a statistical query that operates over a random sample:

```sql
select count(*) as cnt, skewness(filesize_d) as skewness from logs where year_i = 2018
```
Apache Spark is an open-source cluster-computing framework that serves as a fast and general execution engine for large-scale data processing jobs that can be decomposed into stepwise tasks, which are distributed across a cluster of networked computers.

Spark improves on previous MapReduce implementations by using resilient distributed datasets (RDDs), a distributed memory abstraction that lets programmers perform in-memory computations on large clusters in a fault-tolerant manner.

These topics provide information about Spark administration in Fusion Server:

- Spark Components – Spark integration in Fusion, including a diagram
- Spark Getting Started – Starting Spark processes and working with the shell and the Spark UI
- Spark Driver Processes – Fusion jobs run on Spark use a driver process started by the API service
- Spark Configuration – How to configure Spark for maximum performance. The article also provides information about ports, directories, and configuring connections for an SSL-enabled Solr cluster.
- Scaling Spark Aggregations – How to configure Spark so that aggregations scale
- Spark Troubleshooting – How to troubleshoot Spark

Additionally, you can configure and run Spark jobs in Fusion, using the Spark Jobs API or the Fusion UI.
Spark with Fusion AI

With a Fusion AI license, you can also use the Spark cluster to train and compile machine learning models, as well as to run experiments via the Fusion UI or the Spark Jobs API.
Further Reading

• Apache Spark Key Terms, Explained
• Apache Spark on Wikipedia
Spark Components

This diagram shows the Spark components available from Fusion:

- **Fusion API (8765)**: Launches single driver. Runs multiple aggregation jobs. API will restart if it crashes.

- **Spark Driver**: Plans DAG 1 per cluster. Driver program creates a logical plan for executing a DAG of tasks for a job. Actual task execution occurs in executor processes launched by the Spark worker, which sends resource requests to master.

- **Spark Masters (Web UI 8767)**: 1 per cluster Hot standby for HA. Master tracks worker state and delegates work to each worker in cluster.

- **Spark Workers (Web UI 8770)**: At most 1 per node in a cluster. Workers start and monitor processes for failures. Restarts if necessary.

- **ZooKeeper**: Worker spins up many executors, 1 per app.
Spark components in Fusion

- **Application**: An active SparkContext in the Spark Master web UI, which consists of a classpath and a configuration.

  Jobs submitted to the cluster always run as classes in a specific application, that is, using the application’s classpath and configuration.

- **SparkDriver**: The Spark driver program, a JVM process launched by the Fusion API service to execute Fusion jobs in Spark. SparkDriver creates and manages SparkContext for the Fusion application, and stops SparkContext when it’s no longer needed.

- **Spark master (spark-master)**: Agent-managed Fusion service that coordinates worker processes and applications in a Spark cluster.

  You should run at least 2 spark-master processes per cluster to achieve high-availability. ZooKeeper determines which spark-master process is the leader and handles fail-over.

- **Spark worker (spark-worker)**: Agent-managed Fusion service that launches executors for Spark applications. Spark-workers communicate with the master to launch executors for an application.

- **SQL service (sql)**: Agent-managed Fusion service that runs Spark’s thrift-based SQL engine. It provides JDBC access to a Spark cluster.

- **Spark shell (spark-shell)**: Wrapper script provided with Fusion to launch the Spark Scala REPL shell with the correct master URL (pulled from Fusion’s API) and a shaded Fusion JAR added.

- **Custom script job**: A Fusion job that executes a custom Scala script using the Spark shell.

- **Spark Job Workbench**: A toolkit provided by Lucidworks to help build custom Spark jobs using Scala, Java, or Python. See Spark Job Workbench.

- **CoarseGrainedExecutorBackend**: Executor process(es) launched by a spark-worker to execute the tasks for a specific application, such as the spark-shell.

- **Shaded JAR**: The Fusion API service creates an assembly jar (also call an uber jar) that contains all of the dependencies needed to use spark-solr and Fusion classes within a Spark job.

  Classes that conflict with classes on Spark’s classpath are shaded to ensure that Fusion classes use the correct version.
Spark Getting Started

The public GitHub repository Fusion Spark Bootcamp contains examples and labs for learning how to use Fusion's Spark features.

In this section, you'll walk through some basic concepts of using Spark in Fusion. For more exposure, you should work through the labs in the Fusion Spark Bootcamp.

Starting the Spark Master and Spark Worker services

The Fusion run script /opt/fusion/4.2.x/bin/fusion (on Unix) or C:\lucidworks\fusion\4.2.x\bin\fusion.cmd (on Windows) doesn't start the spark-master and spark-worker processes. This reduces the number of Java processes needed to run Fusion and therefore reduces memory and CPU consumption.

Jobs that depend on Spark, for example, aggregations, will still execute in what Spark calls local mode. When in local mode, Spark executes tasks in-process in the driver application JVM. Local mode is intended for jobs that consume/produce small datasets.

One caveat about using local mode is that a persistent Spark UI is not available. But you can access the driver/job application UI at port 4040 while the local SparkContext is running.

To scale Spark in Fusion to support larger data sets and to speed up processing, you should start the spark-master and spark-worker services.

On Unix:

```
./spark-master start
./spark-worker start
```

On Windows:

```
spark-master.cmd start
spark-worker.cmd start
```

Give these commands from the bin directory below the Fusion home directory, for example, /opt/fusion/4.2.x (on Unix) or C:\lucidworks\fusion\4.2.x (on Windows).

Tip

To have the spark-master and spark-worker processes start and stop with bin/fusion start and bin/fusion stop (on Unix) or bin\fusion.cmd start and bin\fusion.cmd stop (on Windows), add them to the group.default definition in fusion.properties. For example:

```
group.default = zookeeper, solr, api, connectors-classic, connectors-rpc, proxy, webapps, admin-ui, log-shipper, spark-master, spark-worker
```
Viewing the Spark Master

After starting the master and worker services, direct your browser to http://localhost:8767 to view the Spark master web UI, which should resemble this:

![Spark Master web UI](image)

If you don't see the master UI and at least one worker in the ALIVE state, check these logs.

**On Unix:**

```
/opt/fusion/4.2.x/var/log/spark-master/spark-master.log
/opt/fusion/4.2.x/var/log/spark-worker/spark-worker.log
```

**On Windows:**

```
C:\lucidworks\fusion\4.2.x\var\log\spark-master\spark-master.log
C:\lucidworks\fusion\4.2.x\var\log\spark-worker\spark-worker.log
```

Use this Fusion API request to get the status of the Spark master:

```
curl http://localhost:8764/api/spark/master/status
```

This request should return a response of the form:

```
[ {
  "url" : "spark://192.168.1.9:8766",
  "status" : "ALIVE",
  "workers" : [
    {
      "id" : "worker-20161005175058-192.168.1.9-8769",
      "host" : "192.168.1.9",
      "port" : 8769,
      "webuiaddress" : "http://192.168.1.9:8770",
      "cores" : 8,
      "coresused" : 0,
      "coresfree" : 8,
      "memoryused" : 0,
      "memoryfree" : 2048,
      "state" : "ALIVE",
      "lastheartbeat" : 1475711489460
    }
  ], ...
} ]
```
If you have multiple Spark masters running in a Fusion cluster, each will be shown in the status but only one will be ALIVE; the other masters will be in STANDBY mode.

| Tip | If you are operating a multi-node Spark cluster, we recommend running multiple Spark master processes to achieve high-availability. If the active one fails, the standby will take over. |

Running a job in the Spark shell

After you have started the Spark master and Spark worker, run the Fusion Spark shell.

On Unix:

```bash
./spark-shell
```

On Windows:

```bash
spark-shell.cmd
```

Give these commands from the bin directory below the Fusion home directory, for example, /opt/fusion/4.2.x (on Unix) or C:\lucidworks\fusion\4.2.x (on Windows).

The shell can take a few minutes to load the first time because the script needs to download the shaded Fusion JAR file from the API service.

If ports are locked down between Fusion nodes, specify the Spark driver and BlockManager ports, for example:

On Unix:

```bash
./spark-shell --conf spark.driver.port=8772 --conf spark.blockManager.port=8788
```

On Windows:

```bash
spark-shell.cmd --conf spark.driver.port=8772 --conf spark.blockManager.port=8788
```

When the Spark shell is initialized, you'll see the prompt:

```
scala>
```

Type `:paste` to activate paste mode in the shell and paste in the following Scala code:
val readFromSolrOpts = Map(
  "collection" -> "system_logs",
  "fields" -> "host_s,level_s,type_s,message_txt,thread_s,timestamp_tdt",
  "query" -> "level_s:[* TO *]"
)

val logsDF = spark.read.format("solr").options(readFromSolrOpts).load
logsDF.registerTempTable("fusion_logs")
var sqlDF = spark.sql(""
| SELECT COUNT(*) as num_values, level_s as level 
| FROM fusion_logs 
| GROUP BY level_s 
| ORDER BY num_values desc 
| LIMIT 10"".stripMargin)
sqlDF.show(10,false)

Press CTRL+D to execute the script. Your results should resemble these results:

```scala
scala> :paste
// Entering paste mode (ctrl-D to finish)

val readFromSolrOpts = Map(
  "collection" -> "system_logs",
  "fields" -> "host_s,level_s,type_s,message_txt,thread_s,timestamp_tdt",
  "query" -> "level_s:[* TO *]"
)

val logsDF = spark.read.format("solr").options(readFromSolrOpts).load
logsDF.registerTempTable("fusion_logs")
var sqlDF = spark.sql(""
| SELECT COUNT(*) as num_values, level_s as level 
| FROM fusion_logs 
| GROUP BY level_s 
| ORDER BY num_values desc 
| LIMIT 10"".stripMargin)
sqlDF.show(10,false)

// Exiting paste mode, now interpreting.

warning: there was one deprecation warning; re-run with -deprecation for details
+----------+-----+
|num_values|level|
+----------+-----+
|3960      |INFO |
|257       |WARN |
+----------+-----+

readFromSolrOpts: scala.collection.immutable.Map[String,String] = Map(collection -> system_logs, fields -> host_s,level_s,type_s,message_txt,thread_s,timestamp_tdt, query -> level_s:[* TO *])
logsDF: org.apache.spark.sql.DataFrame = [host_s: string, level_s: string ... 4 more fields]

Don't worry about WARN log messages when running this script. They are benign messages from Spark SQL

Congratulations, you just ran your first Fusion Spark job that reads data from Solr and performs a simple aggregation!
The Spark master web UI

The Spark master web UI lets you dig into the details of the Spark job. This handy Mastering Apache Spark guide helps you understand the Spark web UI.

In your browser (http://localhost:8767), there should be a job named "Spark shell" under running applications (the application ID will be different than the following screenshot):

Click the application ID, and then click the Application Detail UI link. You'll see this information about the completed job:

Notice the tabs at the top of the UI that let you dig into details about the running application. Take a moment to explore the UI. It can answer these questions about your application:

• How many tasks were needed to execute this job?
• Which JARs were added to the classpath for this job? (Look under the Environment tab.)
• How many executor processes were used to run this job? Why? (Look at the Spark configuration properties under the Environment tab.)
• How many rows were read from Solr for this job? (Look under the SQL tab.)

For the above run, the answers are:

• 205 tasks were needed to execute this job.
• The Environment tab shows that one of the JAR files is named spark-shaded-*.jar and was "Added By User".
• It took 2 executor processes to run this job. Each executor has 2 CPUs allocated to it and the bin/Spark-shell script asked for 4 total CPUs for the shell application.
• This particular job read about 21K rows from Solr, but this number will differ based on how long Fusion has been running.

The key take-away is that you can see how Spark interacts with Solr using the UI.

Spark job tuning

Returning to the first question, why were 202 tasks needed to execute this job?
Details for Query 2

Submitted Time: 2016/10/05 16:17:41
Duration: 0.2 s
Succeeded Jobs: 2

com.lucidworks.spark.SolrRelation@1f0630a9

Filter
number of input rows: 21486
number of output rows: 21486

Project
number of rows: 21486

TungstenAggregate
number of input rows: 21486
number of output rows: 108
data size total (min, med, max): 8.5 MB (4.2 MB, 4.2 MB, 4.2 MB)
spill size total (min, med, max): 0.0 B (0.0 B, 0.0 B, 0.0 B)
The reason is that SparkSQL defaults to using 200 partitions when performing distributed group by operations; see the property `spark.sql.shuffle.partitions`.

Because our data set is so small, let's adjust Spark so that it only uses 4 tasks. In the Spark shell, execute the following Scala:

