

Continuum Documentation

Contents

Getting started with OpCon Continuum

Builder

- Get familiar as a Builder
- Build a service request
- Build a workflow
- Define a frequency
- Builder documentation
- Run commands on legacy LSAM machines
- Manage properties, calendars, and tags
- Use a Null Job
- Order jobs with dependencies
- Run a command
- Run events on job outcomes
- Run a script
- Schedule and build a workflow
- Run SQL against your databases
- Use thresholds and resources
- Version and deploy a workflow
- Wait for a file before your workflow continues
- Embed another workflow with a Workflow Container

Operator

- Get familiar as an Operator
- Respond to a failed connector task
- Operator documentation
- Job statuses and actions
- Monitor the daily run
- Respond to a failed job
- Search across OpCon Continuum
- Use the event log
- Respond to a stuck or failed Wait for File task
- Work the Processes page

Administrator

- Get familiar as an Administrator
- Control which folders agents can access

- Build and run reports
- Set up database connections
- Define custom fields
- Administrator documentation
- Manage agents and agent pools
- Manage connections
- Manage environments
- Manage plugins
- Manage roles and permissions
- Manage workspaces
- Prepare agents for connectors
- Set up and manage relays

Non-OpCon User

- Get familiar as a Non-OpCon User
- Non-OpCon User documentation
- Submit a service request
- What Continuum can automate

Concepts

- How connectors work
- Dashboards
- How the daily run works
- How workflows work
- Self-service portal
- Toolkit overview

Reference

- Agents and agent pools
- Relays
- Plugins
- Environments
- Workspaces
- Run Command task
- Run Script task
- Wait for File task
- SQL Database Executor connector
- Legacy LSAM connectors (UNIX/Linux and Windows)
- Null Job task
- Workflow Container task

Scheduling and build

Frequencies

Calendars

Job dependencies

Thresholds and resources

Workflow events

The Processes page (runtime)

Job and workflow statuses

Operator response guide

Event log

Vision (critical-path monitoring & SLAs)

Notification Manager (groups, triggers, escalation, channels)

Job configuration

Connections

Properties and tags

Custom fields

Workflow versions and deployments

Service requests

Roles and permissions

Reports

Global search

Glossary

Getting started with OpCon Continuum

OpCon Continuum automates the work your organization runs every day, and coordinates how your systems, including your mainframe core, work together. Instead of people running steps by hand and watching for the next one, you build the process once and let it run reliably, in the right order, at the right time.

This page orients you: who does what, what the main pieces are, and where to begin.

⚠ WHAT THIS SOLVES

Running daily operations by hand (people kicking off steps, watching for the next one, reacting to failures in the middle of the night) is slow, error-prone, and doesn't scale as the business grows.

Who uses it

OpCon Continuum is built around four roles. You don't need all of them: many teams wear several hats.

Role	Focus
Builder	Designs the automation: workflows, the jobs in them, and what each job does.
Operator	Watches the daily run and steps in when a job needs attention.
Administrator	Sets up and maintains the environment, agents, and connections.
Non-OpCon User	Understands the value and capabilities; may run published requests via self-service.

When you sign in, you land on a **dashboard** for your role: an Administrator sees environment and agent health, an Operator sees the day's run and any failures, a business user sees the requests they can run.

The main pieces

- **Workflows:** the automated processes: a set of **jobs** that run in order, on a schedule.
- **Connectors:** what a job actually does: run a command or script, wait for a file, run a database operation, or drive a legacy system. Connectors are how Continuum works with the systems you already have.
- **The daily run:** Operators monitor workflows as they run and respond to anything that fails.
- **Toolkit:** reusable building blocks Builders share across workflows: schedules (frequencies), holiday calendars, limits (thresholds and resources), values (properties), and tags.
- **Service requests & self-service:** forms that let business users safely trigger automation without touching the workflows.
- **Agents & connections:** where jobs run, and the secured links to your external systems.
- **Reports, event log, and search:** visibility into what ran and what happened.

Where to begin

- **Builders:** start with [Build a workflow](#), then [order jobs with dependencies](#) and [schedule it](#).
- **Operators:** start with [Monitor the daily run](#) and [Respond to a failed job](#).
- **Administrators:** start with [Manage agents and pools](#), [Manage connections](#), and [Manage plugins](#).
- **Non-OpCon Users:** see [What OpCon Continuum can automate](#) and [Submit a service request](#).

New to the terms? See the [Glossary](#).

Tags: [Concept](#) [Non-OpCon User](#) [Orientation](#) [LMS-Training](#)

*Last updated on **Aug 3, 2026***

Get familiar as a Builder

Reading about Continuum only gets you so far. The fastest way to learn it is to do the core Builder tasks in a test environment. Work through this checklist from top to bottom, checking off each item as you complete it. Each item links to the page that explains it in detail.

Goal: design reliable workflows and the building blocks they use.

Build a workflow

- ☐ Open Workflows and add a job: select **First Job** on an empty workflow, or **New Job** from the **Job Tools** panel.
- ☐ On the **Task Definition** tab, select **Show Task Types** and choose one, such as Run Command or a SQL task, and set its parameters.
- ☐ Set the job's agent assignment: a specific agent or an agent pool.
- ☐ Select **Save & Close**.
- ☐ See [Build a workflow](#).

Order and time the work

- ☐ On the **Dependencies** tab, add a job dependency and try a threshold, resource, or expression dependency.
- ☐ On the **Frequencies** tab, set the job's timing.
- ☐ Review the workflow's build settings, including days in advance and build time.
- ☐ See [Order jobs with dependencies](#) and [Schedule and build a workflow](#).

Toolkit

- ☐ Define a frequency and check its year-calendar preview.
- ☐ Create a threshold, a resource, a property (try marking one **encrypted**), and a tag.

☐ See [Define a frequency](#), [Use thresholds and resources](#), and [Manage properties, calendars, and tags](#).

Events and reuse

- ☐ On the **Events** tab, add an event that fires on a job status such as Failed.
- ☐ Add a Null Job to act as a dependency anchor, and a Workflow Container to embed another workflow as a step.
- ☐ See [Run events on job outcomes](#), [Use a Null Job](#), and [Embed another workflow with a Workflow Container](#).

Service requests, versioning, and deploy

- ☐ Build a service request with inputs and events, then deploy it to an environment.
- ☐ Save a change to a workflow to create a new version, then deploy it as **Latest** or **Pinned**.
- ☐ Find one of your objects using search.
- ☐ See [Build a service request](#), [Version and deploy a workflow](#), and [Search across OpCon Continuum](#).

You're ready when

You can build a multi-job workflow with dependencies, a frequency, and an event; use Toolkit building blocks; publish a service request; and version and deploy to an environment.

Related topics

- [Getting started with OpCon Continuum](#)
- [Glossary](#)

Tags: [Procedure](#) [Builder](#) [Orientation](#)

*Last updated on **Jul 29, 2026***

Build a service request

A **service request** lets a business user trigger automation from a simple form. No workflow knowledge is needed. You design the form fields and the events that fire when it's submitted, then deploy it to an environment so it appears in the self-service portal.

! WHAT THIS SOLVES

Business users need routine automation run for them, but giving them workflow access is risky and running every request by hand is slow and interrupt-driven for the automation team.

What you'll set up

- **Inputs:** the fields the user fills in (text, password, number, date, or a choice field), each optionally required and validated.
- **Events:** what happens on submit. Event parameters can include the user's answers using `${input.<field>}` placeholders.
- **Documentation** and a **confirmation message** to guide the user.
- **Tags** for organizing and for filtering in the portal.

Create a service request

To build a service request, complete the following steps:

1. Go to **Service Requests** and create a new request with a **name**.
2. Add **inputs**: for each, set the type, whether it's required, and any validation.
3. Add **events**: choose the event type and fill in its parameters, using `${input.<field>}` to pass the user's answers.
4. Add **documentation** and a **confirmation message** if helpful.
5. Save (this creates a version), then **deploy** it to an environment.

Deploy it

Deploy the request to each environment where it should be available, choosing **Latest** or a **Pinned** version (and optional effective/expiration dates), the same model as workflows. Use a **transformation** to override a value per environment (for example, point a request at the production system in PROD).



GOOD TO KNOW

- Today's input types are **text, password, number, date, and choice**. (Workflow/job picker inputs are planned but not active yet.)
- Use a **password** input plus a connection/property for secrets. Don't collect credentials in a plain field.
- A request only appears in the portal where it's **deployed and active**.

Related topics

- [Service requests](#) — full configuration reference and troubleshooting
- [Submit a service request](#) (Non-OpCon User)
- [Version and deploy a workflow](#) (Builder)

Last updated: 2026-07-15

Tags:

[Procedure](#)

[Builder](#)

[Service Requests](#)

[LMS-Training](#)

*Last updated on **Aug 3, 2026***

Build a workflow

A workflow in OpCon Continuum is a set of **jobs** that run in a defined order, on a schedule. You build one by adding jobs, telling each what to do, where to run, when it's eligible, and what it depends on. This page covers creating a workflow and adding a job; the related pages cover ordering, scheduling, and responding to outcomes.

! WHAT THIS SOLVES

Business processes are made of many steps that must run in the right order, at the right time, and only when their inputs are ready. Done by hand or with scattered scripts, the order drifts, steps get missed, and no one can see the whole process in one place.

Add a job to a workflow

To create a workflow and add its first job, complete the following steps:

1. Go to **Workflows** and open the workflow you want to edit (or create a new one).
2. Add a job: on an empty workflow select **First Job**, otherwise select **New Job** from the **Job Tools** panel.
3. Enter a **name** for the job.
4. On the **Task Definition** tab, select **Show Task Types** and choose the task type the job should run, then set its parameters. See [the connector pages](#) for each task type.
5. Set the job's **agent assignment**: a specific agent (with optional failover), or an agent pool.
6. Select **Save & Close**. The job is added to the workflow.

Repeat for each job the workflow needs.

Next: shape the workflow

Once your jobs exist, use the other tabs in the job editor and the related pages to finish the design:

- **Order the work:** set what each job waits for. See [Order jobs with dependencies](#).
- **Schedule it:** set when jobs are eligible and the workflow's build settings. See [Schedule and build a workflow](#).
- **Respond to outcomes:** fire actions on job statuses. See [Run events on job outcomes](#).
- **Release it:** version and deploy to an environment. See [Version and deploy a workflow](#).



GOOD TO KNOW

- A job runs a **task type**: the same connectors documented for each kind of work (commands, scripts, SQL, file waits, and more).
- Jobs can run on a **specific agent** (with failover) or across an **agent pool**.
- Mark a job **disabled** to keep it in the design without building it into the run.

Related topics

- [Job configuration](#) — full configuration reference and troubleshooting
- [Order jobs with dependencies](#) (Builder)
- [Schedule and build a workflow](#) (Builder)

Last updated: 2026-06-30

Tags: [Procedure](#) [Builder](#) [Workflows](#) [LMS-Training](#)

*Last updated on **Aug 3, 2026***

Define a frequency

A **frequency** is a reusable scheduling rule that decides which days a job is eligible to run. Define it once in the Toolkit, then jobs across your workflows can share it. As you build the pattern, a **year calendar** previews exactly which dates it will match.

ⓘ WHAT THIS SOLVES

Scheduling rules copied into each job drift apart and mishandle holidays and non-working days, so jobs run on days they shouldn't or skip days they should, and it's hard to verify before it happens.

Choose a pattern

Frequencies are built from a **time period** and, for most, a **type**:

Pattern	Example
Daily	Every day, or every N days from a start date.
Weekly	Specific weekdays (Mon/Wed/Fri), or a position within the week.
Monthly	A day of the month (the 15th), a weekday position ("second Tuesday"), or a period ("last working day").
Yearly	A date (Dec 31), a weekday position, or a period within the year.

You can also choose whether to count **working days only**, what to do when a match lands on a non-working day (skip forward/back, keep, or omit), and a **calendar** of dates to exclude.

Create a frequency

To define a frequency, complete the following steps:

1. Go to **Toolkit → Frequencies** and add a frequency.
2. Enter a **name**.
3. Choose the **time period** and **type**, then set its fields (interval, days, day of month, and so on).
4. Optionally set **working-day** handling and an **exclusion calendar**.
5. Check the **year calendar** preview to confirm the matching dates look right.
6. Save.



GOOD TO KNOW

- A frequency is the *which-days* rule. *When within the day*, retries, and reruns are set on the job when you add the frequency to it. See [Schedule and build a workflow](#).
- Use an **exclusion calendar** (defined in **Toolkit → Calendars**) to skip holidays and blackout days.
- The **year calendar** preview doesn't always reflect exclusion calendars. An excluded date can still appear highlighted. The exclusion still applies when the job runs; only the preview may look off.
- Frequencies are shared across workspaces, so name them clearly.

Related topics

- [Frequencies](#) — full configuration reference and troubleshooting
- [Schedule and build a workflow](#) (Builder)
- [Use thresholds and resources](#) (Builder)

Last updated: 2026-06-30

Tags:

[Procedure](#)

[Builder](#)

[Toolkit](#)

[Scheduling](#)

[LMS-Training](#)

*Last updated on **Aug 3, 2026***

Builder documentation

Design reliable workflows and the building blocks they use — from your first workflow to service requests, versioning, and deployment.



Get familiar as a Builder

A checklist of hands-on tasks so you can learn OpCon Continuum by building a workflow, its building blocks, and a service request in a test environment.



Build a service request

How to build a service request so business users can trigger workflows on demand through a guided form.



Build a workflow

How to build a workflow by adding jobs, dependencies, and schedules so Continuum can run your automation end to end.



Define a frequency

How to define a frequency so workflows run only on the days and timing patterns you intend.



Run commands on legacy LSAM machines

How to run commands on legacy LSAM machines so you can automate work on systems connected through the legacy relay.



Manage properties, calendars, and tags

How to manage properties, calendars, and tags so you can reuse values, control timing, and organize work across workflows.



Use a Null Job

When and how to use a Null Job as a placeholder or dependency anchor to simplify workflow structure.



Order jobs with dependencies

How to order jobs with dependencies so each job runs only after the jobs it relies on reach the right outcome.



Run a command

How to add and configure a Run Command task so a job runs a command or program on an agent.



Run events on job outcomes

How to run events on job outcomes so Continuum reacts automatically when a job succeeds, fails, or reaches a status.



Run a script



Schedule and build a workflow

How to schedule and build a workflow so it is added to the daily run and ready to submit jobs.



Run SQL against your databases

How to run SQL against your databases from a workflow using the database task types.



Use thresholds and resources

How to use thresholds and resources to control concurrency and coordinate jobs that share limited capacity.



Version and deploy a workflow

How to version and deploy a workflow so you can review, promote, and roll back changes safely.



Wait for a file before your workflow continues

How to add and configure a Wait for File task so a workflow waits until a file has arrived and is complete before continuing.



Embed another workflow with a Workflow Container

How to embed one workflow inside another with a Workflow Container so you can reuse and nest automation.

New here? See [Getting started with OpCon Continuum](#).

These pages walk you through the tasks. When you need exhaustive settings, status values, or a troubleshooting table, see [Reference](#).

*Last updated on **Aug 3, 2026***

Run commands on legacy LSAM machines

If you already run legacy LSAM agents, Continuum can drive them directly. You don't need to rebuild that automation to start orchestrating it. The **UNIX Command** and **Windows Command** task types run a program on an existing UNIX/Linux or Windows LSAM machine through the relay, and **Windows - WS_FTP Pro** runs a WS_FTP Pro file transfer on a Windows LSAM machine.

! WHAT THIS SOLVES

You're adopting Continuum but still have working automation on legacy LSAM machines. Rebuilding all of it before you can orchestrate anything would stall the whole move.

Use it when

- You're moving to Continuum but still have automation on legacy LSAM machines.
- A workflow needs to run a program on a UNIX/Linux or Windows LSAM machine.

Add an LSAM command to a workflow

To add an LSAM command, complete the following steps:

1. Go to **Workflows** and open the workflow you want to edit.
2. Add a job: on an empty workflow select **First Job**, otherwise select **New Job** from the **Job Tools** panel.
3. In the job editor, select the **Task Definition** tab.
4. Select **Show Task Types**.
5. In the task type catalog, under **Integration**, select **UNIX Command** or **Windows Command**.
6. Under **Task Parameters**, set the machine and the program to run (see below).
7. Select **Save & Close**.

Settings

Setting	Notes
Machine name	The LSAM machine to run on. It must match the machine name registered with the relay.
Program / start image	The program or script to run on the machine.
Arguments	Command-line arguments for the program.
Working directory	Where the command runs on the machine.
Prerun command	An optional command run before the main one.
Run-as user	UNIX: the user (and group) to run as. Windows: Use Service Account is the only supported option; the job runs under the LSAM's service account.
Environment variables	Extra environment variables (Name = Value) for the job. Now supported on Windows LSAM as well as UNIX.
Exit criteria	Exit-code conditions that mark the job failed.

Windows LSAM commands also support:

- **Run in command shell:** run the command inside a command shell instead of starting the program directly.
- **Output parsing:** scan the job's output for text (**Contains / Does Not Contain**) and set an exit code when it matches, so a job that "succeeds" but logs an error can still be flagged failed. Up to five rules.
- **Custom log file:** the log file that output parsing scans; defaults to the job's standard output.

UNIX commands also support group, process priority, and signal/core-dump failure conditions.

Transfer files with WS_FTP Pro

On a Windows LSAM machine that runs **Ipswitch WS_FTP Pro**, the **Windows - WS_FTP Pro** task type runs a file transfer using WS_FTP Pro's own **site profiles**. You pick the task type the same way as a Windows Command (**Show Task Types → Integration → Windows - WS_FTP Pro**), then fill in the transfer.

Before you start: the source and destination **site profiles** must already be set up in WS_FTP Pro on that machine. The job refers to them **by name** and stores **no passwords or connection details**; WS_FTP Pro supplies those from the profile.

Setting	Notes
Machine name	The Windows LSAM machine to run on (must match the name registered with the relay).
Run-as user	<code>Use Service Account</code> is the only supported option; the job runs under the LSAM's service account.
WS_FTP Pro location	The folder containing <code>wsftppro.exe</code> on the machine.
Source profile / Source file	The WS_FTP Pro site profile name for the source, and the file to transfer.
Destination profile / Destination file	The site profile name for the destination, and the file to write.
File transfer options	Optional extra WS_FTP Pro command-line switches (for example, <code>-binary -delete</code>).

All fields except the options are **required**. Profile names can't contain a colon (:), and none of the text fields can contain a double quote ("). The form won't stop you from saving an invalid value: an empty required field, a stray " or :, or a location that's too long fails when the job is **dispatched**, not while you're editing, so double-check these before you run it.



GOOD TO KNOW

- These run through the **relay** on your existing LSAM machines, not the Universal Agent. If a command reports "no agent available," it's usually relay or machine connectivity; check with your administrator.
- On **Windows LSAM**, jobs run under the LSAM service account. **Per-job environment variables are now supported**; embedded scripts are not, so put your logic in the program or a script that already exists on the machine.

Related topics

- [Legacy LSAM connectors \(UNIX/Linux and Windows\)](#) — full configuration reference and troubleshooting
- [Prepare agents for connectors](#) (Administrator)
- [Respond to a failed connector task](#) (Operator)

Last updated: 2026-07-31

Tags: [Reference](#) [Builder](#) [Connectors](#) [Legacy](#) [LMS-Training](#)

*Last updated on **Aug 3, 2026***

Manage properties, calendars, and tags

Three more Toolkit building blocks help you parameterize, schedule around holidays, and organize your work.

ⓘ WHAT THIS SOLVES

Paths, environment names, and secrets get hard-coded into individual jobs, so one change means editing many places, and credentials end up sitting in plain job definitions.

Properties: reusable values

A **property** is a named value (like a path or an environment name) you can reuse across jobs instead of hard-coding it. Mark a property **encrypted** to store a secret safely.

To create a property, complete the following steps:

1. Go to **Toolkit → Properties** and add a property.
2. Set its **name** and **value**.
3. Mark it **encrypted** if it's a secret.

ⓘ INFO

An encrypted property's value is hidden after saving. That's expected. Use encrypted properties for credentials instead of putting secrets in job parameters.

Calendars: days to skip

A **calendar** is a named set of dates (holidays, blackout days). Reference it from a [frequency](#) as an **exclusion calendar** so jobs skip those days.

To create a calendar, complete the following steps:

1. Go to **Toolkit** → **Calendars** and add a calendar.
2. Add its dates.

Tags: organize and find

A **tag** is a label you apply to jobs (and other objects) to group and search them.

To create a tag, complete the following steps:

1. Go to **Toolkit** → **Tags** and add a tag.
2. Apply it to jobs as you build them.



GOOD TO KNOW

- **Properties** are workspace-scoped; **calendars** and **tags** are shared across workspaces.
- Tags power filtering and global search, so a consistent tag scheme pays off at scale.

Related topics

- [Properties and tags](#) — full configuration reference and troubleshooting
- [Calendars](#) — full configuration reference and troubleshooting
- [Define a frequency](#) (Builder)
- [Build a workflow](#) (Builder)

Last updated: 2026-07-15

Tags: [Procedure](#) [Builder](#) [Toolkit](#)

*Last updated on **Aug 3, 2026***

Use a Null Job

A **Null Job** does no work and finishes immediately. It's a workflow-design building block: a clean anchor point for organizing dependencies and events. It runs inside the platform, so it needs no agent and has no settings.

ⓘ WHAT THIS SOLVES

As a workflow grows, wiring every job to every other for ordering, or hanging events on a real job that does unrelated work, makes the design tangled, hard to read, and fragile to change.

Use it when

- **Simplifying dependencies:** give many jobs a single shared predecessor or successor instead of wiring every job to every other (a clean many-to-one or one-to-many point).
- **Attaching events:** provide a job to hang events on without performing real work.
- **Building on hold:** act as a controllable placeholder when a workflow is built on hold.

Add a Null Job to a workflow

To add a Null Job, complete the following steps:

1. Go to **Workflows** and open the workflow you want to edit.
2. Add a job: on an empty workflow select **First Job**, otherwise select **New Job** from the **Job Tools** panel.
3. In the job editor, select the **Task Definition** tab.
4. Select **Show Task Types**.
5. In the task type catalog, under **Internal**, select **Null Job**.
6. Select **Save & Close**. There are no parameters to set.



GOOD TO KNOW

- A Null Job finishes OK as soon as it runs. It has no failure of its own. If one is waiting, it's waiting on its dependencies, which is by design.

Related topics

- [Null Job task](#) — full configuration reference and troubleshooting
- [Embed another workflow with a Workflow Container](#) (Builder)

Last updated: 2026-07-15

Tags:

[Reference](#)

[Builder](#)

[Workflow design](#)

[LMS-Training](#)

*Last updated on **Aug 3, 2026***

Order jobs with dependencies

Dependencies control **when each job runs**: a job waits until everything it depends on is satisfied. Use them to sequence work (run B after A succeeds), coordinate shared limits, or run recovery steps when something fails.

! WHAT THIS SOLVES

Jobs that only run on a timer either start before the data they need is ready, or plough ahead after an upstream step has failed, producing wrong results that no one catches until much later.

Dependency types

Type	A job waits until...
Job	A predecessor job reaches a result (success , failure , or any), optionally from a previous day.
Threshold	A named threshold reaches a value.
Resource	Enough of a named resource is free (the job reserves it).
Expression	A custom expression is true.

Add a dependency

To add a dependency to a job, complete the following steps:

1. In the job editor, open the **Dependencies** tab.
2. Add a dependency and choose its **type** (job, threshold, resource, or expression).
3. For a **job** dependency, choose the predecessor job, the required **result** (success, failure, or any), and a day offset if it depends on a previous day's run.

4. For a **threshold** or **resource** dependency, choose the threshold/resource and the value or amount required.
5. Select **Save & Close**.

What counts as success, failure, or any

For a **job** dependency, the required result maps to more than one outcome:

Result	The predecessor is satisfied when it...
success	Finished OK, including when it was marked finished OK , fixed , or skipped .
failure	Failed, including a marked-failed , an initialization error , or a job flagged under review .
any	Reached either a success or a failure result (above). It does not mean "any outcome."

A cancelled or missed job never releases its dependents. If a predecessor is **cancelled** or **misses its start time**, it never produced a result, so a dependent (even one set to **any**) keeps waiting. That's intended. If you see a job stuck waiting on a predecessor that was cancelled or missed, clear it from **Processes** with **Force Start** or **Skip**, or fix the upstream job.



GOOD TO KNOW

- A job runs only when **all** its dependencies are satisfied.
- Depending on **failure** or **any** is a valid pattern. For example, a cleanup job runs only when its predecessor fails.
- A **resource** dependency makes a job wait for capacity, so use it to keep too many jobs from running at once against a shared system.

Related topics

- [Job dependencies](#) — full configuration reference and troubleshooting
- [Build a workflow](#) (Builder)
- [Schedule and build a workflow](#) (Builder)

Last updated: 2026-07-28

Tags: [Procedure](#) [Builder](#) [Workflows](#) [LMS-Training](#)

*Last updated on **Aug 3, 2026***

Run a command

The **Run Command** task runs a command line on the agent: call a utility, trigger a batch file, kick off an existing process. It's built in, so there's nothing to set up.

! WHAT THIS SOLVES

An existing program or batch file needs to run as a step in a larger process, on schedule and with its result captured. Triggering it by hand, or with a stray scheduled task on some server, is the manual glue that silently breaks and leaves no record.

Use it when

- You need to run a single program or shell command as a step in a workflow.
- You want to use shell features like pipes or redirects.

Add a Run Command task to a workflow

To add a Run Command task, complete the following steps:

1. Go to **Workflows** and open the workflow you want to edit.
2. Add a job: on an empty workflow select **First Job**, otherwise select **New Job** from the **Job Tools** panel.
3. In the job editor, select the **Task Definition** tab.
4. Select **Show Task Types**.
5. In the task type catalog, under **Script**, select **Run Command**.
6. Under **Task Parameters**, in **Command**, enter the command to run.
7. Set any optional settings (see below).
8. Select **Save & Close**.

Settings

Setting	Notes
Command	The command line to run. Required.
Arguments	Arguments appended to the command.
Working Directory	Where the command runs. Defaults to the agent's working directory.
Environment Variables	Extra environment variables for the command.
Run via Shell	On by default: runs through the system shell so pipes and redirects work. Turn off to run a program directly.
Fail on Error	On by default: a non-zero exit code fails the task. Turn off to succeed regardless.



GOOD TO KNOW

- The command runs **on the agent**, so the program must exist there. If it can't be found, check the command and working directory, or ask your administrator about the agent.
- For security, the command can't read the agent's stored secrets from its environment (variables named like a password or secret are filtered). Pass values you need as **Environment Variables**, or use a connection-based connector for credentials.
- For multi-line logic, use [Run a script](#) instead.

Related topics

- [Run Command task](#) — full configuration reference and troubleshooting
- [Run a script](#) (Builder)
- [Respond to a failed connector task](#) (Operator)

Last updated: 2026-07-15

Tags:

[Reference](#)

[Builder](#)

[Connectors](#)

[LMS-Training](#)

*Last updated on **Aug 3, 2026***

Run events on job outcomes

An **event** runs a command in response to what a job does: for example, send an alert when a job fails, or trigger a follow-up action when a job finishes late.

Events are set per job.

⚠ WHAT THIS SOLVES

When a job fails, finishes late, or ends with an unexpected result, nothing happens unless someone is watching, so problems sit silently and any follow-up waits for a person to notice.

Event triggers

Trigger	Fires when
Job status	The job reaches one of these statuses: Finished OK, Failed, Fixed, Skipped, Under Review, Start Attempted, Still Attempting Start, Late to Start, Late to Finish, Missed Latest Start Time, or Exceeded Max Runtime.
Job exit description	The job's exit value matches a comparison you set (equal to, greater than, a range, and so on).
Job completion expression	A custom expression is true.

Add an event

To add an event to a job, complete the following steps:

1. In the job editor, open the **Events** tab.
2. Add an event and choose the **trigger** (job status, exit description, or completion expression).

