Infor LN Performance Monitoring Whitepaper (On-premises)

Release 10.8.x

Important Notices

The material contained in this publication (including any supplementary information) constitutes and contains confidential and proprietary information of Infor.

By gaining access to the attached, you acknowledge and agree that the material (including any modification, translation or adaptation of the material) and all copyright, trade secrets and all other right, title and interest therein, are the sole property of Infor and that you shall not gain right, title or interest in the material (including any modification, translation or adaptation of the material) by virtue of your review thereof other than the non-exclusive right to use the material solely in connection with and the furtherance of your license and use of software made available to your company from Infor pursuant to a separate agreement, the terms of which separate agreement shall govern your use of this material and all supplemental related materials ("Purpose").

In addition, by accessing the enclosed material, you acknowledge and agree that you are required to maintain such material in strict confidence and that your use of such material is limited to the Purpose described above. Although Infor has taken due care to ensure that the material included in this publication is accurate and complete, Infor cannot warrant that the information contained in this publication is complete, does not contain typographical or other errors, or will meet your specific requirements. As such, Infor does not assume and hereby disclaims all liability, consequential or otherwise, for any loss or damage to any person or entity which is caused by or relates to errors or omissions in this publication (including any supplementary information), whether such errors or omissions result from negligence, accident or any other cause.

Without limitation, U.S. export control laws and other applicable export and import laws govern your use of this material and you will neither export or re-export, directly or indirectly, this material nor any related materials or supplemental information in violation of such laws, or use such materials for any purpose prohibited by such laws.

Trademark Acknowledgements

Publication Information

Release: Infor LN 10.8.x
Publication Date: June 7, 2022
Document code: ln_10.8.x_pbc_user__en-us

Contents

About this guide

This white paper describes the setup of Infor LN performance monitoring using a Windows LN UI server, a WindowsLN application server and a Windows database server running SQL Server.

The monitoring stack is based on these open source applications:

- Telegraf for collecting metrics
- InfluxDB for storing metrics
- Grafana for visualization

Metrics are send to Telegraf by Infor LN and Telegraf can be used to collect system resource information. InfluxDB and Grafana offer a cloud offering. This guide describes the on-premises setup of these products.

Intended audience

This guide is intended for Infor consultants or Infor LN system administrators to setup a monitoring solution for on-premises and single tenant cloud installations.

Support note

Infor does support gathering and sending metrics within the Infor LN or LN UI applications. Setup and configuration of OpenSource products such as InfluxDB, Telegraf and Grafana are not supported by Infor.

Contacting Infor

If you have questions about Infor products, go to Infor Concierge at https://concierge.infor.com/ and create a support incident.

The latest documentation is available from docs.infor.com or from the Infor Support Portal. To access documentation on the Infor Support Portal, select **Search > Browse Documentation**. We recommend that you check this portal periodically for updated documentation.

If you have comments about Infor documentation, contact documentation@infor.com.

Chapter 1: Introduction

It is important to have a good performing ERP system. However, it can be difficult to get the required insight in the performance characteristics of the system.

Therefore, Infor developed application specific metrics to provide insight in the response times and use of the application. When combined with resource utilization metrics, they provide useful information to IT administrators and managers.

This white paper describes how to setup performance monitoring of Infor LN using an open source stack of applications. The monitoring stack described is based on these applications:

Telegraf
Telegraf is a server-based agent for collecting and sending all metrics and events from databases and systems. It can collect resource metrics on the systems where it is installed. Infor LN uses Telegraf as a proxy to deliver the data to a time-series database.

To learn more about Telegraf see: https://www.influxdata.com

InfluxDB
InfluxDB is a powerful time series database to store the metrics send by Telegraf.

To learn more about InfluxDB, see:https://www.influxdata.com.

For the documentation, see: https://docs.influxdata.com/influxdb

Grafana

Grafana is used to query, visualize, alert on, and understand the data. With Grafana you can create, explore and share all of your data through beautiful, flexible dashboards.