```scala
spark.conf.set("spark.sql.shuffle.partitions", "4")
```

You just need to re-execute the final query and `show` command:

```scala
val readFromSolrOpts = Map(
  "collection" -> "logs",
  "fields" -> "host_s,port_s,level_s,message_t,thread_s,timestamp_tdt"
)
val logsDF = spark.read.format("solr").options(readFromSolrOpts).load
logsDF.registerTempTable("fusion_logs")
var sqlDF = spark.sql(""
  | SELECT COUNT(*) as num_values, level_s as level 
  | FROM fusion_logs 
  | GROUP BY level_s 
  | ORDER BY num_values desc 
  | LIMIT 10"").stripMargin
sqlDF.show(10,false)
```

Now if you look at the Job UI, you'll see a new job that executed with only 6 executors! You've just had your first experience with tuning Spark jobs.
Spark Driver Processes

A Spark “driver” is an application that creates a SparkContext for executing one or more jobs in the Spark cluster. The following diagram depicts the driver’s role in a Spark cluster:

In the diagram above, the spark-master service in Fusion is the Cluster Manager.

If your Spark job performs any collect operations, then the result of the collect (or collectAsMap) is sent back to the driver from all the executors. Consequently, if the result of the collect is too big too fit into memory, you will encounter OOM issues (or other memory related problems) when running your job.

All Fusion jobs run on Spark using a driver process started by the API service.

Custom jobs

Fusion supports custom Spark jobs that are written in Scala, Java, or Python jobs and that are built using the Spark Job Workbench, a toolkit provided by Lucidworks. See the examples in the repository for details.

To troubleshoot problems with a custom job, start by looking for errors in the script-job driver log, /opt/fusion/4.2.x/var/log/api/spark-driver-launcher.log (on Unix) or C:\lucidworks\var\fusion\4.2.x\var\log\api\spark-driver-launcher.log (on Windows).

Drivers

Fusion has four types of job drivers:

• Default driver – Executes built-in Fusion jobs, such as a signal aggregation job or a metrics rollup job.

• Script-job driver – Executes custom script jobs; a separate driver is needed to isolate the classpath for custom Scala scripts.

• Spark-shell driver – Wrapper script provided with Fusion to launch the Spark Scala REPL shell with the correct master URL (pulled from Fusion’s API) and a shaded Fusion JAR added. Launched using
Custom-job driver – Executes custom jobs built using the Spark Job Workbench, a toolkit provided by Lucidworks to help build custom Spark jobs using Scala, Java, or Python.

Default driver

Navigate to Collections > Collections Manager and select the system_monitor collection. Then navigate to Collections > Jobs and select one of the built-in aggregation jobs, such as session_rollup. In the diagram above, the spark-master service in Fusion is the Cluster Manager.

You must delete any existing driver applications before launching the job. Even if you haven’t started any jobs by hand, Fusion’s API service might have started one automatically, because Fusion ships with built-in jobs that run in the background which perform rollups of metrics in the system_monitor collection. Therefore, before you try to launch a job, you should run the following command:

```
curl -X DELETE http://localhost:8764/api/spark/driver
```

Wait a few seconds and use the Spark UI to verify that no Fusion-Spark application (for example, Fusion-20161005224611) is running.

In a terminal window or windows, set up a `tail -f` (on Unix, or the equivalent on Windows) on the `api` and `spark-driver-default` logs:

```
tail -f var/log/api/api.log var/log/api/spark-driver-default.log
```

Give this command from the `bin` directory below the Fusion home directory, for example, `/opt/fusion/4.2.x` (on Unix) or `C:\lucidworks\fusion\4.2.x` (on Windows).

Now, start any aggregation job from the UI. It doesn’t matter whether or not this job performs any work; the goal of this exercise is to show what happens in Fusion and Spark when you run an aggregation job. You should see activity in both logs related to starting the driver application and running the selected job. The Spark UI will now show a Fusion-Spark app:

Use the `ps` command to get more details on this process:

```
ps waux | grep SparkDriver
```

The output should show that the Fusion SparkDriver is a JVM process started by the API service; it is not managed by the Fusion agent. Within a few minutes, the Spark UI will update itself:
Notice that the application no longer has any cores allocated and that all of the memory available is not being used (0.0B Used of 2.0 GB Total). This is because we launch our driver applications with `spark.dynamicAllocation.enabled=true`. This setting allows the Spark master to reclaim CPU & memory from an application if it is not actively using the resources allocated to it.

Both driver processes (default and scripted) manage a SparkContext. For the default driver, the SparkContext will be shut down after waiting a configurable (`fusion.spark.idleTime`; default 5 mins) idle time. The scripted driver shuts down the SparkContext after every scripted job is run to avoid classpath pollution between jobs.

**Script-job driver**

Fusion supports custom script jobs.

Script jobs require a separate driver to isolate the classpath for custom Scala scripts, as well as to isolate the classpath between the jobs, so that classes compiled from scripts don’t pollute the classpath for subsequent scripted jobs.

For this reason, the SparkContext that each scripted job uses is immediately shut down after the job is finished and is recreated for new jobs. This adds some startup overhead for scripted jobs.

Refer to the [apachelogs lab](https://example.com) in the [Fusion Spark Bootcamp project](https://example.com) for a complete example of a custom script job.

To troubleshoot problems with a script job, start by looking for errors in the script-job driver log `spark-driver-scripted.log` in `/opt/fusion/4.2.x/var/log/api/` (on Unix) or `C:\lucidworks\var\fusion\4.2.x\var\log\api\` (on Windows).

**Spark drivers in a multinode cluster**

To find out which node is running the Spark driver which node is running the driver when running a multi-node Fusion deployment which has several nodes running Fusion’s API services, you can query the driver status via the following call:

```
curl http://localhost:8764/api/spark/driver/status
```

This returns a status report:
{"/spark-drivers/15797426d56T537184c2": {
    "id": "15797426d56T537184c2",
    "hostname": "192.168.1.9",
    "port": 8601,
    "scripted": false
}}
Spark Configuration

Spark has a number of configuration properties. In this section, we'll cover some of the key settings you'll need to use Fusion's Spark integration.

For the full set of Fusion's spark-related configuration properties, see the Spark Jobs API.

Spark master/worker resource allocation

<table>
<thead>
<tr>
<th>Note</th>
<th>If you co-locate Spark workers and Solr nodes on the same server, then be sure to reserve some CPU for Solr to avoid a compute intensive Spark job from starving Solr of CPU resources.</th>
</tr>
</thead>
</table>

Number of cores allocated

To change the CPU usage per worker, you need to use the Fusion configuration API to update this setting, as in the following example.

```
curl -u user:password -H 'Content-type:application/json' -X PUT -d '6' \n  http://localhost:8764/api/configurations/fusion.spark.worker.cores
```

You can also over-allocate cores to a spark-worker, which usually is recommended for hyper-threaded cores by setting the property spark-worker.envVars to SPARK_WORKER_CORES=<number of cores> in the fusion.properties file on all nodes hosting a spark-worker. For example, a r4.2xlarge instance in EC2 has 8 CPU cores, but the following configuration will improve utilization and performance:

```
spark-worker.envVars=SPARK_WORKER_CORES=16,SPARK_SCALA_VERSION=2.11,SPARK_PUBLIC_DNS=${default.address},SPARK_LOCAL_IP=${default.address}
```

You can obtain the IP address that the Spark master web UI binds to with this API command:

```
curl http://localhost:8765/api/v1/spark/master
```

Tip

<table>
<thead>
<tr>
<th>Tip</th>
<th>We encourage you to set the default.address property in fusion.properties to ensure that all Spark processes have a consistent address to bind to.</th>
</tr>
</thead>
</table>

After making this change to your Spark worker nodes, you must restart the spark-worker process on each.

On Unix:

```
./spark-worker restart
```

Give this command from the bin directory below the Fusion home directory, for example, /opt/fusion/4.2.x.
On Windows:

```bash
spark-worker.cmd restart
```

Give this command from the `bin` directory below the Fusion home directory, for example, `C:\lucidworks\fusion\4.2.x`.

**Memory allocation**

The amount of memory allocated to each worker process is controlled by Fusion property `fusion.spark.worker.memory` which specifies the total amount of memory available for all executors spun up by that Spark Worker process. This is the quantity seen in the memory column against a worker entry in the Workers table.

The JVM memory setting (`-Xmx`) for the spark-worker process configured in the `fusion.properties` file controls how much memory the spark-worker needs to manage executors (and not how much memory should be made available to your job(s)). When modifying the `-Xmx` value, use `curl` as follows:

```bash
curl -u user:password -H 'Content-type:application/json' -X PUT -d '8g' \
    http://localhost:8764/api/configurations/fusion.spark.worker.memory
```

**Tip**

Typically, 512m to 1g is sufficient for the spark-worker JVM process.

The Spark worker process manages executors for multiple jobs running concurrently. For certain types of aggregation jobs you can also configure the per executor memory, but this can impact how many jobs you can run concurrently in your cluster. Unless explicitly specified using the parameter `spark.executor.memory`, Fusion calculates the amount of memory that can be allocated to the executor.

Aggregation Spark jobs always get half the memory of the amount assigned to the workers. This is controlled by the `fusion.spark.executor.memory.fraction` property, which is set to `0.5` by default.

For example, Spark workers have 4 Gb of memory by default and the executors for aggregator Spark jobs are assigned 2 Gb for each executor.

To let Fusion aggregation jobs use more of the memory of the workers, increase `fusion.spark.executor.memory.fraction` property to `1`. Use this property instead of the Spark executor memory property.