3. Set the condition: the status, the comparison and value, or the expression.
4. Enter the **command** to run when the event fires.
5. Select **Save & Close**.



GOOD TO KNOW

- An event firing is the system working as designed (for example, a Failed-status event that sends an alert). It isn't itself a failure.
- Events decide *when* something fires; the **command** decides *what* happens.

Related topics

- [Workflow events](#) — full configuration reference and troubleshooting
- [Build a workflow](#) (Builder)
- [Respond to a failed job](#) (Operator)

Last updated: 2026-07-15

Tags:

[Procedure](#)

[Builder](#)

[Workflows](#)

*Last updated on **Aug 3, 2026***

Run a script

The **Run Script** task runs an embedded PowerShell, Bash, Python, or Shell script on the agent. Use it instead of Run Command when the logic is more than a single command line. It's built in, so there's nothing to set up.

! WHAT THIS SOLVES

A step needs real logic: several commands, a loop, a conditional. Kept in a separate script file on some server, that logic drifts from the job and no one remembers where it lives or what it assumes.

Use it when

- Your step needs several commands, conditionals, or loops.
- You want to keep the script with the job rather than as a separate file.

Add a Run Script task to a workflow

To add a Run Script task, complete the following steps:

1. Go to **Workflows** and open the workflow you want to edit.
2. Add a job: on an empty workflow select **First Job**, otherwise select **New Job** from the **Job Tools** panel.
3. In the job editor, select the **Task Definition** tab.
4. Select **Show Task Types**.
5. In the task type catalog, under **Script**, select **Run Script**.
6. Under **Task Parameters**, choose the **Script Type** and enter your **Script Content**.
7. Set any optional settings (see below).
8. Select **Save & Close**.

Settings

Setting	Notes
Script Type	PowerShell, Bash, Python, or Shell. Shell is the default.
Script Content	The script to run. Required.
Script Arguments	Arguments passed to the script.
Working Directory	Where the script runs. Defaults to the agent's working directory.
Environment Variables	Extra environment variables for the script.
Fail on Error	On by default: a non-zero exit code fails the task. Turn off to succeed regardless.



GOOD TO KNOW

- The interpreter must be **installed on the agent**: PowerShell 7 (`pwsh`) on Linux/macOS, Python, or Bash as needed. Shell is always available. If a script won't start, confirm the interpreter with your administrator.
- PowerShell scripts run without your profile and with execution policy bypassed. Make the script self-contained rather than relying on a console's setup.
- Like Run Command, scripts can't read the agent's stored secrets from the environment. Pass what you need as **Environment Variables** or use a connection-based connector.

Related topics

- [Run Script task](#) — full configuration reference and troubleshooting
- [Run a command](#) (Builder)
- [Prepare agents for connectors](#) (Administrator)

Last updated: 2026-07-15

Tags:

[Reference](#)

[Builder](#)

[Connectors](#)

[LMS-Training](#)

*Last updated on **Aug 3, 2026***

Schedule and build a workflow

Two things decide when work happens: **frequencies** (which days and times a job is eligible) and **build settings** (how and when the workflow becomes a concrete day's run). Holiday **calendars** and **working days** remove days that shouldn't run.

ⓘ WHAT THIS SOLVES

A workflow scheduled wrong (wrong days, no holiday handling, or built too early) either runs when it shouldn't or isn't ready when the business needs it.

Set when a job runs

To set a job's schedule, complete the following steps:

1. In the job editor, open the **Frequencies** tab.
2. Add the frequency (or frequencies) the job should run on.
3. Set the job's timing for that frequency: start-time offset, late-to-start and late-to-finish thresholds, and maximum run time.
4. Optionally enable **failure retries** (how many attempts, how far apart) and **success reruns**.
5. Select **Save & Close**.

Set how the workflow builds

In the workflow's settings, set how the day's run is created:

- **Days in advance** and **build time**: when the run is built.
- **Build on hold**: build the run held, so it won't start until you release it.
- **Holiday calendar** and **working days**: exclude days the workflow shouldn't run.
- **Retention**: how long built runs are kept.



GOOD TO KNOW

- A job appears in a day's run only when its **frequency matches** that date and the day isn't excluded by a calendar or working-days rule.
- The late-to-start, late-to-finish, and max-run-time settings drive the matching job statuses you (and Operators) can act on.
- **Build on hold** is the safe way to stage a workflow without it running until you're ready.

Related topics

- [Scheduling and build](#) – full configuration reference and troubleshooting
- [Order jobs with dependencies](#) (Builder)
- [Define a frequency](#) (Builder)

Last updated: 2026-06-30

Tags:

[Procedure](#)

[Builder](#)

[Workflows](#)

[LMS-Training](#)

*Last updated on **Aug 3, 2026***

Run SQL against your databases

Bring your databases into a workflow: run a T-SQL script when nightly core processing finishes, kick off and watch a SQL Server Agent job, refresh a reporting database, or run an SSIS package as one step in a larger flow. The **SQL Database** connectors handle SQL Server, MySQL, Oracle, and any ODBC-compatible database.

Each SQL task runs against a **connection**: the server and login, set up once by your administrator and reused by every job that needs it. You never put credentials in the job.

ⓘ WHAT THIS SOLVES

Getting a database step into an automated workflow usually means credentials embedded in a script and a bespoke runner: insecure, and a job no one else can safely maintain.

Choose the right task type

Task type	Use it to
MS SQL Script	Run a T-SQL script against SQL Server
MS SQL Job	Start and monitor a SQL Server Agent job
MS SQL DTEXec (SSIS)	Run an SSIS package (Windows agents only)
MySQL Script	Run a script against MySQL
Oracle Script	Run a SQL/PL-SQL script against Oracle
Other DB Script (ODBC)	Run SQL against any ODBC-compatible database

Add a SQL task to a workflow

To add a SQL task, complete the following steps:

1. Go to **Workflows** and open the workflow you want to edit.
2. Add a job: on an empty workflow select **First Job**, otherwise select **New Job** from the **Job Tools** panel.
3. In the job editor, select the **Task Definition** tab.
4. Select **Show Task Types**.
5. In the task type catalog, under **Database**, select the task type you need (for example, **MS SQL Script**).
6. Attach the **connection** for the target database (ask your administrator if it isn't listed).
7. Under **Task Parameters**, enter your script or settings (see below).
8. Select **Save & Close**. The job is added to the workflow.

Settings

For the script task types (MS SQL, MySQL, Oracle, Other DB):

Setting	Notes
SQL Script / Script File Path	Provide the SQL inline, or point to a <code>.sql</code> file on the agent. Use one or the other.
Script Variables / Bind Variables	Substitute values into the script. MS SQL/MySQL/ODBC use <code>\$(Name)</code> placeholders; Oracle uses bind variables. This is a plain-text substitution, not a parameterized query. Use it for trusted values like an environment name, never for untrusted data.
Output File Path	Optionally write the results to a file on the agent. Not available for the Other DB (ODBC) task type.
Query Timeout	Maximum seconds for the operation. Default is 300 : raise it for long-running jobs.
Fail on Error	On by default. Turn off to let the workflow continue even if the SQL errors.

Task-specific settings:

- **MS SQL Job:** give the **Job Name** and a **Monitor Mode** (wait until it completes, watch during execution, or monitor until an end time), with a polling interval.
- **MS SQL DTEXec (SSIS):** choose the **Package Source** (file, SQL Server, or SSIS Catalog) and point to the package.



GOOD TO KNOW

- Results shown in the task output are capped at **1000 rows**. Use **Output File Path** or the database's own export for bulk data.
- **SSIS** runs only on **Windows** agents.
- Connections, drivers, and agent prerequisites are set up by your administrator. See [Set up database connections](#).

Related topics

- [SQL Database Executor connector](#) — full configuration reference and troubleshooting
- [Set up database connections](#) (Administrator)
- [Respond to a failed connector task](#) (Operator)

Last updated: 2026-07-15

Tags: [Reference](#) [Builder](#) [Connectors](#) [Database](#) [LMS-Training](#)

*Last updated on **Aug 3, 2026***

Use thresholds and resources

Thresholds and **resources** let jobs coordinate beyond simple "run after" ordering: gate a job on a value, or limit how many jobs hit a shared system at once.

ⓘ WHAT THIS SOLVES

Simple "run after" ordering can't express "wait until enough is done" or "don't let too many jobs hit this system at once", so jobs either start too early or overwhelm a shared database or host.

Thresholds: gate a job on a value

A **threshold** is a named number. A job can depend on it and wait until it reaches a required value: useful for "wait until N things are done" style gating.

To use a threshold, complete the following steps:

1. Go to **Toolkit → Thresholds** and create a threshold with a name.
2. In a job's **Dependencies** tab, add a **threshold** dependency and set the value it requires.

Resources: limit concurrency

A **resource** is a named pool with a total capacity. A job reserves units while it runs and releases them when it finishes, so you can cap how many jobs run against a shared system (a core database, say) at the same time.

To use a resource, complete the following steps:

1. Go to **Toolkit → Resources** and create a resource with a **total** capacity.
2. In a job's **Dependencies** tab, add a **resource** dependency and set the units it needs.



GOOD TO KNOW

- A job waiting on a resource shows a *waiting on resource* status until units free up; a job waiting on a threshold shows *waiting on threshold*.
- Thresholds and resources are **workspace-scoped**: a job can only use ones in its workspace.
- Size a resource's **total** to the real limit of the system you're protecting.

Related topics

- [Thresholds and resources](#) — full configuration reference and troubleshooting
- [Order jobs with dependencies](#) (Builder)
- [Define a frequency](#) (Builder)

Last updated: 2026-06-30

Tags:

[Procedure](#)

[Builder](#)

[Toolkit](#)

[LMS-Training](#)

*Last updated on **Aug 3, 2026***

Version and deploy a workflow

Editing a workflow saves a new **version**; running it in an environment is a separate step called **deployment**. Keeping these separate lets you test a change before it reaches production.

⚠ WHAT THIS SOLVES

If editing a workflow immediately changes what runs in production, there's no safe way to test a change first, and no clean way back when one goes wrong.

Versions

Every saved change creates a new version, with an optional change description and full history. You can review earlier versions and **roll back** to one. The older versions are never lost.

Deploy to an environment

Deploy a workflow to an environment and choose how it tracks versions:

- **Latest:** the environment always runs the newest version.
- **Pinned:** the environment runs a fixed version until you change it.

You can also set an **effective date** and **expiration date** so a deployment is active only within a window. The same workflow can be deployed to several environments at once: for example, **pinned** to a tested version in production while running **latest** in a test environment.



GOOD TO KNOW

- **A change isn't appearing in an environment?** The deployment is probably **pinned** to an older version, the new version hasn't been deployed, or the **effective date** is in the future.
- **Rolling back** means deploying or pinning an earlier version: no work is lost.
- Deliberate **production-vs-test drift** is normal: pin production, run latest in test.

Related topics

- [Workflow versions and deployments](#) — full configuration reference and troubleshooting
- [Build a workflow](#) (Builder)
- [Manage environments](#) (Administrator)

Last updated: 2026-07-15

Tags:

[Procedure](#)

[Builder](#)

[Workflows](#)

*Last updated on **Aug 3, 2026***

Wait for a file before your workflow continues

Sometimes a workflow shouldn't start until the data it needs has actually landed: a nightly export from your core banking system, a file dropped by a partner, a report another team produces. The **Wait for File** task holds your workflow until that file is there and ready, so the work that follows always runs against complete data.

It's built in, so there's nothing to install. Add it to any workflow and point it at a file.

! WHAT THIS SOLVES

Without it, a workflow can start the moment it's scheduled: before the file it needs has arrived, or while that file is still being written. The job then runs against missing or half-written data, and the failure often isn't obvious until the results are wrong downstream.

Use it when

- A downstream job must not run until an upstream file arrives.
- A file is delivered on an unpredictable schedule and you want the workflow to react the moment it's ready.
- A file is written gradually and you need to be sure it's **finished** before processing it.

How it works

The task watches a file path and checks it on a set interval. As soon as the file exists and meets the conditions you set (size, freshness, stability), the task completes and your workflow moves on. If the file never shows up in time, you decide whether that's a failure or simply nothing to do.

Add a Wait for File task to a workflow

A Wait for File task runs as a job in your workflow. To add one, complete the following steps:

1. Go to **Workflows** and open the workflow you want to edit.
2. Add a job: on an empty workflow, select **First Job**; otherwise select **New Job** from the sidebar.
3. In the job editor, select the **Task Definition** tab.
4. Select **Show Task Types**.
5. In the task type catalog, under **Utility**, select **Wait for File**.
6. Under **Task Parameters**, in the **File Path** field, enter the full path to the file:
for example, `/tmp/incoming/report.csv`.
7. Set any optional conditions you need (see [Settings](#)).
8. Select **Save & Close**. The job is added to the workflow and runs the Wait for File task.

Settings

Setting	Required	What it does
File path	Yes	The file to watch for. It must be in a folder your administrator allows the agent to access.
Check interval	No	How often to look for the file. Default is every 10 seconds.
Minimum file size	No	Wait until the file reaches at least this size. Use it to avoid picking up an empty or partial file.
Maximum file age	No	Require a <i>recent</i> file: one modified within this many seconds. Use it when only a freshly delivered file should count.
Stable for	No	Wait until the file stops changing size for this long. Ideal for files still being written.
Delete after detection	No	Remove the file once it's found and meets every condition. Off by default.
Fail on timeout	No	If the file doesn't arrive in time: leave this on to fail the workflow, or turn it off to let the workflow continue. On by default.

When the file doesn't arrive in time

The task waits up to the timeout set on the job. If that time passes and the file still isn't ready:

- **Fail on timeout on** (default): the task fails, so the workflow can alert or branch.
- **Fail on timeout off**: the task succeeds and the workflow simply continues.

Tips for reliable file waits

- **Avoid partial files.** If a file is written gradually, set **Stable for** (and/or a **Minimum file size**) so the task only completes once the file is whole. Without it, a fast workflow can grab a file mid-write.
- **Only accept fresh files.** Use **Maximum file age** when an old file left in the folder should *not* count, but be aware that a file older than this limit will never satisfy the task, so leave it off if any existing file is fine.
- **One-and-done deliveries.** Turn on **Delete after detection** when each file should be processed once and cleared, but leave it off if a later job in the workflow still needs the file.



GOOD TO KNOW

- Wait for File runs on **Universal Agents**, on Windows, Linux, and macOS.
- The folders an agent can watch are controlled by your administrator. If a path is rejected, ask your administrator whether that location is permitted. See [Prepare agents for connectors: File access](#).

Related topics

- [Wait for File task](#) – full configuration reference and troubleshooting
- [Respond to a stuck or failed Wait for File task](#) (Operator)

Last updated: 2026-07-15

Tags:

[Reference](#)

[Builder](#)

[Task types](#)

[LMS-Training](#)

*Last updated on **Aug 3, 2026***

Embed another workflow with a Workflow Container

A **Workflow Container** runs another workflow as a single step inside the current one. Build a reusable workflow once (a standard end-of-day sequence, say) and call it from many parent workflows instead of rebuilding it each time. It runs inside the platform, so it needs no agent.

! WHAT THIS SOLVES

The same sequence (a standard end-of-day close, a shared validation routine) gets rebuilt inside every workflow that needs it, so each copy drifts and a single fix has to be made in many places.

Use it when

- You have a sequence used by several workflows and want to maintain it in one place.
- You want to break a large workflow into smaller, reusable pieces.

Add a Workflow Container to a workflow

To add a Workflow Container, complete the following steps:

1. Go to **Workflows** and open the workflow you want to edit.
2. Add a job: on an empty workflow select **First Job**, otherwise select **New Job** from the **Job Tools** panel.
3. In the job editor, select the **Task Definition** tab.
4. Select **Show Task Types**.
5. In the task type catalog, under **Internal**, select **Workflow Container**.
6. Under **Task Parameters**, in **Workflow**, choose the workflow to embed (by name).
7. Select **Save & Close**.

Settings

Setting	Notes
Workflow	The workflow to embed, by name. Required. It must already be deployed.



GOOD TO KNOW

- The container stays **running** until the embedded workflow finishes, then settles to match.
- A workflow can't embed itself or any workflow that already contains it (no loops), and nesting has a depth limit (10 levels by default).
- When troubleshooting, remember a single job may be an entire embedded workflow. Follow it to the nested run.

Related topics

- [Workflow Container task](#) — full configuration reference and troubleshooting
- [Use a Null Job](#) (Builder)
- [Respond to a failed connector task](#) (Operator)

Last updated: 2026-07-15

Tags: [Reference](#) [Builder](#) [Workflow design](#) [LMS-Training](#)

Last updated on Aug 3, 2026

Get familiar as an Operator

Reading about Continuum only gets you so far. The fastest way to learn it is to do the core Operator tasks in a test environment. Work through this checklist from top to bottom, checking off each item as you complete it. Each item links to the page that explains it in detail.

Goal: watch the daily run and respond correctly when something needs attention.

Monitor the run

- ☐ Go to Processes and select the environment you're monitoring.
- ☐ Switch between **Workflow View** (the diagram) and **Job View** (the table).
- ☐ Open a job's detail to see its status, timing, dependencies, and output.
- ☐ See [Monitor the daily run](#).

Read statuses

- ☐ Learn the status families (Held, Waiting, Running, Finishing, Finished, and Failed) and what a waiting job is waiting on.
- ☐ Recognize that a brief Finishing state is transitional, not stuck.
- ☐ See [Job statuses and actions](#).

Respond

- ☐ On a failed job, review the cause, then choose **Restart**, **Mark Fixed**, **Under Review**, or **Cancel**.
- ☐ On a waiting or held job, try **Force Start**, **Skip**, **Hold**, or **Release**.
- ☐ See [Respond to a failed job](#).

Trace and triage

- ☐ In the event log, use **View correlated events** to trace an incident by its correlation ID.
- ☐ Triage a failed connector task from its output and route it to the Builder or Administrator.
- ☐ See [Use the event log](#) and [Respond to a failed connector task](#).

You're ready when

You can read the run at a glance, interpret any status, take the right action on a failed or waiting job, and trace an incident through the event log.

Related topics

- [Getting started with OpCon Continuum](#)
- [Glossary](#)

Tags: [Procedure](#) [Operator](#) [Orientation](#)

*Last updated on **Jul 29, 2026***

Respond to a failed connector task

When a connector task fails or stalls, the **task output** almost always tells you what happened. Read it first, then use the tables below to act or route it to the right team.

Routing rule of thumb: problems with the task's *settings* go to the **Builder**; problems with *connections, agents, drivers, or the relay* go to the **Administrator**.

ⓘ WHAT THIS SOLVES

A connector task fails or stalls mid-run, and it's not obvious whether the fix belongs to the operator, the builder, or the administrator. It bounces between teams while the run waits.

Commands and scripts (Run Command, Run Script)

Output / symptom	Meaning	What to do
"could not start" / command not found	The program or interpreter isn't on the agent	Builder confirms the command; Administrator confirms it's installed on the agent.
Script won't start (PowerShell/Python)	The interpreter isn't installed on the agent	Administrator installs the interpreter (e.g. <code>pwsh</code> , <code>python3</code>).
Failed with a non-zero exit code	The command/script reported an error	Builder checks the command and whether Fail on Error should apply.
Ended at the timeout, exit code 128	The command ran past the job's timeout and was stopped	Builder raises the job timeout or fixes the command.

SQL Database tasks

Output / symptom	Meaning	What to do
Certificate / TLS error (SQL Server)	The server's certificate isn't trusted	Administrator enables Trust Server Certificate or installs a trusted cert.
"Login failed"	Bad credentials or auth mismatch	Administrator fixes the connection.
Can't connect (ODBC/Oracle)	Driver/client missing on the agent	Administrator installs the driver/client.
Fails at ~5 minutes	Hit the default 300-second query timeout	Builder raises Query Timeout .
Results look cut off	Output is capped at 1000 rows	Expected: Builder uses an output file or DB export for bulk data.
"connection ... not found"	No connection attached to the task	Builder attaches the right connection.

Internal tasks (Null Job, Workflow Container)

Output / symptom	Meaning	What to do
Null Job sits waiting	Waiting on its dependencies (by design), or on hold	Operator checks predecessors; release if intentionally on hold.
Workflow Container stays running a long time	The embedded (nested) workflow hasn't finished	Open the nested workflow instance and troubleshoot it there.
Container failed: "does not exist" / "circular" / "depth exceeded"	The embedded workflow reference is wrong, loops, or nests too deep	Builder fixes the workflow reference.

Legacy LSAM commands

Output / symptom	Meaning	What to do
"No agent available"	The relay is down or the LSAM machine is unreachable	Administrator checks the relay and LSAM connectivity.
Windows job rejected at dispatch	The run-as user isn't the service account	Builder sets the user to <code>Use Service Account</code> .
Marked failed though it ran	An exit-criteria condition matched	Builder confirms the failure conditions match the program's real codes.

File waits

For a Wait for File task that's stuck or timed out, see [Respond to a stuck or failed Wait for File task](#). The task output names the exact criterion it's waiting on.

When you escalate

Include the **task output**, the **task type**, and (for database or LSAM tasks) the **connection** or **machine name** involved, but never credentials. Note whether the same operation works when run directly (in a terminal, a database client, or on the LSAM machine).

Related topics

- [Respond to a stuck or failed Wait for File task](#) (Operator)

Last updated: 2026-07-15

Tags: [Troubleshooting](#) [Operator](#) [Connectors](#) [LMS-Training](#)

*Last updated on **Aug 3, 2026***

Operator documentation

Watch the daily run and step in when a job needs attention — read any status at a glance and take the right recovery action.



Get familiar as an Operator

A checklist of hands-on tasks so you can learn OpCon Continuum by monitoring a run and responding to jobs in a test environment.



Respond to a failed connector task

How to diagnose and respond to a failed connector task so you can get a stalled workflow moving again.



Job statuses and actions

A reference to job statuses and the operator actions available for each, so you know how to respond during the daily run.



Monitor the daily run

How to monitor the daily run so you can track progress and act on jobs that need attention.



Monitor critical paths and SLAs with Vision

How to group jobs into Vision cards with SLAs so you can see at a glance whether critical work is on track to finish on time.



Respond to a failed job

How to respond to a failed job by reviewing the cause and choosing the right recovery action.



Search across OpCon Continuum

How to search across Continuum so you can quickly find jobs, workflows, and other objects.



Use the event log

How to use the event log to trace what happened during a run using correlation IDs and event statuses.



Respond to a stuck or failed Wait for File task

How to respond to a stuck or failed Wait for File task so a waiting workflow can continue or be escalated.



Work the Processes page

How to use the Processes page to watch the daily run and act on jobs -- filter, search, and take single or bulk actions.

New here? See [Getting started with OpCon Continuum](#).

These pages walk you through the tasks. When you need exhaustive settings, status values, or a troubleshooting table, see [Reference](#).

*Last updated on **Aug 3, 2026***

Job statuses and actions

A quick reference for what a job's status means and what you can do about it.

ⓘ WHAT THIS SOLVES

A job's status is only useful if you know what it means and what you're allowed to do about it. Guess wrong, and you restart something that shouldn't be, or leave a run stalled.

What a status is telling you

Status group	Examples	What it means
Held	On Hold	Won't run until released.
Waiting	Waiting on a dependency, start time, resource, machine; Late to Start; Marked to be skipped	Eligible but waiting on something. Usually fine. "Marked to be skipped" means you skipped it and it will drop out when its turn comes.
Running	Attempting to start, Running, Late to Finish	Actively starting or running.
Finishing	(the "pending" states)	Done; completion is being processed. Transitional, not stuck.
Finished	Finished OK, Fixed	Completed successfully.
Failed	Failed, Initialization Error, Missed Start Time	Needs attention.
Other terminal	Cancelled, Skipped, Under Review	Ended without running to success.

What you can do, by situation

The job is...	You can	To...
On Hold	Release · Force Start · Skip · Cancel	Let it run, push it through, skip it, or stop it.
Waiting on a dependency/time/resource/machine	Force Start · Skip · Hold · Cancel	Run it now, skip it, hold it, or stop it.
Running	Kill	Stop a job that's actively running.
Failed	Restart · Mark Fixed · Under Review · Skip · Cancel	Re-run, accept as resolved, flag for review, skip it, or abandon.
Missed Start Time	Force Start · Skip · Cancel	Run it now, skip it, or abandon.
Under Review	Mark Fixed · Restart · Cancel	Resolve, re-run, or abandon.

Actions are status-specific: the run only offers an action a job's status allows. **Cancel** stops a job *before* it runs; **Kill** stops one that's *already running*. **Restart** is available once a job has stopped (Failed, Cancelled, Skipped, ...); **Release** only from On Hold.

Mark Finished OK and **Mark Failed** are manual overrides available from most live states: use them to force a job's outcome (successful or failed) so the jobs waiting on it move on. (A **container job**, one whose body is another workflow, can only be **Killed** while it's running.)

What the actions do

Action	Effect
Hold	Pause the job so it won't run until released.
Release	Take a job off hold.
Force Start	Start a waiting job now, regardless of what it's waiting on.
Skip	Mark the job so it doesn't run. In a live workflow the skip is applied in sequence (when the job would have been its turn to run), and only then do its dependents stop waiting on it. Skipping a job that has already failed or missed its start time takes effect promptly.
Restart	Run a stopped job again (failed, cancelled, skipped, or already finished).
Mark Fixed	Treat a failed job as resolved (work done outside the system).
Under Review	Flag a failed job for follow-up.
Cancel	Stop a job that <i>hasn't started running yet</i> ; dependents stop waiting on it.
Kill	Stop a job that's <i>actively running</i> . (Cancel doesn't apply once a job is running.)
Mark Finished OK	Force a job to a successful outcome so its dependents proceed.

Action	Effect
Mark Failed	Force a job to a failed outcome so its dependents react accordingly.

Related topics

- [Job and workflow statuses](#) – full configuration reference and troubleshooting
- [Respond to a failed job](#) (Operator)
- [Monitor the daily run](#) (Operator)

Last updated: 2026-07-31

Tags: [Reference](#) [Operator](#) [Processes](#) [LMS-Training](#)

*Last updated on **Aug 3, 2026***

Monitor the daily run

The **Processes** view is where you watch the day's automation run: which workflows are in progress, and which jobs are waiting, running, finished, or failed. Use it to step in when something needs attention.

! WHAT THIS SOLVES

Without one place to watch the day's automation, a stalled or failed job goes unnoticed until a business process is already late.

Two ways to look at the run

- **Workflow View:** a diagram of each workflow instance and its jobs, colored by status. Best for seeing the shape of a run and where it's blocked. Filter by **status** using simple groups (**Building, Waiting, In Process, Held, Completed, Cancelled**) to narrow to just the workflows you care about.
- **Job View:** a filterable, sortable table of job instances. Best for finding specific jobs by name, status, or time.

Watch the run

To monitor the run, complete the following steps:

1. Go to **Processes** and select the environment you're monitoring.
2. Use **Workflow View** to see workflows and their jobs, or switch to **Job View** to search and filter jobs.
3. Select a job to open its detail: status, timing, dependencies, output, and logs.
4. Use **Refresh** to pull the latest state.

From the workflow diagram you can also jump to the matching jobs in **Job View** to act on them.

Reading status at a glance

Jobs move through **waiting** → **running** → **finished** states. A **waiting** job is usually fine. It's waiting on a dependency, its start time, a resource, or an agent. A job briefly in a *completion-processing* state is finishing, not stuck. Focus your attention on **failed** jobs and jobs running late. See [Job statuses and actions](#).

Start a workflow manually

Use the **Schedule Workflow** action to run a workflow outside its normal schedule. Choose the workflow and the schedule date.

Related topics

- [Respond to a failed job](#) (Operator)
- [Job statuses and actions](#) (Operator)

Last updated: 2026-07-15

Tags: [Procedure](#) [Operator](#) [Processes](#) [LMS-Training](#)

Last updated on Aug 3, 2026

Respond to a failed job

When a job fails, your goal is to find out why, fix or account for it, and get the run moving again. The job's own output is the fastest path to the cause.