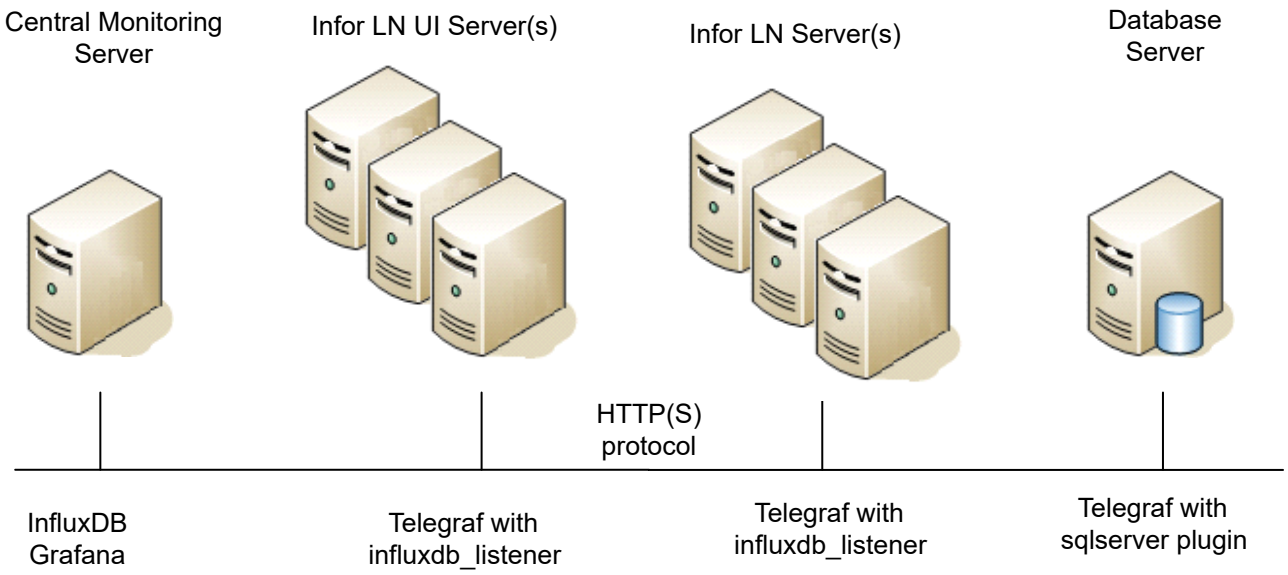
To learn more about Grafana, see: https://www.grafana.com

Note: You can send Infor LN metrics directly to InfluxDB without using Telegraf. Instead of using InfluxDB, another time series database supported by Telegraf can also be used. When desired, you can use another visualization application instead of Grafana. This is outside the scope of this white paper.

Recommended deployment

To monitor Infor LN, we recommend to set up a central monitoring system running InfluxDB for data storage and Grafana for visualization. On each monitored system the Telegraf agent must be installed.

This diagram shows the recommended deployment scenario of Infor LN monitoring solution:



Architecture

When enabled, Infor LN sends different types of metrics in the InfluxDB lineprotocol format to a HTTP endpoint configured by the administrator.

This endpoint can be the InfluxDB HTTP webservice itself or a Telegraf agent running the `influxdb_listener` plugin. Using Telegraf is recommended because it can act as an API or proxy and multiple destinations can be defined. After the data is available in InfluxDB, a dashboard can be created using Grafana to visualize the data in a graph.

Note: You can save the output to another database such as OpenTSDB or to a file. In that case the Telegraf agent is required to translate the InfluxDB line protocol to OpenTSDB format.

Prerequisites

These utilities or software levels are required:

- Download Telegraf from https://influxdata.com and extract to `c:\monitoring\telegraf` on each system to be monitored
- Download InfluxDB v2.x from https://influxdata.com and extract to `c:\monitoring\influxdb` on the Central Monitoring Server
  Note: InfluxDB listens for incoming requests on port 8086 by default. Ensure this is not blocked by a firewall.

- Download InfluxDB Cloud CLI from https://influxdata.com and extract to `c:\monitoring\influxdb` on the Central Monitoring Server
- Download Grafana from https://grafana.com and extract to `c:\monitoring\grafana` on the Central Monitoring Server
- Minimum version Infor LN porting set: 9.3b
- Minimum version Infor Enterprise Server: 10.7
- Minimum version LN UI: 12.2
- Minimum version Infor C4WS: 12.2.8

Note: This guide is based on open source versions of Telegraf v1.22, InfluxDB v2.2 and Grafana v8.4.6 OSS.

Chapter 2: Setting up monitoring

You can set up a central monitoring system for Infor LN with InfluxDB, Telegraf and Grafana.

InfluxDB setup

Complete these steps on the Central Monitoring Server to set up InfluxDB:

1. Open a command prompt as administrator and generate an InfluxDB configuration file:

   Specify this command:

   ```
   c:/monitoring/influxdb/influxd.exe print-config > c:/monitoring/influxdb/config.yaml
   ```

2. Edit the configuration file `config.yaml` and specify these lines:

   ```
   bolt-path: c:/monitoring/influxdb/data/influxd.bolt
   ```

   ```
   engine-path: c:/monitoring/influxdb/engine
   ```

3. Start InfluxDB in this folder: `c:/monitoring/influxdb/influxd.exe`

   Use nssm to run InfluxDB as a service and restart it automatically after a system reboot.