```bash
curl -u user:password -H 'Content-type:application/json' -X PUT -d '1' \
    http://localhost:8764/api/configurations/fusion.spark.executor.memory.fraction
```

After making these changes and restarting the workers, when we run a Fusion job, we get the following:
Cores per driver allocation

The configuration property `fusion.spark.cores.fraction` lets you limit the number of cores used by the Fusion driver applications (default and scripted). For example, in the screenshot above, we see 18 total CPUs available.

We set the cores fraction property to 0.5 via the following command:

```bash
curl -u user:password -H 'Content-type:application/json' -X PUT -d '0.5' \ http://localhost:8764/api/configurations/fusion.spark.cores.fraction
```

This cuts the number of available cores in half, as shown in the following screenshot:

Ports used by Spark in Fusion

This table lists the default port numbers used by Spark processes in Fusion.

<table>
<thead>
<tr>
<th>Port number</th>
<th>Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>4040</td>
<td>SparkContext web UI</td>
</tr>
<tr>
<td>7337</td>
<td>Shuffle port for Apache Spark worker</td>
</tr>
<tr>
<td>8767</td>
<td>Spark master web UI</td>
</tr>
<tr>
<td>8770</td>
<td>Spark worker web UI</td>
</tr>
<tr>
<td>Port number</td>
<td>Process</td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>8766</td>
<td>Spark master listening port</td>
</tr>
<tr>
<td>8769</td>
<td>Spark worker listening port</td>
</tr>
<tr>
<td>8772 (spark.driver.port)</td>
<td>Spark driver listening port</td>
</tr>
<tr>
<td>8788 (spark.blockManager.port)</td>
<td>Spark BlockManager port</td>
</tr>
</tbody>
</table>

If a port is not available, Spark uses the next available port by adding 1 to the assigned port number. For example, if 4040 is not available, Spark uses 4041 (if available, or 4042, and so forth).

Ensure that the ports in the above table are accessible, as well as a range of up to 16 subsequent ports. For example, open ports 8772 through 8787, and 8788 through 8804, because a single node can have more than one Spark driver and Spark BlockManager.

**Spark-related directories and files in Fusion**

The following directories and files are for Spark components and logs in Fusion.

**Spark components**

These directories and files are for Spark components:

<table>
<thead>
<tr>
<th>Path (relative to Fusion home)</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>bin/spark-master</td>
<td>Script to manage (start, stop, status, etc.) the Spark Master service in Fusion</td>
</tr>
<tr>
<td>bin/spark-worker</td>
<td>Script to manage (start, stop, status, etc.) the Spark Worker service in Fusion</td>
</tr>
<tr>
<td>bin/sql</td>
<td>Script to manage (start, stop, status, etc.) the SQL service in Fusion</td>
</tr>
<tr>
<td>bin/spark-shell</td>
<td>Wrapper script to launch the interactive Spark shell with the Spark Master URL and shaded JAR</td>
</tr>
<tr>
<td>apps/spark-dist</td>
<td>Apache Spark distribution; contains all JAR files needed to run Spark in Fusion</td>
</tr>
<tr>
<td>apps/spark/hadoop</td>
<td>Hadoop home directory used by Spark jobs running in Fusion</td>
</tr>
<tr>
<td>apps/spark/driver/lib</td>
<td>Add custom JAR files to this directory to include in all Spark jobs</td>
</tr>
<tr>
<td>Path (relative to Fusion home)</td>
<td>Notes</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td><code>apps/spark/lib</code></td>
<td>JAR files used to construct the classpath for the spark-worker, spark-master, and sql services in Fusion</td>
</tr>
<tr>
<td><code>var/spark-master</code></td>
<td>Working directory for the spark-master service</td>
</tr>
<tr>
<td><code>var/spark-worker</code></td>
<td>Working directory for the spark-worker service; keep an eye on the disk usage under this directory as temporary application data for running Spark jobs is saved here</td>
</tr>
<tr>
<td><code>var/spark-workDir-*</code></td>
<td>Temporary work directories are created in when an application is running. They are removed after the driver is shut down or closed.</td>
</tr>
<tr>
<td><code>var/sql</code></td>
<td>Working directory for the SQL service</td>
</tr>
<tr>
<td><code>var/api/work/spark-shaded-*</code>.jar</td>
<td>The shaded JAR built by the API service; contains all classes needed to run Fusion Spark jobs. If one of the jars in the Fusion API has changed, then a new shaded jar is created with an updated name.</td>
</tr>
</tbody>
</table>

**Spark logs**

These directories and files are for configuring and storing Spark logs:

Path (relative to Fusion home)

Notes

**Log configuration**

- `conf/spark-master-log4j2.xml`
  
  Log configuration file for the spark-master service

- `conf/spark-worker-log4j2.xml`
  
  Log configuration file for the spark-worker service

- `conf/spark-driver-log4j2.xml`
  
  Log configuration file for the Spark Driver application launched by Fusion; this file controls the log settings for most Spark jobs run by Fusion

- `conf/spark-driver-scripted-log4j.xml`
  
  Log configuration file for custom script jobs and Parallel Bulk Loader (PBL) based jobs

- `conf/spark-driver-launcher-log4j2.xml`
Log configuration file for jobs built using the Spark Job Workbench

`conf/spark-executor-log4j2.xml`

Log configuration file for Spark executors; log messages are sent to STDOUT and can be viewed from the Spark UI

`conf/sql-log4j2.xml`

Log configuration file for the Fusion SQL service

Logs

`var/log/spark-master/*`

Logs for the `spark-master` service

`var/log/spark-worker/*`

Logs for the `spark-worker` service

`var/log/sql/*`

Logs for the `sql` service

`var/log/api/spark-driver-default.log`

Main log file for built-in Fusion Spark jobs

`var/log/api/spark-driver-scripted.log`

Main log file for custom script jobs

`var/log/api/spark-driver-launcher.log`

Main log file for custom jobs built using the Spark Job Workbench

**Connection configurations for an SSL-enabled Solr cluster**

You'll need to set these Java system properties used by SolrJ:

- `javax.net.ssl.trustStore`
- `javax.net.ssl.trustStorePassword`
- `javax.net.ssl.trustStoreType`

For the following Spark configuration properties:

- `spark.executor.extraJavaOptions`
- `fusion.spark.driver.jvmArgs`
- `spark.driver.extraJavaOptions`
> curl -H 'Content-type:application/json' -X PUT \\
-\d '\-Djavax.net.ssl.trustStore=/opt/app/jobs/ssl/solrtrust.jks -Djavax.net.ssl.trustStorePassword=changeit \\
-\Djavax.net.ssl.trustStoreType=jks' \\
"http://localhost:8764/api/configurations/spark.executor.extraJavaOptions"

> curl -H 'Content-type:application/json' -X PUT \\
-\d '\-Djavax.net.ssl.trustStore=/opt/app/jobs/ssl/solrtrust.jks -Djavax.net.ssl.trustStorePassword=changeit \\
-\Djavax.net.ssl.trustStoreType=jks' \\
"http://localhost:8764/api/configurations/fusion.spark.driver.jvmArgs"

> curl -H 'Content-type:application/json' -X PUT \\
-\d '\-Djavax.net.ssl.trustStore=/opt/app/jobs/ssl/solrtrust.jks -Djavax.net.ssl.trustStorePassword=changeit \\
-\Djavax.net.ssl.trustStoreType=jks' \\
"http://localhost:8764/api/configurations/spark.driver.extraJavaOptions"
Scaling Spark Aggregations

Consider the process of running a simple aggregation on 130M signals. For an aggregation of this size, it helps to tune your Spark configuration.

Speed up tasks and avoid timeouts

One of the most common issues encountered when running an aggregation job over a large signals data set is task timeout issues in Stage 2 (foreachPartition). This is typically due to slowness indexing aggregated jobs back into Solr or due to JavaScript functions.

The solution is to increase the number of partitions of the aggregated RDD (the input to Stage 2). By default, Fusion uses 25 partitions. Here, we increase the number of partitions to 72. Set these configuration properties:

- `spark.default.parallelism` – Default number of partitions in RDDs returned by transformations like `join`, `reduceByKey`, and `parallelize` when not specified by the user:

  curl -u user:password -H 'Content-type:application/json' -X PUT -d '72' "$FUSION_API/configurations/spark.default.parallelism"

- `spark.sql.shuffle.partitions` – Number of partitions to use when shuffling data for joins or aggregations.

  curl -u user:password -H 'Content-type:application/json' -X PUT -d '72' "$FUSION_API/configurations/spark.sql.shuffle.partitions"

After making these changes, the `foreachPartition` stage of the job will use 72 partitions:

Increase rows read per page

You can increase the number of rows read per page (the default is 10000) by passing the rows parameter when starting your aggregation job; for example:
For example, we were able to read 130M signals from Solr in 18 minutes at ~120K rows/sec using rows=20000 vs. 21 minutes using the default 10000.

**Improve job performance**

You can increase performance when reading input data from Solr using the `splits_per_shard` read option, which defaults to 4. This configuration setting governs how many Spark tasks can read from Solr concurrently. Increasing this value can improve job performance but also adds load on Solr.
Spark Troubleshooting

This article contains tips and techniques for troubleshooting Spark.

Log API endpoints for Spark jobs

Log endpoints are useful for debugging Spark jobs on multiple nodes. In a distributed environment, the log endpoints parse the last N log lines from different Spark log files on multiple nodes and output the responses from all nodes as text/plain (which renders nicely in browsers) sorted by the timestamp.

The REST API Reference documents log endpoints for Spark jobs. The URIs for the endpoints contain /api/spark/log.

The most useful log API endpoint is the spark/log/job/ endpoint, which goes through all Fusion REST API and Spark logs, filters the logs by the jobId (using MDC, the mapped diagnostic context), and merges the output from different files.

For example, to obtain log content for the job jobId:

```
curl -u user:password "$FUSION_API/spark/log/job/jobId"
```

Note

Log endpoints will only output data from log files on nodes on which the API service is running.

Specific issues

These are some specific issues you might encounter.

Job hung in waiting status

Check the logs for a message that looks like:

```
2016-10-07T11:51:44,800 - WARN  [Timer-0:Logging$class@70] - Initial job has not accepted any resources; check your cluster UI to ensure that workers are registered and have sufficient resources
```

If you see this, then it means your job has requested more CPU or memory than is available. For instance, if you ask for 4g but there is only 2g available, then the job will just hang in WAITING status.

Lost executor due to heartbeat timeout

If you see errors like the following:

```
```

2016-10-07T11:51:44,800 - WARN  [Timer-0:Logging$class@70] - Initial job has not accepted any resources; check your cluster UI to ensure that workers are registered and have sufficient resources
```
This is most likely due to an OOM in the executor JVM (preventing it from maintaining the heartbeat with the application driver). However, we’ve seen cases where tasks fail, but the job still completes, so you’ll need to wait it out to see if the job recovers.

