System Health and Debug

Health Checks

System health checks are performed by the Pipeline managers.

Debug

To view the Aspire debug console, see http://localhost:50505/aspire after starting up the Aspire server. You can also access it from the Admin UI, using the **Debug Console** link that appears for each server.

On this page:

Health Checks

System health checks are performed by the Pipeline managers. Health checks check the overall health of the system for things like:

- Jobs which encounter exception errors
- Jobs which are too slow
- Components which are still initializing after startup

Once configured, the health of the entire server, as well as detailed information on each health check, is available through the admin RESTful interface.

Configure Health Checks

Pipeline managers need to be configured to perform health checks to determine how Aspire is performing.

Configuration

To configure health checks for your system add a <healthChecks> section to the pipeline managers for which you desire health checks. Details on how to configure each type of health check are given below. Multiple health checks can be configured per pipeline manager.

The following is an example configuration:

```
<component name="MyPipelineManager" subType="pipeline" factoryName="aspire-application">
   <healthChecks>

     <timestamp name="Test Timestamp" redThreshold="4000" yellowThreshold="1000"
                history="5" />

     <initialization name="Test Long Initializer">
        <check componentRef="/pipeline/LongInitializer"/>
        <check componentRef="/pipeline/Concat-Test"/>
     </initialization>

     <jobCount redThreshold="1" />

   </healthChecks>

   <!-- Configure your pipelines here -->

   <!-- Configure your components here -->

</component>
```

Note that all health checks can have a "user friendly name" attached to them (the @name attribute).

Once health checks are configured for your pipeline managers, they will be automatically accumulated and made available as requested.

Types

There are multiple types of health checks available. The types currently available include:

- **Initialization** - Checks to see if components are fully initialized
- **Timestamp** - Creates timestamps when jobs start and when they are complete. Jobs which take too long to complete can be flagged as either "RED" or "YELLOW".
- **Job Count** - Provides status on total jobs started, completed successfully, and completed with an error. Health is based on the count of jobs which failed.
- **Latency** - Computes a moving average of the time it takes to complete a job.

Usage

Health Checks can be:

- **INITIALIZING** - Either the Aspire System itself is still loading configurations, or a component has a long initialization which is still in progress.
- **GREEN** - Everything okay
- **YELLOW** - One or more health checks is showing an issue
- **RED** - Jobs are failing or jobs are too slow to satisfy the service levels

The health checks for all pipeline managers across the system are accumulated into a single health check for the entire server. URLS are also available for accessing and managing health checks:

- Entire server health: http://localhost:50505/aspire?cmd=health
- Health check details: http://localhost:50505/aspire?cmd=healthDetail
    - Gives the details for all health checks across all pipeline managers.
- Clearing an individual health check: http://localhost:50505/aspire?cmd=healthClear&id=<health-check-id>
- Clearing all Health Checks: http://localhost:50505/aspire?cmd=healthClearAll .

Health Check 1: Initialization

This health check is used to check components to see if they are initializing. If they are, the health of the system will be returned as "INITIALIZING".

Configuration:

```
<initialization name="Test Long Initializer">
    <check componentRef="/pipeline/LongInitializer"/>
    <check componentRef="/pipeline/Concat-Test"/>
</initialization>
```

- *<check>* - Specifies the component to check.

Attribute:

- *@componentRef* - The path name to the component. This *must be* the full path name of the component.

Health details:

- This health check returns YELLOW if a component is not available.
- This health check returns RED if there is an error accessing the component or getting the component's status.
    - A health of RED supercedes a health of INITIALIZING.

Note: It is expected that the initialization health check will be performed automatically in a future release of Aspire, at which point this health check will be deprecated.

Health Check 2: Job Count

This health check provides a total count of jobs and will determine the health of the system based on the total number of failed jobs that occur.

Configuration:

```
<jobCount name="Count of Document Jobs" redThreshold="3" yellowThreshold="1"/>
```

Attributes:

- *@redThreshold* - If total number of failed jobs is greater than or equal to this number, system health is RED. Should be 1 or more.
    - If @redThreshold is "0", your system will be RED all the time! So, set it to 1 or more.

- *@yellowThreshold* - If total number of failed jobs is greater than or equal to this number, system health is YELLOW. Should be 1 or more.