4. To manage InfluxDB with a web interface, go to `http://localhost:8086`

   Set up the initial user and bucket. In this example the organization is called monitoring. You can choose your own name, but you require this name in step 8.

5. Go to **data > API Tokens > admin's Token**.

   Copy the admin token to clipboard.

6. Open a command prompt and go to this folder:

   ```
   c:/monitoring/influxdb
   ```

7. Specify the INFLUXDB_HOME dir:

   ```
   set INFLUXDB_HOME=c:/monitoring/influxdb
   ```

8. Create an InfluxDB CLI config file:

   ```
   influx.exe config create --active --config-name monitoring --host-url http://localhost:8086
    --org monitoring --token API_TOKEN --configs-path %INFLUXDB_HOME%/configs
   ```

   Replace the organization with your organization and the API token with the admin API Token from step 5.

9 Create a bucket (inforlnmonitoring) and a retention policy to automatically delete data after some time. In this case, the data is removed after 2 years.

```
influx.exe bucket create -d "Infor LN Monitoring" -n inforlnmonitoring -r 730d --configs-path
 %INFLUXDB_HOME%/configs
```

Save the returned bucket ID as it is required later in the process.

10 Create a database retention policy mapping to access the data using the InfluxQL language:
Replace BUCKET_ID with the ID of step 9. Example:

```
influx.exe v1 dbrp create --bucket-id BUCKET_ID --db inforlnmonitoring --default --rp 2year
 --configs-path %INFLUXDB_HOME%/configs
```

Example:

```
influx.exe v1 dbrp create --bucket-id 13d046bab7215fcf --db inforlnmonitoring --default --rp
 2year --configs-path %INFLUXDB_HOME%/configs
```

11 Create an authorization token to access the newly created bucket:

```
influx.exe auth create --read-buckets --write-buckets -d inforlnmonitoring --configs-path
%INFLUXDB_HOME%/configs
```

Save the returned token as it is required later in the process. Example:
For more details, see the InfluxDB v2.x documentation.

Telegraf setup

Complete these steps on all systems to be monitored to set up Telegraf:

1 Download a Telegraf config template from 2071972 and save it to this folder:

`c:/monitoring/telegraf/telegraf.conf`

2 Edit the config file `telegraf.conf` and review/modify the sections to reflect your situation. Always use forward slashes in directories:

```
[agent]
      logfile = "C:/monitoring/telegraf/telegraf.log"
```

3 Add or uncomment these sections to the Telegraf config file and modify these sections:

```
[[outputs.influxdb_v2]]
## The URLs of the InfluxDB cluster nodes.
urls = ["http://yourmcentralonitoringserver:8086"]
## Token for authentication.
token = "token from InfluxDB setup step 9"
## Organization is the name of the organization you wish to write to; must exist.
organization = "monitoring"
```

```
## Destination bucket to write into.
bucket = "inforlnmonitoring"
```

The `influxdb_listener` input plugin is enabled by default in the template. It is used to accept incoming data from Infor LN and listens only on the local IP address 127.0.0.1 and port 8186. This is for security and performance reasons. It prevents injection of data to InfluxDB from other sources than the local system. Performance can decrease when using a user-friendly name such as localhost instead of the IP address.

4 On the Infor LN UI Server, add this section to the `telegraf.conf` file:

```
[[inputs.http_response]]
   urls = ["http://localhost:8312/webui"]
```

5 On the Database Server, add this section to the `telegraf.conf` file:

```
[[inputs.sqlserver]]
 servers = [
  "Server=localhost\\instance;
  Port=1433;
  User Id=telegraf;
  Password=mystrongpassword;
  app name=telegraf;
  log=1;"
  ]
 database_type = "SQLServer"
```

Replace the instance name, port and password.

For more information about the SQL Server input plugin, see https://github.com/influxdata/Telegraf/tree/master/plugins/inputs/sqlserver

You must create a login providing VIEW permissions on every SQL Server instance to monitor with this script. Replace the password.

```
USE master;
GO
CREATE LOGIN [telegraf] WITH PASSWORD = N'mystrongpassword';
GO
GRANT VIEW SERVER STATE TO [telegraf];
GO
GRANT VIEW ANY DEFINITION TO [telegraf];
GO
```

6 To include network monitoring, ping test, you can include the Telegraf ping plug-in:

https://github.com/influxdata/Telegraf/tree/master/plugins/inputs/ping

7 The Telegraf config file can be copied to other systems that must be monitored. Review the input sections for each system.