! WHAT THIS SOLVES

A failed job stops the run. Without a clear path to the cause and the right action, it sits unresolved, or gets restarted before the real problem is fixed, so it just fails again.

Steps

To respond to a failed job, complete the following steps:

1. In **Processes**, open the failed job to see its detail.
2. Read the job **output and logs** to find the cause. In the **Completion** section (and on the **Output** tab), check the **Exit Code** and **Termination** description. The exit code shows as a badge, **green for 0** and **red for any non-zero** value.
3. Resolve the cause. Many failures are configuration or connection issues, so escalate to the **Builder** (job settings) or **Administrator** (connections, agents) as needed.
4. Choose how to clear the job (see below).

The **Exit Code** is the code the command, script, or connector returned; it's a diagnostic detail, not the job's status. A failed job that reports **no** exit code shows a dash (—) rather than a 0, so "failed with code 0" reads differently from "failed, code unknown."

Your options on a failed job

Action	Use it when
Restart	The cause is fixed and you want the job to run again.
Mark Fixed	The work was completed or resolved outside the system, and the job should count as done.
Under Review	You need to flag the job for follow-up without resolving it yet.
Cancel	The job should not run, and downstream jobs should stop waiting on it.

Restart and Mark Fixed are available from a **Failed** job; Mark Fixed is also available from a job **Under Review**. If an option isn't offered, the job isn't in a status that accepts it.

Before you restart

Confirm the underlying problem is actually resolved. Otherwise the job will just fail again. If the failure is in the job's configuration, the **Builder** needs to change it before a restart will succeed.

When to escalate

Escalate (with the job output, the job's status, and the workflow instance) when the failure isn't explained by the job's settings or a connection/agent issue, or when a job's status doesn't match what actually happened.

Related topics

- [Operator response guide](#) — full configuration reference and troubleshooting
- [Job statuses and actions](#) (Operator)
- [Respond to a failed connector task](#) (Operator)

Last updated: 2026-07-28

Tags:

[Procedure](#)

[Operator](#)

[Processes](#)

[LMS-Training](#)

*Last updated on **Aug 3, 2026***

Search across OpCon Continuum

Global search finds objects across the whole platform from one place: workflows, jobs, calendars, frequencies, thresholds, resources, tags, properties, scripts, and service requests, and lets you act on or export the results.

! WHAT THIS SOLVES

Finding a specific workflow, job, or object across a large, busy platform by paging through screens is slow, especially mid-incident, when you need it now.

Search

To search, complete the following steps:

1. Go to **Search**.
2. Choose which **types** to include and enter your query.
3. Narrow with the available **filters**, and page through results.
4. Use a row's actions to **go to** the object, **open it in a new tab**, **copy** it (workflows and scripts), or **delete** it (most types).
5. **Export** the results to CSV or Excel if you need them outside the app.

Save and schedule a search

Save a search (its types, query, filters, and sort) to re-run later. You can also **schedule** a saved search to be **emailed** to you daily, weekly, or monthly.

💡 GOOD TO KNOW

- **Copy** is available for workflows and scripts; **delete** is available for most types but not jobs.
- Very large multi-type result sets are capped with a warning. Narrow your filters to get a complete export.

Related topics

- [Global search](#) — full configuration reference and troubleshooting
- [Build and run reports](#) (Administrator)
- [Use the event log](#) (Operator)

Last updated: 2026-07-15

Tags:

[Procedure](#)

[Operator](#)

[Search](#)

*Last updated on **Aug 3, 2026***

Use the event log

The **event log** records everything the platform processed: every action, where it came from, and whether it worked. When you need to understand *what happened*, especially across a chain of related actions, this is where you look.

ⓘ WHAT THIS SOLVES

When something went wrong, piecing together what actually happened (which action, from where, and whether it worked) from scattered places is slow and error-prone.

Find what you need

To investigate with the event log, complete the following steps:

1. Go to **Event Log**.
2. Filter by **status** (Received, Processing, Completed, Failed, Expired), **event type**, **environment**, **source**, or a **date range**.
3. Open an event to see its detail: what it changed, anything it triggered, and (if it failed) the error and retry attempts.
4. Use **View correlated events** to see everything that shares the event's **correlation ID**: the full chain from the original action.

Keep it live

Turn on **live refresh** (every 5, 10, 30, or 60 seconds) to watch events as they happen, or pause and refresh manually.



GOOD TO KNOW

- To trace an incident, start at the failed event and open its **correlated** view. You'll see the triggering action and everything it set off.
- **Expired** is different from **Failed**: expired means the event wasn't processed in time.
- A "completed" event means it was processed. The work it kicked off has its own status in Processes.

Related topics

- [Event log](#) — full configuration reference and troubleshooting
- [Monitor the daily run](#) (Operator)
- [Respond to a failed job](#) (Operator)

Last updated: 2026-07-15

Tags:

[Procedure](#)

[Operator](#)

[Event Log](#)

[LMS-Training](#)

*Last updated on **Aug 3, 2026***

Respond to a stuck or failed Wait for File task

A **Wait for File** task is waiting for a file to arrive and meet its conditions. If one is stuck waiting or has failed, you can usually tell exactly why from the task's own output, and whether the fix is yours, the Builder's, or the Administrator's.

ⓘ WHAT THIS SOLVES

A Wait for File task sits waiting or fails, and it's unclear whether the file is late, in the wrong place, or being rejected by a condition. The whole workflow is stuck with no obvious cause.

Start with the task output

The task writes a line every time it checks the file. The most recent line tells you what it's waiting on right now. Open the task output and read the last line before doing anything else.

What the output is telling you

Last output line	What it means	What to do
File not found, checking again in ...s	The file hasn't arrived yet	Confirm the upstream source actually produced the file (partner delivery, export job, etc.). If it should be there, the path may be wrong: escalate to the Builder .
Path exists but is not a file	A folder is sitting where the file should be	Escalate to the Builder : the file path points at a directory.
File size ... < minimum ...	The file is present but smaller than required, often still being written	Wait for the source to finish writing. If it's already complete and just small, escalate to the Builder to adjust the minimum size.
File age ...s > maximum ...s	The file exists but is older than allowed, so it will never qualify	The task is waiting for a <i>fresh</i> file. Confirm whether a new file is due. If the existing file should count, escalate to the Builder .
File size changed ... / File stable for ...s, need ...s	The file is still being written and hasn't settled	Wait for the source process to finish. If it never settles, escalate to the Builder .
... is outside allowed directories	The agent isn't permitted to read that location (task fails immediately)	Escalate to the Administrator : the folder must be allowed on the agent.

Last output line	What it means	What to do
Timeout after ... s	The file didn't arrive (and meet its conditions) in time	See When a task times out .

When a task times out

A timeout means the file never became ready within the task's time limit. Two outcomes are possible, by design:

- **The task failed:** the Builder set it to fail when the file is missing. Treat it like any failed task: confirm whether the file was supposed to arrive, then rerun once it has, or escalate.
- **The task succeeded anyway:** the Builder set the file as optional, so the workflow continued on purpose. No action needed.

Before rerunning, **confirm the file is actually present and complete**. Otherwise the task will simply wait and time out again.

Who to escalate to

- **Builder:** anything about the task's configuration: the file path, the size/age/stability conditions, or whether a timeout should fail the workflow.
- **Administrator:** "outside allowed directories" errors, or any case where the agent can't reach a folder it should.

When you escalate, include the **task output** and the **file path** the task was watching.

Related topics

- [Wait for a file before your workflow continues](#) (Builder)

Last updated: 2026-07-15

Tags: [Troubleshooting](#) [Operator](#) [Task types](#) [LMS-Training](#)

*Last updated on **Aug 3, 2026***

Work the Processes page

Processes is your daily-run screen: the live view of the workflows and jobs running in the selected **environment**. This page is where you watch the run and act on it.

! WHAT THIS SOLVES

During the run you need to see what's happening and act fast (push a waiting job through, stop one that's misbehaving, or clear a batch of stuck jobs at once) without hunting through separate screens or selecting one row at a time.

Pick your view

A toggle switches between two views (your choice sticks if you reload):

- **Job View** (the default): every job in the run, one row each. All columns sort.
- **Workflow View**: the workflows themselves. Filter it by status **group** (Building, Waiting, In Process, Held, Completed, Cancelled) rather than individual statuses.

From here you can also **Schedule Workflow** to add a workflow to the run, and **refresh** the view.

Find what you're looking for

- Use the **search** command menu to jump to a job.
- Use the **filters** (look for the funnel icon on a section header) to narrow the list; the frequency filter uses on/off switches. Hover any control for a tooltip explaining it.

Changing the page, sort, search, or a filter **clears your current selection**, so a bulk action never acts on rows you can no longer see.

Act on one job

Open a job's action menu to act on it. You'll only ever see the actions its **status** allows, so you can't pick one that would fail. For what each action does, including when to use **Kill** vs **Cancel**, see [Job statuses and actions](#).

Select a job to open its **detail panel** (read-only), with quick links to its **History** and its **Workflow**.

Act on many jobs at once (bulk actions)

1. Tick the check boxes for the rows you want (or select-all).
2. Once **two or more** are selected, a **Bulk actions** button appears.
3. Open it. The menu shows **only the actions valid for every selected row**, plus **Delete**.
4. Choose an action; it runs across the selection and a message confirms the result.

This works in both Job View and Workflow View.

Add a job to a running workflow

Sometimes a workflow is already running (or has finished for the day) and you need one more of its jobs to run: a step that was skipped, or one you want to run again. From **Workflow View**, open a workflow instance's action menu and choose **Add Job**.

ⓘ WHAT THIS SOLVES

A job that belongs to the workflow didn't run this time (maybe its frequency didn't qualify for the date, or it was left out), and you need it to run now, in the same instance, without rebuilding or rescheduling the whole workflow.

In the **Add Job to Workflow Instance** dialog:

1. **Pick the job.** You'll see every job defined on the workflow. Each is tagged so you know what will happen:

- *(no tag)*: the job isn't in this run yet and will be added.
 - **Will Replace**: the job already ran and finished; adding it again replaces that finished copy (you'll be asked to confirm first).
 - **Job is Active** (greyed out): the job is already running or waiting, so you can't add another.
 - **No Frequency** (greyed out): the job has no frequency defined, so it can't be added.
2. **Choose a frequency** if the job has more than one (if it has just one, it's picked for you).
 3. **Adjust properties** if you need to. The grid is pre-filled with the job's defaults; leave it alone to use them as-is.
 4. **Add on hold** if you want the job held instead of released to run right away.
 5. **Add a reason** (optional). It's saved to the job's history for the audit trail.

Choose **Add Job**. A released job starts as soon as it's eligible; a held job waits until you release it. If you add a job to a workflow that already **completed**, the workflow reopens so the new job can run. The added job shows up in the run and on the workflow's timeline, marked as added.

Some jobs can't be added this way yet: jobs that run as multiple instances, workflow-container (sub-schedule) jobs, and jobs set to run on every agent in a pool. You also can't add a job to a **cancelled** workflow instance.

Delete a job or workflow instance

Delete removes a job or a whole workflow instance from the run (a soft delete). Deleting a job also frees anything waiting on it, so downstream jobs stop hanging on a step that's no longer there.

You can't delete a workflow instance while it's **actively running**. Hold or let it finish first. Deletion is allowed when it's waiting, held, completed, or cancelled.

Related topics

- [The Processes page \(runtime\)](#) — full configuration reference and troubleshooting
- [Monitor the daily run](#) (Operator)

- [Job statuses and actions](#) (Operator)
- [Respond to a failed job](#) (Operator)

Last updated: 2026-07-29

Tags: [Procedure](#) [Operator](#) [Processes](#) [LMS-Training](#)

*Last updated on **Aug 3, 2026***

Get familiar as an Administrator

Reading about Continuum only gets you so far. The fastest way to learn it is to do the core Administrator tasks in a test environment. Work through this checklist from top to bottom, checking off each item as you complete it. Each item links to the page that explains it in detail.

Goal: stand up and maintain the environment, agents, connections, and access.

Agents and pools

- ☐ Open an agent pool, then an individual agent.
- ☐ Read an agent's status (Online, Offline, Busy, Draining, or Disabled), its last heartbeat, and its detected runtimes.
- ☐ Adjust an agent's maximum concurrent jobs.
- ☐ See [Manage agents and agent pools](#).

Plugins

- ☐ Open Plugins and view a plugin's detail.
- ☐ Enable an external plugin. Note that built-in and legacy LSAM plugins are enabled out of the box, but external plugins are not.
- ☐ Find where to set a plugin's default version.
- ☐ See [Manage plugins](#).

Connections

- ☐ Create a database connection and use **Test Connection** to confirm it works.
- ☐ Edit the connection and confirm credential fields appear blank. That's intentional.
- ☐ See [Manage connections](#) and [Set up database connections](#).

Agent readiness

- ☐ Review the file paths an agent is allowed to use, and where script interpreters and database drivers need to be installed.
- ☐ See [Prepare agents for connectors](#).

Environments, workspaces, and custom fields

- ☐ View or create an environment and check its service health.
- ☐ Create a workspace and note what's scoped to it.
- ☐ Define a custom field, and try a **select** field with a few options.
- ☐ See [Manage environments](#), [Manage workspaces](#), and [Define custom fields](#).

Roles and permissions

- ☐ Open Roles and browse the permission catalog.
- ☐ Create a role: choose its permissions and set its scopes.
- ☐ Run permission simulation for the role and read the results.
- ☐ Know the caveat: role-based access is being rolled out, so these permissions currently describe intended access rather than being enforced.
- ☐ See [Manage roles and permissions](#).

Reporting

- ☐ Run a job-history report for a date range and environment, then export it.
- ☐ See [Build and run reports](#).

You're ready when

You can inspect an agent, enable a plugin, create and test a connection, set up a workspace, and run a report.

Related topics

- [Getting started with OpCon Continuum](#)
- [Glossary](#)

Tags: [Procedure](#) [Administrator](#) [Orientation](#)

*Last updated on **Jul 29, 2026***

Control which folders agents can access

Note: This is a placeholder page included so cross-references from other personas resolve in the test site. The full Administrator content has not been authored yet.

Administrators control which directories a Universal Agent is allowed to read and write. Tasks such as [Wait for File](#) can only watch paths that fall within these allowed directories; a path outside them is rejected and the task fails immediately.

*Last updated on **Jun 29, 2026***

Build and run reports

Reports answer questions about your automation (what's configured, what ran, and who changed what) and let you export the results. Operators and administrators both use them; auditors rely on the governance reports.

! WHAT THIS SOLVES

Questions about what's configured, what ran, and who changed what get answered by hand (screenshots, ad-hoc queries, spreadsheets), which is slow and leaves auditors without a repeatable record.

What you can report on

Category	Examples
Configuration & reference	Agents, jobs, scripts, calendars, frequencies, tags, properties, resources, thresholds
Operational	Job history, event details by date
Governance & access	Audit history

Build a report

To build and run a report, complete the following steps:

1. Go to **Reports** and choose what to report on.
2. Select the **columns** you want and arrange their order.
3. Add **filters** to narrow the results (text, number, single-select, or date/time; for example, a date range and an environment for job history).
4. Set the **sort** order.
5. Run the report to see results, then **export** to CSV or Excel if needed.

Save it for reuse

Save a report (its columns, filters, and sort) to re-run later. This is handy for a recurring month-end or audit pull.



GOOD TO KNOW

- Results are paginated and large sets show a truncation note; exports stream the full set.
- Reports are **read-only**: running one never changes anything.
- For a result set delivered on a schedule (by email), use a saved **search** instead. See [Search across OpCon Continuum](#).

Related topics

- [Reports](#) — full configuration reference and troubleshooting
- [Use the event log](#) (Operator)
- [Search across OpCon Continuum](#) (Operator)

Last updated: 2026-07-15

Tags:

[Procedure](#)

[Administrator](#)

[Reports](#)

[LMS-Training](#)

*Last updated on **Aug 3, 2026***

Set up database connections

Connectors that work with databases (the SQL Database Executor task types) don't carry their own credentials. Instead, you create a **connection** once: the server, database, and login for a system, stored centrally and reused by any job that needs it. Credentials are kept secure and delivered to the agent only at run time; they never appear in a job definition or in task output.

Set up a connection for each database a workflow needs to reach: for example, your core banking database, a reporting warehouse, or a partner system.

❗ WHAT THIS SOLVES

Database jobs need a server and login. Put those in each job and credentials sprawl across workflows and show up in definitions and task output, which is a security and maintenance problem.

Connection types

Connection	Required	Optional	Use with
SQL Server	Server, Database	Port (default 1433), Username, Password, Windows Authentication, Trust Server Certificate	MS SQL Script, MS SQL Job, MS SQL DTEXEC (SSIS)
MySQL	Host, Database, Username	Port (default 3306), Password	MySQL Script
Oracle	Connection String	Username, Password	Oracle Script
ODBC	Connection String	Username, Password	Other DB Script (ODBC)

SQL Server connections

Connections to SQL Server are **always encrypted (TLS)**. Two settings deserve attention:

- **Trust Server Certificate:** leave this off when the server presents a certificate from a trusted certificate authority. Turn it **on** only when the server uses a self-signed or internal-CA certificate; otherwise the connection fails on a certificate error. Installing a properly trusted certificate is the more secure long-term option.
- **Windows Authentication:** turn this on to connect with a Windows account instead of a SQL login.

Best practices

- Use a **least-privilege database account** scoped to what the workflows actually need.
- Create a **separate connection per environment** (for example, test vs. production) so jobs can't accidentally run against the wrong database.
- Keep connection credentials current. An expired or rotated password shows up as a login failure on every job that uses the connection.

Agent prerequisites

Some connections require software on the **agent** that runs the task:

- **ODBC**: the ODBC driver/DSN named in the connection string must be installed on the agent.
- **Oracle**: the agent must meet the Oracle client prerequisites.

See [Prepare agents for connectors](#).

Troubleshooting

Symptom	Cause	Fix
Connection fails on a certificate error (SQL Server)	Self-signed / internal-CA certificate, Trust Server Certificate off	Turn on Trust Server Certificate , or install a trusted certificate.
"Login failed"	Wrong credentials, or SQL vs. Windows auth mismatch	Verify the login; set Windows Authentication to match the server.
ODBC connection fails	Driver/DSN not installed on the agent	Install the ODBC driver/DSN on the agent.
Oracle connection fails	Wrong connection string, or missing Oracle client on the agent	Verify the connection string; confirm agent prerequisites.

Related topics

- [Prepare agents for connectors](#)
- [SQL Database connectors](#) (Builder)

Last updated: 2026-07-15

Tags: [Reference](#) [Administrator](#) [Connections](#) [Database](#)

[LMS-Training](#)

*Last updated on **Aug 3, 2026***

Define custom fields

Custom fields let you add your own metadata to capture information that matters to your organization. Custom fields apply to **jobs**.

! WHAT THIS SOLVES

The metadata your organization needs to track on a job (an owner, a ticket number, a cost center) has nowhere to live, so it ends up crammed into job names or simply isn't captured.

Field types

Type	Use for
Text / Text area	Short or long free text
Number	Numeric values
Boolean	Yes/no
Select	A fixed list of choices (you provide the options)
Date	A date

Create a custom field

To define a custom field, complete the following steps:

1. Go to **Custom Fields** and add a field.
2. Enter a **name** and choose an **input type**.
3. For a **select** field, add the **options**.
4. Optionally set it **required**, give it a **default value** and a **description**, and set its **display order**.

5. Save.



GOOD TO KNOW

- A **select** field must have at least one option.
- Set **display order** to control where fields appear.
- Custom fields apply to jobs today.

Related topics

- [Custom fields](#) — full configuration reference and troubleshooting
- [Manage workspaces](#) (Administrator)

Last updated: 2026-07-15

Tags:

[Procedure](#)

[Administrator](#)

[Administration](#)

*Last updated on **Aug 3, 2026***

Administrator documentation

Set up and maintain the environment: agents, connections, environments, workspaces, plugins, and access.



Get familiar as an Administrator

A checklist of hands-on tasks so you can learn OpCon Continuum administration by setting up agents, connections, and access in a test environment.



Control which folders agents can access

Note: This is a placeholder page included so cross-references from other



Build and run reports

How to build, run, and export reports so you can track job activity, agent health, and schedule outcomes.



Set up database connections

How to set up and test database connections so jobs can run SQL against your databases securely.



Define custom fields



Manage agents and agent pools

How to add, configure, and group agents into pools so work runs on the right machines and stays available.



Manage connections

How to create, edit, and secure connections so jobs can reach the external systems they automate.



Manage environments

How to create and manage environments so you can separate and promote work across development, test, and production.



Manage notifications

How to set up notification groups, triggers, actions, and escalation so the right people are told when an agent, workflow, or job needs attention.



Manage plugins

How to enable, disable, and update plugins so you can extend Continuum with built-in and external capabilities.



Manage roles and permissions

How roles, permissions, and scopes control who can see and change what, and how to assign them.



Manage workspaces

How to create and manage workspaces so you can organize work and control access by team or function.



Prepare agents for connectors

How to prepare agents with the interpreters, drivers, and permissions each connector needs before running connector tasks.



Set up and manage relays

How to register, install, and maintain a relay so your on-prem legacy LSAM machines can run jobs dispatched from the cloud.

New here? See [Getting started with OpCon Continuum](#).

These pages walk you through the tasks. When you need exhaustive settings, status values, or a troubleshooting table, see [Reference](#).

*Last updated on **Aug 3, 2026***

Manage agents and agent pools

Agents run your jobs; **agent pools** group agents so work can be routed to whichever one is available. As an administrator you keep agents healthy and organized so workflows always have somewhere to run.

! WHAT THIS SOLVES

If workflows target a single named machine, one agent going down or filling up stalls the work. No one has a clear view of which agents are healthy or overloaded.

Agent pools

Group related agents into a pool, then let workflows target the pool instead of a single machine. Create a pool with a **name** (the name can't be changed later) and a description. The pool view shows how many jobs are queued and running.

Read an agent's state

Each agent shows the essentials you need at a glance:

- **Status:** Online, Offline, Busy, Draining (finishing up, taking no new work), Disabled, or **Unknown**. *Unknown* (shown amber) applies to a legacy agent whose relay can't be reached, so its state can't be confirmed. This is different from *Offline*, which means the agent itself is down.
- **Last heartbeat:** when it last checked in ("Never" means it hasn't yet).
- **Type and OS:** e.g. a Universal Agent on Windows or Linux.
- **Load:** current jobs vs. its maximum concurrent jobs (which you can adjust).
- **Update status:** whether the agent version is up to date or an update is available.
- **Runtimes:** detected tools (Node, Python, .NET, Java, PowerShell, Docker), useful for the Run Script task.

How workflows pick an agent

When a Builder designs a job, they choose how it's placed: a **specific agent** (with failover backups), the **least-busy agent in a pool**, or **all agents in a pool**. A legacy **LSAM** job can instead target a **legacy agent group**: either the least-busy member or every member. A job is only sent to an agent whose type and OS match the work. A job pinned to a **specific agent** (or to a group with no member online) waits rather than failing, so check the target if such a job sits waiting.

Manage legacy agents and groups

Legacy LSAM agents are registered on the **Agents** page, on its **Legacy Agents & Groups** tab (alongside the **Agent Pools** tab): add, edit, or remove a legacy agent there. Changes take effect automatically: the connecting relay picks them up within about a heartbeat, with no restart. Give each legacy agent a name that's unique for its relay.

On the same page you can gather legacy agents into a **legacy agent group**. Every member of a group must be the **same legacy type** (for example, all UNIX), and a group can span more than one relay. A Builder can then point a legacy job at the group instead of a single machine, to run on the least-busy member or on every member at once. Removing a group leaves its member agents and their work untouched.

Troubleshooting

Symptom	Cause	Fix
A job is waiting on a machine	No matching agent is online (wrong type/OS, or all offline/draining/disabled). A job pinned to a specific agent waits for <i>that</i> agent.	Bring a matching agent online; check the job's required agent type, or reassign the job.
An agent shows offline / stale heartbeat	The agent host or its connection is down	Check the agent machine and network.
A legacy agent shows Unknown (amber)	Its relay can't be reached, so the agent's state is uncertain, not necessarily down	Check the relay; the agent leaves <i>Unknown</i> on its own once the relay reconnects.
A legacy job targeting a group waits	No member of the group is online	Bring a group member online; if members show <i>Unknown</i> , check the relay.
Work piles up on one agent	Its concurrency cap is reached	Raise the agent's maximum concurrent jobs, or route through a pool.

Some agent details (CPU, memory, disk, installed plugins) aren't populated yet and show "—".

Related topics

- [Agents and agent pools](#) – full configuration reference and troubleshooting
- [Prepare agents for connectors](#) (Administrator)
- [Manage plugins](#) (Administrator)

Last updated: 2026-07-20

Tags:

[Procedure](#)

[Administrator](#)

[Agents](#)

[LMS-Training](#)

*Last updated on **Aug 3, 2026***

Manage connections

A **connection** stores the endpoint and credentials for an external system once, so jobs can use it without anyone embedding secrets in a workflow. You create and maintain connections; Builders attach them to jobs.

! WHAT THIS SOLVES

Every job that reaches an external system needs its endpoint and credentials. Embed those in each workflow and secrets sprawl, rotation is painful, and a changed password breaks jobs everywhere at once.

How connections work

- Each connection is created from a **connection type**, which defines the fields it needs. Connection types are provided by plugins: today these are the **database** types (SQL Server, MySQL, Oracle, ODBC). More types arrive as plugins are added.
- Credentials are **stored encrypted** and are **never shown again** after you save them.
- Every connection has a **Test Connection** action that checks it works and reports the result.

Create a connection

To create a connection, complete the following steps:

1. Go to **Connections** and add a connection.
2. Enter a **name** and choose the **connection type** (the type can't be changed after creation).
3. Fill in the configuration fields the type requires.
4. Select **Test Connection** to confirm it works.
5. Save.

For the specific fields of each database type (and the SQL Server certificate/auth options), see [Set up database connections](#).

Editing safely

When you edit a connection, credential fields appear **blank**. That's intentional. Leave them blank to keep the current values, or re-enter them only if they've changed.

Troubleshooting

Symptom	Cause	Fix
Test connection fails	Wrong credentials, untrusted certificate, missing driver, or the system is unreachable	See the database connector guidance; fix the connection or the agent prerequisite.
A connection type isn't available	Its plugin isn't enabled	Enable the plugin. See Manage plugins .
A job says its connection type isn't found	No connection of that type is attached to the job	Have the Builder attach the right connection.

Related topics

- [Connections](#) — full configuration reference and troubleshooting
- [Set up database connections](#) (Administrator)
- [Manage plugins](#) (Administrator)

Last updated: 2026-07-08

Tags: [Procedure](#) [Administrator](#) [Connections](#) [LMS-Training](#)

*Last updated on **Aug 3, 2026***

Manage environments

An **environment** is a target (such as test or production) that workflows and service requests are deployed to. Each environment has its own runtime and its own deployed content, so you can move automation from test to production deliberately.

! WHAT THIS SOLVES

Without separate targets for test and production, there's no safe place to try a change. Automation goes straight to production, or test work accidentally runs against live systems.

Create an environment

To create an environment, complete the following steps:

1. Go to **Environments** and add an environment.
2. Enter a **name**, choose its **type** (Dev, Test, Staging, or Production), and add a description.
3. Save.

Creating an environment starts its **provisioning** in the background. A new environment may take a short time before its runtime is fully ready. Check its **service health** and heartbeat.

What an environment shows

- Its **type** and **status** (active, inactive, or degraded).
- Its registered **services** and their **health** (healthy, unhealthy, or unknown).