Another situation when this can occur is when a shuffle size (incoming data for a particular task) exceeds 2GB. This is hard to predict in advance because it depends on job parallelism and the number of records produced by earlier stages. The solution is to re-submit the job with increased job parallelism.

**Spark Master won't start on EC2**

See [this article](#) for a solution.
Monitoring

Fusion Server has functions built in for observability and monitoring with Fusion’s tools or third-party monitoring tools.
Fusion system metrics

Fusion system metrics are logged and available in the UI, in log files, and through the REST API.

- For an overview of noteworthy metrics that are logged and indexed, see System Metrics.
- See System Administration with the Fusion UI.
- Read about System Admin APIs, including APIs for metrics and cluster status, in the REST API Reference Guide.
Additional tools for monitoring Fusion

Read about health checks for components of your Fusion deployment in Checking System State.

Read about using notifications or alerts in Messaging Service.
For more information

In the Lucidworks Knowledge Base:

- An example using Zabbix to monitor Fusion.
- General guidelines about monitoring Solr.

From our blogs:

- An example of using a third-party tool, Pager Duty, to monitor Fusion.
Checking System State

As described in Fusion Server Directories, Files, and Ports in the Deployment Guide, Fusion runs several components as separate JVMs running on different ports. Each of the components is capable of reporting its status. The proxy component reports status for all of the other components.
Full System Check

To see whether each component has been started, a simple API call to the proxy (running on port 8764 by default) will return the status of each component of the system.

curl http://localhost:8764/api

The response should look similar to the following. If 'ping' is true for each service, all of the system components are running.

```json
{
  "status": {
    "db": {"ping":true},
    "apolloZk": {"ping":true},
    "apollo": {"ping":true},
    "connectors": {"ping":true},
    "fusion": {"ping":true},
    "startTime": "2018-02-16T16:23:51Z",
    "initMeta": {"initializedAt": "2018-02-15T21:50:43Z", "version": "local",
    "nativeRealmInstalled": "2018-02-15T21:53:44Z"},
    "enabledRealms": [{"name": "native", "realmType": "native"}, {"name": "webapps-jwt-realm", "realmType": "jwt"}],
    "version": "local"
  }
}
```
Solr Health Check

The Fusion UI and API services are not accessible if ZooKeeper and Solr are not in healthy state. A Solr health check can be performed with a ping request, for example:

curl http://localhost:8983/solr/collection-name/admin/ping

The response will be a JSON response, similar to the following:

```json
{
  "responseHeader": {
    "zkConnected": true,
    "status": 0,
    "QTime": 0,
    "params": {
      "q": "{!lucene}*:*",
      "distrib": "false",
      "df": "_text_",
      "rows": "10",
      "echoParams": "all"
    }
  },
  "status": "OK"
}
```

The **status** should be **OK** if Solr is able to return results for queries.
REST API Services Health Check

All of the Fusion API backend services (except Connectors and the UI) are started at port 8765 when the run.sh script is executed. The Fusion UI depends on all these services.

If all the services are started without any issues, then the below ping request should return the response `ok`.

```
curl http://localhost:8765/api/v1
```

As an alternative check, you can also query the `system/status` endpoint, which returns the status.

```
curl http://localhost:8764/api/system/status
```

The response looks like something like this:

```json
[ {
  "status" : "ok",
  "node" : "http://127.0.0.1:8984/connectors/v1/system",
  "messages" : [ ]
}, {
  "status" : "ok",
  "node" : "http://127.0.0.1:8764/api/system",
  "messages" : [ ]
} ]
```
Connectors Classic Health Check

You can perform a health check of the Connectors Classic Service by sending a ping request to port 8984. Similar to the previous ping request, the returned response is `ok` if the service started successfully.

```
curl http://localhost:8984/connectors/v1
```

As an alternative check, you can also query the `system/status` endpoint, which returns the status.

```
curl http://localhost:8984/connectors/v1/system/status
```

The response looks like something like this:

```
[ {
    "status" : "ok",
    "node" : "http://127.0.0.1:8984/connectors/v1/system",
    "messages" : [ ]
}, {
    "status" : "ok",
    "node" : "http://127.0.0.1:8764/api/system",
    "messages" : [ ]
} ]
```
System Metrics

By default, Fusion captures system metrics in the `var/log/metrics/metrics.log` file, then indexes them asynchronously in the `system_monitor` system collection:

- Host/server metrics (CPU, memory, disk space usage, and so on)
- Service metrics (process CPU, Java heap memory usage, and so on)

You can view these metrics in the DevOps Center.

Configuration

Most aspects of metrics collection can be configured in the `fusion.properties` file:

- Metrics collection can be disabled with `default.collectMetrics = false`
- The frequency of metrics collection can be adjusted with `default.collectMetricsIntervalSecs = 30`
- Metrics can be shipped to a different Solr cluster or collection by adjusting the `log-shipper.solrZk.connect` and `log-shipper.metricsSolrCollection` properties.

The retention period for system metrics is 30 days by default and can be configured in the `delete-old-system-metrics` Fusion task job, available in all apps.

How to adjust the system metrics retention period

1. Navigate to Collections > Jobs.
2. Select the `delete-old-system-metrics` job.
3. In the job configuration pane, scroll down to REQUEST ENTITY (AS STRING).
4. Change `30DAYS` to the desired period of time to retain system metrics.
Metrics document fields

Both host and service metrics are stored as a single Solr document with a timestamp and the fields described below.

All metrics

- **id**
  Unique autogenerated document identifier.

- **node_s**
  Unique identifier of a Fusion node / server (autogenerated).

- **timestamp_tdt**
  Timestamp of a metric.

- **view_s**
  Type of a metric, either "host" or "service_instance".

- **type_s**
  Store type of a metric, either "latest" or "history".
  - "history" is a snapshot of a metric at a particular time.
  - "latest" is a single document per host or service with the latest state that is constantly updated over time. It allows easy retrieval and aggregation for just the "latest" / "recent" state of the system.

Host metrics

- **CPU**
  - **cpu_load_d**
    Normalized CPU load, such as a floating number value in the range of \([0.0,1.0]\).
  - **cpu_sys_d**, **cpu_user_d**, **cpu_wait_d**, **cpu_combined_d** and **cpu_idle_d**
    Break down of CPU load per type. Those are also floating number values in the range of \([0.0,1.0]\).
  - **load_average_d**
    System load average for the last minute, not normalized.
  - **processors_l**
    Number of CPU cores according to JVM.

- **Memory**
  - **memory_total_l** and **memory_free_l**
    Total and free amounts of physical memory in bytes.
  - **swap_total_l** and **swap_free_l**
Total and free swap in bytes.

- Disk space
  - `disk_total_l` and `disk_free_l`
  
  Disk sizes of a partition where Fusion is installed (where `var` and `data` folders reside).

- Uptime
  - `host_uptime_l`
  
  Total uptime of a host operating system (in milliseconds).
  - `agent_uptime_l`
  
  Uptime of Fusion agent service (in milliseconds).

- Various Info
  - `os_name_s`, `os_arch_s` and `os_version_s`
  
  OS details according to JVM.
  - `addresses_ss`
  
  List of IP addresses according to network configuration.
  - `hostname_s`
  
  Main hostname or IP address of a server.

**Service metrics**

- `service_s`
  
  The name of the service (that is, `api`, `solr`, and so on) to which this metric pertains.

- `status_s`
  
  Status of the service according to Agent (that is, `RUNNING`, `STARTING`, and so on).

- `pid_i`
  
  Process ID.

- `address_s`
  
  IP address or hostname that is configured for this service to run on (or the default).

- Generic Java Metrics
  - `java_process_cpu_load_d`
    
    Normalized CPU load used by this service.
  
  - `java_heap_max_l`, `java_heap_used_l` and `java_non_heap_used_l`
    
    JVM memory metrics.
- **java_open_file_descriptors_i**
  Number of open files according to JVM.

- **java_loaded_classes_i** and **java_unloaded_classes_i**
  JVM class loading metrics, useful for spotting problems with dynamic redeployment of Web applications.

- **java_threads_i**
  Total JVM threads.

- **gc_collection_count_l** and **gc_collection_time_l**
  GC metrics like number of invocations and total time spent.

- **Jetty Metrics**
  All Jetty based services provide low-level Jetty metrics such as the following:

  - **jetty_request_time_mean_f**
    Mean request time according to Jetty.

  - **jetty_threads_i**
    Number of Jetty threads

  - **jetty_responses_5xx_l**, **jetty_responses_4xx_l**, and so on
    Number of responses per status.

- **Solr Metrics**
  - **solr_index_size_l**
    Total Solr index size in bytes hosted on a Solr node.

  - **solr_docs_l**
    Total number of Solr documents hosted on a Solr node.

  - **solr_requests_l**
    Total number of Solr requests to all cores on a Solr node.

- **ZooKeeper Metrics**
  - **zk_connections_i**
    Number of ZooKeeper connections to ZooKeeper node.

  - **zk_znodes_l**
    Number of ZooKeeper nodes.

  - **zk_watches_i**
    Number of Zk watches.
Number of ephemeral ZooKeeper nodes.

ZooKeeper size in bytes.

- API metrics
  - api_query_pipelines_http_one_minute_rate_f, api_query_pipelines_http_mean_f, and so on
    Query pipeline metrics like rate of query requests to the HTTP endpoint or to Solr and mean response times.
  - api_index_pipelines_http_one_minute_rate_f, api_index_pipelines_http_mean_f, and so on
    Index pipeline metrics like rate of index requests to the HTTP endpoint or to Solr and mean response times.

- Proxy metrics
  - proxy_active_sessions_l
    Number of active auth sessions.
  - proxy_sessions_one_minute_rate_f
    Rate of new auth sessions per minute (per node). This metric is captured once per second, then presented as a moving average over the last minute.

Legacy metrics collection

The metrics collection features described below are deprecated in Fusion 4.2 and will be removed in a future release.

In version 4.1 and earlier, Fusion automatically creates the system collection system_metrics. It is empty until you manually enable system_metrics indexing; see below for instructions. In version 4.2 and above, Fusion creates the collection when you enable system_metrics indexing.

| Note | The jobs that produce legacy metrics are not automatically linked to your app when you enable legacy metrics, so they will not automatically appear in the Jobs panel. You can find them in the Object Explorer and link them to your app. |

How to enable metrics indexing in the Fusion UI

1. Navigate to System > System > Metrics.
2. Enable Record System Metrics Over Time.

How to enable metrics indexing using the REST API

```
curl -u admin:password123 -H 'Content-type:application/json' -X PUT -d 'true'
"http://localhost:8764/api/configurations/com.lucidworks.apollo.metrics.enabled"
```

There are around 600 different metrics available. In this topic we've highlighted a few that are likely to be the most
useful or interesting to you.