8 Set up Telegraf as Windows service:

```
> c:/monitoring/Telegraf/Telegraf.exe --service install --config
  c:/monitoring/Telegraf/Telegraf.conf
> net start Telegraf
```

Grafana setup

Complete these steps on the Central Monitoring Server to set up Grafana:

1. Open a command prompt as administrator and go to this folder:

   `c:/monitoring/grafana`

2. Run this command:

   `c:/monitoring/grafana/bin/grafana-server.exe`

   Run Grafana as a Windows service and restart it automatically after a system reboot, you can use nssm.

   Grafana runs by default on port 3000. In case of Windows permission issues with port 3000, see http://docs.grafana.org/installation/windows.

   You must open this port in a firewall to access Grafana from other systems.

3. Open a web browser, preferably Firefox or Chrome, and go to `http://localhost:3000`
4. Log in to Grafana with user `admin` and password `admin`.
5. Go to datasources, `http://localhost:3000/datasources`
6. Add a new InfluxDB datasource to the access the `inforlnmonitoring` database. Specify this information:

   Name
   `InforLnMonitoring`

   Query Language
   InfluxQL

   URL
   http://localhost:8086

   Add custom HTTP headers

   Header
   Authorization

   Value:
   API TOKEN

   Token from step 9 in the set up procedure of InfluxDB.

   Example
   `Token pQamJ_Oy8MOewrL66VrewRv5FCK6mZu5VUTsjrYELBkBI6HMWQi43dpnjEwernQ==`

   Database
   `inforlnmonitoring`

7. Click **Save & test**.
8. Download the Infor LN Grafana dashboards from Infor Support Portal KB 2071972
9. Go to `http://localhost:3000/dashboard/import`
10. Import the downloaded dashboards using “Upload JSON file”.
11. Select datasource `InforLnMonitoring` and click **Import**.

12 This dashboard requires the Grafana-piechart-panel plugin. To install, go to `http://localhost:3000/plugins/grafana-piechart-panel`

13 Click **install**.

14 Optionally, load dashboard 1902, Telegraf & Influx Windows Host Overview, with `http://localhost:3000/dashboard/import`

15 Select the `InforLnMonitoring` datasource and click **Import**.

The dashboards are available at `http://localhost:3000/dashboards` to monitor several systems and get more insight in the performance of Infor LN on SQL Server.

Note: We do not recommend that you run the browser on the systems you are monitoring. Rendering these dashboards is CPU intensive and saturates a full CPU core.

Chapter 3: Configuring Infor LN for monitoring

Configure Infor LN, Infor LN UI and C4WS for monitoring.

Metrics

Metrics are defined to gain understanding in the Infor LN application performance. Using extensions, you can add your own metrics.

These metrics are currently defined:

- **Initial LN Access Time**
  Provides insight in the time taken by LN when a user accesses LN for the first time after logging into Infor Ming.le.

- **Single Session Startup Duration**
  Provides insight in the time it takes to startup an LN session from end-user experience.

- **C4WS BDE performance**
  Provides insight in performance of BDE transactions.

- **LN Session Usage Distribution**
  Provides insight into the distribution of LN sessions.

- **Synthetic LN Application Monitoring**
  Put a small synthetic load on CPU, Network and Database to get an idea of the generic application performance. Note that the results can be influenced when the load on the system increases.

Infor LN setup

Complete these steps on the Infor LN server to configure Infor LN monitoring:

1 Modify this file: `%BSE%/lib/defaults/all`

2 Specify this information:

```
monitor_enable:1
monitor_session_start:1
monitor_url:127.0.0.1:8186
monitor_tags:bse=${BSE},environment_type=Production,user=${USER}
```

Note: `monitor_tags` are used in Grafana to filter on certain metrics. You can configure your own `monitor_tags` in `key=value` format. In this way you can send data from a test environment and production environment to the same database. Filter the metrics in the Grafana dashboard. The metrics are sent to the local Telegraf agent running the `influxdb_listener` plugin.

3 To enable synthetic tests, create this file:

`%BSE%/lib/defaults/ttsit`

Ensure the BSE services framework is active and the Infor LN job daemon is running.

4 Specify this content:

```
# ttsit BSE Service resources
# svc_reload_resources_interval: >= 60, default = 300 seconds (5 mins)
# svc_exec_tests_interval: >= 300, default = 3600 seconds (1 hour)
# NOTE: tests should be finished before svc_reload_resources_interval period ends!
svc_reload_resources_interval:300

# Run ttsit2200m000 every 6 hours and send output to InfluxDB:
svc_exec_tests_interval:21600

# Run 1 CPU test for 30 seconds and wait 10 seconds between
cpu:1
cpu_exec_count:1
cpu_wait_secs:10
cpu_params:SECONDS=30

# Run 1 network test and fetch 50000 records from tttxt010
nw:1
nw_exec_count:1
nw_wait_secs:10
nw_params:RECORDS=50000

# Run 1 database test using 1, 2 and 4 parallel bshells
# Insert, update and delete 10000 records on table ttcon010
db:1
db_exec_count:1
db_wait_secs:10
db_params:SERVERS=4,RECORDS=10000
```