GOOD TO KNOW

- The Environments screen is reached by direct link (it isn't in the main navigation).
- If a brand-new environment isn't running workflows yet, it may still be **provisioning**. Confirm service health before treating it as a fault.
- A provisioning failure is logged, not surfaced as an error on the create request. If an environment's service health doesn't turn healthy after a reasonable wait, check service health directly rather than expecting a failure notification.
- Workflows and service requests run in an environment only once they're **deployed** there. See [Version and deploy a workflow](#).

Related topics

- [Environments](#) — full configuration reference and troubleshooting
- [Manage workspaces](#) (Administrator)
- [Version and deploy a workflow](#) (Builder)

Last updated: 2026-07-08

Tags: [Procedure](#) [Administrator](#) [Administration](#)

*Last updated on **Aug 3, 2026***

Manage plugins

Plugins provide the task types (connectors) your Builders can use, and some provide connection types. You control which plugins are **enabled** for your organization and which version new jobs use.

! WHAT THIS SOLVES

Builders need the right connectors available on a consistent version. Without central control, task types are missing when a workflow needs them, or jobs quietly run on versions that differ across the org.

The plugin catalog

The Plugins area lists each plugin with its category, version, and task types. Search by name or filter by category to find one, then open it to see its details and the connection types it provides.

Enable or disable a plugin

To enable or disable a plugin, complete the following steps:

1. Go to **Plugins** and find the plugin.
2. Use its **enabled** toggle to turn it on or off for your organization.

What's on by default:

- **Built-in** task types (Run Command, Run Script, Wait for File), **internal** ones (Null Job, Workflow Container), and **legacy LSAM** are **enabled** out of the box.
- **External** plugins (such as the **SQL Database Executor**) are **disabled by default** and must be turned on before Builders can use them.

Versions

A plugin can have more than one version available; the **default version** is what new jobs pick up. Set the default when you want new jobs to move to a newer version.



GOOD TO KNOW

- If a Builder says a task type (often SQL) "isn't there," the usual cause is that its external plugin hasn't been **enabled** yet.
- Disabling a plugin removes its task types from new jobs. Check what's already in use before turning one off.

Related topics

- [Plugins](#) — full configuration reference and troubleshooting
- [Manage connections](#) (Administrator)
- [How connectors work / the connector catalog](#) (Builder)

Last updated: 2026-07-08

Tags:

[Procedure](#)

[Administrator](#)

[Plugins](#)

[LMS-Training](#)

*Last updated on **Aug 3, 2026***

Manage roles and permissions

Evolving. This page is accurate for the current build, but the underlying feature is still in active development. Details can change between releases.

Roles define what people can do, and **scopes** define where they can do it. You assign a set of **permissions** to a role and then scope the role to the workspaces and environments it applies in.

! WHAT THIS SOLVES

Everyone either has too much access or waits on an administrator for everything. Without scoped roles there's no way to prove who can do what, and where, when an audit asks.

Roles

A role has a **name**, a set of **permissions**, and one or more **scopes**. The built-in **Administrator** role has every permission everywhere and can't be changed. You can create roles, duplicate an existing one as a starting point, and edit or remove the roles you create.

Permissions

A permission is an **action** on an **object type**: `view`, `create`, `edit`, `delete`, or `execute`. Object types fall into scope models:

- **Workspace**: things that live in a workspace (workflows, scripts, properties, thresholds, resources, service requests, agents, agent pools, connections).
- **Global**: tenant-wide settings (workspaces, frequencies, calendars, reports, custom fields, tags, roles, audit history, and more).
- **Workspace + environment**: runtime things (workflow and job instances, deployments, the self-service portal).

Granting `create`, `edit`, `delete`, or `execute` automatically includes `view` for that object. You can't manage what you can't see.

Scopes

Scope a role to a **workspace**, an **environment**, both, or leave either open to mean "any":

Scope	Applies
Any workspace, any environment	Everywhere
A workspace, any environment	That workspace
Any workspace, an environment	That environment
A workspace and an environment	Just that combination

Configure a role

To set up a role, complete the following steps:

1. Go to **Roles** and add a role (or duplicate one).
2. Enter a **name**.
3. Select the **permissions** the role should grant.
4. Set the **scopes**: the workspace/environment combinations where it applies.
5. Save.

Check what a role grants

Use **permission simulation**: choose a role (and optionally a workspace and environment) to see each permission as **granted** or **denied**, with the reason. It's the quickest way to confirm a role does what you intend. The **permission catalog** lists every available permission, grouped by object type.

Related topics

- [Roles and permissions](#) — full configuration reference and troubleshooting

- [Manage workspaces](#) (Administrator)
- [Manage environments](#) (Administrator)

Last updated: 2026-07-15

Tags: [Procedure](#) [Administrator](#) [Administration](#) [LMS-Training](#)

*Last updated on **Aug 3, 2026***

Manage workspaces

A **workspace** groups and isolates content: workflows, scripts, properties, thresholds, resources, and service requests. Use workspaces to keep teams' or applications' content separate. Every organization starts with a built-in **General** workspace.

ⓘ WHAT THIS SOLVES

One big shared space means teams step on each other's content and can't be given access to just their own, so changes and mistakes ripple across unrelated automation.

Create a workspace

To create a workspace, complete the following steps:

1. Go to **Workspaces** and add a workspace.
2. Enter a **name** (it must be unique, and can't be "General") and an optional description.
3. Save.

What lives in a workspace

Workflows, scripts, properties, thresholds, resources, and service requests are **scoped to a workspace**. (Frequencies, calendars, and tags are shared across all workspaces.)



GOOD TO KNOW

- The **General** workspace is built in and can't be deleted.
- A workspace **can't be deleted while it still contains objects**. You'll see exactly what's left to move or delete first.
- Who can work in a workspace is governed by **roles** scoped to that workspace, not by adding members to the workspace. See [Manage roles and permissions](#).

Related topics

- [Workspaces](#) — full configuration reference and troubleshooting
- [Manage roles and permissions](#) (Administrator)
- [Manage environments](#) (Administrator)

Last updated: 2026-07-15

Tags:

[Procedure](#)

[Administrator](#)

[Administration](#)

*Last updated on **Aug 3, 2026***

Prepare agents for connectors

Most connectors run on an agent, and some need software or permissions in place on that agent first. This guide covers what to set up so connectors run reliably. A job is only dispatched to an agent that matches the task's operating system and agent type. If nothing matches, the job can't be assigned.

⚠️ WHAT THIS SOLVES

A connector fails at run time because the agent is missing a driver, an interpreter, or access to a path. The failure looks like a job problem when it's really a setup gap on the machine.

File access

The **Wait for File** task, and the script-file and output-file options of the SQL connectors, can only read or write files in directories the agent is allowed to use. By default the allowed directories are:

- `/tmp`
- `/var/tmp`
- the operating system temp directory
- the agent's working directory

To allow other locations, set the `OPCON_ALLOWED_FILE_PATHS` environment variable on the agent to a comma-separated list of paths, then restart the agent. Setting this variable **replaces** the default list rather than adding to it, so include any default directories you still need. A path outside the allowed list fails immediately with an "outside allowed directories" error.

Script interpreters

The **Run Script** task uses the interpreter installed on the agent. Continuum does not provide it. Make sure the interpreter for the script types your Builders use is present:

Script type	Requirement on the agent
Shell	Always available (<code>cmd.exe</code> on Windows, <code>/bin/sh</code> on Linux/macOS)
PowerShell	<code>powershell.exe</code> on Windows; PowerShell 7+ (pwsh) must be installed on Linux/macOS
Bash	<code>/bin/bash</code> present
Python	<code>python</code> (Windows) / <code>python3</code> (Linux/macOS) installed and on <code>PATH</code>

Database drivers and tooling

For the **SQL Database Executor** connectors:

- **ODBC** task types need the relevant ODBC driver/DSN installed on the agent.
- **Oracle** task types need the agent to meet the Oracle client prerequisites.
- **MS SQL DTEXec (SSIS)** runs on **Windows agents only** and needs SSIS tooling installed on that agent.

Connection endpoints and credentials are configured separately. See [Set up database connections](#).

Legacy LSAM agents

The **UNIX/Linux LSAM** and **Windows LSAM** connectors don't use the Universal Agent. They run through the **relay**, which forwards jobs to your existing LSAM machines. For these to work:

- The **relay** must be running and the target **LSAM machines** reachable. A "no agent available" error for an LSAM command points here, not at the Universal Agent.

- A task's **machine name** must match the LSAM agent name registered with the relay.
- **Windows LSAM (current release)** runs jobs under the LSAM's own service account; arbitrary run-as users and embedded scripts are not yet supported. Per-job environment variables are supported.

Troubleshooting

Symptom	Cause	Fix
File task fails with "outside allowed directories"	The path isn't in the agent's allowed list	Add the location to <code>OPCON_ALLOWED_FILE_PATHS</code> (keep the defaults you need) and restart the agent.
Run Script task can't start the interpreter	Interpreter not installed on the agent (often <code>pwsh</code> or <code>python3</code>)	Install the interpreter on the agent and confirm it's on <code>PATH</code> .
SQL/ODBC/Oracle task can't connect	Driver/client missing on the agent	Install the ODBC driver/DSN or Oracle client on the agent.
SSIS task can't be assigned	Not a Windows agent, or SSIS tooling missing	Target a Windows agent with SSIS tooling installed.
LSAM command reports "no agent available"	Relay down or LSAM machine unreachable	Check the relay and LSAM connectivity.

Related topics

- [Set up database connections](#)

- [Wait for a file before your workflow continues](#) (Builder)

Last updated: 2026-07-15

Tags: [Reference](#) [Administrator](#) [Agents](#) [Connectors](#) [LMS-Training](#)

*Last updated on **Aug 3, 2026***

Set up and manage relays

A **relay** lets your on-prem legacy **LSAM** machines run jobs that OpCon Continuum dispatches from the cloud. You install it on your side, it connects **outbound** to OpCon (no inbound firewall changes), and it hands work to the LSAM machines behind it.

! WHAT THIS SOLVES

Your legacy LSAM machines live on-prem, but the scheduler runs in the cloud, and opening inbound firewall holes to reach them is a non-starter. The relay bridges that gap with a single outbound connection, so cloud-scheduled work reaches on-prem machines without exposing them.

How a relay connects

- It connects **outbound only**: nothing inbound needs to be opened to the internet.
- It authenticates with a `credentials.json` file you download when you register it.
- One relay can front several legacy agents, and a legacy agent group can span more than one relay.

1. Register the relay

1. Go to **Agents → Legacy Agents & Groups** and choose **Register Relay**.
2. On the success screen:
 - **Download credentials.json**: the file the relay needs to authenticate.
 - Note the **connection details** shown: the `AGENT_SERVICE_URL` and path prefix for your environment. You'll pass these to the installer (or to Docker) when you install.
 - Optionally copy a **run snippet** (Docker / Linux / Windows) or use the **download link** / `docker pull` for the relay software.

The **client secret is shown only once**. If you lose it, you don't need to start over. You can [rotate the credentials](#) later.

2. Install the relay

Pick the platform that fits your environment. **Docker is recommended**: it bundles everything the relay needs, so there's no runtime to install on the host.

Platform	Notes
Docker (recommended)	Bundles the relay and the LSAM connector: no host runtimes required. Mount the folder holding <code>credentials.json</code> , pass the connection variables with <code>-e</code> , expose port <code>8080</code> .
Linux (systemd)	Run the installer with your connection details, drop <code>credentials.json</code> in <code>/etc/opcon/</code> , then <code>systemctl enable --now opcon-relay</code> . The installer writes the settings to <code>/etc/opcon/relay.env</code> for you.
Windows	Run the installer as Administrator with your connection details, place <code>credentials.json</code> under <code>C:\ProgramData\OpCon\Relay</code> , and start the <code>OpConRelay</code> service.

The native installers take the connection details as **arguments** and configure the relay for you, so you don't set environment variables by hand. Use the values shown in the register dialog:

- **Linux:** `sudo ./install.sh --agent-service-url <url> [--agent-service-path-prefix /agent-api]`
- **Windows:** `.\install.ps1 -AgentServiceUrl <url> [-AgentServicePathPrefix /agent-api]`

`--agent-service-url` / `-AgentServiceUrl` is **required**. Use the path prefix (e.g. `/agent-api`) for cloud environments and leave it empty for local/dedicated

setups. To change a setting on Linux later, edit `/etc/opcon/relay.env` and restart the service.

No runtimes to install. Every install method is self-contained: Docker *and* the native Linux/Windows packages bundle everything the relay needs. You don't need Node.js or .NET on the host.

3. Confirm it's running

- **Health check:** `GET http://localhost:8080/health` returns **200** when ready (**503** while starting).
- **Docker:** `docker ps` shows the health status.
- **Linux:** `systemctl status opcon-relay` (logs: `journalctl -u opcon-relay`); **Windows:** `Get-Service OpConRelay` (logs: `relay-stdout.log` / `relay-stderr.log` in the log directory).
- **In OpCon:** the relay appears and its legacy agents come online.

Rotate credentials

If the client secret is lost or you want to cycle it, **regenerate the credentials** on the relay instead of deleting and re-registering it:

1. Open the relay and choose **Regenerate credentials**.
2. Download the new `credentials.json` (the new secret is shown **once**).
3. Replace the file on the relay host and restart it.

The **old credentials stop working immediately** after rotation. A lost secret can't be used to connect once you've rotated.

Keep it secure

- Protect `credentials.json` with restrictive permissions (the installers do this for you, so keep them if you replace the file manually).
- The relay needs **no inbound internet ports**. Port `8080` (`/health`) is only for local health checks.

- Treat the client secret like a password: never share or commit it.

When something's wrong

Symptom	Likely cause	Fix
Startup error about <code>credentials.json</code> keys	The file was edited or is out of date	Re-download it from the register dialog and use it unchanged.
Opaque <code>403</code> / <code>404</code> on startup	Wrong <code>AGENT_SERVICE_URL</code> or missing path prefix	Use the exact connection values shown in the register dialog.
Relay stays "unhealthy"	It never finished starting (credentials or connectivity)	Check the relay logs; verify the credentials and connection variables.
Legacy agents show Unknown (amber)	The relay is stale/down, not the agents	Check the relay's health and connectivity first.

Related topics

- [Relays](#) — full configuration reference and troubleshooting
- [Manage agents and agent pools](#) (Administrator)
- [Prepare agents for connectors](#) (Administrator)

Last updated: 2026-07-27

Tags: [Procedure](#) [Administrator](#) [Agents](#) [LMS-Training](#)

*Last updated on **Aug 3, 2026***

Get familiar as a Non-OpCon User

Reading about Continuum only gets you so far: the fastest way to learn it is to submit a request yourself in a test environment. Work through this checklist from top to bottom, checking off each item as you complete it. Each item links to the page that explains it in detail.

Goal: understand what Continuum can automate for you, and run a piece of automation yourself without building or configuring anything.

Understand what you can automate

- ☐ Read what kinds of systems and work Continuum can automate in business terms.
- ☐ See [What Continuum can automate](#).

Find and submit a request

- ☐ Sign in and land on your Business dashboard; note the self-service catalog and your recent activity.
- ☐ Go to **Self Service** and use the **search box** or **tag filters** to find a request.
- ☐ Open the request and read any guidance shown at the top of the form.
- ☐ Fill in the fields and select **Submit**.
- ☐ Confirm you see a message telling you whether it was submitted successfully.
- ☐ See [Submit a service request](#).

You're ready when

You can find a request in the self-service catalog, understand what it will do, submit it, and know what a successful submission means: the request was accepted and the automation is running in the background.

Related topics

- [Getting started with OpCon Continuum](#)
- [Glossary](#)

Tags: [Procedure](#) [Non-OpCon User](#) [Orientation](#)

*Last updated on **Aug 3, 2026***

Non-OpCon User documentation

Understand the value and capabilities Continuum delivers, then find and submit a service request through self-service — no building or configuring required.



Get familiar as a Non-OpCon User

A checklist of hands-on tasks so you can learn how to find and submit a service request through OpCon Continuum self-service.



Submit a service request

How to submit a service request so you can trigger an automated workflow without building it yourself.



What Continuum can automate

A business-level overview of what Continuum can automate and why connectors matter for your bank or credit union.

New here? See [Getting started with OpCon Continuum](#).

Last updated on Aug 3, 2026

Submit a service request

The **self-service portal** lets you run a piece of automation yourself (request a report, kick off a process, ask for a restore) by filling in a short form. You don't need to know anything about the workflows behind it.

ⓘ WHAT THIS SOLVES

When you need routine automation run (a report, a restore, a process kicked off), waiting on the automation team by email or ticket is slow, and there's no safe way to do it yourself.

Find and submit a request

To submit a service request, complete the following steps:

1. Go to **Self Service**. You'll see the requests available in your environment.
2. Use the **search box** or the **tag filters** to find the one you need.
3. Select a request to open its form. Read any guidance shown at the top.
4. Fill in the fields, then select **Submit**.
5. Confirm when prompted. You'll see a message telling you whether it was submitted successfully.

💡 GOOD TO KNOW

- You only see requests that have been made available in **your environment**: if something's missing, ask the team that publishes requests to deploy it there.
- A successful submission means your request was **accepted** and the automation has been kicked off, and the actual work then runs in the background.
- If you see a partial-failure message, try again or contact your automation team.

Related topics

- [What OpCon Continuum can automate](#) (Non-OpCon User)

Last updated: 2026-07-03

Tags: [Procedure](#) [Non-OpCon User](#) [Self Service](#) [LMS-Training](#)

*Last updated on **Aug 3, 2026***

What Continuum can automate

Banks and credit unions run on a web of systems (the core banking platform, databases, file exchanges with partners, and homegrown scripts) that all have to run in the right order, at the right time, every day. Continuum automates that work end to end, and coordinates how those systems talk to one another, including the **mainframe core**.

The part that matters most for adoption is **connectors**: the building blocks that let Continuum work with the systems you already have, instead of replacing them.

❗ WHAT THIS SOLVES

The daily work that keeps a bank or credit union running depends on many systems running in the right order, on time, every day. Done by hand it's slow, fragile, and hard to scale, and a single missed or mistimed step can delay the whole day's processing.

What a connector lets Continuum do

Each connector adds a kind of work a workflow can perform. Today these include:

- **Wait for data to arrive** before processing it: for example, hold a workflow until the nightly core export file has fully landed, so nothing runs against half-written data.
- **Run programs and scripts** on your servers: the utilities and batch jobs you already rely on.
- **Work with databases**: SQL Server, MySQL, Oracle, and ODBC-compatible systems: run a script, kick off a database job, or refresh a reporting database as a step in a larger flow.
- **Reuse legacy automation**: drive your existing UNIX/Linux and Windows LSAM machines without rebuilding them, so you can adopt Continuum gradually.

- **Compose reusable workflows:** build a standard sequence once and reuse it across processes.

Why it matters

- **Reliability for mission-critical processing.** Steps run in the right order and only when their data is ready, which reduces missed windows and manual intervention.
- **Use what you already own.** Connectors meet your systems where they are, including legacy agents and core systems, protecting existing investments.
- **A clear audit trail.** Every run is recorded, which is useful when examiners or auditors ask what ran, when, and whether it succeeded.
- **Faster to adopt.** Built-in connectors need no setup, and existing automation can be brought in as-is.

Who does what

- **Administrators** set up the environment and the connections to your systems.
- **Builders** design the workflows and choose the connectors for each step.
- **Operators** watch the daily run and respond if something needs attention.

As a sponsor, you don't configure any of this, but knowing what Continuum can connect helps you speak to its value and spot where it can remove manual work in your organization.

Related topics

- [Run SQL against your databases](#) (Builder)
- [Wait for a file before your workflow continues](#) (Builder)

Last updated: 2026-07-03

Tags:

[Concept](#)

[Non-OpCon User](#)

[Connectors](#)

[LMS-Training](#)

*Last updated on **Aug 3, 2026***

Last updated: 2026-06-24

How connectors work

Behavior-level overview of the connector model.

A **connector** is what lets Continuum do a specific kind of work — run a command, wait for a file, run a SQL script against a core banking database, call into a legacy system. Each connector contributes one or more **task types**. When a Builder adds a job to a workflow, they choose a task type; that choice determines what the job does and what settings it needs.

This is the surface customers care about most: connectors are how Continuum integrates with the systems a bank or credit union already runs.

The four ways a connector runs

How a connector runs determines what (if anything) has to be set up before a customer can use it.

How it runs	What it means	Setup needed	Examples
Built-in	Ships with the Universal Agent	None	Run Command, Run Script, Wait for File
Downloadable	Delivered to the agent when first needed	None by the customer; the connector is published to the platform	SQL Database Executor
Platform-internal	Runs inside the platform — no agent involved	None	Null Job, Workflow Container
Legacy (LSAM)	Runs on existing legacy LSAM agents through the relay	A relay and reachable LSAM agents	UNIX/Linux LSAM, Windows LSAM

flowchart LR

J[Job in a workflow] → V[Platform validates the task settings]

V → R{How the connector runs}

R -- Built-in → UA[Universal Agent runs it]

R -- Downloadable → UA2[Agent downloads it, then runs it]

R -- Platform-internal → P[Platform runs it – no agent]

R -- Legacy LSAM → RL[Relay forwards to the LSAM machine]

UA → O[Outcome reported back]

UA2 → O

P → O

RL → O

Connections

Some connectors need a saved **connection** – the endpoint and credentials for an external system. The database connectors are the main example: a SQL script task can't run until it knows which server, database, and login to use. Built-in utility connectors (Run Command, Run Script, Wait for File) need **no** connection because they act on the agent itself.

Credentials in a connection are delivered to the connector securely at run time and are not exposed in the job definition. (Connection setup and security is an Administrator topic.)

Where a task type can run

A task type can require a specific **operating system** or **agent type**. The platform only dispatches a job to an agent that matches – for example, an SSIS package task targets Windows agents, and a UNIX LSAM command targets UNIX LSAM agents. If no matching agent is available, the job can't be assigned.

Before a job runs

When a Builder configures a task, its settings are **validated against the connector's parameter definition** before any work is sent to an agent. A task with missing or invalid settings is rejected up front rather than failing on the agent. Each connector also runs in isolation on the agent, so one connector's failure doesn't affect others.

The connector catalog

Connector	Task type(s)	How it runs	Needs a connection?	Runs on
Run Command	Run Command	Built-in	No	Universal Agent (Windows/Linux/macOS)
Run Script	Run Script (PowerShell, Bash, Python, Shell)	Built-in	No	Universal Agent (Windows/Linux/macOS)
Wait for File	Wait for File	Built-in	No	Universal Agent (Windows/Linux/macOS) — see Wait for File task
SQL Database Executor	MS SQL DTEExec (SSIS), MS SQL Job, MS SQL Script, MySQL Script, Oracle Script, Other DB (ODBC)	Downloadable	Yes — a database connection (SQL Server, MySQL, Oracle, or ODBC)	Universal Agent; OS varies by task type
Null Job	Null Job	Platform-internal	No	No agent — completes immediately
Workflow Container	Workflow Container	Platform-internal	No	No agent — runs another workflow as a step
UNIX/Linux LSAM	UNIX Command	Legacy (relay)	No	Legacy UNIX/Linux LSAM agent

Connector	Task type(s)	How it runs	Needs a connection?	Runs on
Windows LSAM	Windows Command	Legacy (relay)	No	Legacy Windows LSAM agent

Things to know

- **Null Job** is a workhorse for workflow design, not a no-op to dismiss: it's used to aggregate dependencies, hang events off a single point, and build a workflow on hold.
- **Workflow Container** embeds one workflow inside another — when troubleshooting a workflow, remember a job may itself be an entire child workflow.
- **Legacy LSAM** task types only work through a configured relay; "no agent available" for a UNIX/Windows Command usually points at the relay or LSAM connectivity, not the Universal Agent.
- **SQL Database Executor** reaches out to external databases, so it is the connector most likely to hit connection, credential, and driver problems.

Related topics

- [Wait for File task](#)

Tags: [Concept](#) [Non-OpCon User](#) [Connectors](#) [Task types](#) [LMS-Training](#)

*Last updated on **Aug 3, 2026***

Last updated: 2026-06-24

Dashboards

Evolving. This page is accurate for the current build, but the underlying feature is still in active development — details can change between releases.

Orientation is in [Getting started](#).

OpCon Continuum opens on a **dashboard** tailored to the user. The platform routes each user to the dashboard that matches their role (remembering their last-viewed choice), so the first thing they see is relevant to what they do.

The three dashboards

Dashboard	For	Typical widgets
Administrator	Keeping the environment healthy	Environment health, agents, relays, agent-pool utilization
Operator	Watching the daily run	Failed jobs, event summary, workflow/job status counts, agent-pool job queue
Business	Non-technical users	Self-service catalog, the user's events

Things to know

- The dashboards are **read-only summaries** — they surface state from other areas (Processes, Agents, Events, Self Service); they don't change anything.
- Which dashboard a user lands on follows their role/last choice, so two users may see different home screens — that's expected.

Related topics

- [How the daily run works](#)
- [Getting started](#)

Tags: [Concept](#) [Operator](#) [Dashboards](#)

*Last updated on **Aug 3, 2026***

Last updated: 2026-06-24

How the daily run works

See [How workflows work](#) for the design side.

The **daily run** is what happens after a workflow is built: its jobs move through a series of statuses as dependencies are satisfied, work is dispatched to agents, and results come back. Operators watch and act on the run in the **Processes** view.

Lifecycle

```
flowchart TD
    A[Workflow built] --> B[Jobs waiting:<br/>dependency / start time / resource / machine]
    B --> C{Eligible?}
    C -- Yes --> D[Job starts on an agent]
    C -- On hold --> H[Held until released]
    H --> C
    D --> E[Job running]
    E --> F[Completion processing<br/>*_PENDING_TERM]
    F --> G{Result}
    G -- Finished OK --> I[Dependents released]
    G -- Failed --> J[Operator responds]
    I --> K[Workflow completes]
```

Status families

Every job status falls into one of these families (the [statuses reference](#) lists them all):

Family	Meaning
Held	Held; won't run until released.
Waiting	Eligible but waiting on something — a dependency, its start time, a resource, or an agent.
Running	Actively starting or running.
Pending-term	Finished, but completion is still being processed. Transitional, not stuck.
Terminal	Done — finished OK, failed, cancelled, skipped, etc.

Two-phase completion

When a job finishes, it first enters a **pending-term** status (e.g. `FINISHED_OK_PENDING_TERM`) while completion is made durable, then settles to its final status. Dependent jobs are **not** released until that settles — so a brief pending-term state is normal, not a hang. (Cancellation and initialization errors release dependents immediately.)

Recovery

The run survives a restart: the engine rebuilds its in-memory state from its event store and snapshots, so in-flight jobs are reconciled rather than lost.

Things to know

- A job **waiting** is usually working as designed — identify *what* it's waiting on (dependency, start time, resource, machine) before treating it as a fault.
- **Pending-term** is transitional. If a job is stuck there for a long time, that's worth investigating; a momentary pending-term is not.
- **On hold** is intentional until released.

Related topics

- [Job and workflow statuses](#)
- [Operator response guide](#)

Tags: [Concept](#) [Operator](#) [Processes](#) [LMS-Training](#)

*Last updated on **Aug 3, 2026***

Last updated: 2026-06-24

How workflows work

Behavior-level overview of the workflow model.

A **workflow** in OpCon Continuum is a scheduled, dependency-ordered set of **jobs**. It defines *what* runs (the jobs), *when* it's eligible to run (frequencies, calendars, build settings), *in what order* (dependencies), and *what should happen* in response to outcomes (events). Each job runs a **task type** — the connectors documented separately — on an **agent**.

Terminology note: in the configuration and runtime model a workflow is represented as a **schedule**. Treat "workflow" (the UI term) and "schedule" (the underlying term) as the same thing.

What a workflow contains

Part	What it is
Jobs	The units of work. Each runs a task type on an agent and has its own timing, dependencies, and events.
Frequencies	Rules for which days/times a job (and the workflow) is eligible to run.
Dependencies	Conditions that must be met before a job runs.
Events	Actions (commands) fired in response to job outcomes.
Build settings	How and when the workflow is built into a day's run.
Calendars / working days	Holiday calendars and working days that shape eligibility.