The `/system/metrics` endpoint of the System API lists all the metrics that the system is currently collecting. Metrics are returned for the current instance only; Fusion instances do not aggregate metrics between nodes.

**Types of Metrics Collected**

There are several types of metrics:

- **Gauges**: These are single values, valid for the point in time at which the metrics are collected.
- **Counters**: These are values that are incremented or decremented over time.
- **Meters**: These measure the rate of events over time. They include a mean rate, as well as a 1-, 5- and 15-minute moving average. Most of these moving averages are exponentially weighted, so that more recent values contribute more heavily than older values; exceptions to this rule have the word "unweighted" in their name.
- **Histograms**: These measure the distribution of values. They will report the minimum, maximum, mean, and the values at the 50th, 75th, 95th, 98th, 99th, and 99.9th percentiles.
- **Timers**: A timer is a meter combined with a histogram; it measures the length of time that a particular operation takes (both mean duration and moving averages) as well as the distribution of those durations.

Many of the metrics are for internal use by the system. However, Fusion may ask for a dump of the metrics data (using the System API endpoint) to help diagnose performance issues. Some metrics are also subject to change pending performance tuning and additional testing.

**Metrics of Particular Interest**

**Slow Web Service Calls**

For each web service endpoint in the system, the system keeps a list of the last several requests whose request time has been in the 99th percentile – that is, examples of the top 1% of slow requests for that endpoint. These are recorded as `com.lucidworks.apollo.resources.serviceName.methodName.weighted.slow.examples`, where serviceName is the name of the service and methodName is the name of a valid method for that service.

This information might be helpful when diagnosing performance issues. Here is an example of the 5 slowest calls to the `getCollectionMetrics` method of the `CollectionResource` service:
"com.lucidworks.apollo.resources.CollectionResource.getCollectionMetrics.weighted.slow.examples" : {
    "value" : [ {
        "requestUri" : "http://localhost:8764/api/collections/lws5_metrics/stats",
        "queryParams" : { },
        "userPrincipal" : null,
        "method" : "GET",
        "cookies" : { }
    }, {
        "queryParams" : { },
        "userPrincipal" : null,
        "method" : "GET",
        "cookies" : { }
    }, {
        "queryParams" : { },
        "userPrincipal" : null,
        "method" : "GET",
        "cookies" : { }
    }, {
        "requestUri" : "http://localhost:8764/api/collections/lws5_metrics/stats",
        "queryParams" : { },
        "userPrincipal" : null,
        "method" : "GET",
        "cookies" : { }
    }, {
        "requestUri" : "http://localhost:8764/api/collections/lws5_metrics/stats",
        "queryParams" : { },
        "userPrincipal" : null,
        "method" : "GET",
        "cookies" : { }
    } ]
}

System Memory

There are several memory-related metrics reported:

- **mem.heap.used**: the current amount of heap memory, in bytes, used by the system.
- **mem.heap.max**: the maximum amount of heap memory, in bytes, that the system could use.
- **mem.heap.usage**: the percentage (0 - 1.0) of available heap memory that the system is currently using (this is equal to \( \frac{\text{mem.heap.used}}{\text{mem.heap.max}} \)).
- **mem.non-heap.used**: the current amount of non-heap memory (also called "off-heap memory"), in bytes, used by the system.
- **mem.non-heap.max**: the maximum amount of non-heap memory, in bytes, that the system could use.
- **mem.non-heap.usage**: the percentage (0 - 1.0) of available non-heap memory that the system is currently using (this is equal to \( \frac{\text{mem.non-heap.used}}{\text{mem.non-heap.max}} \)).
- **mem.total.used**: the current total amount of memory (heap plus non-heap), in bytes, used by the system.
- **mem.total.max**: the maximum amount of total memory (heap plus non-heap), in bytes, that the system could use.

Here is an example of **mem.heap.used**: 
Query and Index Pipeline Stage Metrics

For each query pipeline and index pipeline stage, Fusion collects aggregate performance metrics for successful executions and for errors. All executions for each stage are stored in a metric named `stages.stageType.stageName.process`, where stageType is the type of stage, and stageName is the name of a specific stage.

Here is an example of a request to get the performance metrics for an index pipeline stage named 'solr-default' (`stages.solr-index.solr-default.process`), which is included with Fusion:

```json
{"version": "3.0.0",
"gauges": {},
"counters": {},
"histograms": {},
"meters": {},
"timers": {
"stages.solr-index.solr-default.process": {
"count": 109195,
"mean": 0.128585,
"max": 0.004011065175097276,
"min": 0.0022500000000000003,
"p50": 0.0030645000000000004,
"p75": 0.0033495,
"p95": 0.005410449999999992,
"p98": 0.014195759999999965,
"p99": 0.02462230000000001,
"p999": 0.12850243700000002,
"stddev": 0.007408363728123277,
"rate": 11.957732876922531,
"rate": 8.78428947811962,
"rate": 9.037172472578138,
"mean_rate": 9.214233776748047,
"duration_units": "seconds",
"rate_units": "calls/second"
}
}
```

This shows the number of uses of the stage ("count"), the maximum and minimum times, the mean, the 50th, 75th, 95th, 98th, 99th, and 99.9th percentiles (p50, p75, and so on.), and the mean rates over 1-, 5- and 15-minute intervals ('m1_rate', and so on.). In this case, the pipeline has been used 109,195 times, with a mean rate of 9.214 events per second, with only .003 events in the 50th percentile.
Metrics for successful completions of stages are stored in metrics named `stages.index.stageType.stage.stageName.ok` or `stages.query.stageType.stage.stageName.ok`, depending on if the stage is part of an index pipeline or a query pipeline. Here is an example of the mean rates for successful runs of the 'solr-default' index pipeline stage (`stages.index.solr-index.stage.solr-default.ok`):

```json
{
    "version" : "3.0.0",
    "gauges" : { },
    "counters" : { },
    "histograms" : { },
    "meters" : {
        "stages.index.solr-index.stage.solr-default.ok" : {
            "count" : 110855,
            "m15_rate" : 5.270163206842968,
            "m1_rate" : 8.485969925086419,
            "m5_rate" : 8.0678529981572,
            "mean_rate" : 9.18230056255745,
            "units" : "events/second"
        }
    },
    "timers" : { }
}
```

This shows the number of uses of the stage ("count") and the mean rates over 1-, 5- and 15-minute intervals ('m1_rate', and so on.). From the above, we can see that the solr-default stage has been executed 110,855 times, with a mean rate of 9.18 events per second.

If you prefer to see the metrics for the entire stage type, you can omit the stage name entirely, and simply get metrics for the stage type. This takes the form of `stages.index.stageType.ok` (for an index pipeline) or `stages.query.stageName.ok` (for a query pipeline). Here is an example, using the solr-index stage type:

```json
{
    "version" : "3.0.0",
    "gauges" : { },
    "counters" : { },
    "histograms" : { },
    "meters" : {
        "stages.index.solr-index.ok" : {
            "count" : 116425,
            "m15_rate" : 6.178851947720613,
            "m1_rate" : 8.814380052133192,
            "m5_rate" : 8.585203640734829,
            "mean_rate" : 9.19499774409566,
            "units" : "events/second"
        }
    },
    "timers" : { }
}
```

In this example, we see that the solr-index stage has been successfully run 116,425 times, with a mean rate of 9.19 events per second.
Web Service Endpoint Metrics

For each web service endpoint, Fusion keeps a timer recording the duration and rate of requests. The duration is calculated using an exponentially-weighted moving average with a heavy bias toward measurements from the last 5 minutes.

These metrics have names in the form: `com.lucidworks.apollo.resources.serviceName.methodName.weighted.timer`, or for a specific example, `com.lucidworks.apollo.resources.CollectionResource.getCollectionMetrics.weighted.timer`:

```
"com.lucidworks.apollo.resources.CollectionResource.getCollectionMetrics.weighted.timer" : {
    "count" : 2624,
    "max" : 0.134712,
    "mean" : 0.031589107976653694,
    "min" : 0.022424000000000003,
    "p50" : 0.028440000000000003,
    "p75" : 0.036908,
    "p95" : 0.044644499999999995,
    "p98" : 0.05026944,
    "p99" : 0.05448510000000004,
    "p999" : 0.134693411,
    "stddev" : 0.00936497282768644,
    "m15_rate" : 0.07113433590025664,
    "m1_rate" : 0.06387037028343223,
    "m5_rate" : 0.06218407166715861,
    "mean_rate" : 0.0663172057583814,
    "duration_units" : "seconds",
    "rate_units" : "calls/second"
}
```

Solr Request Metrics

The system keeps track of the performance of requests to each Solr server that it communicates with.

The metrics have names in the form `solr.solrIdentifier.requestType`. The solrIdentifier is the address of the Solr instance, and the requestType can be 'get-requests', 'post-requests' or 'put-requests'.

This example shows get-requests to a Solr instance that is found on '10.0.1.8' and port 8983:
From this we can see that there have been 3,170 GET requests to that Solr instance, and the mean response rate is .03 requests per second.

**Changing Metric Collection Frequency**

The default frequency to collect metrics is 60 seconds. Since the metrics are stored in a system collection (and a Solr instance), the data can grow to be quite large over time. If you do not need metrics collection to happen as frequently (perhaps during initial implementation), you can change the frequency by modifying the `com.lucidworks.apollo.metrics.poll.seconds` configuration parameter with the Configurations API.

For example:

```
curl -u user:pass -X PUT -H 'Content-type: application/json' -d '600'
http://localhost:8764/api/configurations/com.lucidworks.apollo.metrics.poll.seconds
```

To disable metrics, you could set the `com.lucidworks.apollo.metrics.poll.seconds` parameter to `-1`.

```
curl -u user:pass -X PUT -H 'Content-type: application/json' -d '-1'
http://localhost:8764/api/configurations/com.lucidworks.apollo.metrics.poll.seconds
```
Messaging and Alerting

Fusion’s messaging services provide implementations to send messages and alerts to any application or device capable of displaying the supported message types. Read a primer on Fusion’s messaging services on our blog.

Supported Message Types

Fusion supports these types of messages and alerts:

- **logging**

  This service logs any message sent to it in the configured logger. You can use it in an index pipeline or a query pipeline.

- **Slack**

  Slack is a team messaging service with document integration and a focus on collaborative communication. See the Slack Index Stage and the Slack Query Stage.

- **SMTP**

  Email, via the Simple Mail Transfer Protocol. See the Email Index Stage and the Email Query Stage.