5 Restart Infor LN.

Infor LN UI setup

Complete these steps on the Infor LN UI server to configure Infor LN UI monitoring:

1 Edit the `webuiProperties.xml` file in this folder: `C:\Infor\ese\lnui\config`

2 After the closing tag `</environments>` specify this information:

```
<metrics>
   <dispatcher name="Telegraf">
        <type>HTTP_POST</type>
        <url>http://127.0.0.1:8186/write</url>
        <!-- <size>40</size> -->
    </dispatcher>
    <periodic name="lnapplogincount">
        <interval>60</interval>
    </periodic>
</metrics>
```

Ensure the correct `influxdb_listener` URL is used.

3 Restart Infor LN UI.

Configure C4WS

Complete these steps on the Infor LN server to configure Infor LN monitoring

Complete these steps on the Infor LN C4WS server to configure the Infor LN BDE Performance monitoring:

1 Edit the `c4wsProperties.xml` file.

You can find the file in this folder: `C:\Infor\ese\c4ws\config`.

Add this:

```
<c4ws>
…
  <metrics>
    <dispatcher>
        <type>HTTP_POST</type>
        <url>http://127.0.0.1:8186/write</url>
     </dispatcher>
  </metrics>
</c4ws>
```

Ensure the correct `influxdb_listener` URL is used.

2 Restart C4WS.

Chapter 4: Creating custom metrics

Every customer has different requirements, it is difficult to set up generic transaction monitoring.

Therefore, you can create your own transaction monitoring with several small extensions to the standard software. The basics are explained to create transaction monitoring.

For more details see the *Infor LN programmers guide*.

Infor LN Monitoring events

Monitoring events are prepared by defining an event class.

Several types of attributes can be added to an event class:

- **Tags**

  A tag is an identifying attribute of an event. It is always of type string. All events of a certain class have the same value for a tag. By default several tags are added to each event class, as defined by the `monitor_tags` resource.

- **Fields**

  A field is a non-identifying attribute of an event. It can be of type string, long, boolean or double. All events of a certain class have the same value for a field.

- **Metrics**

  A metric is a non-identifying attribute of an event. It can be of type string, long, boolean or double. For each event of a certain class the value for a metric must be provided. When no value for a defined metric is provided, the metric is not represented in the message send to the monitoring system.

  After a class is prepared, events of this class can be triggered. The values for the metrics to be represented in the message generated must be provided.

  The messages are send asynchronously, so the flow of the session is not interrupted. Success of triggering an event indicates the message is send. When the actual sending fails, a bshell message is generated. This message falls outside the flow of the sending session. When messages are generated faster than they can be send, they are combined into larger messages going to the same URL.

Monitoring Intervals

Monitoring intervals are a special type of monitoring events.

In addition to the configured tags, fields and metrics an additional metric (duration) is present. This duration is the time, in seconds with millisecond granularity, between the start of the interval and the actual creation of the message at the end of the interval. The value of this metric is generated automatically.

Definitions

The Infor LN monitoring API uses several concepts that are different from, but are related to, concepts used to describe InfluxDB points.

This table shows the concepts and the description:

Infor LN concept	Explanation, including relation to InfluxDB concepts.
Monitor Event class	A typing of InfluxDB points. All events of an event class have the same InfluxDB measurement and tags. Some fields have the same value for all events in an event class. See Field. The name of the class maps to the InfluxDB measurement.
Monitor Event	An InfluxDB point is generated.
Monitor Interval class	A special type of event, which is used to register durations
Monitor Interval	An event, with a 'duration' metric already added. The interval is created when a start function is called. Calling the stop function triggers the event, with the time, in milliseconds, between start and stop as the value in the duration metric.
Tag	Identical to an InfluxDB tag. This can be
Field	An InfluxDB field, that is identical for each event of a certain event class.
Metric	An InfluxDB field, that is specific for each event.

The InfluxDB timestamps are not visible on the 3GL interface.

A field can also be implemented by sending a metric with the same value at each event. The concept of a field makes this more user friendly.

At this point no restrictions are specified for the tags.

Bshell functions

In addition to the possible errors returned by these functions, they all can return a `monitor_general_error`

This indicates a not monitor specific error and its exact nature can be examined by viewing logs and traces.

monitor_define_event_class

Creates a monitor event class.