Jobs

A job is configured with:

- A **task definition** — the task type to run (e.g. Run Command, SQL Script) plus its parameters.
- An **agent assignment** (how the job is placed on an agent — see below).
- One or more **frequencies** with runtime settings (timing, retries, reruns).
- **Dependencies** (what must be true before it runs).
- **Events** (what to do on certain outcomes).
- **Properties, tags, custom fields**, and flags like **disabled** and **multi-instance**.

How jobs are ordered: dependencies

A job runs only when all its dependencies are satisfied. There are four types:

Dependency	Satisfied when
Job	A predecessor job reaches the required result — success , failure , or any — optionally on a day offset and a specific frequency.
Threshold	A named threshold reaches a value.
Resource	Enough of a named resource is available (the job reserves it).
Expression	A custom expression evaluates true.

Job dependencies are how the run is sequenced; threshold and resource dependencies coordinate shared state and capacity across jobs.

When jobs run: frequencies and build

Frequencies define eligibility. Each job's frequency carries runtime settings, including:

- **Build status** — `released`, `onHold`, `doNotSchedule`, `toBeSkipped`, or `disabled`.
- **Start-time offsets** and **late-to-start / latest-start / late-to-finish** offsets.
- **Max run time** and **estimated run time**.
- **Failure retries** (whether to retry, how many times, minutes between attempts).
- **Success reruns** (none, recurring instances, or a restart offset).

Build turns the definition into a day's run. Build settings include how many days in advance to build, the build time, whether to build **on hold**, whether to overwrite on build, and how long to keep instances. Holiday **calendars** and **working days** remove non-eligible days.

Where jobs run: agent assignment

Mode	Behavior
Specific agent	Runs on a chosen agent, with optional secondary/tertiary/quaternary failover agents.
Pool – least-tasked	Runs on the least-busy agent in a pool.
Pool – run-all	Runs on every agent in a pool.

Responding to outcomes: events

Events fire a **command** in response to a job outcome. Triggers:

- **Job status** — a status such as `FinishedOk`, `Failed`, `LateToStart`, `LateToFinish`, `Skipped`, `ExceededMaxRuntime`, and others.
- **Job exit description** — a comparison (equal, greater than, range, etc.) against the job's exit value.
- **Job completion expression** — a custom expression.

Workflows can also carry schedule-level events.

The run lifecycle (link to runtime)

A workflow is **built** into a day's run, then jobs run as their dependencies are satisfied, moving through statuses to completion. Operators monitor and act on that run in the **Processes** view. See the runtime/Processes docs for status meanings and operator actions.

flowchart TD

A[Workflow built into the day's run] → B{Job dependencies satisfied?}

B -- No → B

B -- Yes → C[Job placed on an agent]

C → D[Task type runs]

D → E{Job outcome}

E -- Success → F[Dependent jobs released]

E -- Failure → G[Events fire / Operator responds]

G → H{Retry or restart?}

H -- Yes → C

H -- No → I[Workflow finishes]

F → I

Versions and deployments

Workflows are **versioned** — each saved change creates a new version, and history can be reviewed or rolled back. A version is **deployed** to an environment to run there; deployments are tracked, so the same workflow can move from test to production deliberately.

Things to know

- **Build on hold** is intentional — a workflow built on hold won't run until released; it's not stuck.
- **Disabled** jobs and `doNotSchedule` / `toBeSkipped` build statuses are by design — check the job's frequency settings before treating a "missing" job as a fault.
- **Multi-instance** jobs/workflows can run more than once concurrently; expect multiple instances.
- A **job dependency on failure or any** is valid design — a downstream job can be meant to run when its predecessor fails.

Related topics

- [How connectors work](#)
- [How the daily run works](#) (runtime monitoring)

Tags: [Concept](#) [Builder](#) [Workflows](#) [LMS-Training](#)

*Last updated on **Aug 3, 2026***

Last updated: 2026-06-24

Self-service portal

The build side is [service requests](#).

The **self-service portal** is where a business user runs the automation a Builder has published — without seeing workflows or jobs. They pick a request, fill in a short form, and submit.

What the user sees

- A **catalog** of the service requests **deployed to their environment**, searchable and filterable by **tag**. Each card shows the request name, a short description, and its tags.
- Selecting one opens its **form** — any **documentation** and **confirmation message** the Builder set, then the input fields.
- On **Submit**, a confirmation dialog appears; confirming fires the request's events and shows a success or error result.

What happens on submit

The user's answers are substituted into the request's events, which are then fired (together if there are several). The user sees whether it succeeded; if some events fail, the portal reports a partial failure to try again.

Things to know

- The catalog is **environment-scoped** — a user only sees requests **deployed and active** in their environment. "I can't see request X" usually means it isn't deployed there.
- Access is permission-gated: submitting requires the self-service view permission for the environment. Role-based access is still being rolled out, so in the current release these permissions describe *intended* access rather than being enforced. See [Roles and permissions](#).

- A submission **fires events**; the actual work is the automation those events trigger, so a "submitted OK" result means the events were accepted, not that the downstream work is finished.

Related topics

- [Service requests](#)

Tags:

[Concept](#)[Non-OpCon User](#)[Self Service](#)[LMS-Training](#)

*Last updated on **Aug 3, 2026***

Last updated: 2026-06-24

Toolkit overview

The **Toolkit** holds the reusable building blocks Builders use across workflows — scheduling rules, shared limits, organization, and parameter values. Defining them once in the Toolkit keeps workflows consistent and maintainable.

Item	What it is	Scope
Frequencies	Scheduling patterns (daily/weekly/monthly/yearly rules) that decide which days a job is eligible.	Global
Calendars	Named sets of dates (holidays/blackouts) used to exclude days.	Global
Thresholds	Named numeric values used to gate jobs (flow control).	Workspace-scoped
Resources	Named capacity pools that jobs reserve and release.	Workspace-scoped
Properties	Named key/value pairs (some encrypted) used to parameterize jobs.	Workspace-scoped
Tags	Labels for organizing and searching jobs and other objects.	Global

Why scope matters

- **Workspace-scoped** (Properties, Thresholds, Resources) default to the tenant's **General** workspace and are only visible/usable within their workspace. A "missing" item is often in a different workspace.
- **Global** (Frequencies, Calendars, Tags) are shared across all workspaces.

How workflows use them

- A job references **frequencies** (when it runs), **calendars** (days to skip), **thresholds** and **resources** (as dependencies), **properties** (values at runtime), and **tags** (organization). See [How workflows work](#) and [Job dependencies](#).

Related topics

- [Frequencies](#) · [Calendars](#) · [Thresholds and resources](#) · [Properties and tags](#)

Tags: [Concept](#) [Builder](#) [Toolkit](#) [LMS-Training](#)

Last updated on Aug 3, 2026

Last updated: 2026-07-20

Agents and agent pools

Task walkthrough: [Manage agents and agent pools](#). This page is the full configuration and troubleshooting reference.

An **agent** is the software that runs jobs on a machine. **Agent pools** group agents so jobs can be routed to whichever agent is available. The **Universal Agent** runs the built-in and downloadable connectors; legacy LSAM machines are reached through the relay.

Agent pool

Field	Notes
<code>name</code>	Unique; cannot be changed after creation .
<code>description</code>	Optional, editable.

Pool views also show **queued** and **running** job counts.

Agent

Field	Values / notes
<code>name</code> , <code>hostname</code>	Identity.
<code>os</code>	<code>windows</code> , <code>linux</code> , <code>darwin</code> .
<code>architecture</code>	<code>x64</code> , <code>arm64</code> .
<code>agentVersion</code>	Installed agent version.
<code>agentType</code>	<code>UNIVERSAL</code> , plus legacy <code>WINDOWS</code> / <code>UNIX</code> ; other values (<code>IBM_i</code> , <code>MCP</code> , <code>SAP</code> , <code>ZOS</code>) exist in the type list.
<code>status</code>	<code>ONLINE</code> , <code>OFFLINE</code> , <code>BUSY</code> , <code>DRAINING</code> , <code>DISABLED</code> , <code>UNKNOWN</code> . <code>UNKNOWN</code> = relay-down (reachability can't be determined because the agent's relay went stale) — distinct from <code>OFFLINE</code> = agent-down. Shown amber, with a text label. Applies to legacy LSAM agents reached via a relay.
<code>updateStatus</code>	<code>UP_TO_DATE</code> , <code>UPDATE_AVAILABLE</code> , <code>UPDATING</code> , <code>UPDATE_FAILED</code> .
<code>lastHeartbeat</code>	Last check-in; shows Never until the first heartbeat.
<code>currentJobCount</code> / <code>maxConcurrentJobs</code>	Load vs. its concurrency cap (the cap is editable).
<code>capabilities.runtimes</code>	Detected runtimes: node, python, dotnet, java, powershell, docker.

The agent detail view also lists recent commands. **Stub (not populated yet)**: CPU cores, memory, free disk, and the installed-plugins section show .

How jobs are routed to agents

A job's **agent assignment** (set in the workflow) carries a `kind` (**agent-pool** or **legacy-group**) and a mode:

Assignment	Mode	Behavior
Specific agent	—	Primary agent, with optional secondary/tertiary/quaternary failover .
Agent pool	least-tasked	The least-busy agent in the pool.
Agent pool	run-all	Every agent in the pool.
Legacy agent group	least-tasked	The least-tasked ONLINE member of the group (interim selection: random among ONLINE).
Legacy agent group	run-on-all	Every member — fanned into one job instance per member , frozen at build.

Legacy **LSAM** task types dispatch through a **legacy agent group** (or a specific legacy agent), not a Universal-Agent pool — the assignment selector only offers groups **compatible** with the task type (groups are type-homogeneous; see the Support notes). See [Job configuration](#). The platform only routes a job to an agent whose **type/OS** matches the task type's requirements.

Group dispatch gating. A **least-tasked** group job needs **≥1 member ONLINE**; like specific-agent gating it **fails open** on a stale/`UNKNOWN` relay feed (feed freshness is tracked per group) so the scheduler never deadlocks on untrusted status. A **run-on-all** group job resolves its members to a frozen snapshot **at build** and reuses the multi-instance loop (one instance per member).

Things to know

- **Heartbeat "Never"** means the agent hasn't checked in yet (or has lost connectivity) — a common first thing to check for "no agent available."
- **DRAINING** means the agent is finishing current work and not taking new jobs (e.g. before an update); **DISABLED** means it's been taken out of rotation.
- **UNKNOWN vs OFFLINE for legacy agents:** **UNKNOWN** (amber) means the **relay** is stale, so a legacy LSAM agent's reachability can't be confirmed — *not* that the agent is down. **OFFLINE** means the agent itself is down. When a relay recovers, its agents leave **UNKNOWN** on the next heartbeat. Status updates within ~30s (the heartbeat cadence).
- **Legacy agents and groups are managed in-product** on the **Agents** page, on its **Legacy Agents & Groups** tab (a sibling of the **Agent Pools** tab; the two former navigation entries were merged into a single **Agents** item). Register/edit/delete a legacy agent there; the relay **hot-reconciles** its LSAM connections within one heartbeat — no relay restart needed. (Agent names are unique per relay.)
- **Legacy agent groups are type-homogeneous** (all members share one legacy type) and may **span relays**. A group is an operational **dispatch target** for a legacy job (least-tasked or run-on-all; see the routing table). Deleting a group **soft-deletes the group only** — never its member agents or their queues.
- CPU/memory/disk and installed-plugins on the agent detail are **not implemented yet** — the  is expected, not missing data.

Troubleshooting

Symptom	Likely cause	Resolution
Job can't be assigned / <code>WAIT_MACHINE</code>	No matching agent ONLINE (type/OS mismatch, all draining/disabled/offline)	Check agent status/type vs the task's requirements (Administrator).
Specific-agent job sits in <code>WAIT_MACHINE</code>	The job targets one specific agent that is OFFLINE on a fresh status feed — it holds rather than fail, and proceeds when that agent is ONLINE again. (For universal agents, BUSY/DRAINING/DISABLED also hold; ONLINE releases.)	Bring the target agent online, or reassign the job (Administrator/Builder).
Specific-agent job advances despite an offline-looking target	By design: when the status feed is stale or UNKNOWN , the gate fails open so the scheduler never deadlocks on untrusted status.	Confirm the target agent/relay is actually reachable; investigate the stale feed (Administrator).
Least-tasked group job sits in <code>WAIT_MACHINE</code>	No group member is ONLINE on a fresh feed — it holds and proceeds when a member comes online. (Fails open if the relay feed is stale/ <code>UNKNOWN</code> .)	Bring a group member online; check the relay if the feed is stale (Administrator).
Agent shows OFFLINE / heartbeat stale	Connectivity or the agent process is down	Check the agent host and network (Administrator).

Symptom	Likely cause	Resolution
Legacy agent shows <code>UNKNOWN</code> (amber)	Its relay is stale/down — agent reachability can't be determined (not the same as the agent being down)	Check the relay; on recovery, agents leave <code>UNKNOWN</code> within a heartbeat (Administrator).
Jobs pile up on one agent	<code>maxConcurrentJobs</code> reached, or pool routing	Raise the cap or use a pool (Administrator / Builder).

Tags: [Reference](#) [Administrator](#) [Agents](#) [LMS-Training](#)

*Last updated on **Aug 3, 2026***

Last updated: 2026-07-27

Relays

See [Agents and agent pools](#) and [LSAM legacy connectors](#).

Task walkthrough: [Set up and manage relays](#). This page is the full configuration and troubleshooting reference.

A **relay** is the on-prem software that lets legacy **LSAM** machines run jobs dispatched from OpCon Continuum in the cloud. It connects **outbound only** to the OpCon agent-service (no inbound firewall changes), authenticates with a downloaded `credentials.json`, and bridges dispatched work to the LSAM machines behind it. Legacy agents are reached *through* a relay; a relay may front more than one legacy agent and legacy agent groups may span relays.

The relay is a pure **credentials consumer** — its only inputs are a `credentials.json` file and a few connection environment variables. Relay support covers registration and credential download, credential rotation, artifact distribution, installation, and a `/health` readiness endpoint.

Register a relay

In the UI: **Agents → Legacy Agents & Groups → Register Relay** (backend `POST /api/v1/relays/register`, tenant-scoped). On the success screen the dialog provides everything needed to stand up the relay:

On the register dialog	Detail
Client ID / Client Secret	The credential pair. The secret is shown once — mask/show toggle and per-value copy.
Download credentials.json	Writes the strict <code>{ relayId, clientId, clientSecret }</code> file the relay requires at startup.
Connection details	The runtime-derived <code>AGENT_SERVICE_URL</code> and the <code>/agent-api</code> path prefix for the environment.
Run snippets	Per-platform Docker / Linux / Windows run commands, pre-filled with the connection values.
Relay download links + <code>docker pull</code>	The latest published release (version + Docker image + native artifact), served from the relay-releases manifest.

`credentials.json` must match the exact three-key shape — a **strict** schema is enforced at boot, so an extra key such as the agent-only `agentPoolId` causes a parse failure. The relay package ships a template at `config/credentials.json.example`.

Connection environment variables

Variable	Required	Default	Purpose
<code>AGENT_SERVICE_URL</code>	✓	—	Base URL of the agent-service for the environment (from the register dialog).
<code>AGENT_SERVICE_PATH_PREFIX</code>	⚠ cloud only	<code>""</code>	Service path prefix for ALB/CloudFront routing, e.g. <code>/agent-api</code> . Omit for local/dedicated-origin.
<code>CREDENTIALS_FILE</code>	✓	<code>~/.opcon-relay/credentials.json</code>	Absolute path to <code>credentials.json</code> (installers pre-set this).
<code>WORK_POLLER_WAIT_SECONDS</code>	✗	<code>10</code>	Edge long-poll wait (seconds, max 20). Keep ≤ 10 on-prem behind CloudFront/envoy (~15s upstream timeout) to avoid 504s.
<code>HEALTH_PORT</code>	✗	<code>8080</code>	Port for the HTTP <code>/health</code> readiness endpoint.
<code>LOG_LEVEL</code>	✗	<code>info</code>	<code>trace</code> <code>debug</code> <code>info</code> <code>warn</code> <code>error</code> <code>fatal</code> .

The relay reads its configuration **entirely from environment variables**. There is **no relay.yaml** — the old YAML template has been removed. `config/relay.env.example` in the relay package is the authoritative reference for every variable.

A wrong `AGENT_SERVICE_URL` / missing `AGENT_SERVICE_PATH_PREFIX` typically surfaces as an opaque `403/404` from the cloud edge. Use the exact values from the register dialog.

Install

Every artifact is **self-contained** — Docker, the Linux/Windows native packages, and the LSAM connector all bundle their own runtimes, so **no host Node or .NET runtime is required on any platform**. The runtimes are bundled into the native artifact, and a later release slimmed the Windows-portable artifact and made the connector self-contained to start, closing out the host-runtime requirement. Full per-platform steps are in the `README.md` at the root of the relay package.

The native installers take the connection details as **arguments** and set the environment for you; only `credentials.json` is a file. `--agent-service-url` (Linux) / `-AgentServiceUrl` (Windows) is **required** — the installer aborts without it.

- **Docker:** mount the config dir holding `credentials.json` to `/opt/opcon-relay/config`, pass the connection env vars with `-e`, expose `8080`. Health via `docker inspect ... .State.Health.Status`.
- **Linux:** `sudo ./install/install.sh --agent-service-url <url> [--agent-service-path-prefix /agent-api] [--work-poller-wait-seconds 10]` → app under `/opt/opcon-relay`, config `/etc/opcon`, `opcon-relay` systemd unit; credentials at `/etc/opcon/credentials.json` (0600). The installer **generates** `/etc/opcon/relay.env` from its arguments and systemd loads it via `EnvironmentFile` — edit that file and restart to change values. Logs via `journalctl -u opcon-relay`.
- **Windows:** `.\deploy\install\install.ps1 -AgentServiceUrl <url> [-AgentServicePathPrefix /agent-api] [-WorkPollerWaitSeconds 10]` (Admin) → `C:\Program Files\OpCon\Relay`, config `C:\ProgramData\OpCon\Relay` (ACL: Administrators + SYSTEM); service `OpConRelay`. The service is managed by the **bundled nssm.exe** (copied into the install dir and registered), which sets the environment on the service and captures stdout/stderr to `relay-stdout.log` / `relay-stderr.log`. NSSM is now the **only** supported Windows service mechanism — the old `-UseNSSM` switch and the `sc.exe` native-service path have been removed, because the relay cannot run reliably as a bare native Windows service (it fails with error 1053).

Readiness / health

The relay runs a minimal zero-dependency `/health` server: **200** `{"status": "ok"}` when ready, **503** `{"status": "starting"}` during startup, 404 otherwise. Readiness follows the relay's own startup (ready after the core loops start, not-ready on connector max-restarts, closed on shutdown). The Docker `HEALTHCHECK` uses it with a 40s start-period to cover connector startup.

Rotate credentials

If a `clientSecret` is lost, **rotate** rather than delete + re-register. UI: the **Regenerate credentials** dialog on the relay; backend `POST /api/v1/relays/:id/credentials/rotate` (tenant-scoped, authenticated). It generates a new `clientId/clientSecret`, rotates the active credential **in place**, and returns the new pair **once**; it records who performed the rotation and when.

After rotation, **old credentials fail OAuth**: refresh-token validation verifies the token's `clientId` still resolves to an *active* credential, so a pre-rotation refresh token cannot keep minting ~24h access tokens. Cross-tenant/missing relay → 404.

Relay ↔ agent-service runtime

- The relay reports in on `POST /api/v1/relays/:id/heartbeat` and fetches its members from `GET /api/v1/relays/self/agents` (relay-authenticated).
- Legacy agent changes made in-product **hot-reconcile** to the relay within about one heartbeat — no relay restart. See [LSAM legacy connectors](#).
- When a relay goes stale, its legacy agents show **UNKNOWN** (amber) — *relay-down*, distinct from **OFFLINE** (agent-down). Specific-agent `WAIT_MACHINE` gating **fails open** on a stale/`UNKNOWN` feed so the scheduler can't deadlock. See [Agents and agent pools](#) and [Operator response guide](#).

Security notes

- Keep `credentials.json` at **0600** (Linux/Docker) or a restricted ACL (Windows); installers set this — preserve it if you replace the file by hand.
- **Outbound only** — no inbound internet ports. `8080 (/health)` is for local/orchestrator checks, not external exposure.
- Treat `clientSecret` like a password; never commit or share it.

Troubleshooting

Symptom	Likely cause	Fix
Boot error: "Unrecognized key" / "relayId is required"	Edited/old <code>credentials.json</code> with wrong keys	Re-download from the UI; the file must be exactly <code>{ relayId, clientId, clientSecret }</code> .
Opaque <code>403 / 404</code> from the cloud on startup	Wrong <code>AGENT_SERVICE_URL</code> / missing <code>AGENT_SERVICE_PATH_PREFIX</code>	Use the exact connection values from the register dialog.
Container stuck <code>unhealthy</code>	Relay never reached ready (auth/connector)	<code>docker logs</code> ; verify credentials and connection env.
Install aborts: <code>-- agent-service-url</code> is required	Installer run without the connection URL	Pass <code>--agent-service-url</code> (Linux) / <code>-AgentServiceUrl</code> (Windows) with the value from the register dialog.
Windows install fails: "Bundled nssm.exe not found"	Incomplete/partial relay artifact	Re-download the relay artifact; the package must ship <code>nssm.exe</code> alongside <code>install.ps1</code> .
Old credentials still work after a suspected leak	Credentials were never rotated	Rotate credentials (invalidates the old pair, incl. refresh tokens).
Legacy agents show <code>UNKNOWN</code>	The relay is stale/down	Check the relay (health, logs, connectivity) — not the individual agents.

Tags: [Reference](#) [Administrator](#) [Agents](#) [LMS-Training](#)

Last updated on **Aug 3, 2026**

Last updated: 2026-06-24

Plugins

Plugins define the **task types** (connectors) jobs can run — see [How connectors work](#).

Task walkthrough: [Manage plugins](#). This page is the full configuration and troubleshooting reference.

A **plugin** packages one or more task types (and sometimes connection types). The Plugins area lets an administrator see what's available and **enable or disable** plugins per tenant.

Plugin catalog

Each plugin shows a **display name**, internal **name**, **category**, **version**, and its **task types**; plugins that provide connection types link out to create a connection of that type.

Categories: DATABASE, FILE_TRANSFER, SCRIPT, CONTAINER, INTEGRATION, UTILITY, CUSTOM, INTERNAL.

Installation and enablement (per tenant)

Field	Notes
pluginName	e.g. builtin:command, sql-executor.
installedVersions	Versions available to the tenant.
defaultVersion	The version new jobs use.
isActive	Whether the plugin is enabled for the tenant.

Default visibility

Plugin prefix	Default
<code>builtin:*</code> (e.g. Run Command, Run Script, Wait for File)	Enabled by default
<code>internal:*</code> (Null Job, Workflow Container)	Enabled by default
<code>opcon.lsam.*</code> (legacy LSAM)	Enabled by default
<i>no prefix</i> (external, e.g. <code>sql-executor</code>)	Disabled by default — must be opted in

Things to know

- "A task type isn't available to build with" usually means its plugin is disabled — external plugins (like SQL) are **off by default** and must be enabled per tenant.
- Disabling a plugin removes its task types from new jobs; existing jobs that use it are a separate concern — confirm before disabling.
- Multiple versions can be installed; the **default version** is what new jobs pick up.

Troubleshooting

Symptom	Likely cause	Resolution
SQL (or other external) task type missing in the builder	The external plugin is disabled (default)	Enable the plugin for the tenant (Administrator).
New jobs use an unexpected plugin version	The default version changed	Check/set the default version (Administrator).

Tags: [Reference](#) [Administrator](#) [Plugins](#)

*Last updated on **Aug 3, 2026***

Last updated: 2026-06-24

Environments

Task walkthrough: [Manage environments](#). This page is the full configuration and troubleshooting reference.

An **environment** is a target — such as test or production — that workflows and service requests are deployed to. The registry service is the source of truth for environment identity and the services registered to each.

Model

Field	Notes
<code>name</code>	Unique per tenant.
<code>description</code>	Optional.
<code>environmentType</code>	<code>DEV</code> , <code>TEST</code> , <code>STAGING</code> , or <code>PRODUCTION</code> .
<code>status</code>	<code>active</code> , <code>inactive</code> , or <code>degraded</code> .
<code>metadata</code>	Free-form key/value used by provisioning.
<code>lastHeartbeatAt</code>	Last service heartbeat.

Registered services

Each environment has one or more **services**:

Field	Notes
serviceType	runtime-service or notification-service.
healthStatus	healthy, unhealthy, or unknown.
baseUrl, version	Endpoint and version.

How an environment is created

Creating an environment kicks off **asynchronous provisioning** (a background process that stands up the per-environment runtime service). Provisioning runs **fire-and-forget**: a provisioning failure is logged but does **not** fail the create request. So a newly created environment can exist before its runtime is fully ready.

Things to know

- The **Environments** screen isn't in the main navigation — it's reachable by direct link.
- A new environment that "isn't working yet" may still be **provisioning** — check the environment's service health (unknown/unhealthy) and heartbeat.
- degraded/unhealthy indicate the environment's services aren't fully healthy — a deployment target issue, not necessarily a workflow problem.

Troubleshooting

Symptom	Likely cause	Resolution
New environment not running workflows	Provisioning hasn't finished or failed (fire-and-forget)	Check service health/heartbeat; re-provision if needed (Administrator → escalate).
Service shows <code>unhealthy</code> / <code>unknown</code>	The registered service isn't reporting healthy	Check the service endpoint and heartbeat (Administrator).

Contact support when

- Provisioning silently failed (environment exists but its runtime service never registers/heartbeats).

Include the environment id/name, type, status, and the registered services' health.

Tags: [Reference](#) [Administrator](#) [Administration](#)

*Last updated on **Aug 3, 2026***

Last updated: 2026-06-24

Workspaces

Task walkthrough: [Manage workspaces](#). This page is the full configuration and troubleshooting reference.

A **workspace** groups and isolates content. Every tenant gets a system **General** workspace automatically; administrators create additional workspaces as needed.

Model

Field	Notes
<code>name</code>	Unique per tenant; cannot be "General" (reserved).
<code>description</code>	Optional.
<code>isSystem</code>	The General workspace is a system workspace — immutable and can't be deleted.

What's scoped to a workspace

Workflows, properties, thresholds, resources, scripts, and service requests. (Frequencies, calendars, and tags are **global**, not workspace-scoped — see [Toolkit overview](#).)

Membership

There is **no per-user workspace membership** today. Access to a workspace's content is governed by **role scopes** (a role can be scoped to a workspace), not by adding users to a workspace — see [Roles and permissions](#).

Things to know

- A workspace **can't be deleted while it still contains objects** — the error lists what's left (e.g. "2 workflows, 1 property") and they must be reassigned or deleted first. System workspaces can never be deleted.
- "Missing" workspace-scoped items (a property/threshold/resource/script) are usually in a **different workspace** — confirm the workspace before treating it as lost.

Troubleshooting

Symptom	Likely cause	Resolution
Can't delete a workspace	It's the system workspace, or it still has objects	Move/delete its objects first; the General workspace can't be deleted (Administrator).
A user can't see content you expect	Their role scope doesn't cover that workspace (note: enforcement caveat in roles doc)	Check role scopes (Administrator).

Tags: [Reference](#) [Administrator](#) [Administration](#)

*Last updated on **Aug 3, 2026***

Last updated: 2026-06-24

Run Command task

Behavior-level reference for diagnosing Run Command tasks.

Task walkthrough: [Run a command](#). This page is the full configuration and troubleshooting reference.

The **Run Command** task runs a command line on the agent machine. It's a **built-in** task type (`builtin:command`), so no setup or connection is required.

Property	Value
Task type	Run Command (<code>run-command</code>)
Runs on	Universal Agent only
Operating systems	Windows, Linux, macOS
Connections required	None
Category	Script

How it works

The agent runs the command on the agent machine and captures its output.