- **PagerDuty**

  Alerting and monitoring. See the pagerduty index stage and the PagerDuty Query Stage.

Messaging Service Configuration

The Message Services as a whole can be configured via the Configurations API with these attributes:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rateLimit</td>
<td>The time, in milliseconds, to wait between sending messages on a per-second basis. This does not synchronize throttling between requests.</td>
</tr>
<tr>
<td>storeAllMessages</td>
<td>Boolean flag that indicates whether messages should be indexed and stored. By default, only scheduled messages are stored, as they need to be retrieved by the scheduler at a later time. Storing all messages can be useful for auditing the system, but it will have an impact on the system storage requirements.</td>
</tr>
</tbody>
</table>

Enabling Messaging Services

The logging service is enabled by default, but Slack, email, and PagerDuty messaging services must be explicitly enabled.

You can do this through the UI at System > System > Messaging Services, or through the Messaging API.

To see which messaging services are currently enabled:
String Templates

String templates are libraries used for structured text generation outputs. They are a powerful way of doing variable substitution into a provided template using values contained in documents, requests, and contexts. String templates are made available in the Messaging Service System setup, where users can fill in these portions with document or query values from the working system.

See Messaging Services Templates for details.

Triggering Messages and Alerts

The messaging services can be invoked in several ways:

- Via the Scheduler API to send messages at designated intervals.
- Via the Messaging API.
- Through an index or query pipeline; see Messaging and Alerting Pipeline Stages below.

| Note | By default, only scheduled messages are stored. To configure Fusion to store all messages, see Messaging Service Configuration above. The default collection for message storage is system_messages, which is created on startup. |

Messaging and Alerting Pipeline Stages

The Messaging Service supports these pipeline stages:

- Email Message Query Stage
- Email Message Index Stage
- PagerDuty Message Query Stage
- PagerDuty Message Index Stage
- Slack Message Query Stage
- Slack Message Index Stage
- SetPropertyIndex Stage

The pipeline stages above send messages and alerts when specific conditions are met. Conditions can be specified using regular expressions, database lookups, and more. Any upstream stage can affect how Fusion behaves when a match occurs, so pay special attention to the order in which stages occur.

Additionally, the Set Property Index Stage allows conditions to be specified before messages are sent.

Setting Properties Upstream

Fusion includes two index pipeline stages that are useful for setting properties on indexed documents so that you can evaluate those properties in one of the messaging stages, either downstream in the index pipeline or in a query pipeline:
• The Set Property Index Stage can be used to set a property on a document, or a context, by evaluating one or more simple conditions. It is an index-only, conditional stage that allows the setting of properties without the use of JavaScript.

• The JavaScript Index Stage provides a more sophisticated means of setting properties upstream from the messaging stages.
Dashboards

Dashboards are Fusion's built-in analytics component. Most of the components of dashboards, called panels, are for data visualization. Some panels display textual information; for example, records in a table or documents.

| Note | Additional analytics functionality is available in the DevOps Center and App Insights. A Fusion AI license is needed to enable App Insights. |

Dashboards permit near-real-time views of data. For some types of panels, you can configure how frequently the panel gets new data.

Dashboards also permit users to interact with data. For example, users can submit queries, select facets, and zoom in and out on histograms.

A Fusion dashboard contains one or more controls for search query inputs and one or more quantitative displays over the results for that query.

Dashboards run as a client-side application in a web browser, using JavaScript components for HTML5. Client-side JavaScript provides a dynamic, responsive browsing experience. Solr facets provide the quantifications required for visualizations, for example, charts, graphs, tables, and maps (for geospatial data). Dashboards can also have tabular displays for drilling down to the individual documents in a results set.

The underlying browser application is Banana, an open-source visualization tool based on Kibana 3. Where Kibana communicates with Elasticsearch, Banana communicates with Solr. Banana can be downloaded from: https://github.com/LucidWorks/banana/.

Fusion dashboards were first developed for log file analysis. Log file entries contain a timestamp plus some amount of additional event information. Fusion's date/time processing aggregates these timestamps at different levels of granularity allowing for visualization of historical trends in system and user behaviors.

See these additional topics for details:

- Create Dashboards
- Use Dashboards
- Manage Dashboards
- Input Panels
- Display Panels:leveloffset: +1
Create Dashboards

Create dashboards that meet the needs of users.

| Tip | When you create a dashboard, you don’t build and build and build some more and then, at the end of the process, see if it works. If you configure the dashboard to use existing data and add a Filtering panel, then panels in the dashboard work as you add panels and configure them. |
Requirements for a dashboard

Requirements for a dashboard are:

- **Input panel(s)** – You must add one or more input panels that let you specify which data the dashboard will display.
- **Filtering panel** – You must add a filtering panel to ensure that the dashboard functions correctly.
- **Output panel(s)** – You must add one or more output panels to display the data.
Steps for creating a dashboard

The steps for creating a dashboard are:

1. Choose the best starting point.
2. Configure dashboard settings.
3. Lay out the dashboard. A dashboard consists of rows and panels, and possibly columns (if you use Column panels).
4. Configure the panels as you add them. Below we present general tasks that apply to all panels and tasks that apply to specific types of panels.
5. Save the dashboard.
6. Test the dashboard.
1. Choose a starting point

Choose a starting point for your dashboard:

- **A blank dashboard** – Build your dashboard from scratch.
- **An existing built-in dashboard** – Leverage an existing dashboard and modify it to meet your needs.

To open the dashboard

1. From the Fusion workspace, click Analytics ➔ **Dashboards** or System ➔ **Log Viewer**.

2. In the upper right corner of the default dashboard, click the link for the dashboard you want to use as a starting point.
2. Configure dashboard settings

Configure dashboard settings, which affect all panels on a dashboard.

1. **Choose where Fusion sends queries from a dashboard** – Fusion can send queries to a Fusion query pipeline or to Solr. At the top right of the dashboard, click **Configure dashboard**. On the Solr tab, select **Use Fusion** to use a Fusion query pipeline. Deselect **Use Fusion** to send queries to Solr.

2. **Choose a default collection to query** – On the **Solr** tab, specify the name of the collection to query. Fusion queries this collection when the dashboard opens.

   To use the same dashboard for multiple collections, let the dashboard user choose a collection. On the **Controls** tab, select **Show Collections Picker**.

3. **Specify global query parameters to append to all dashboard queries** (optional). On the **Solr** tab, enter **Global Query Parameters**. For example, to set the default search field to the field `message_t`, specify the global query parameter `&df=message_t`. 
3. Lay out the dashboard

- Add a row to a dashboard – Click Add a Row.

- Configure a row – At the left of the row, click Configure row.

- Add a panel to an empty row – Click Add panel to empty row. Click Add panel and select the Panel Type. Configure the panel, and then click Add Panel.

- Add a panel to a row that already has panels – Click Add a panel to this row. Click Add panel and select the Panel Type. Configure the panel, and then click Add Panel.

- See that a row is full – On the left side of a row, the Row Full indicator shows you that the row is full.

- Resize a panel Click Configure. On the General tab, choose a Span (width) between 1 (one twelfth) and 12 (twelve twelfths).

- Drag a panel to a different location Click-and-hold the panel type (in the upper right corner of the panel), and drag the panel to an empty location or on top of a different panel (which moves out of the way).

- Remove a panel – At the top right of the panel you want to remove, click Remove Panel.

- Place panels in a column – A Column panel lets you lay out panels in a column.

- Add text – A Text panel lets you add text to a dashboard, for example, for user instructions (how to use the dashboard to interact with data), descriptions of content (what am I seeing), etc.
4. Configure panels

- **Configure a panel** – At the top right of the panel, click **Configure**. The primary tab for configuration is **Panel**. This is what the tabs let you do. Not all tabs are present for all panel types.
  - General – The name and width of the panel, whether it is editable, and whether the panel will let users inspect the queries.
  - Panel – The primary tab for configuring the panel. This configuration affects how data is displayed, but also in some cases, which data is displayed.
  - Info – Optionally, add a help message that Fusion displays when a user inspects the queries for a panel.
  - Fields – Add and remove fields in a Table panel.
  - Paging – Control pagination of a Table panel.
  - Queries – View the panel query. Optionally, add one or more custom Solr query parameters. Fusion adds these to the panel query.

For the Histogram and Range Facet panels, you can perform some configuration by clicking View. Click View ▼ to hide the configuration.

- **Display help for a setting** – Hover over the Help icon.
- **Inspect a query** – At the top right of a panel, click **Inspect** to inspect the query that a panel sends to Solr. Fusion displays both the panel query and the custom query (if any).
- **Make a panel not editable** – Click **Configure**. On the **General** tab, deselect the **Editable** check box.
- **Make a panel editable** – Remove the panel, and then add it back. Click **Remove Panel**, and then click **Add panel to empty row** or **Add Panel** to add the panel back.
5. Configure input panel settings

What goes into input panels determines what comes out on display panels. This determination has these parts: – panel configuration, panel contributions to the query, and user interactions with the panels.

- **Add a custom query for a panel** (optional; only possible for some panel types) – Some panels let you specify a custom query to add to the panel query. The combined query is sent to the Fusion query pipeline or to Solr.

- **Filter** – Filters select subsets of the data in a collection, for example, records in a time range or in which the field Gender has the value Female. A filtering panel is required for data interactivity to work. The filtering panel displays the filters that are in effect. You can also use a Filtering panel to add, remove, edit, and toggle filters.

- **Facet** – A user can use a Facet panel to choose data in a specific facet of one or more facets. For example if the facets Gender and Graduation Year are defined, a user could choose the Gender Male and the Graduation Year 2010.
6. Configure specific types of panels

- **Add a query-entry field to a Query panel** – Click Add Search Box.
- **Delete a search box from a Query panel** – Hover over the search box you want to delete, and then click Delete.
- **Move a column in a table to the left** – At the top of the column you want to move, click Move Column to Left.
- **Move a column in a table to the right** – At the top of the column you want to move, click Move Column to Right.
7. Save a dashboard

Click Save Dashboard. You can save a dashboard to Solr, a file, or a GitHub gist. We recommend that you always save dashboards to Solr (even if you also save them as files or gists).

- **Save in Solr** – Click Save Dashboard. Under Solr, enter a name for the dashboard, and then click. Ignore the Make Public checkbox.

- **Export to File** – Click Save Dashboard > . Fusion downloads the dashboard as a file in the browser's download directory. The filename is named `dashboard\_name-large\_number`, for example, `Storefront-1502124761166`. The file is a JSON file. It lacks the file type `.json`, which isn't really needed (you could add it). If you want to place the file in a different location, move it.

- **Save as a GitHub gist** – When you save a dashboard as a GitHub gist, you don't need to log in to Github first. Fusion stores the gist as an anonymous gist.