Synopsis: `function global long monitor_define_event_class (const string event_class_name)`

This table shows the arguments and the result:

Description		
Arguments	event_class_name	The name of the measurement that is used. A restricted character set applies.
Result	On success	Returns a number > 0, which identifies the monitor event class that is created.
	On failure	No monitor event class is created, and the function returns a number < 0. Possible values: • monitor_not_enabled • monitor_error_invalid_class_name

monitor_define_interval_class

Creates a monitor interval class

synopsis: `function global long monitor_define_interval_class( const string interval_class_name )`

A monitor interval class is a special monitor event class, that has 1 additional metric (‘duration’), the value for this metric is automatically determined, being the time between the start of a timer and the actual occurrence of the event.

This table shows the arguments and the result:

Description		
Arguments	interval_class_name	The name of the measurement that is used for this event class. A restricted character set applies.

Description		
Result	On success	Returns a number > 0, which identifies the monitor interval class that is created
	On failure	No monitor interval class is created, and the function returns a number < 0. Possible values: • monitor_not_enabled • monitor_error_invalid_class_name

monitor_add_tag

Adds a tag to an existing monitor event/interval class.

Adding a tag to a monitor interval class, adds the tag to all three monitor event classes.

Synopsis: `function global long monitor_add_tag ( long event_class_id, const string tag_name, const string tag_value )`

This table shows the arguments and the result:

Arguments	event_class_id	The id of the event or interval the tag must be added to.
	tag_name	The name of the tag. A restricted character set applies.
	tag_value	The value of the tag. A restricted character set applies.
Result	On success	Returns 0
	On failure	Returns an error code (value < 0). Possible values: • monitor_error_unknown_class • monitor_error_invalid_tag_name • monitor_error_invalid_tag_value

monitor_add_field

Adds a field to an existing monitor event/interval class.

synopsis: `function global long monitor_add_field ( long event_class_id, const string field_name, void field_value )`

A field is similar to a metric, but all events of a class have the same value for this field. A field differs from a tag in that it does not identify the event, it provides supplementary information. Furthermore, a field is not restricted to string for its type.

This table shows the arguments and the result:

Description		
Arguments	event_class_id	The id of the event or interval the field must be added to.
	field_name	The name of the field. A restricted character set applies.
	field_value	The value of the field. The type of the actual argument is restricted to: • long • boolean • string • double
Result	On success	Returns 0
	On failure	Returns an error code (value < 0) Possible values: • monitor_error_unknown_class • monitor_error_invalid_field_name • monitor_error_invalid_field_type

monitor_add_metric

Adds a field to an existing monitor event/interval class.

Adding a metric to a monitor interval class, adds the metric to all three monitor event classes.

Synopsis: `function global long monitor_add_metric ( long event_class_id, const string metric_name, long metric_type )`

This table shows the arguments and the result:

Description		
Arguments	event_class_id	The id of the event or interval the metric must be added to.
	metric_name	The name of the metric. A restricted character set applies
	metric_type	The type of the metric. These types exist: • VAR.TYPE.LONG • VAR.TYPE.BOOLEAN • VAR.TYPE.STRING • VAR.TYPE.DOUBLE

Description		
Result	On success	Returns 0
	On failure	Returns an error code (value < 0) Possible values: • monitor_error_unknown_class • monitor_error_invalid_field_name • monitor_error_invalid_field_type

monitor_start_interval

Start a timer for an interval of the indicated class.

synopsis: `function global long monitor_start_interval( long interval_class_id, ... )`

This table shows the arguments and the result:

Description		
Arguments	interval_class_id	The id of the interval class the timer must be started for.
Result	On success	The interval is remembered, and the function returns the interval id (value > 0).
	On failure	No event is reported, nor an interval remembered. The function returns an error code (value < 0) Possible value: monitor_error_unknown_class

monitor_stop_interval

An interval event is reported to the monitoring system.

synopsis: `function global long monitor_stop_interval( long interval_id, ... )`

The event is off the interval class the interval identified belongs to, and contains all tags and fields as defined for this interval class. In addition, the metrics that are given in this call are added to the reported event.

This table shows the arguments and the result:

Description		
Arguments	interval_class_id	The id of the interval class the timer must be reported for.

Description		
	<odd variable argument>	A string value identifying a previously specified metric that is added to the reported event
	<even variable argument>	A value for the metric identified in the <odd variable argument> just before it. The value must be of a type which corresponds to the metric specification.
Result	On success	The interval event is reported, the interval forgotten, and the function returns 0..
	On failure	No event is reported, nor is the interval forgotten. The function returns an error code (value < 0) Possible values: • monitor_error_unknown_interval • monitor_error_unknown_class • monitor_error_unknown_field • monitor_error_missing_field_value • monitor_error_field_value_of_wrong_type • monitor_error_invalid_field_value

monitor_event

An event will be reported to the monitoring system.

synopsis: `function global long monitor_event( long event_class_id, ... )`

The event is off the class identified and contains all tags and fields as defined for this class. In addition, the metrics that are given in this call are added to the reported event.