Health details will show:

- Total count of jobs initiated
- Total count of jobs which completed successfully
- Total count of jobs which failed with an unhandled exception
- Total count of jobs outstanding (equals total initiated minus total completed)

Health Check 3: Time Stamps

Timestamp health checks are used to check the duration of every job and are typically used for occasional jobs (for example, nightly) that take a long time to run (i.e., hours).

Configuration:

```
<timestamp name="Rebuild Dictionary Token Stats" history="5" redThreshold="10000" yellowThreshold="2000"/>
```

Attribute:

- *@history* - The number of past timestamps to display on the health detail page.
- *@redThreshold* - (milliseconds) If a job takes this much time to complete (or more), health will be RED.
- *@yellowThreshold* - (milliseconds) If a job takes this much time to complete (or more), health will be at least YELLOW.

History:

A history of old timestamps will be kept and displayed in the health check details.

Note that only the most recent timestamp will contribute to the health of the overall system.

Health Check 4: Latency

Latency health checks compute a moving average of the time it takes to complete a job and then will flag RED or YELLOW if average job latency rises above specified thresholds.

Averages are computed over a specified number of jobs (@jobsToAverage). This method will also compute a peak average latency and give a history of averages for previous time periods.

Configuration: (defaults to 15 minute intervals over 24 hours)

```
 <latency name="Process Single Document" jobsToAverage="5" isSticky="true"
         redThreshold="15000" yellowThreshold="5000" />
```

Configuration: (specify the interval and history length)

```
<latency name="Process Single Document" jobsToAverage="5" isSticky="true"
         redThreshold="15000" yellowThreshold="5000"
         interval="3600000" history="48" />
```

Attributes for Moving Averages:

- *@jobsToAverage* - The number of jobs to include in the moving average which is used to compute health.
- *@isSticky* - Specifies whether peak values are "sticky", that is, they hang around until cleared. For example, if your jobs slow down and the health is RED, it will remain RED (i.e., "sticky") even if they start to speed up again. Can be "true" or "false".
- *@redThreshold* - (ms) If the moving average of job latencies rises above this threshold, health will be RED.
- *@yellowThreshold* - (ms) The average latency threshold for declaring the health to be YELLOW.

Attributes for History Presentation:

- *@interval* - (ms) The size of each interval in the history. Measured in milliseconds. If not supplied, defaults to 900000 (15 minutes).
- *@history* - (count) The number of history intervals to save. Histories are a rolling window from the current time back through this number of intervals. Defaults to 96 intervals (equals 24 hour's worth of intervals).

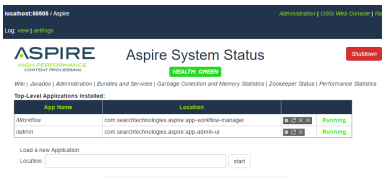
Notes:

- Histories are independent of moving average computations. The moving averages are computed independently from the history display. Therefore, it is possible for history intervals to show long latencies and for the health status to still be GREEN. This would occur when a history interval contains fewer than the number of jobs specified in @jobsToAverage.

- Don't make @jobsToAverage too small. If this number is too small, then the computations will become unreliable.
- Failed jobs are not added to latency numbers. Latencies are only computed for successful jobs, since these are the only ones which will provide reliable measurement data.

Debug

To view the debug console for Aspire, go to http://localhost:50505/aspire after starting up the Aspire server.



(You can also access it from the System Admin user interface, using the Debug Console link that appears for every server.)

The debug console provides status information and some low-level admin controls for Aspire. Some of these functions can also be performed using the System Admin user interface. Through the debug console you can:

- Load New Applications
  - Each Aspire application will be an application.xml file which has components and pipelines for processing content.
  - Multiple configurations can be loaded into the same instance of Aspire.
    - For example, you can have a separate configuration for each database that you want to process with Aspire.
- Refresh Applications
  - This will reload the application.xml file for a configuration, loading all new configuration settings and automatically loading components as needed
- Browse Applications
  - Every component in an Aspire Application Configuration has a separate web page.
- Component Status / Component Controls
  - Many of the components will use the debug console to display additional status about the component or to allow for administrator control over the component (reloading XSL transforms, etc.)
  - For example, you can turn on statistics for pipeline managers to see which pipeline stages are the slowest.
- Update Components
  - Not only can component configurations be updated, but the binary Java code for individual components can be updated as well. Click on "Check For Updates". If a new version of a component is available from the Maven Repository, you will be given the option to update the component.
  - This is possible because Aspire is built on top of OSGi, which allows for Jar files to be dynamically reloaded as needed. The entire Aspire system has been built to allow for dynamic component updates.
- View the OSGi Web Console
  - The OSGi web console allows you to interact with Apache Felix directly. The username /password is "admin/admin".
- View the Health of the System
  - Clicking on the "Health" Bar will show you detailed health about the system.
  - Note that health checks need to be configured in the system, and are not turned on by default (other than the application startup health)