To save a dashboard as a gist, the dashboard must be configured to allow that. Click Configure dashboard. On the Controls tab under Allow saving to, select Gist. At the top right of the dashboard, click Save. Under Gist, enter a name for the dashboard. The name becomes the title of the dashboard, which is stored in the JSON file as the value of "title". The filename is `kibana-dashboard.json` Then click the icon. Fusion displays the URL at which the gist resides. If you save the dashboard again, you get a different anonymous gist and a different URL. For example:

```
Gist

Storefront

Last gist:
https://gist.github.com/dbf12ce975f2746360fec52a32ab22f
```

**Important**

Visit the URL immediately. Save the URL. Bookmark it. Write it down. You can't find the gist by searching for it, and Gist identifiers don't persist for long in the Fusion UI. (The last gist saved is available in the Load dialog for the duration of the browser session.)
8. Test the dashboard

Ensure that the dashboard displays the expected data when users use the dashboard. Adjust the panels present, their placement, and their configuration based on user feedback.
Use Dashboards

Fusion ships with built-in dashboards that you can access in the Fusion UI.

Use built-in dashboards

Fusion includes the following built-in dashboards. To access these dashboards, from the Fusion UI, click System > Log Viewer.

Service Logs dashboard

The Service Logs dashboard (lucidworks-service-logs.json) is also the default dashboard (default.json).

You can modify the default dashboard to suit your needs. Each user can also choose a different dashboard as a browser default (for a specific browser).

To open the default dashboard from the Fusion workspace, click Analytics > Dashboards or System > Log Viewer. Alternatively, on any dashboard, click Goto saved default.

Access Logs dashboard

The Access logs dashboard (lucidworks-access-logs.json) lets you analyze logs from the system_logs collection where type_s:http, that is Fusion access over the Web, for example, from the Fusion UI and web apps that submit Fusion requests.
Combined Logs dashboard

The Combined Logs dashboard (lucidworks-combined-logs.json) lets you analyze all logs from the system_logs collection.

Blank dashboard

An entirely blank dashboard (blank.json) is available. To open this dashboard from the default dashboard, click Blank Dashboard at the top right.

Fusion Metrics dashboard

The Fusion Metrics dashboard (lucidworks-metrics.json) displays key Fusion metrics from the Fusion collection system_metrics, if this type of metrics collection is enabled. See System Metrics.
Note

In Fusion 4.2 and above, metrics collection is automatically enabled and metrics are indexed to the `system_monitor` collection. Use the DevOps Center to view them.

To open the Fusion Metrics dashboard from the default dashboard, click **Metrics Dashboard** at the top right.

**Search Analytics dashboard**

The Fusion Search Analytics dashboard (**lucidworks-searchanalytics.json**) displays search analytics derived from collection logs and from queries to Solr (**/api/solr**).
To open the Search Analytics dashboard from the default dashboard, click **Search Analytics** at the top right.

**Fusion Signals dashboard**

The Fusion Signals dashboard (**lucidworks-signals.json**) is a time-series dashboard that you can use to monitor signals collections. It uses the signal timestamp as the time field.

To open the Fusion Signals dashboard from the default dashboard, click **Lucidworks Fusion Signals** at the top right.

**Default Non-Time-Series dashboard**

Fusion includes a default non-time-series dashboard (**default-nts.json**). This is an example:
To open the default non-time-series dashboard, at the top right of any dashboard, click New > www.

**Default Time-Series dashboard**

Fusion includes a default time-series dashboard (default-ts.json). This is an example:

![Default Time-Series dashboard](image)

To open the default time-series dashboard, at the top right of any dashboard, click New > www.

**Open a dashboard**

- **Open the default dashboard** – Click Goto saved default www
- **Open a pre-built dashboard** – See Use Built-in Dashboards.
- **Load a dashboard from Solr** – At the top right of the open dashboard, click Load www. Saved dashboards are listed at the bottom of the dialog box. Click a dashboard to open it. Filter dashboards by entering an exact-match string in the Filter box. If there are more that 20 saved dashboards, use the pagination controls to find a dashboard. To view the next set of pages, click www. To view the prior set of pages, click www. You must click a page number to view the dashboards on that page. This is an example of the controls:

![Dashboard pagination controls](image)

- **Load a dashboard from a file** – At the top right of the open dashboard, click Load www. Click Choose File www, browse to and select the file, and then click Open.
- **Load a dashboard from a GitHub gist** – Perform these steps:
  1. At the top right of the open dashboard, click Load www
  2. Enter the gist URL or number (the last part of the URL). For example:
3. Click Get 🐶. A link that opens the dashboard appears below a thin line. The link displays the dashboard name. Click the link to open the dashboard. This is an example of the Gist-loading dialog:

![Gist button](https://gist.github.com/anonymous/#b10353f36c9fcaaade7db98889e9678b#)

**Gist**

https://gist.github.com/anonymous

**Get**

gist:9be99306e676f07ad1dc7a72922dc394

**Dashboards in**

gist:9be99306e676f07ad1dc7a72922dc394 [click to load]

**Storefront**

- **Set a default dashboard for a browser** Click Save Dashboard 🗂 > Set as Browser Default.
- **Clear the default dashboard for a browser** – Click Save Dashboard 🗂 > Clear Browser Default 🚫. The default dashboard for the browser becomes the system default dashboard `default.json` (the Fusion Logs dashboard).

**Interact with data**

- **Choose the collection from which to obtain data** – If a dashboard was configured to let users choose among collections, then you can choose the collection from the Collections drop-down list at the top right of the dashboard.
- **Search** – Enter a query term or phrase in a search bar in a Query panel, and then click Search 🔍.
- **Retrieve all data** – In the Query panel, enter : , and then click Search 🔍.
- **Clear a search box and return to the most general query** (: ) – Click Clear Search Box 🗂.
- **See that a query is in progress for a panel** – At the top right of a panel, the Waiting for Data 🌀 indicator shows you that the panel is waiting for the data. If the indicator spins and spins while other panels display data, the panel is not configured correctly.
- **Select a specific data item** – In a table, click a row. In a graphical-visualization panel, click a data item. Not all
panel types let you select data in this way.

• **Export data** – At the top of a panel, click Export.

• **Hide a row** – At the left of the row, click Hide Row.

• **Expand a hidden row** – At the left of the hidden row, click Expand Row.

**Interact with specific types of panels**

Following are interactions for specific types of panels.

**Table panels**

• **Show the field list** – At the top left of a Table panel, click Show Field List.

• **Hide the field list** – At the top left of a Table panel, click Hide Field List.

• **Sort a column in a table into ascending order** – At the top of a column, click Sort Ascending.

• **Sort a column in a table into descending order** – At the top of a column, click Sort Descending.

• **View the next page of hits** – At the top or bottom of a page of hits in a Table panel, click Next Page.

• **View the previous page of hits** – At the top or bottom of a page of hits in a Table panel, click Previous Page.

• **View the first page of hits** – At the top left of a Table panel, click View First Page.

• **Show details for a record in a table** – Click a row in a table. Click Table, JSON, or Raw to view the details in those formats.

• **Hide details for a record in a table** – At the top right of the details, click Hide Details.

**Full Text Search panels**

• **Show details for a record in a Full Text Search panel** – Click a row in a Full Text Search panel. Click Table, JSON, or Raw to view the details in those formats.

• **Hide details for a record in a Full Text Search panel** – At the top right of the details, click Hide Details.

**Histogram and Range Facet panels**

• **Zoom in on a subset of data** (Histogram and Range Facet panels) – On the histogram, drag the mouse across the range of data you want to zoom in on.
Filtering panels

- **Edit a filter** – Click Edit Filter.
- **Toggle a filter on and off** Click Toggle Filter.
- **Remove a filter** – For the filter that you want to remove, click Remove Filter.

  **Note** Exercise caution when removing filters. Some panels depend on specific filters being present. Fusion also creates filters based on your interactions with data. You can add a filter back, but there is no undo. You might want to just toggle the filter off instead.

Heat Map panels

- **Transpose rows and columns in a heat map** – In a Heat Map panel, click Transpose Rows and Columns.

**Filter by Fusion request ID**

To only display information about log entries with a specific Fusion request ID, you can filter the contents of a logs dashboard by the Fusion request ID.
You can do this in these ways:

- Click a link in the error message or toast.
- Filter in a logs dashboard.
- Append a query filter parameter to the dashboard URL.

**Click a link in the error message or toast**

In an API error in the Fusion UI or in an error toast, a *View logs dashboard* link is present. Click the link to view the dashboard using a filter for that request ID.

API error with link in Fusion UI:

![API error with link in Fusion UI](image)

API error toast with link:

![API error toast with link](image)
Filtered Service Logs dashboard:

Clicking a View logs dashboard link opens the Service Logs dashboard and filters by the Fusion request ID. Notice the filtering query for the `mdc_requestId_s` field. In this example, the fusion-request-id is `BWQSjPPYgB`. 

6/29/18 4:00 PM
Failed to simulate results from working pipeline.
View logs dashboard

6/29/18 4:00 PM
Failed to simulate results from working pipeline.
View logs dashboard
Filter in a logs dashboard

In the Service Logs, Access Logs, or Combined Logs dashboard, you can filter by Fusion request ID.

To do so:

1. Obtain the Fusion request ID returned by a request. For example, notice the Fusion request ID **BWQSjPPYgB** returned by this API command:
2. Open a logs dashboard.

3. Specify the Fusion request ID in the Filter panel, with a query for the `mdc_requestId_s` field that specifies the value of `fusion-request-id` for which you want to search.

In this example, the `fusion-request-id` is `BWQSjPPYgB`:
Append a query filter parameter to the dashboard URL

As an alternative to constructing a query filter in the dashboard, you can append a query filter parameter to the dashboard URL. In the query filter, use the field name `mdc_requestID_s`.

Example 1: Default dashboard (Service Logs)

Example 2: Combined Logs dashboard

Example 1: Default dashboard (Service Logs)
Open the Service Logs dashboard from a job history

1. Open the job history. For most jobs, click Collections > Jobs, select a job, and then click Job History. For datasource jobs, click Indexing > Datasources, select a datasource, and then click Job History.

   In the job history, you will see a link named runId:

   ![Job History](image)

2. Click the link to view the Service Logs dashboard filtered by the runId:

   ![Combined Logs](image)
Manage Dashboards

Set Permissions

Users who manage or use dashboards need specific permissions to do so.

Permissions to Manage Dashboards

The built-in roles admin and developer both have sufficient permissions to manage dashboards.

For users who manage dashboards, set these permissions in roles, and then assign the roles to users:

```
GET:/solr/{id}/*:id=*_collection-name_*
GET:/solr/{id}/admin/luke:id=*_collection-name_*
GET,POST,PUT,DELETE:/solr/system_blobs/*
GET,POST,PUT,DELETE:/collections/system_blobs
```

Permissions to Use Dashboards

The built-in role search doesn’t have sufficient permissions to use dashboards.