This table shows the arguments and the result:

Description		
Arguments	event_class_id	The id of the event class the event must be reported for.
	<odd variable argument>	A string value identifying a previously specified metric that will be added to the reported event.
	<even variable argument>	A value for the metric identified in the <odd variable argument> just before it. The value must be of a type which corresponds to the metric specification.

Description		
Result	On success	The event is reported, the interval forgotten, and the function returns 0.
	On failure	No event is reported, and the function returns an error code (value < 0) Possible values: • monitor_error_unknown_class • monitor_error_unknown_field • monitor_error_missing_field_value • monitor_error_field_value_of_wrong_type • monitor_error_invalid_field_value

monitor_remove_interval

Removes an interval

Synopsis: `function global long monitor_remove_interval( long interval_id )`

When this function succeeds, subsequent use of this interval id results in a monitor_error_unknown_interval error.

This table shows the arguments and the result:

Description		
Arguments	interval_id	The id of the interval that must be removed.
Result	On success	Returns 0.
	On failure	Returns an error code (value < 0) Possible value: monitor_error_unknown_interval

monitor_remove_interval_class

Removes an interval class.

Synopsis: `function global long monitor_remove_interval_class( long interval_class_id )`

In addition to the interval class itself, all associated intervals are removed.

When this function succeeds, subsequent use of this interval class id results in a monitor_error_unknown_class error.

This table shows the arguments and the result:

Description		
Arguments	interval_id	The id of the interval class that must be removed.
Result	On success	Returns 0.
	On failure	Returns an error code (value < 0) Possible value: monitor_error_unknown_class

monitor_remove_event_class

Removes an event class.

Synopsis: `function global long monitor_remove_event_class( long event_class_id )`

When this function succeeds, subsequent use of this event class id results in a monitor_error_unknown_class error.

This table shows the arguments and the result:

Description		
Arguments	event_class_id	The id of the event class that must be removed.
Result	On success	Returns 0.
	On failure	Returns an error code (value < 0) Possible value: monitor_error_unknown_class

Monitor Error Codes

All monitor functions return a Long. When this result is greater than or equal to zero, the function succeeded. When it is less than zero the function failed.

These are the possible error codes:

- monitor_not_enabled

  The current configuration does not send any messages to the monitoring system. No event/interval class is created.
- monitor_error_invalid_class_name

  The name provided contains characters that are not allowed.
- monitor_error_unknown_class

  The event/interval class id does not identify a valid class.
- monitor_error_invalid_tag_name

The name provided contains characters that are not allowed.

- monitor_error_invalid_tag_value
  The string value provided contains characters which are not allowed.
- monitor_error_invalid_field_name
  The name provided contains characters that are not allowed.
- monitor_error_invalid_field_type
  The type provided is not one of the defined monitor field types.
- monitor_error_invalid_field_value
  The value provided cannot be used, for example, a string value containing a newline character.
- monitor_error_unknown_interval
  The interval id does not identify a valid interval.
- monitor_error_unknown_field
  The metric name provided in an event is unknown for the monitor event class.
- monitor_error_missing_field_value
  The argument providing the metric value is missing.
- monitor_error_field_value_of_wrong_type
  The type of the value provided doesn't correspond to the type specified for this metric.
- monitor_general_error
  Function failed, see logs/traces for more information

Example

This example instruments the warehousing confirm shipment process using an Infor LN extension.

Note: Implementing monitoring using extensions can have caveats. When a pop up window is displayed to specify a device after the you clicked **Confirm shipment**, extra wait time is included in the measurement. This makes the measurement inaccurate. In the example the number of shipment lines is counted. In reality the number of actual processed shipment lines can differ. In the example is no error handling included. These steps are required to add a performance monitor to a form command in a standard session:

1. Create a session extension.
2. Add the extension type Standard Form Command for the form command to monitor.
3. Implement the Declaration hook for global variables. See this code:

```
table twhinh430
long g.monitor.conf_shipment, g.monitor.interval.id
```

4. Implement the Function hook to code helper functions. See this code:

```
function long get.num.shipment_lines()
{
```

```
long cnt
cnt = 0
select count(whinh431.shpm):cnt
from whinh431
where whinh431.shpm = :whinh430.shpm
selectdo
endselect
return(cnt)
}
```