- **Run via Shell on** (default) — the command runs through the system shell (`cmd.exe` on Windows, `/bin/sh` on Linux/macOS), so pipes, redirects, and shell built-ins work. Any arguments are appended to the command and safely quoted.
- **Run via Shell off** — the command is launched directly as a process. Shell features are **not** available; the command must be an executable and arguments are passed to it directly.

Standard output and standard error are both captured; standard error is appended to the task output under a `--- STDERR ---` marker. The agent also sets `OPCON_WORK_ITEM_ID` and `OPCON_CORRELATION_ID` environment variables that the command can read.

Configuration reference

Parameter	Type	Required	Default	Limits	Notes
<code>command</code>	string	Yes	—	1–65536 chars	The command line to run.
<code>arguments</code>	list of strings	No	none	≤100 items, each ≤8192 chars	Appended to the command.
<code>workingDirectory</code>	string	No	agent working directory	≤4096 chars	Directory to run in.
<code>environment</code>	name→value map	No	none	each value ≤32768 chars	Extra environment variables for the command.
<code>shell</code>	boolean	No	<code>true</code>	—	Run through the system shell (pipes/redirects) vs. direct process.
<code>failOnError</code>	boolean	No	<code>true</code>	—	<code>true</code> fails the task on a non-zero exit code; <code>false</code> succeeds regardless.

The task timeout is set on the job, not here. When the timeout is reached the agent stops the command (graceful stop, then a forced stop after 5 seconds).

Outcomes

Result	Exit code	When
Success	0	Command exited 0.
Success despite non-zero exit	the command's code	Command exited non-zero and <code>failOnError</code> is <code>false</code> .
Failed — command error	the command's code	Command exited non-zero and <code>failOnError</code> is <code>true</code> .
Failed — invalid parameters	1	A parameter fails validation (e.g. empty <code>command</code>).
Failed — could not start	1	The command or program was not found or could not start.
Stopped — timeout or cancel	128	The process was terminated by a signal (timeout or cancellation).

Environment handling

The command inherits the agent's environment **minus sensitive variables**, which are filtered out before the command runs. Filtered variables are:

- any variable whose name contains `SECRET` or `PASSWORD`, and
- these known credential variables: `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, `AWS_SESSION_TOKEN`, `DATABASE_URL`, `DB_PASSWORD`, `API_KEY`, `SECRET_KEY`, `PRIVATE_KEY`, `OPCON_CLIENT_SECRET`.

Variables the Builder sets in the task's **Environment Variables**, plus the injected `OPCON_WORK_ITEM_ID` and `OPCON_CORRELATION_ID`, are always passed. This filtering is by design — a command cannot read the agent's stored secrets through its environment.

Troubleshooting

Symptom	Likely cause	Resolution
Task fails immediately with "Invalid task definition"	<code>command</code> empty, or a value over its limit	Correct the parameters (Builder).
Command works in a terminal but the task reports it can't start	Executable not on the agent's <code>PATH</code> , or wrong working directory	Confirm the command exists on the agent ; set Working Directory ; for agent setup escalate to the Administrator .
Pipes, redirects, or shell built-ins don't work	Run via Shell is off	Turn Run via Shell on (Builder).
Command can't see an environment variable (often a password/secret)	The variable was filtered by design	Pass it as a task Environment Variable , or use a connection-based connector for credentials (Builder / Administrator).
Task fails on a non-zero exit code that you consider acceptable	Fail on Error is on	Turn Fail on Error off (Builder).
Task ends at the timeout with exit code 128	The command ran past the job's timeout and was stopped	Increase the job timeout, or fix the command so it completes in time (Builder).
Task assigned but never runs	Target agent is not a Universal Agent	Reassign to a Universal Agent.

Contact support when

- A command that exists on the agent and runs manually still fails to start as a task, with a permission or spawn error.
- Output is captured incorrectly, or exit codes don't match what the command actually returns.

Include the task output, the parameter values (`command`, `arguments`, `shell`, `workingDirectory`), and the agent OS.

Tags: [Reference](#) [Builder](#) [Task types](#) [LMS-Training](#)

*Last updated on **Aug 3, 2026***

Last updated: 2026-06-24

Run Script task

Behavior-level reference for diagnosing Run Script tasks.

Task walkthrough: [Run a script](#). This page is the full configuration and troubleshooting reference.

The **Run Script** task runs an embedded script — PowerShell, Bash, Python, or Shell — on the agent machine. It's a **built-in** task type (`builtin:script`), so no setup or connection is required. Use it instead of Run Command when the logic is more than a single command line.

Property	Value
Task type	Run Script (<code>run-script</code>)
Runs on	Universal Agent only
Operating systems	Windows, Linux, macOS
Connections required	None
Category	Script

How it works

The script content is written to a temporary file on the agent, run with the matching interpreter, and the temporary file is removed afterward. The interpreter **must be installed on the agent** — Continuum does not provide it.

Script type	Interpreter used	Notes
powershell	powershell.exe on Windows, pwsh on Linux/macOS	Run with <code>-NoProfile -NonInteractive - ExecutionPolicy Bypass</code> . pwsh (PowerShell 7+) must be installed on non-Windows agents.
bash	/bin/bash	Bash must be present.
python	python on Windows, python3 on Linux/macOS	Python must be installed and on <code>PATH</code> .
shell	cmd.exe on Windows, /bin/sh on Linux/macOS	Default. Always present on the matching OS.

Standard output and standard error are captured; standard error is appended under a `--- STDERR ---` marker. The agent sets `OPCON_WORK_ITEM_ID` and `OPCON_CORRELATION_ID` for the script, and filters sensitive environment variables exactly as [Run Command](#) does.

Configuration reference

Parameter	Type	Required	Default	Limits	Notes
<code>scriptType</code>	string	Yes	<code>shell</code>	one of <code>powershell</code> , <code>bash</code> , <code>python</code> , <code>shell</code>	Selects the interpreter.
<code>script</code>	string	Yes	—	1–1,048,576 chars (1 MB)	The script content.
<code>arguments</code>	list of strings	No	none	≤100 items, each ≤8192 chars	Passed to the script.
<code>workingDirectory</code>	string	No	agent working directory	≤4096 chars	Directory to run in.
<code>environment</code>	name→value map	No	none	each value ≤32768 chars	Extra environment variables.
<code>failOnError</code>	boolean	No	<code>true</code>	—	<code>true</code> fails the task on a non-zero exit code; <code>false</code> succeeds regardless.

The task timeout is set on the job, not here. When the timeout is reached the agent stops the script (graceful stop, then a forced stop after 5 seconds).

Outcomes

Result	Exit code	When
Success	0	Script exited 0.
Success despite non-zero exit	the script's code	Script exited non-zero and <code>failOnError</code> is <code>false</code> .
Failed — script error	the script's code	Script exited non-zero and <code>failOnError</code> is <code>true</code> .
Failed — invalid parameters	1	A parameter fails validation (e.g. empty <code>script</code> , unknown <code>scriptType</code>).
Failed — interpreter missing	1	The interpreter for the chosen script type is not installed on the agent.
Stopped — timeout or cancel	128	The process was terminated by a signal (timeout or cancellation).

Troubleshooting

Symptom	Likely cause	Resolution
Task fails immediately with "Invalid task definition"	<code>script</code> empty, <code>scriptType</code> not one of the four values, or a value over its limit	Correct the parameters (Builder).
Task fails to start a PowerShell script on Linux/macOS	<code>pwsh</code> (PowerShell 7+) is not installed on the agent	Install <code>pwsh</code> on the agent, or use a different script type (Administrator / Builder).
Python script won't start	<code>python</code> / <code>python3</code> not installed or not on <code>PATH</code> on the agent	Install Python on the agent and confirm it's on <code>PATH</code> (Administrator).
PowerShell script behaves differently than in an interactive console	The script runs with <code>-NoProfile</code> and <code>-ExecutionPolicy Bypass</code>	Don't rely on profile setup or local execution policy; make the script self-contained (Builder).
Script can't see an environment variable (often a password/secret)	Filtered by design (same rule as Run Command)	Pass it as a task Environment Variable , or use a connection-based connector for credentials (Builder / Administrator).
Task fails on a non-zero exit you consider acceptable	Fail on Error is on	Turn Fail on Error off (Builder).
Task ends at the timeout with exit code 128	The script ran past the job's timeout and was stopped	Increase the job timeout, or make the script finish sooner (Builder).
Task assigned but never runs	Target agent is not a Universal Agent	Reassign to a Universal Agent.

Contact support when

- A script fails to start even though its interpreter is installed and runs the same script manually on the agent.
- Output or exit codes don't match what the script actually produces.

Include the task output, the `scriptType`, the agent OS, and confirmation that the interpreter is installed on the agent.

Tags:

[Reference](#)

[Builder](#)

[Task types](#)

[LMS-Training](#)

*Last updated on **Aug 3, 2026***

Wait for File task

Behavior-level reference for diagnosing Wait for File tasks.

Task walkthrough: [Wait for a file before your workflow continues](#). This page is the full configuration and troubleshooting reference.

The **Wait for File** task pauses a workflow until a file appears at a given path and meets optional size, age, and stability criteria. It can delete the file once detected. It is a **builtin** task type (`builtin:file-wait`), so no plugin download is required.

Property	Value
Task type	Wait for File (<code>file-wait</code>)
Runs on	Universal Agent only
Operating systems	Windows, Linux, macOS
Connections required	None
Category	Utility

How it works

When the task starts, the agent polls the file path on a fixed interval until either **all configured criteria are met** (success) or the **task timeout** is reached.

On each check, criteria are evaluated in this order. The first unmet criterion ends the check and the agent waits one interval before rechecking:

- Exists and is a file** — if the path is missing, or exists but is a directory, keep waiting.
- Minimum size** — if `minSizeBytes` is set and the file is smaller, keep waiting.
- Maximum age** — if `maxAgeSeconds` is set and the file was last modified longer ago than that, keep waiting.

4. **Stability** — if `stableForSeconds` is set, the size must stay unchanged for that long.
Any size change restarts the stability timer.

When every criterion is satisfied, the task records the file size and modified time, optionally deletes the file, and finishes successfully.

Configuration reference

Parameter	Type	Required	Default	Range	Notes
<code>filePath</code>	string	Yes	—	1–4096 chars	Must resolve inside an allowed directory (see Path restrictions).
<code>checkIntervalSeconds</code>	number	No	<code>10</code>	1–3600	How often the agent rechecks the file.
<code>minSizeBytes</code>	number	No	<i>unset</i>	≥ 0	Wait until the file is at least this many bytes.
<code>maxAgeSeconds</code>	number	No	<i>unset</i>	≥ 1	File must have been modified within this many seconds (i.e. be recent).
<code>stableForSeconds</code>	number	No	<i>unset</i>	1–300	Wait until the size is unchanged for this long. Use for files still being written.
<code>deleteAfterDetection</code>	boolean	No	<code>false</code>	—	Delete the file after all

Parameter	Type	Required	Default	Range	Notes
					criteria are met.
<code>failOnTimeout</code>	boolean	No	<code>true</code>	—	If the timeout is reached: <code>true</code> fails the task, <code>false</code> finishes it successfully.

The task timeout is not set here. Wait for File honors the **task's own timeout** (configured on the job, not in these parameters). `failOnTimeout` only decides what happens *when* that timeout is reached.

Outcomes

Result	Exit code	When
Success	0	All criteria met before the timeout.
Success on timeout	0	Timeout reached and <code>failOnTimeout</code> is <code>false</code> .
Failed — timeout	1	Timeout reached and <code>failOnTimeout</code> is <code>true</code> .
Failed — invalid parameters	1	A parameter fails validation (e.g. <code>filePath</code> empty, value out of range).
Failed — path not allowed	1	<code>filePath</code> resolves outside the allowed directories. Fails immediately, without waiting.
Failed — cancelled	1	The task is cancelled while waiting.

The task output is a running log of each check ("File not found, checking again in 10s", "File size 512 < minimum 1024", "File found and meets all criteria", etc.), which is the fastest way to see *which* criterion a stuck task is waiting on.

Path restrictions

`filePath` must resolve inside a directory the agent allows; anything else fails immediately rather than waiting. By default the allowed directories are:

- `/tmp`
- `/var/tmp`
- the operating system temp directory
- the agent's working directory

An administrator can override the list with the `OPCON_ALLOWED_FILE_PATHS` environment variable on the agent (comma-separated paths). Setting it **replaces** the defaults — it does not add to them.

Troubleshooting

Symptom	Likely cause	Resolution
Task fails instantly with a "path is outside allowed directories" error	<code>filePath</code> is not under an allowed directory	Move the file into an allowed directory, or have an admin add the location to <code>OPCON_ALLOWED_FILE_PATHS</code> on the agent and restart it.
Task waits forever, then times out — file is clearly present	<code>maxAgeSeconds</code> set too low; an existing-but-old file never satisfies the age check	Raise or remove <code>maxAgeSeconds</code> . Use it only when you specifically need a <i>freshly written</i> file.
Task never succeeds on a file that keeps growing	<code>stableForSeconds</code> set, but the file is still being written, so the stability timer keeps resetting	Confirm the producing process has finished; raise <code>checkIntervalSeconds</code> / <code>stableForSeconds</code> to span the write, or remove the producer's lock/partial writes.
Task succeeds but downstream job gets an empty/partial file	No <code>minSizeBytes</code> or <code>stableForSeconds</code> , so the task matched the file mid-write	Add <code>minSizeBytes</code> and/or <code>stableForSeconds</code> so the file is complete before the task finishes.
File arrives but is never picked up; output says "Path exists but is not a file"	A directory exists at <code>filePath</code>	Correct the path to point at the file, not its folder.

Symptom	Likely cause	Resolution
Task finished OK but the file is gone	<code>deleteAfterDetection</code> is <code>true</code>	Expected. Disable it if a later job needs the file.
Expected a failure on a missing file, but the task finished OK	<code>failOnTimeout</code> is <code>false</code>	Set <code>failOnTimeout</code> to <code>true</code> if a missing file should fail the workflow.
Task assigned but never runs	Target agent is not a Universal Agent	Wait for File runs only on Universal Agents. Reassign to a Universal Agent.

Contact support when

- The task fails with an error that is **not** one of the cases above (e.g. a permission or I/O error on a path that is allowed and present).
- The file demonstrably meets every configured criterion within the timeout but the task still times out.

Include the task output log, the parameter values, the agent OS, and the agent's effective `OPCON_ALLOWED_FILE_PATHS` value.

Tags: [Reference](#) [Builder](#) [Task types](#) [LMS-Training](#)

*Last updated on **Aug 3, 2026***

Last updated: 2026-06-24

SQL Database Executor connector

Behavior-level reference for diagnosing SQL Database tasks. This connector reaches out to external databases, so expect connection, certificate, authentication, and driver issues.

Task walkthrough: [Run SQL against your databases](#). This page is the full configuration and troubleshooting reference.

The **SQL Database Executor** runs SQL scripts and database operations against SQL Server, MySQL, Oracle, and ODBC-compatible databases. It is a **downloadable** connector (`sql-executor`, category Database): the platform delivers it to the Universal Agent the first time it's needed — there's nothing for the customer to install on the agent. Every task type requires a **connection** (endpoint + credentials), which an Administrator configures.

Property	Value
Connector ID	<code>sql-executor</code>
Runs on	Universal Agent (downloaded on first use; runs as its own process on the agent)
Connection required	Yes — one database connection per task
Task types	6 (see below)

How it works

The agent runs the connector against the database named in the task's **connection**. Credentials come from that connection — they are delivered to the connector securely at run time and never appear in the job definition or the task output. The connector connects, runs the script or operation, captures the results, and reports success or failure.

- **Query results** are returned in the task output, formatted as a table. Output is capped at **1000 rows** (a `(... more rows truncated)` note is added); individual values are capped at 50 characters. Large result sets are for verification, not bulk data export.
- **Output to file** — `outputFilePath` (where supported) writes the results to a file on the agent; the path is validated against the agent working directory.
- **Query timeout** — `timeoutSeconds` (default **300**) bounds how long a single operation may run, independent of the job timeout.

Task types

Task type	Name	Connection	Operating systems	Purpose
MS SQL Script	mssql-script	SQL Server	Windows, Linux, macOS	Run T-SQL scripts
MS SQL Job	mssql-job	SQL Server	Windows, Linux	Run/monitor a SQL Server Agent job
MS SQL DTEXec (SSIS)	mssql-dtexec	SQL Server	Windows only	Run an SSIS package
MySQL Script	mysql-script	MySQL	Windows, Linux, macOS	Run MySQL scripts
Oracle Script	oracle-script	Oracle	Windows, Linux	Run SQL/PL-SQL scripts
Other DB Script (ODBC)	otherdb-script	ODBC	Windows, Linux	Run SQL via an ODBC/OleDB connection

Connections (Administrator-configured)

Connection	Required fields	Optional fields	Notes
SQL Server (database-sqlserver)	server, database	port (default 1433), username, password, useWindowsAuth (default false), trustServerCertificate (default false)	Connections are TLS-encrypted . A self-signed/internal CA certificate fails unless trustServerCertificate is on. useWindowsAuth uses NTLM.
MySQL (database-mysql)	host, database, username	port (default 3306), password	
Oracle (database-oracle)	connectionString	username, password	The connection string carries the host/service details.
ODBC (database-odbc)	connectionString	username, password	The target ODBC driver/DSN must exist on the agent.

Common script-task parameters

These apply to **MS SQL Script**, **MySQL Script**, **Oracle Script**, and **Other DB (ODBC)**:

Parameter	Type	Required	Default	Limits	Notes
<code>script</code>	string	one of <code>script</code> / <code>scriptFile</code>	—	1–1,048,576 chars	Inline SQL.
<code>scriptFile</code>	string	one of <code>script</code> / <code>scriptFile</code>	—	—	Path to a <code>.sql</code> file on the agent (validated against the working directory).
<code>outputFilePath</code>	string	No	—	—	Write results to this file (not available for ODBC).
<code>variables</code> / <code>bindVariables</code>	name→value map	No	—	—	Substituted into the script. MS SQL/MySQL/ODBC use SQLCMD-style <code>\$(Name)</code> . Oracle uses <code>bindVariables</code> .
<code>timeoutSeconds</code>	integer	No	<code>300</code>	1–86400	Per-query timeout.
<code>failOnError</code>	boolean	No	<code>true</code>	—	<code>false</code> reports a warning and still succeeds .

Behavior notes:

- Exactly **one** of `script` or `scriptFile` must be provided.
- For SQL Server, scripts are split on `GO` (on its own line) and run as separate batches.
- Variable substitution is plain text replacement**, not parameterized queries — it's meant for trusted config values (environment names, paths), not untrusted data.

MS SQL Job (mssql-job)

Parameter	Type	Required	Default	Notes
jobName	string	Yes	—	Name of the SQL Server Agent job.
monitorMode	enum	No	until-completion	until-completion, during-execution, or with-end-time.
endTime	string HH:MM	when with-end-time	—	Stop monitoring at this time.
pollingIntervalSeconds	integer	No	30	5–300. How often job status is checked.
maxMonitorHours	integer	No	24	1–168. Upper bound on monitoring before timing out.

MS SQL DTEXec / SSIS (mssql-dtexec)

Parameter	Type	Required	Default	Notes
packageSource	enum	Yes	file	file, sql, or ssis-catalog.
packagePath	string	for file/sql	—	Package file path or SQL Server location.
projectName, folderName	string	for ssis-catalog	—	SSIS Catalog project/folder.
parameters	name→value map	No	—	SSIS package parameters.
configurationFile	string	No	—	SSIS configuration file path.
use32BitRuntime	boolean	No	false	Run the 32-bit DTEXec runtime.

SSIS runs only on **Windows** agents and depends on SSIS tooling being present on that agent.

Outcomes

Result	Exit code	When
Success	0	The script/operation completed.
Completed with warning	0	An error occurred but <code>failOnError</code> is <code>false</code> — the output carries a <code>WARNING:</code> note.
Failed	1	An error occurred and <code>failOnError</code> is <code>true</code> (connection, auth, SQL error, missing file, etc.).

Troubleshooting

Symptom	Likely cause	Resolution
"Required connection type ... not found"	No connection (or the wrong type) is attached to the task	Attach the matching connection (SQL Server/MySQL/Oracle/ODBC) to the task (Builder).
SQL Server connection fails on a certificate/TLS error	Server uses a self-signed or internal-CA certificate; <code>trustServerCertificate</code> is off	Enable Trust Server Certificate on the connection, or install a trusted cert (Administrator).
SQL Server "login failed"	Wrong username/password, or SQL vs. Windows auth mismatch	Verify credentials; set Windows Authentication correctly on the connection (Administrator).
ODBC task fails to connect	The ODBC driver/DSN named in the connection string isn't installed on the agent	Install the ODBC driver/DSN on the agent (Administrator) .
Oracle task fails to connect	Connection string host/service wrong, or Oracle client prerequisites missing on the agent	Verify the connection string; confirm agent prerequisites (Administrator).
"Script file not found" / "Permission denied reading script file"	<code>scriptFile</code> path is wrong or unreadable by the agent	Correct the path (relative to the agent working directory) and file permissions (Builder/Administrator).
SSIS task never runs / can't be assigned	Assigned to a non-Windows agent, or SSIS tooling absent	Target a Windows agent that has SSIS tooling (Administrator/Builder).
Task fails at ~5 minutes on a long query	<code>timeoutSeconds</code> default is 300	Raise <code>timeoutSeconds</code> (max 86400) for long operations (Builder).
Result set looks cut off	Output is capped at 1000 rows	Expected — use <code>outputFilePath</code> or the database's own export for bulk data (Builder).
A SQL error didn't fail the task	<code>failOnError</code> is <code>false</code>	Turn Fail on Error on if the SQL error should fail the workflow (Builder).

Symptom	Likely cause	Resolution
<code>\$(Variable)</code> left un-substituted in the script	Variable not supplied, or name typo; Oracle uses <code>bindVariables</code> not <code>variables</code>	Supply the variable; for Oracle use bind variables (Builder) .

Contact support when

- A connection succeeds manually (e.g. from the agent host with the same credentials) but the task still fails to connect.
- A script runs correctly in a database client but returns different results or row counts through the connector.
- SSIS/DTEXec behavior can't be explained by the package, agent tooling, or parameters.

Include the task type, the connection type (never the credentials), the task output/error, the agent OS, and whether the same operation works directly against the database.

Tags: [Reference](#) [Builder](#) [Connectors](#) [Database](#) [LMS-Training](#)

Last updated on **Aug 3, 2026**

Last updated: 2026-07-31

Legacy LSAM connectors (UNIX/Linux and Windows)

Behavior-level reference.

Task walkthrough: [Run commands on legacy LSAM machines](#). This page is the full configuration and troubleshooting reference.

These connectors run commands on **existing legacy LSAM agents** — the agents many customers already run before moving to the Universal Agent. They let a Continuum workflow drive UNIX/Linux and Windows LSAM machines without rebuilding that automation.

Connector	Task type	Agent type	Purpose
UNIX/Linux LSAM (<code>opcon.lsam.unix</code>)	UNIX Command (<code>unix-command</code>)	UNIX LSAM	Run a command or script on a UNIX/Linux LSAM machine
Windows LSAM (<code>opcon.lsam.windows</code>)	Windows Command (<code>windows-command</code>)	Windows LSAM	Run a command on a Windows LSAM machine
Windows LSAM (<code>opcon.lsam.windows</code>)	Windows - WS_FTP Pro (<code>windows-wsftp-pro</code>)	Windows LSAM	Run a WS_FTP Pro file transfer on a Windows LSAM machine

How it works

Unlike built-in tasks, these don't run on a Universal Agent. The platform dispatches the job through the **relay**, which forwards it to the legacy LSAM

machine; the LSAM runs the work and reports completion back. Two consequences matter:

- A working **relay** and reachable **LSAM machines** are required. "No agent available" for an LSAM command points at the relay or LSAM connectivity, not the Universal Agent.
- The job is **routed by** `machineName`, which must match the LSAM agent name registered with the relay. A mismatch means the job can't be placed.
- **Legacy agents are now registered in-product**, on the **Agents** page, on its **Legacy Agents & Groups** tab (a sibling of the **Agent Pools** tab): register, edit, or delete a legacy agent there. The relay **hot-reconciles** its LSAM connections within one heartbeat of a change — no relay restart needed. Agent names are **unique per relay** (a duplicate is rejected).
- A legacy agent's status can be `ONLINE`, `OFFLINE`, or `UNKNOWN`. `UNKNOWN` (amber) means the **relay** went stale, so the agent's reachability can't be confirmed — distinct from `OFFLINE` (the agent itself is down). On relay recovery, agents leave `UNKNOWN` within a heartbeat. See [Agents and pools](#).

Shared parameters

Parameter	Required	Limits	Notes
<code>machineName</code>	Yes	1–24 chars	Target LSAM machine; must match the agent name registered with the relay.
<code>startImage</code>	Yes	1–4009 chars (Windows: 1–4000)	Path to the program or script to run.
<code>parameters</code>	No	≤2005 chars (Windows: ≤2000)	Command-line arguments, appended to <code>startImage</code> with a space.
<code>preRunImage</code>	No	≤2005 chars (Windows: ≤4000)	A prerun command run before the main job.
<code>exitCriteria</code>	No	≤5 conditions	Exit-code conditions that mark the job failed . Each is an operator (EQ, NE, LT, GT, LE, GE) and a value; if any condition matches the exit code, the job fails.

Windows length limits reconciled to Classic (2026-07-29 release). The Windows job's `startImage`/`parameters`/`workingDir`/`preRunWorkingDir`/`preRunImage` max-lengths were aligned to the legacy Windows job limits (command line 4000, directories 255). The UNIX limits are unchanged.

UNIX/Linux LSAM — additional parameters

Parameter	Required	Limits	Notes
<code>user</code>	Yes	1–31 chars	UNIX user name or UID to run the job as.
<code>groupId</code>	Yes	1–31 chars	UNIX group name or GID.
<code>niceValue</code>	No	-20 to 19	Process priority (-20 highest, 19 lowest).
<code>signalCriteria</code>	No	≤5 conditions	Signal conditions that mark the job failed (same operator/value form as <code>exitCriteria</code>).
<code>coreDumpFlag</code>	No	Y / N	Whether a core dump fails the job.
<code>jobSubType</code>	No	E or F	E = embedded script, F = file watcher.
<code>environmentVariables</code>	No	—	Environment variables to set for the job.

Windows LSAM — additional parameters

The Windows LSAM job was brought to functional parity with the legacy basic Windows job as of the 2026-07-29 release. In addition to the shared parameters above:

Parameter	Required	Limits	Notes
user	Yes	fixed	Only Use Service Account is accepted (a single-option list) – the job runs under the LSAM's own service account. A missing/blank value defaults to Use Service Account; any other value is rejected at dispatch (RC_NO_RUNAS).
workingDir	No	≤255 chars	Working directory for the job.
preRunWorkingDir	No	≤255 chars	Working directory for the prerun command.
runInCommandShell	No	boolean (default off)	Run the command inside a command shell (relay emits FC 3030 only when enabled).
environmentVariables	No	—	Environment variables (Name=Value) to set for the job. Now active – the relay emits FC 3036, mirroring the UNIX path.
outputParsing	No	≤5 rules	Scan the job output for text and set an exit code when it matches. Each rule is a searchOperation (Contains / Does Not

Parameter	Required	Limits	Notes
			Contain), a <code>stringToFind</code> (1–255 chars), and an <code>exitCodeToSet</code> (relay FCs 3032/3033/3034).
<code>customLogFilePath</code>	No	≤255 chars	Log file that output parsing scans. Defaults to the job's standard output (relay FC 3035).