For users who manage dashboards, set these permissions in roles, and then assign the roles to users:

```
GET:/solr/{id}/*:id=*_collection-name_*
GET:/solr/{id}/admin/luke:id=*_collection-name_*
GET:/solr/system_blobs/*
GET:/collections/system_blobs
```

Manage Dashboards

• **Create a new dashboard** – Create a new dashboard:
  
  ◦ Time-series dashboard – At the top right of a dashboard, click New > .
  
  ◦ Non time-series dashboard – At the top right of a dashboard, click New > .
  
  ◦ Start from an example or template – Open an example dashboard or template. At the top right of the open dashboard, click Load .
  
  ◦ Start from a blank dashboard – Open the New Dashboard template blank.json.

  In all of these cases, save the dashboard as a new dashboard.

  For more information about creating dashboards, see Create Dashboards.

• **Configure a dashboard** – At the top right of the dashboard, click Configure Dashboard .

• **Set the default search field** – At the top right of the dashboard, click Configure Dashboard  On the Solr tab under Global Query Parameters, include a &df parameter; for example, &df=product.
• **Let dashboard users choose the collection** – At the top right of the dashboard, click Configure Dashboard 🛠. On the Controls tab, select Show Collections Picker.

• **Save a dashboard** – Click Save Dashboard 🔗. You can save a dashboard to Solr, a file, or a GitHub gist. We recommend that you always save dashboards to Solr (even if you also save them as files or gists). For more information, see Save a Dashboard.

• **Delete a dashboard saved in Solr** – At the top right of the open dashboard, click Load 🔄. Saved dashboards are listed at the bottom of the dialog box. Next to the dashboard you want to delete, click 🌐. Click OK to confirm that you want to delete the dashboard.

• **Delete a dashboard saved in a file** – Navigate to the file and delete it.

  • **Delete a dashboard saved as a Github gist** – Open the gist URL. Click 🚧 Delete.
Input Panels

Input panels let you control which data output panels display. The values you specify in input panels become parts of the Solr queries that output panels use to obtain data.

You can use these input panels:

- **Query Panel** – Enter a free-form query (one or more query terms) in a search bar. Add additional search bars to a query panel to submit separate queries. Some visualization panels (for example, Rangefacet) keep the data separate so you can compare it. This is an example of a Query panel with two search boxes. The parameters for query strings are `Primary_Specialty:Urology` and `Primary_Specialty:Psychiatry`. Here is the Query Panel and the resulting Rangefacet panel:

  ![Query Panel Example](image1)

- **Time Picker Panel** – Apply a time range to time-series data. The time range can be:
  - Relative – A time range starting now and reaching backward in time, for example, the last 15 minutes or one hour
• **Absolute** – A specific time-and-date range, for example, from 06/01/2017 00:00:00 to 06/30/17 23:59:59

• **Since** – A time range since a specific date and time, for example, since 01/01/2017 00:00:00.

• **Facet** – A Facet panel can facet any data field.

• **Filtering** – A Filtering panel lets you apply field-based filters to the hits returned by the query. The filters apply to all display panels.
Query Syntax

Enter a search term or phrase in the search box of the Query panel. Autocompletion provides a list of possibly related prior searches. Finish typing your search query or select an autocompleted query, and then click Search.

Enter queries in a Query panel using Apache Lucene Query Parser syntax. You enter the parameter for the query string (for example, Susan or Gender:F), not the query string (for example, q=Susan or q=Gender:F).

Rules for the Simplest Cases

Here are some syntax rules for the simplest cases:

- A single term is either field: value or value. With value, the search is over all fields.
- For an exact-case match, you must specify the field name.
- Surround a term that contains spaces in double quotation marks (" ").
- You can't specify :value or :"value" to search over all fields; that syntax doesn't work.
- To retrieve all records, use the search term :
- For OR logic, enter OR between the terms, or just use spaces. For AND logic, enter AND between the terms.

Examples

These are examples of the query syntax:

<table>
<thead>
<tr>
<th>Goal</th>
<th>Syntax and example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single term in any field; no blanks in term</td>
<td>term (matches any case)</td>
</tr>
<tr>
<td>Single term in any field; blanks in term</td>
<td>&quot;term&quot; (matches any case)</td>
</tr>
<tr>
<td>Multiple terms, each in any field; with OR logic; no blanks in terms</td>
<td>term1 term2 ...</td>
</tr>
<tr>
<td>Goal</td>
<td>Syntax and example</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>Multiple terms, each in any field; with OR logic; blanks in terms</td>
<td>&quot;term1&quot; &quot;term2&quot; ... (matches any case)</td>
</tr>
<tr>
<td>Multiple terms, each in any field; with AND logic (in the same record); no blanks in terms</td>
<td>term1 AND term2 ...</td>
</tr>
<tr>
<td>Multiple terms, each in any field; with AND logic (in the same record); blanks in terms</td>
<td>&quot;term1&quot; AND &quot;term2&quot; ... (matches any case)</td>
</tr>
<tr>
<td>Single term in a specific field; no blanks in term</td>
<td>field:term (matches any case)</td>
</tr>
<tr>
<td>Single term in a specific field; blanks in term</td>
<td>&quot;field:term&quot; (matches exact case)</td>
</tr>
<tr>
<td>Multiple terms, each in a specific field; with OR logic; no blanks in terms</td>
<td>field1:term1 field2:term2 ...</td>
</tr>
<tr>
<td>Multiple terms, each in a specific field; with OR logic; blanks in terms</td>
<td>&quot;field1:term1&quot; &quot;field2:term2&quot; ... (matches exact case)</td>
</tr>
<tr>
<td>Multiple terms, each in each in a specific field; with AND logic (in the same record); no blanks in terms</td>
<td>field1:term1 AND field2:term2 ...</td>
</tr>
<tr>
<td>Multiple terms, each in each in a specific field; with AND logic (in the same record); blanks in terms</td>
<td>&quot;field1:term1&quot; AND &quot;field2:term2&quot; ... (matches exact case)</td>
</tr>
</tbody>
</table>

For more information about the query syntax, see [Standard Query Parser Parameters](#).

| Tip                                                                 | You can use a Text panel to advise the user regarding the syntax of search terms. Also, if you want the dashboard user to explore subsets of the data, use a Filtering panel or a Facet panel to achieve that. Don't expect users to enter complex search expressions. For example, add the field `gender` as a facet, instead of expecting the user to add `AND gender:male` to a search expression. |

---

**Inspect a Panel Query**

You can't inspect the panel query in the Query panel. You can inspect the panel query in other panels, for example, in a Histogram or Heatmap panel. You can see the contributions that the different parts of the query make. In this example, there are no global query parameters.

**Query:**

```
start_station_name:"Broadway and E 14 St" AND gender:Male
```

**Panel query for a Histogram panel:**

The part of the query from the Query panel is the first part, from `q=` through `Male`. 
Display Panels

Display panels on a dashboard display information about the data in a single collection. Fusion has these types of display panels:

- **Layout** – Use these display panels to organize the panels on a dashboard.
- **Textual Information** – Use these panels to display textual information. For example, use a Table panel to display the values of fields in a set of records.
- **Graphical Visualization** – Use these panels to help users to visualize data. For example, display category data in a bar chart and geographical data on a map.

What Data is Displayed

These things determine which data Fusion displays in a display panel:

- Where the query is sent (to a Fusion query pipeline or to Solr)
- The collection that contains the data. You can specify this when you create a dashboard, or permit the user to choose the collection.
- Global query parameters (optional)
- Input panel configuration settings (possibly including a panel query)
- A user’s selection of the collection (if permitted)
- A user’s query
- A user’s interactions with the data

Layout Panels

These layout panels can help you organize a dashboard:

- **Column** – Lay out panels in a column within a row or part of a row. (Instead of the usual left-to-right layout within rows.)
- **Text** – Add text to a dashboard, for example, to instruct the user regarding how to use the dashboard, or to describe the data.

Textual Information Panels

These panels display textual information:

- **Doc Viewer** – Display documents as single pages of information, and let the user page through documents. Paging through documents doesn’t affect the data displayed in other panels.
- **Full Text Search** – Provide full text search capability.

Note: Make a Full Text Search panel wide. Start with Span = 12. Depending on the content, you might be able to make it narrower. If you make a Full Text Search panel too narrow, content and controls can be inaccessible.

- **Hits** – A Hits panel displays statistical information about the hits returned by the query, including all filters that are applied to the query. The default information is the count (number of hits). You can display the count, minimum,
mean, maximum, sum, standard deviation, the sum of squares, and/or the number of hits that lack a value for a field.

- **Table** – Display data in a table.

  **Note**: Make a Table panel wide. Start with Span = 12. Depending on the content, you might be able to make it narrower. If you make a Table panel too narrow, content and controls can be inaccessible.

---

### Graphical Visualization Panels

Most display panels are graphical visualization panels. They visually present the data requested in the input panel(s), subject to configuration of the display panel and to any global filtering. Panels for visualizing information graphically are:

- **Bar** – Graph data as frequencies in a bar chart.

- **Better Map** – Display geolocated points in clustered groups on a map. As you zoom out, points are clustered into fewer groups. As you zoom in, points are clustered into fewer groups (and at some point are not clustered). The Better Map panel *doesn’t* use the geospatial search capabilities of Solr. It transfers more data than the Map panel and generally requires more computation, while showing less data. If you have a time filter, the panel will show the most recent points in your search, up to your defined limit. This panel is best used after filtering the results through other queries and filter queries, or when you want to inspect a recent sample of points.
• **Heat Map** – Display a heat map, that is, a graphical representation of data along two facet axes.

• **Histogram** – Display a histogram. When used in count mode, this is a bucketed chart of the current query, including all applied time and non-time filters. When used in values mode, the histogram plots the value of a specific field over time, and lets the user group the values based on the values of a second field.
- **Map** – Display a map of shaded regions using a field that contains a 2-letter country code or U.S. state code. Regions with more hits are shaded darker. Instead of a count, you can choose to shade regions based on the minimum, maximum, mean, or sum. The map panel uses facets, so it is important that you set field values to the appropriate 2-letter codes at index time.

- **Multi-series** – Graph multiple fields of the same type that vary with time in the same graph. The X-axis is used for the time series field. Values in the other fields are graphed along the Y-axis.

- **Range Facet** – Display a histogram of a numeric field. The Range Facet panel is similar to the time series histogram. It lets you select ranges and zooming in/out to the desired numeric range. Range selections in the panel are reflected across the entire dashboard. With multiple search boxes in a Query panel, the RangeFacet panel can display multiple data sets, as shown in this example:
• **Scatter Plot** – Display a scatter plot between two variables or four variables.

• **Sunburst** – Display a sunburst plot based on facets.
- **Tag Cloud** – Display a tag cloud of the top N words in a field.
• **Terms** – Display the count, minimum, maximum, mean, or sum for a Solr facet as a bar chart, pie chart, or table.