So you can see, the debug console contains quite a lot of useful functionality.

In general, the debug console is self-documenting. This section will cover some of the inner workings of the interface.

Configuration of the debug console

See Changing the Aspire Port Address for information about changing the port address or General Settings for adding a username and password to the debug console.

Every component has a web page

Every component configured in Aspire has a web page. The web page can be accessed using the component's full name as: http://server:50505/aspire/{component-full-name} (see Naming Components for more details on component names).

Navigation

The component manager and pipeline manager components contain links to all of the sub-components they contain. The Aspire application component (the component which reponds to the http://localhost:50505/aspire address) also has links to all of the installed applications.

In this way, you can usually navigate to any component by clicking the links from parent components to sub-components.

Component commands

Components can also receive commands from the admin user interface. Web commands to individual components have many different purposes:

- Turn on or off debugging
- Turn on or off additional statistics gathering

- To run diagnostic tests on components
- To view additional information on a component

All component commands have a "cmd" parameter which specifies the name of the command to execute. For example, to get the status of a component use:

```
http://localhost:50505/aspire?cmd=status
```

Some commands also take additional URL parameters.

In general, it is not necessary to document the component commands, since they should all be available from the System Administration user interface itself.

Common commands

The following commands are common to all components in Aspire. The format for each command is:

```
http://server:50505/aspire/{component-name}?cmd={command}&{parameters}
```

Command	Parameters	Description
status	none	Provides status on the command. This provides the same result as navigating to the command in the user interface.
log	display	Displays the log file for the specified command. For example, http://localhost:50505/aspire?cmd=log&display will display the log file for the Aspire application component.
log	debug=true	Turns on debug logging. Debug messages can be viewed with the log/display command.
log	debug=false	Turns off debug logging.

-== XML and XSLT ===

All commands executed by the System Administration user interface return straight XML to the web browser. You can test this by doing a "view source" on any page in the interface (using Firefox; if using IE9, use the developer tools to see the original XML).

In order to convert the XML to a user-friendly display, the XML is transformed to XHTML and CSS using an XSLT stylesheet. The style sheet is automatically located by the Aspire application and if it exists, it is added as an <?xml-stylesheet> processing command to the top of the resulting XML returned by the component. The web browser will then fetch the XSLT transform and will transform the XML into XHTML for display.

Locate the XSL transform

XSL transforms for the debug console are bundled with the Jar file which contains the component code. This increases the portability of components, since the System Administration user interface (i.e., the XSL transform) is included and installed automatically with the component software itself.

However, the Aspire application will first attempt to load the transform from the following directory:

```
{Aspire Home}/resources/{component-implementation-class}
```

This is done for user interface development, since it is much easier to edit files in the Aspire Home directory than it is to edit files inside a component Jar file.

For example, if the component's Java implementation class is "com.searchtechnologies.aspire.components.RDBConnectionStage", then the Aspire application will first attempt to load the transform from the following directory:

```
{Aspire Home}/resources/com.searchtechnologies.aspire.components.RDBConnectionStage
```

If the XSLT is not in Aspire Home (and it typically is not), then it will then attempt to locate it in the component's Jar file, under the "resources/{component-implementation-class} directory. In this way, most of the components are bundled with the XSL transform used for their administration user interface.

XSL transform file name

The XSL transform names are the same as the Aspire URL command names. That is, the command names passed to the "cmd" parameter on the URL.

For example, "status.xsl" is the XSL transform used for all status reports.

If the XSL file for the specific command can not be located, the XSL transform called "result.xsl" will be located. This is the fallback transform used to present the result for most components.

Finally, if no "result.xsl" transform can be located, the XML will be sent directly to the admin user without a transform.