Windows LSAM limitations – read before troubleshooting. On Windows LSAM today:

- Jobs run **only** under the LSAM service account (no arbitrary run-as user yet).
- **Embedded scripts are not supported** — the `jobSubType` field is not available on the Windows job; use `startImage` + `parameters` or a script already on the machine.
- Windows returns a single integer exit code, so `signalCriteria` and `coreDumpFlag` do not apply.
- `environmentVariables` **are now active on Windows**. Previously they were declared but not emitted.

Windows LSAM — WS_FTP Pro file transfer

As of the 2026-07-31 release there is a second Windows LSAM task type, **Windows - WS_FTP Pro** (`windows-wsftp-pro`), registered alongside `windows-command` on the same `opcon.lsam.windows` plugin (Windows LSAM agents only). It runs a **WS_FTP Pro file transfer** on the machine.

WS_FTP Pro has **no dedicated LSAM field codes**. The transfer runs as an ordinary Windows **Run Program** job: the relay assembles the form fields into a `wsftppro.exe` command line (FC 3003) plus a working directory (FC 3004).

There is **no** `startImage/parameters` on this task type — the command line is built entirely from the fields below.

Authentication is delegated to WS_FTP Pro *site profiles* configured on the agent. The job references a source and a destination profile **by name only** and carries **no credentials**; the named profiles must already exist in the WS_FTP Pro installation on that machine.

Parameter	Required	Limits	Notes
<code>machineName</code>	Yes	1–24 chars	Target Windows LSAM machine; must match the registered agent name.
<code>user</code>	Yes	fixed	Only <code>Use Service Account</code> (single-option list; a missing/blank value defaults to it).
<code>wsFTPProLocation</code>	Yes	1–253 chars, no <code>"</code>	Folder containing <code>wsftppro.exe</code> . Capped at 253 so the quote-wrapped working directory (FC 3004) stays within its 255 cap.
<code>sourceProfile</code>	Yes	no <code>"</code> or <code>:</code>	WS_FTP Pro site profile name for the source.
<code>sourceFile</code>	Yes	no <code>"</code>	Source file to transfer.
<code>destinationProfile</code>	Yes	no <code>"</code> or <code>:</code>	WS_FTP Pro site profile name for the destination.
<code>destinationFile</code>	Yes	no <code>"</code>	Destination file to write.
<code>fileTransferOptions</code>	No	free text	Extra <code>wsftppro.exe</code> command-line switches (e.g. <code>-binary -delete</code>); appended verbatim, only when non-empty.

Validation happens at the relay, not the form. Plugin task-schema patterns and lengths are **UI render hints only** — a job is not checked against them when you save it, so a job with a bad value **saves cleanly and then fails when it runs**. The relay is the enforcement point and rejects: a missing required field (`RC_MISSING_WSFTP_FIELD`); a double quote in any of the five required text fields, or a colon in either profile field (`RC_INVALID_WSFTP_FIELD` — `:` delimits profile from file in the assembled command line); a command line over 4000 chars (`RC_CMDLINE_TOO_LONG`); or a working directory over 255 chars (`RC_WORKDIR_TOO_LONG`). A trailing backslash in `sourceFile`/`destinationFile` (e.g. `C:\archive\`) is preserved losslessly — the relay doubles it so it can't escape the closing quote and swallow later options.

Outcomes

Result	When
Success	The job ran and no <code>exitCriteria</code> (or, on UNIX, <code>signalCriteria</code> /core-dump) condition marked it failed.
Failed — by criteria	An exit code (or UNIX signal/core dump) matched a failure condition.
Failed — dispatch rejected	The job couldn't be dispatched (e.g. Windows <code>user</code> is not <code>Use Service Account</code> , or <code>machineName</code> doesn't match a registered LSAM agent).

Troubleshooting

Symptom	Likely cause	Resolution
"No agent available" for an LSAM command	The relay is down, or the LSAM machine is unreachable	Check the relay and LSAM connectivity (Administrator) — this is not a Universal Agent issue.
Legacy agent shows UNKNOWN (amber)	Its relay is stale/down — reachability can't be determined (not the same as the agent being down)	Check the relay; agents leave UNKNOWN within a heartbeat once it recovers (Administrator).
Newly registered legacy agent isn't picked up	Relay hot-reconcile hasn't run yet, or registration failed (e.g. duplicate name per relay)	Wait one heartbeat; confirm the agent was registered on the Legacy Agents page with a name unique to that relay (Administrator).
Job can't be placed / unknown machine	machineName doesn't match the LSAM agent name registered with the relay	Correct machineName to the registered agent name (Builder/Administrator).
Windows job rejected at dispatch	user is set to something other than Use Service Account	Set user to Use Service Account for Windows LSAM (the only accepted value) (Builder).
Windows job ignores an embedded script	Embedded scripts (jobSubType) are not supported on Windows LSAM	Use startImage + parameters, or a script already on the machine (Builder).
WS_FTP Pro job rejected	A required field is blank, a required text field contains	Fill every required field; remove " from all five text

Symptom	Likely cause	Resolution
at dispatch	a <code>"</code> , a profile field contains a <code>:</code> , or the assembled command line/working directory is too long	fields and <code>:</code> from the profile names; shorten the location/options (Builder). The form won't catch these — they surface only at dispatch.
WS_FTP Pro transfer fails to authenticate	The referenced source/destination site profile doesn't exist on the agent, or its stored credentials are wrong	Create/repair the named profiles in the WS_FTP Pro installation on that machine (Administrator); the job carries no credentials of its own.
Windows job ignores environment variables	Environment variables for Windows LSAM are supported as of the 2026-07-29 release	Confirm the environment is running a build that includes this change; env vars are emitted as FC 3036 (Builder/Administrator).
Job ran but is marked failed	An <code>exitCriteria</code> (or UNIX <code>signalCriteria/core-dump</code>) condition matched	Confirm the criteria match the program's real success/failure codes (Builder).
UNIX job fails with a permissions error	<code>user</code> / <code>groupId</code> lacks rights to run <code>startImage</code>	Use a user/group with the right permissions on the LSAM machine (Builder/Administrator).

Contact support when

- The relay and LSAM machine are confirmed healthy and `machineName` is correct, but jobs still can't be dispatched or never report completion.

- Exit/signal handling doesn't match what the LSAM machine actually returned.

Include the connector (UNIX/Windows), `machineName`, `startImage` / `parameters`, the configured `exitCriteria`, and the relay/LSAM status.

Tags: [Reference](#) [Builder](#) [Connectors](#) [Legacy](#) [LMS-Training](#)

*Last updated on **Aug 3, 2026***

Last updated: 2026-06-24

Null Job task

Behavior-level reference.

Task walkthrough: [Use a Null Job](#). This page is the full configuration and troubleshooting reference.

The **Null Job** is a placeholder that does no work and finishes successfully the moment it runs. It's an **internal** task type (`internal:null-job`) — it runs inside the platform, not on an agent, so it needs no agent and no connection. It's a workflow-design building block, not a no-op to dismiss.

Property	Value
Task type	Null Job (<code>null-job</code>)
Runs on	The platform — no agent
Connections required	None
Parameters	None
Category	Internal

What it's for

- **Dependency aggregation** — give many jobs a single common successor (many-to-one) or a single common predecessor, so other dependencies can hang off one clean point instead of wiring every job to every other.
- **Running events** — provide a job to attach events to without performing real work.
- **Building a workflow on hold** — act as a controllable placeholder when a workflow is built in an on-hold state.

How it works

When the Null Job runs, the platform completes it immediately — start and end time are set to the same moment — and it finishes **OK**. There are no parameters to configure and nothing to run.

Outcomes

Result	When
Finishes OK (immediately)	Always, as soon as the job runs.

A Null Job has no failure path of its own. If a Null Job isn't completing, the cause is upstream of the task itself (see Troubleshooting).

Troubleshooting

Symptom	Likely cause	Resolution
Null Job never runs / sits waiting	Its dependencies haven't been satisfied yet — the job is waiting, not stuck	Check the predecessor jobs it depends on (Operator).
Null Job stays on hold	The workflow (or job) was built in an on-hold state by design	Release it when ready (Operator).
"Why is this empty job here?"	It's an intentional aggregation/event/anchor point	Expected — confirm the workflow design before removing it (Builder).

Contact support when

- A Null Job whose dependencies are all satisfied, and which is not on hold, still does not complete immediately.

Include the workflow instance, the job name, and the status of the job's dependencies.

Tags:

[Reference](#)

[Builder](#)

[Task types](#)

[LMS-Training](#)

*Last updated on **Aug 3, 2026***

Last updated: 2026-06-24

Workflow Container task

Behavior-level reference.

Task walkthrough: [Embed another workflow with a Workflow Container](#). This page is the full configuration and troubleshooting reference.

The **Workflow Container** embeds another workflow as a step inside the current one. When the container job runs, the platform builds and runs the referenced workflow; the container stays running until that nested workflow finishes. It's an **internal** task type (`internal:workflow-container`) — it runs inside the platform, with no agent and no connection.

Troubleshooting reminder: when reviewing a workflow, a single job may actually be an entire child workflow. Follow the container to the nested instance.

Property	Value
Task type	Workflow Container (<code>workflow-container</code>)
Runs on	The platform — no agent
Connections required	None
Category	Internal

Configuration reference

Parameter	Type	Required	Notes
<code>workflow.name</code>	string	Yes	Name of the workflow to embed.
<code>workflow.id</code>	string	No	The workflow's identifier — optional, used for faster/exact lookup.

How it works

When the container job runs, the platform:

1. Validates that the referenced workflow exists and is deployable.
2. Checks for a **circular reference** — a workflow cannot embed itself or any workflow already in its parent chain.
3. Checks the **nesting depth** — nesting is limited (default **10** levels deep).
4. Builds the nested workflow (inheriting the parent's schedule date) and moves the container job to **running**.

The container job remains **running** until the nested workflow completes, then settles to the matching outcome.

Outcomes

Result	Status	When
Nested workflow building/running	Running	Reference is valid; the nested workflow has been built and is running.
Failed — missing reference	Failed	No <code>workflow.name</code> was provided.
Failed — workflow not found	Failed	The referenced workflow doesn't exist or isn't deployable.
Failed — circular reference	Failed	The referenced workflow is already an ancestor (the error names the cycle path).
Failed — nesting too deep	Failed	The nesting depth limit (default 10) was exceeded.

Troubleshooting

Symptom	Likely cause	Resolution
"Referenced workflow ... does not exist or is not deployable"	Wrong name, or the target workflow isn't deployed	Correct <code>workflow.name/id</code> ; deploy the target workflow (Builder).
"Circular workflow reference detected"	The embedded workflow references one of its own ancestors	Remove the cycle; an embedded workflow can't include a parent (Builder).
"Maximum nested workflow depth ... exceeded"	Containers nested more than the limit (default 10)	Flatten the design so nesting stays within the limit (Builder).
Container job stays running for a long time	The nested workflow hasn't finished	Open the nested workflow instance and troubleshoot it there (Operator).
"Workflow-container job missing workflow reference"	The job has no workflow selected	Set the workflow reference on the job (Builder).

Contact support when

- The referenced workflow exists, is deployable, isn't circular, and is within the depth limit, but the container still fails to build it.
- A container reports its nested workflow finished, but its own status doesn't settle to match.

Include the parent workflow instance, the container job name, the referenced workflow name/ID, and the nested workflow instance ID if one was created.

Tags: [Reference](#) [Builder](#) [Task types](#) [LMS-Training](#)

*Last updated on **Aug 3, 2026***

Last updated: 2026-06-24

Scheduling and build

The frequency *pattern* itself is defined in the Toolkit; this doc covers how scheduling and build **behave**. See [How workflows work](#).

Task walkthrough: [Schedule and build a workflow](#). This page is the full configuration and troubleshooting reference.

Build turns a workflow definition into a concrete day's run. **Frequencies** decide which days and times a job is eligible, and **calendars / working days** remove days that don't qualify.

Build settings (per workflow)

Setting	Notes
<code>daysInAdvanceToBuild</code>	How many days ahead the run is built.
<code>daysToBuild</code>	How many days are built at once.
<code>buildTime</code>	Time of day the build runs (HH:mm).
<code>buildOnHold</code>	When on, the run is built on hold and won't run until released.
<code>overwriteOnBuild</code>	Whether a rebuild overwrites the existing run.
<code>deleteAfterDays</code>	How long built instances are kept.
<code>useHolidayCalendar</code> / <code>additionalHolidayCalendar</code>	Apply a holiday calendar to exclude days.
<code>workingDays</code>	The days of week the workflow runs.
<code>startTime</code>	The workflow's start time (HH:mm).
<code>conflictOtherDays</code> , <code>multiInstance</code>	Conflict handling and whether multiple instances may run.

When `autoBuildSettings` is null, auto-build is disabled (the run is built manually).

Per-frequency runtime settings (per job)

Each job frequency carries its own runtime behavior:

Setting	Notes
<code>buildStatus</code>	<code>released</code> , <code>onHold</code> , <code>doNotSchedule</code> , <code>toBeSkipped</code> , or <code>disabled</code> .
<code>priority</code>	Relative scheduling priority.
<code>startTimeOffset</code> (+ <code>startTimeOffsetFormat</code> <code>absolute</code> / <code>relative</code>)	When the job is eligible to start.
<code>lateToStartOffset</code> , <code>latestStartOffset</code>	Thresholds for late-to-start and the latest allowed start.
<code>lateToFinishOffset</code> (+ format)	Threshold for late-to-finish.
<code>maxRunTime</code>	Run time after which the job is considered to have exceeded its limit.
<code>estimatedRunTime</code> (+ <code>estimationSource</code> <code>calculated</code> / <code>history</code> / <code>userDefined</code>)	Expected run time.
<code>failureRetries</code>	<code>enabled</code> , <code>maximumAttempts</code> , <code>minutesBetweenAttempts</code> .
<code>successReruns</code>	<code>none</code> , <code>recurringInstances</code> (with instance times), or <code>restartOffset</code> (interval + max runs).

These offsets drive the job-status events (LateToStart, LateToFinish, MissedLatestStartTime, ExceededMaxRuntime) covered in [Workflow events](#).

Troubleshooting

Symptom	Likely cause	Resolution
A job didn't appear in the day's run	Its frequency didn't match the date, the day is excluded by a calendar/working-days rule, or its build status is <code>doNotSchedule</code> / <code>toBeSkipped</code> / <code>disabled</code>	Check the job's frequency, calendar, and build status (Builder).
The whole workflow didn't build	Auto-build disabled (<code>autoBuildSettings</code> null), or build time not yet reached	Confirm auto-build settings or trigger a manual build (Builder / Operator).
Workflow built but nothing runs	Built on hold (<code>buildOnHold</code>)	Release it when ready — it's not stuck (Operator).
Job flagged <code>LateToStart</code> / <code>LateToFinish</code>	The job passed its configured offset	Expected signal; investigate why the job started/finished late (Operator).
Job keeps retrying or rerunning	<code>failureRetries</code> or <code>successReruns</code> configured	Confirm the retry/rerun settings match intent (Builder).
Old instances disappear	<code>deleteAfterDays</code> retention reached	Expected — adjust retention if longer history

Symptom

Likely cause

Resolution

is needed
(**Builder**).

Contact support when

- A job's frequency, calendar, working days, and build status all indicate it should build, but it does not appear in the run.

Include the workflow, the job, its frequency/calendar settings, and the build date in question.

Tags:

[Reference](#)[Builder](#)[Workflows](#)[Scheduling](#)[LMS-Training](#)

*Last updated on **Aug 3, 2026***

Last updated: 2026-06-24

Frequencies

Frequencies are **global** (not workspace-scoped).

Task walkthrough: [Define a frequency](#). This page is the full configuration and troubleshooting reference.

A **frequency** is a named scheduling rule (a **pattern**) that decides which calendar dates a job is eligible to run on. Jobs reference frequencies; the per-job runtime settings (build status, timing, retries) live with the job (see [Scheduling and build](#)).

Pattern model

A pattern is defined by a **time period** and, for most, a **type**:

- **Time period:** `daily`, `weekly`, `monthly`, `yearly`.
- **Type:** `byDate`, `byWeekday`, or `byPeriod` (daily has no type).

Pattern	Key fields
Daily	<code>interval</code> (every N days), <code>startDate</code> , <code>repeatAnnually</code>
Weekly · byWeekday	<code>interval</code> , <code>daysOfWeek</code> (e.g. Mon/Wed/Fri)
Weekly · byPeriod	<code>interval</code> , <code>anchor</code> (start/end), <code>offsetDays</code>
Monthly · byDate	<code>interval</code> , <code>dayOfMonth</code> (1–31, clamped to month length)
Monthly · byWeekday	<code>interval</code> , <code>weekOrdinal</code> (first...fifth/last), <code>daysOfWeek</code> — e.g. "second Tuesday"
Monthly · byPeriod	<code>interval</code> , <code>anchor</code> (start/middle/end), <code>offsetDays</code> (–30...30) — e.g. "last working day"
Yearly · byDate	<code>interval</code> , <code>month</code> (1–12), <code>dayOfMonth</code> , <code>nonWorkingDayAction</code> (required)
Yearly · byWeekday	<code>interval</code> , <code>weekOrdinal</code> , <code>daysOfWeek</code>
Yearly · byPeriod	<code>interval</code> , <code>anchor</code> , <code>offsetDays</code>

Shared options

Option	Values	Notes
<code>dayType</code>	<code>working</code> / <code>any</code>	Whether to count only working days.
<code>nonWorkingDayAction</code>	<code>skipToNext</code> / <code>skipToPrevious</code> / <code>keep</code> / <code>omit</code>	What to do when a match lands on a non-working day.
<code>exclusionCalendar</code>	a Calendar reference	Dates to exclude (see Calendars).

Forecast

The frequency detail page shows a **year calendar** that highlights the dates the pattern matches, calculated from the pattern for the selected year (and auto-scrolls to the first matching year). This is how a Builder validates a pattern before using it.

Things to know

- A frequency is the **pattern**; *when within the day* and *retry/rerun* behavior are the job's per-frequency runtime settings, not the frequency itself.
- **Known limitation:** the forecast calculation does not yet fully apply **exclusion calendars** — a date excluded by a calendar may still appear highlighted in the forecast even though the build honors it. You may therefore see a forecast that disagrees with the dates a workflow actually runs on.
- New frequencies default to a **daily, interval 1, starting today, any-day** pattern.

Tags: [Reference](#) [Builder](#) [Toolkit](#) [Scheduling](#) [LMS-Training](#)

*Last updated on **Aug 3, 2026***

Last updated: 2026-06-24

Calendars

Calendars are **global** (not workspace-scoped).

Task walkthrough: [Manage properties, calendars, and tags](#). This page is the full configuration and troubleshooting reference.

A **calendar** is a named set of dates — typically holidays or blackout days. Calendars are used to **exclude** days from scheduling.

Fields

Field	Notes
name	Unique calendar name.
description	Optional.
dates	A list of dates, each YYYY-MM-DD.

How calendars are used

- As an **exclusion calendar** on a [frequency](#) — matched dates that fall on a calendar date are excluded.
- As an **additional holiday calendar** on a workflow's schedule (see [Scheduling and build](#)).

Things to know

- Calendars are **shared** across workspaces.
- See the exclusion-calendar **forecast limitation** noted in [Frequencies](#): a date excluded by a calendar may still appear in a frequency's forecast even though the build honors the exclusion.

Tags: [Reference](#) [Builder](#) [Toolkit](#)

*Last updated on **Aug 3, 2026***

Last updated: 2026-07-28

Job dependencies

See [How workflows work](#).

Task walkthrough: [Order jobs with dependencies](#). This page is the full configuration and troubleshooting reference.

A job runs only when **all** of its dependencies are satisfied. Dependencies are how a workflow is sequenced and how jobs coordinate shared state and capacity.

Dependency types

Type	Satisfied when	Key settings
Job	A predecessor job reaches the required result	<code>predecessorJob</code> , <code>predecessorResult</code> (<code>success</code> / <code>failure</code> / <code>any</code>), <code>predecessorDaysOffset</code> , optional <code>predecessorFrequency</code>
Threshold	A named threshold reaches a value	<code>threshold</code> , <code>thresholdValue</code>
Resource	Enough of a named resource is available (the job reserves it)	<code>resource</code> , <code>resourceRequired</code>
Expression	A custom expression evaluates true	<code>expression</code>

How job dependencies resolve

flowchart TD

A[Job eligible to run] → B{All dependencies satisfied?}

B -- "Job dep: predecessor reached required result?" →

C{...}

B -- "Threshold dep: value reached?" → C

B -- "Resource dep: enough available?" → C

B -- "Expression dep: evaluates true?" → C

C -- All yes → D[Job runs]

C -- Any no → E[Job keeps waiting]

E → B

- **predecessorResult** matters: **success** waits for the predecessor to finish OK, **failure** waits for it to fail, and **any** proceeds on either. A dependency on **failure**/**any** is valid design (e.g. a cleanup job that runs when its predecessor fails).
- **predecessorDaysOffset** lets a job depend on a predecessor from a different day's run.
- **Resource** dependencies reserve capacity — a job waits if not enough of the resource is free.

Which predecessor statuses satisfy each result

The result condition is not a single status — each maps to the **set** of terminal statuses that release the dependent:

predecessorResult	Satisfied by	Notes
success	FINISHED_OK, MARKED_FINISHED_OK, SKIPPED, FIXED	A skipped or marked-fixed predecessor counts as success.
failure	FAILED, MARKED_FAILED, INITIALIZATION_ERROR, UNDER_REVIEW	UNDER_REVIEW is a deliberate non-terminal release point — the operator has acknowledged the failure, so recovery work starts without waiting for it to be marked Fixed or Failed.
any	The union of the two lists above	any is the ignore-error union, <i>not</i> "any outcome."

- **CANCELLED** and **MISSED_START_TIME** **satisfy nothing** under any condition — a predecessor that never produced a result cannot release its dependents. A job depending (even with **any**) on a predecessor that ends up cancelled or missing its start time will **wait forever**; that's the intended behavior, not a bug. Clear it with **Force Start / Skip** or fix the upstream job.
- ***_PENDING_TERM** statuses do **not** satisfy dependents — the predecessor must reach its final terminal status first.

Troubleshooting

Symptom	Likely cause	Resolution
Job sits waiting and never runs	A dependency isn't satisfied — identify which one	Check each dependency: predecessor result, threshold value, resource availability, expression (Operator → Builder if the dependency is mis-set).
Job waits on a predecessor that already finished OK	The dependency expects <code>failure</code> or a different result/day offset	Confirm <code>predecessorResult</code> and <code>predecessorDaysOffset</code> match intent (Builder).
Job never gets a resource	Not enough of the named resource is available; another job holds it	Check resource counts and which jobs hold it (Builder / Operator).
Job runs earlier/later than expected relative to another	Missing or extra job dependency	Review the dependency wiring (Builder).

Contact support when

- All dependencies are demonstrably satisfied but the job still does not become eligible.

Include the workflow instance, the job, and the state of each of the job's dependencies.

Tags: [Reference](#) [Builder](#) [Workflows](#) [LMS-Training](#)

*Last updated on **Aug 3, 2026***

Last updated: 2026-06-24

Thresholds and resources

Both are **workspace-scoped** (default to the General workspace).

Task walkthrough: [Use thresholds and resources](#). This page is the full configuration and troubleshooting reference.

Thresholds and **resources** coordinate flow and capacity across jobs. Jobs reference them as [dependencies](#).

Thresholds

A **threshold** is a named numeric value used to gate jobs.

Field	Notes
<code>name</code>	Unique threshold name.
<code>value</code>	The current numeric value (non-negative integer).
<code>description</code>	Optional.

A job with a **threshold dependency** waits until the threshold reaches the value the dependency requires — a simple way to gate work on a count or a signal.

Resources

A **resource** is a named capacity pool that jobs reserve from and release back.

Field	Notes
<code>name</code>	Unique resource name.
<code>total</code>	Total capacity (non-negative integer).
<code>description</code>	Optional.

A job with a **resource dependency** reserves the number of units it needs; it waits if not enough are free, and releases them when it completes. Each reservation is tracked as an allocation (per environment, workflow instance, and job) and is released on completion.

Things to know

- Use **resources** to cap concurrency against a shared system (e.g. only N jobs hitting the core at once) — a job waits in a `WAIT_RESOURCE_DEPENDENCY` status until units free up.
- A job waiting on a threshold sits in `WAIT_THRESHOLD_DEPENDENCY` until the value is reached.
- Both are **workspace-scoped** — a job can only use thresholds/resources in its workspace.

Troubleshooting

Symptom	Likely cause	Resolution
Jobs never get a resource	Not enough <code>total</code> , or units aren't being released (held by stuck jobs)	Check the resource total and which jobs hold allocations (Builder / Operator).
A threshold-gated job never runs	The threshold value isn't reaching the required level	Confirm what sets the threshold value and the dependency's required value (Builder).

Tags: [Reference](#) [Builder](#) [Toolkit](#) [LMS-Training](#)

*Last updated on **Aug 3, 2026***

Last updated: 2026-06-24

Workflow events

See [How workflows work](#).

Task walkthrough: [Run events on job outcomes](#). This page is the full configuration and troubleshooting reference.

An **event** fires a **command** in response to a job outcome — for example, sending a notification when a job fails, or triggering another action when a job finishes late. Events are configured per job; workflows can also carry schedule-level events.

Event trigger types

Trigger (<code>eventTriggerType</code>)	Fires based on	Key settings
Job status (<code>jobStatus</code>)	The job reaching a specific status	<code>jobStatus</code> (see values below)
Job exit description (<code>jobExitDescription</code>)	A comparison against the job's exit value	<code>comparisonOperator</code> , <code>comparisonValue</code> , plus <code>comparisonEndRange</code> when the operator is <code>Range</code>
Job completion expression (<code>jobCompletionExpression</code>)	A custom expression	<code>expression</code>

Every event carries a `command` to run and an optional `frequency` that scopes when it applies.

Job status values

FinishedOk · Failed · Fixed · Skipped · UnderReview · StartAttempted ·
StillAttemptingStart · LateToStart · LateToFinish ·
MissedLatestStartTime · ExceededMaxRuntime

Comparison operators (for exit-description events)

EqualTo · NotEqualTo · GreaterThan · GreaterThanOrEqualTo · LessThan ·
LessThanOrEqualTo · Contains · Range (uses comparisonValue +
comparisonEndRange)

Things to know

- An event firing is **expected behavior**, not an error — e.g. a Failed-status event that sends an alert is the system working as designed.
- If an expected event didn't fire, check the trigger configuration first: the status/exit comparison may not have matched, or the event's frequency may have excluded the run.
- Events run a **command** — what that command does is defined by the command itself; the event only decides *when* it fires.

Tags: [Reference](#) [Builder](#) [Workflows](#)

Last updated on Aug 3, 2026

Last updated: 2026-07-29

The Processes page (runtime)

For what each status/action *means*, see [Job and workflow statuses](#) and [Operator response guide](#).

Task walkthrough: [Work the Processes page](#). This page is the full configuration and troubleshooting reference.

Processes is the operator's daily-run screen: the live view of workflow instances and their jobs for the selected **environment**. It is environment-scoped (the environment picker sits above it) and breadcrumbs to a single **Processes** crumb.

Two views

A **view toggle** switches the body between two views; the choice is URL-synced, so a reload returns to the same view. **Job View** is the default.

View	Shows	Notes
Job View	Every job instance in the run	Default. All column headers sort.
Workflow View	Workflow instances (the containers)	No "Built" column. Filters by coarse status groups , not raw statuses (see below). "View in Workflow view" links from a job.

The page also hosts the **Schedule Workflow** action (schedule a workflow into the run) and a manual **refresh**.

Workflow View status-group filter

The Workflow View **Status(s)** filter offers coarse groups — **Building, Waiting, In Process, Held, Completed, Cancelled** — each expanding to its raw statuses for the query. `UNKNOWN` is intentionally ungrouped. See [Job and workflow statuses](#).

Filters and search

- **Command-menu search**: the job search field is a command menu rather than a plain text box; it opens/focuses as a menu and drives the search.
- Filter section headers use a **funnel glyph**; the frequency filter uses **switches** (not check boxes). Saved filters are available via a selector.
- **Tooltips** annotate the Processes and Advanced Search controls.