5 Implement the Before Command hook to activate the monitor. See this code:

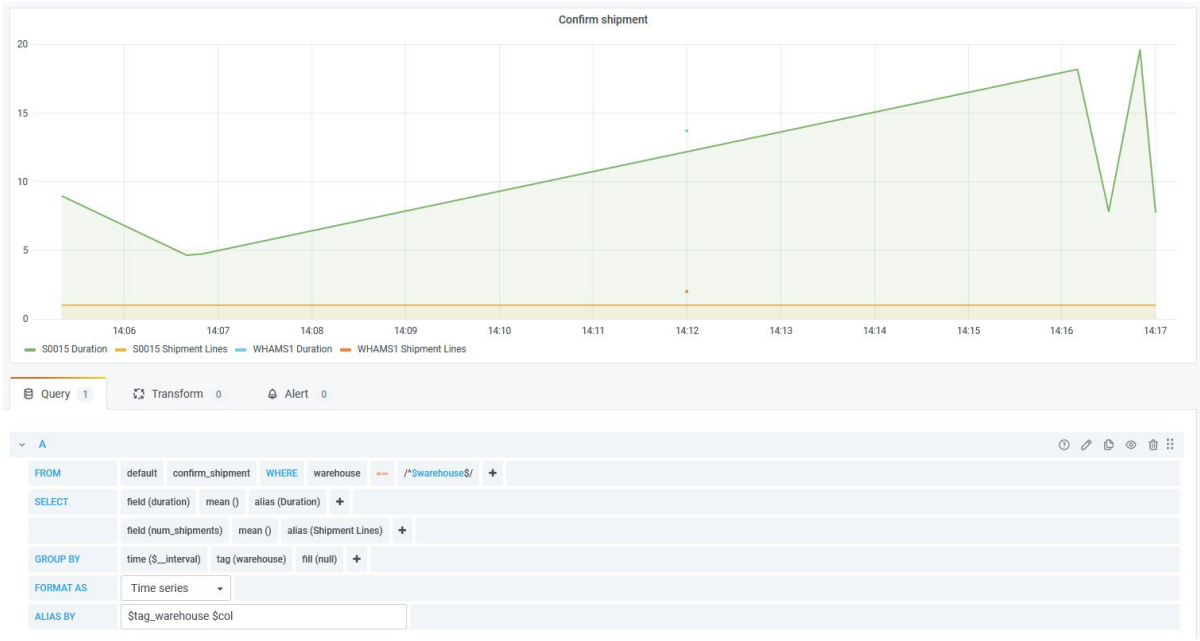
```
function extern void function.confirm.shipment.before.command()
{
  long num.shipment_lines, ret
  string warehouse(20)
  ret = monitor_define_interval_class("confirm_shipment")
  if ret > 0 then
          g.monitor.conf_shipment = ret
          warehouse = whinh430.sfco
          ret = monitor_add_tag(g.monitor.conf_shipment, "warehouse", strip$(warehouse))
  endif
  if ret = 0 then
          num.shipment_lines = get.num.shipment_lines()
          ret = monitor_add_field(g.monitor.conf_shipment, "num_shipment_lines", num.ship
ment_lines)
    else
          | Any error will disable monitoring
           g.monitor.conf_shipment = 0
    endif
    | Before start processing
    if g.monitor.conf_shipment > 0 then
            ret = monitor_start_interval(g.monitor.conf_shipment)
            if ret > 0 then
                    g.monitor.interval.id = ret
            else
                    | Error stop monitoring
                    g.monitor.interval.id = 0
                    ret =
monitor_remove_interval_class(g.monitor.conf_shipment)
                    g.monitor.conf_shipment = 0
            endif
    endif
}
```

6 Implement the After Command hook to register the measured time. See this code:

```
function extern void function.confirm.shipment.after.command()
{
    long ret
    | After processing
    if g.monitor.interval.id > 0 then
            ret = monitor_stop_interval(g.monitor.interval.id)
            g.monitor.interval.id = 0
            ret =
monitor_remove_interval_class(g.monitor.conf_shipment)
            g.monitor.conf_shipment = 0
    endif
}
```

After the form command is used to confirm shipments, the measurements flow into InfluxDB and a Grafana dashboard can be created.

See this screen shot:


Confirm shipment
20
15
10
5
0
14:06
14:07
14:08
14:09
14:10
14:11
14:12
14:13
14:14
14:15
14:16
14:17
S0015 Duration
S0015 Shipment Lines
WHAMS1 Duration
WHAMS1 Shipment Lines
Query 1
Transform 0
Alert 0
A
FROM
default
confirm_shipment
WHERE
warehouse
=~
/^$warehouse$/
SELECT
field (duration)
mean ()
alias (Duration)
field (num_shipments)
mean ()
alias (Shipment Lines)
GROUP BY
time ($__interval)
tag (warehouse)
fill (null)
FORMAT AS
Time series
ALIAS BY
$tag_warehouse $col