Changing page, sort, search, or a filter **clears the current row selection** (so a bulk action never acts on rows scrolled out of view).

Row actions (single job)

Each row has an action menu offering only the actions the job's **status** accepts — the status-gated set (see [Job statuses and actions](#)): Hold, Release, Cancel, **Kill**, Restart, Force Start, Skip, Mark Fixed, Under Review, **Mark Finished OK**, **Mark Failed**. Cancel applies to a job that hasn't started running; **Kill** is its running-job counterpart. A running **container** job offers **Kill** only.

Bulk actions

Both views support multi-row selection (per-row check boxes + select-all). A **Bulk actions** button appears once **2+ rows** are selected. Its menu is the **intersection of the valid actions across the selection** (an action shows only if every selected row accepts it) plus **Delete**; actions run as a batch and a confirmation message reports the result in both Job View and Workflow View.

Delete (jobs and workflow instances)

Delete is a **manual intervention** (soft-delete), not a lifecycle action, surfaced ad hoc rather than in the action set:

- **Delete a job instance** drops the dependency edge from every dependent (same- and cross-workflow) so downstream jobs re-qualify against their remaining conditions instead of hanging on a predecessor that no longer exists; dropped edges are persisted so a restart doesn't re-block them. Rejected while the job is in a pending-termination status.
- **Delete a workflow instance** soft-deletes the instance and cascades to its child jobs (parity with job delete), tearing them out of the runtime dependency graph and re-qualifying surviving cross-workflow dependents. Allowed only from a **deletable** status — `WAIT_TO_START`, `WAIT_CONTAINER_JOB`, `ON_HOLD`, `PARENT_HOLD`, `COMPLETED`, `CANCELLED` — and **blocked while actively running** (`BUILDING`, `IN_PROCESS`, `STARTED_BY_USER`, `UNKNOWN`), or when a child is running/pending-term.

Add a job to a workflow instance

An operator can add a job that is **defined on the workflow** to a **running (or resident) instance** of that workflow — the equivalent of the legacy **Add Jobs** wizard (`$JOB:ADD` / `$JOB:ADDHLD`), re-shaped to Continuum's synchronous in-process model. The trigger is an **Add Job** item in the **Workflow View** row action menu; it opens the **Add Job to Workflow Instance** dialog and posts to `POST /workflow-instances/:workflowInstanceId/jobs`.

Candidate list. Candidates come from the **live workflow definition**, not the instance's frozen config snapshot — so a job added to the definition *after* the instance was built still appears. The dialog shows a sorted radio list of every defined job, each badged with its **disposition**:

Disposition	Badge	Selectable?
addable	<i>(none)</i>	Yes — not currently in the instance
replace	Will Replace	Yes — already ran and is in a terminal status; adding replaces it
active	Job is Active	No — already present in a non-terminal status
no-frequency	No Frequency	No — the job defines no frequency, so it can't be instantiated

Multi-instance, workflow-container / sub-schedule, and pool run-on-each-agent jobs are excluded from the list and rejected — this is not supported yet. The backend re-fetches the deployment and stays authoritative on whether a job is actually addable.

Dialog inputs.

- **Frequency** — a choice is required when the job defines more than one; auto-selected when it defines exactly one.
- **Instance Properties** — a property grid pre-filled with the job's default property set. Sent to the backend **only if edited**; an untouched add keeps the deployment's (environment-transformed) property values. A property marked `encrypted` but carrying a plaintext value is rejected.
- **Add on hold** — check box. Released → the job lands at `WAIT_START_TIME` (so it runs regardless of whether its frequency qualified for the date); on hold → `ON_HOLD`.
- **Reason** — optional free text recorded on the audit history entry.

Duplicate gating. A job already active in the instance rejects; a job present only in a **terminal** status is **replaced** (the dialog warns first, then soft-deletes the prior instance and re-points its dependents onto the new job).

Dependency wiring. Same-workflow dependencies are resolved by GUID and pruned against the jobs actually materialized in the instance; **reverse-**

dependents (jobs waiting on the added job) are rewired from the live definition; an add that would create a **dependency cycle** is rejected (400).

Reopen. Adding a job to a **COMPLETED** resident instance reopens it (`COMPLETED → IN_PROCESS`) via a job recount so the new job can dispatch; if the best-effort graph sync fails, the instance is left **COMPLETED** rather than churning. The add records an **ADDED** job-history entry at the landing status (the job appears on the workflow timeline; audit parity with Classic's "Job Added").

Preconditions & scope. The add is rejected (409) when the workflow instance is **CANCELLED** or **COMPLETED-and-not-resident** (evicted); the UI disables **Add Job** for **CANCELLED**. A target instance in another **environment** is masked as not-found.

Job detail

Selecting a job opens the **job detail panel** — a **read-only** view (the inline edit affordance has been removed). Its header actions are **History** (left) and **Workflow** (right). The hover overlay on a diagram node is a separate, lighter tooltip.

Tags: [Reference](#) [Operator](#) [Processes](#) [LMS-Training](#)

*Last updated on **Aug 3, 2026***

Last updated: 2026-07-31

Job and workflow statuses

The status names below are the authoritative set.

Task walkthrough: [Job statuses and actions](#). This page is the full configuration and troubleshooting reference.

Job statuses

Held

Status	Meaning
ON_HOLD	Held; won't run until released.

Waiting

Status	Meaning
RELEASED	Released and eligible to proceed.
WAIT_START_TIME	Waiting for its scheduled start time.
WAIT_JOB_DEPENDENCY	Waiting on a predecessor job.
WAIT_THRESHOLD_DEPENDENCY	Waiting on a threshold value.
WAIT_RESOURCE_DEPENDENCY	Waiting for enough of a resource.
WAIT_EXPRESSION_DEPENDENCY	Waiting on an expression to evaluate true.
WAIT_JOB_CONFLICT	Waiting because of a conflict with another job.
WAIT_MACHINE	Waiting for an agent/machine to be available. A specific-agent job holds here while its target agent is OFFLINE on a <i>fresh</i> status feed (universal agents also hold on BUSY/DRAINING/DISABLED), and proceeds once it's ONLINE. If the feed is stale or UNKNOWN , the gate fails open (advances) so the scheduler can't deadlock on untrusted status.
WAIT_TO_START / WAIT_TO_START_FORCED	Queued to start (forced = via Force Start).
LATE_TO_START	Past its late-to-start threshold, not yet started.
JOB_TO_BE_SKIPPED	Skipped by an operator in a live workflow but not yet resolved — a deferred skip. No

Status	Meaning
	end time, and its dependents are not released yet. The engine resolves it to terminal <code>SKIPPED</code> in sequence once the job's own gates are satisfied. See the Skip note under Which actions a job status accepts .
<code>PRERUN_FAILED</code>	The prerun command failed.

Running

Status	Meaning
<code>ATTEMPT_TO_START</code> / <code>START_ATTEMPTED</code> / <code>STILL_ATTEMPTING_START</code>	Starting the job.
<code>PRERUN_ACTIVE</code>	The prerun command is running.
<code>JOB_RUNNING</code>	Running.
<code>JOB_RUNNING_TO_BE_TERMINATED</code>	Running, marked to be terminated.
<code>LATE_TO_FINISH</code>	Running past its late-to-finish threshold.

Pending-term (transitional)

`FINISHED_OK_PENDING_TERM` • `FAILED_PENDING_TERM` •
`MARKED_FINISHED_OK_PENDING_TERM` • `MARKED_FAILED_PENDING_TERM` •
`SKIPPED_PENDING_TERM` • `UNDER_REVIEW_PENDING_TERM` • `FIXED_PENDING_TERM`
 — completed, completion processing in progress; dependents not yet released.

Terminal

Status	Meaning	Category
<code>FINISHED_OK</code> / <code>MARKED_FINISHED_OK</code>	Completed OK (marked = set by a user)	Success
<code>FIXED</code>	Marked fixed; treated as resolved	Success
<code>FAILED</code> / <code>MARKED_FAILED</code>	Failed (marked = set by a user)	Failure
<code>INITIALIZATION_ERROR</code>	Failed to initialize	Failure
<code>MISSED_START_TIME</code>	Did not start within its allowed window	Failure
<code>CANCELLED</code>	Cancelled	—
<code>SKIPPED</code>	Skipped	—
<code>UNDER_REVIEW</code>	Flagged for review	—

Workflow statuses

Status	Meaning
BUILDING	Being built into a run.
WAIT_TO_START	Built, waiting to start.
WAIT_CONTAINER_JOB	Waiting on a container (nested workflow) job.
IN_PROCESS	Running.
ON_HOLD / PARENT_HOLD	Held / held because its parent is held.
STARTED_BY_USER	Started manually.
COMPLETED / CANCELLED	Terminal.
UNKNOWN	Indeterminate.

Workflow status-group filter

In **Processes → Workflow View**, the **Status(s)** filter offers coarse **groups** rather than the raw statuses above. Each group expands to its constituent statuses:

Group	Includes
Building	BUILDING
Waiting	WAIT_TO_START, WAIT_CONTAINER_JOB
In Process	IN_PROCESS, STARTED_BY_USER
Held	ON_HOLD, PARENT_HOLD
Completed	COMPLETED
Cancelled	CANCELLED

`UNKNOWN` is intentionally **ungrouped** (not user-facing). Unlike the job query, the workflow API filters by individual `status` only, so selecting a group expands to its statuses for the query.

Which actions a job status accepts

The runtime accepts **11** lifecycle actions per job status, aligned to the job-UI matrix (the UI offers only the actions a status accepts, so a user is never shown an action that would fail):

Action	Accepted when the job is...
Hold	A waiting/ <code>WAIT_*</code> state, <code>RELEASED</code> , <code>LATE_TO_START</code> , <code>WAIT_TO_START(_FORCED)</code> , or <code>PRERUN_FAILED</code> .
Release	<code>ON_HOLD</code> .
Cancel	A held, waiting, or non-running failed state — <code>ON_HOLD</code> , <code>RELEASED</code> , any <code>WAIT_*</code> , <code>LATE_TO_START</code> , <code>WAIT_TO_START(_FORCED)</code> , <code>PRERUN_FAILED</code> , <code>JOB_TO_BE_SKIPPED</code> , <code>MISSED_START_TIME</code> , <code>INITIALIZATION_ERROR</code> , <code>FAILED</code> , <code>MARKED_FAILED</code> , or <code>UNDER_REVIEW</code> . Not a running job — use Kill there.
Kill	A running state: <code>ATTEMPT_TO_START</code> , <code>START_ATTEMPTED</code> , <code>STILL_ATTEMPTING_START</code> , <code>PRERUN_ACTIVE</code> , <code>JOB_RUNNING</code> , or <code>LATE_TO_FINISH</code> . Kill is the running-job counterpart to Cancel.
Restart	A stopped/terminal state: <code>FAILED</code> , <code>MARKED_FAILED</code> , <code>INITIALIZATION_ERROR</code> , <code>CANCELLED</code> , <code>SKIPPED</code> , <code>FINISHED_OK</code> , <code>MARKED_FINISHED_OK</code> , <code>FIXED</code> , <code>UNDER_REVIEW</code> , or <code>JOB_TO_BE_SKIPPED</code> .
Force Start	A dependency/ <code>WAIT_*</code> state, <code>ON_HOLD</code> , <code>RELEASED</code> , <code>WAIT_START_TIME</code> , <code>WAIT_TO_START(_FORCED)</code> , <code>LATE_TO_START</code> , <code>PRERUN_FAILED</code> , <code>MISSED_START_TIME</code> , or <code>JOB_TO_BE_SKIPPED</code> .
Skip	A waiting/ <code>WAIT_*</code> state, <code>ON_HOLD</code> , <code>RELEASED</code> , <code>LATE_TO_START</code> , <code>PRERUN_FAILED</code> , <code>MISSED_START_TIME</code> , or a failed state (<code>FAILED</code> , <code>MARKED_FAILED</code> , <code>INITIALIZATION_ERROR</code> , <code>UNDER_REVIEW</code>).
Mark Fixed	<code>FAILED</code> , <code>MARKED_FAILED</code> , <code>INITIALIZATION_ERROR</code> , or <code>UNDER_REVIEW</code> .
Under Review	<code>FAILED</code> , <code>MARKED_FAILED</code> , or <code>INITIALIZATION_ERROR</code> .

Action	Accepted when the job is...
Mark Finished OK	Broad manual override accepted from most non-terminal states (held, waiting, starting, running, failed, under review, MISSED_START_TIME, FIXED); forces a successful outcome so dependents proceed.
Mark Failed	Broad manual override accepted from most non-terminal states plus FINISHED_OK / MARKED_FINISHED_OK / FIXED; forces a failed outcome so dependents react.

Container jobs: a running **container job** (a step whose body is a nested workflow) accepts **Kill** only — the **Mark Finished OK** / **Mark Failed** overrides are refused while it runs, because its outcome is driven entirely by the nested workflow.

Skip is deferred and in-sequence. Skipping a job in a **live** workflow no longer takes effect immediately. The job is marked JOB_TO_BE_SKIPPED (no end time, dependents still waiting), and the engine resolves it to terminal SKIPPED **in sequence** — once the job's own TIME/JOB/THRESHOLD/RESOURCE gates are satisfied, i.e. when it would otherwise have been its turn to start (SKIPPED_PENDING_TERM → SKIPPED, which stamps the end time and releases dependents). A job that has already reached that point — a FAILED or MISSED_START_TIME job past its start window — resolves promptly. The skip is written **terminal immediately** only when the workflow is already terminal or the job isn't resident in the running graph (the in-sequence sweep could never reach it); a residual mark on a workflow being **cancelled or closed** is finalized to SKIPPED as part of that action. A JOB_TO_BE_SKIPPED mark does not block a workflow **close**, and it never reopens a finished workflow.

Workflow actions. The UI offers, per workflow-instance status:

Workflow status	UI actions
WAIT_TO_START	Hold · Start
ON_HOLD	Release · Start
IN_PROCESS	Hold · Close
STARTED_BY_USER	Hold
WAIT_CONTAINER_JOB, PARENT_HOLD, BUILDING, COMPLETED, CANCELLED, UNKNOWN	(none)

- **Start** force-starts a built-but-not-yet-running instance now, ignoring its scheduled start time (moves it to `STARTED_BY_USER`, which is sweep-eligible and promotes to `IN_PROCESS`).
- **Close** force-completes an `IN_PROCESS` instance; it succeeds only once the remaining jobs are all failed/under-review, and can't be undone.
- **Cancel** is retained as an event/programmatic endpoint but is **not surfaced in the UI menu**; it's accepted on any non-terminal status.
- **Delete** (soft-delete of the instance + its child jobs) is allowed from `WAIT_TO_START`, `WAIT_CONTAINER_JOB`, `ON_HOLD`, `PARENT_HOLD`, `COMPLETED`, or `CANCELLED` — blocked while actively running (`BUILDING`, `IN_PROCESS`, `STARTED_BY_USER`, `UNKNOWN`). Delete is a manual intervention, not a lifecycle action.

Tags: [Reference](#) [Operator](#) [Processes](#) [LMS-Training](#)

*Last updated on **Aug 3, 2026***

Last updated: 2026-07-31

Operator response guide

Guides an operator through the daily run. See [How the daily run works](#) and [Job and workflow statuses](#).

Task walkthrough: [Respond to a failed job](#). This page is the full configuration and troubleshooting reference.

Situation → response

Situation	What it means	Action
Job in a <code>WAIT_*</code> dependency status	Waiting on a predecessor job, threshold, resource, or expression	Confirm the blocker; if it should proceed now, Force Start . If it shouldn't run, Skip .
Job <code>WAIT_MACHINE</code>	No agent/machine available — a specific-agent job holds here while its target agent is offline	Check agent/relay availability (Administrator); the job proceeds when the target is back online. A legacy agent showing <code>UNKNOWN</code> (amber) means its relay is stale — check the relay, not the agent.
Job <code>WAIT_START_TIME</code>	Not yet at its start time	Normal; Force Start only if it must run early.
Job in a running state (<code>JOB_RUNNING</code> , starting, <code>LATE_TO_FINISH</code>)	Actively running	To stop it, use Kill — Cancel applies only <i>before</i> a job starts running. To force its downstream outcome, Mark Finished OK or Mark Failed (a running container job accepts Kill only).
Job <code>ON_HOLD</code>	Held intentionally	Release when ready.
Job <code>LATE_TO_START</code> / <code>LATE_TO_FINISH</code>	Past a timing threshold	Investigate the delay; the job is still progressing.

Situation	What it means	Action
Job <code>FAILED</code>	The job failed	Read the job output/logs; Restart once fixed, Mark Fixed if resolved outside the system, Under Review to flag it, or Cancel .
Job <code>MISSED_START_TIME</code>	Didn't start in its window	Decide whether to Skip or rerun via the next build.
Job stuck in <code>*_PENDING_TERM</code>	Completion processing not settling	Transitional normally; if prolonged, escalate (see below).
Workflow <code>ON_HOLD</code> / <code>PARENT_HOLD</code>	Held (or parent held)	Release when ready; releasing the parent releases children.

Responding to a failed job

1. Open the job's detail and read its output/logs to find the cause.
2. Resolve the cause (often a connector/config issue — route to the **Builder** or **Administrator**).
3. Then choose: **Restart** (re-run), **Mark Fixed** (resolved outside the system), **Under Review** (flag for follow-up), or **Cancel** (abandon).

Reading the exit code

A completed job's detail now shows its real **Exit Code** (return code) and a **Termination** description in the **Completion** section — and on the **Output** tab beside the run date. The Exit Code renders as a badge: **green for 0** (success), **red for any non-zero** value. This is a **data field only** — it does **not** change the job's status classification (a job can be `FAILED` with the exit code still shown).

- Previously the return code always displayed a dash even for a job that finished OK with code 0; a real numeric code (including 0) is now persisted and shown for both universal-agent and legacy/LSAM jobs.
- A **failed** job with **no numeric code reported** by the agent shows – (not a spurious 0) — distinguish "failed with code 0" from "failed, code unknown."
- Mixed-version agents are supported: an older agent that omits the code leaves the field blank rather than breaking the report.

Which action is available

Actions are status-gated (see [Job and workflow statuses](#)). There are 11 job actions (realigned to the Classic matrix). Key points for triage:

- **Cancel** stops a job that hasn't started running; **Kill** stops one that *is* running. They don't overlap — offer **Kill**, not **Cancel**, for anything in a running status.
- **Restart** is available from any stopped state (FAILED, MARKED_FAILED, CANCELLED, SKIPPED, INITIALIZATION_ERROR, UNDER_REVIEW, JOB_TO_BE_SKIPPED, and the finished states) — not only FAILED / CANCELLED.
- **Mark Fixed** from FAILED / MARKED_FAILED / INITIALIZATION_ERROR / UNDER_REVIEW; **Release** only from ON_HOLD; **Force Start/Skip** from waiting states.
- **Skip is deferred and in-sequence.** In a live workflow, Skip marks the job JOB_TO_BE_SKIPPED rather than skipping it on the spot: it becomes terminal SKIPPED — and its dependents are released — only when it would otherwise have been its turn to start (its TIME/JOB/THRESHOLD/RESOURCE gates satisfied). Skipping a job that has already reached that point (FAILED, MISSED_START_TIME) resolves promptly. So after selecting Skip, expect the job to sit in "Marked to be skipped" until its turn comes — that is normal, not a stuck action.
- **Mark Finished OK / Mark Failed** are broad manual overrides that force a job's outcome so dependents proceed/react; accepted from most non-terminal states. A **running container job** accepts **Kill** only (the mark-overrides are refused while it runs).

If an action isn't offered, the job isn't in a status that accepts it.

Contact support when

- A job sits in a *_PENDING_TERM status indefinitely.
- A status-appropriate action is rejected by the system, or a job's status doesn't reflect what the agent actually reported.

Include the workflow instance, the job, its status, and the job output.

Tags: [Reference](#) [Operator](#) [Processes](#)

*Last updated on **Aug 3, 2026***

Last updated: 2026-06-24

Event log

Task walkthrough: [Use the event log](#). This page is the full configuration and troubleshooting reference.

The **event log** is the record of everything the platform processed — every event, where it came from, what it did, and whether it succeeded. It's the primary tool for tracing *what happened* and *why* across a chain of related actions.

What an event shows

List view: type (e.g. `$JOB:HOLD`), environment, **status**, **source**, actor (who/what triggered it), received time, and processing duration.

Detail view adds: the **payload**, the **changes** it made (entity, field, previous → new value), any **events it triggered** as side effects, an **error** (code + message) if it failed, and the **retry attempts**.

Statuses and sources

Field	Values
Status	RECEIVED, PROCESSING, COMPLETED, FAILED, EXPIRED
Source	API, CLI, AGENT, SCHEDULER, WEBHOOK, UI, JOB_EVENT

Filtering, correlation, and live view

- **Filters:** status, event type (pattern), environment, source, date range, actor name, correlation ID.
- **Correlation:** every event carries a **correlation ID**; a dedicated view shows all events that share one — follow a single action through everything it caused.

- **Live polling:** refresh every 5/10/30/60 seconds (10s default), with play/pause and manual refresh; page size 25/50/100 (50 default).

Things to know

- To trace an incident, start from a failed event and open its **correlation** view to see the full chain (the triggering action and everything it set off).
- A `FAILED` event's detail includes the **error** and the **retry attempts** — the fastest place to see why something didn't process.
- `EXPIRED` means the event wasn't processed within its window — different from `FAILED`.

Contact support when

- An event is `FAILED`/`EXPIRED` with no actionable error, or its **changes**/`triggeredEvents` don't match what actually happened in the system.

Include the event id, correlation id, type, status, and the error/attempts from the detail view.

Tags: [Reference](#) [Operator](#) [Event Log](#) [LMS-Training](#)

*Last updated on **Aug 3, 2026***

Last updated: 2026-07-15

Vision (critical-path monitoring & SLAs)

Evolving. This page is accurate for the current build, but the underlying feature is still in active development — details can change between releases.

⚠️ **Known gap in this build.** The Vision interface is new and still changing, and the evaluate-and-fire loop that actually raises SLA alerts is **switched off by default**. You can set up cards and SLA targets, but nothing will fire until that loop is enabled for your environment.

Task walkthrough: [Monitor critical paths and SLAs with Vision](#). This page is the full configuration and troubleshooting reference.

Vision monitors whether the right work is finishing **on time**, by tag, against **service-level agreements (SLAs)**. A **Vision board** is a hierarchy of **cards**; Vision evaluates each card against a chosen schedule ("anchor") date and surfaces status, and — when enabled — fires the card's configured actions.

Cards

Two card **types** (`type` is chosen at creation and **immutable**):

Type	Purpose
Tag card	Monitors the jobs whose definition tags match a tag pattern , on a schedule date, in one source environment .
Group card	Carries no pattern/frequency of its own — rolls up the stats and state of its child cards.

Cards form a tree via `parentCardId` (a child's parent must be a **group** card; cycles/self-parent rejected). **There is no "card category" concept** — it existed briefly and was removed.

Card configuration

Setting	Type	Required	Default	Notes
<code>name</code>	string	Yes	—	
<code>type</code>	<code>tag</code> <code>group</code>	Yes	<code>tag</code> (UI)	Immutable after create
<code>description</code>	string	No	null	
<code>parentCardId</code>	id	No	null	Must be a group card
Day offset (<code>offsetDays</code>)	int	No	0	Range -5..+5 ; resolves the card's schedule date to the <i>Nth qualifying frequency day</i>
<code>tagPattern</code>	string (≤255)	Tag: Yes	—	Exact, or <code>*</code> / <code>?</code> wildcards; group cards must not set it
<code>sourceEnvironmentId</code>	id	Tag: Yes at create	null	The environment whose runtime supplies the card's job data. Cannot be unbound once set. Group cards must not set it

Setting	Type	Required	Default	Notes
<code>frequencies</code>	array	Tag: ≥ 1	—	Priority-ordered (first match claims the day); group cards must not set any

Each **frequency association** on a tag card holds: a **frequency** (by id/name), a list of **SLAs**, and a list of **triggers**.

SLA (per card × frequency)

Setting	Type	Required	Default (UI)	Notes
<code>requirement</code>	<code>expectedStart</code> <code>expectedEnd</code>	Yes	<code>expectedEnd</code>	UI: "Expected start" / "Expected end"
<code>time</code>	<code>HH:mm</code> (24h)	Yes	<code>00:00</code>	
<code>offsetDays</code>	int	Yes	0	Range -5..+5 ; raw calendar-day offset on the target (the cross-midnight mechanism, e.g. end 02:00 +1 day)

An empty SLA list is valid ("monitored only").

Triggers & actions (per card × frequency)

A **trigger** = a **condition** (see below) → one or more **actions**. Each action:

Field	Required	Notes
<code>eventType</code>	Yes	Must be a routable standard OpCon event (validated against the event routing table) — a notify/job/webhook event
<code>parameters</code>	No	Token-substituted with live card values before emit
<code>repeatAfterMinutes</code>	No	Range 1..60 ; omit/0 = fire once
<code>environmentId</code>	No	Delivery target ("remote instance") — distinct from the card's <code>sourceEnvironmentId</code>

How evaluation works

For an anchor date, the engine snapshots the whole board, and for each **tag** card: picks the qualifying frequency (priority order), resolves the schedule date (via the day-offset walk), resolves the card's **source environment** to that environment's runtime, and aggregates matched jobs by `tagPattern` + schedule date **from that environment**. Group cards roll up their children.

Matched jobs fall into five buckets: **waiting, running, finishedOk, failed, notRun**. A card then exposes **four orthogonal signals** (there is no single OK/at-risk/breached enum):

Signal	Values	Meaning
progress	waiting · inProcess · finished · null	count-driven; null = no matched jobs
severity	none · normal · warn · error	warn = any <i>notRun</i> ; error = any <i>failed</i>
conditions (list)	FAILED · PARTIAL_FAILED · FINISHED_OK · LATE_TO_START · LATE_TO_FINISH · ESTIMATED_LATE_TO_START · ESTIMATED_LATE_TO_FINISH	count- and SLA/time-driven; a card can hold several at once. These drive triggers. On-track SLA = no condition
lifecycle	unknown · calculating · noScheduleDate	short-circuits the above when set

SLA logic: target = (scheduleDate @ time) + offsetDays calendar days (UTC). **Breached** (LATE_*) when past target with nothing started/finished (or an actual is past target); **at-risk** (ESTIMATED_LATE_*) when a forecast exceeds target but now is before it.

Cadence & the fire loop

No push updates — Vision **polls on an adaptive timer**: roughly every **20 seconds**, tightening to about **5 seconds** when an on-track SLA is close enough to its target that it could flip to late within the next cycle. **The evaluate-and-fire loop is off by default** — status can still be read on demand in the UI, but automated trigger firing does not run until it is enabled for your environment. Fire-state is persisted, so it survives a restart; firing is at-most-once per episode with retry and optional repeat.

Environment-awareness

Each tag card is bound to its own `sourceEnvironmentId`; environments are resolved once per tick. A card whose environment can't be used gets a per-card `environmentStatus` — `ok` · `unset` · `not-registered` · `unhealthy` · `unresolvable` — and shows **unknown** lifecycle, rather than sinking the whole board.

Events

There are no Vision-specific events. The earlier `$VISION:CARD_*` taxonomy has been removed. A qualifying trigger emits the **standard OpCon event you configured on that trigger's action** (token-substituted, idempotent) — so Vision routes conditions to standard events rather than having an event family of its own.

Permissions

Workspace-scoped object `vision-card` — **four** permissions (there is **no** `vision-card-category` permission):

Permission	Description
<code>vision-card.view</code>	View Vision cards
<code>vision-card.create</code>	Create new Vision cards
<code>vision-card.edit</code>	Modify existing Vision cards
<code>vision-card.delete</code>	Delete Vision cards

The interface

Mounted at `/vision/*`. Two views: **Live** (default) and **Cards**. Header: "Vision — Tag-based critical-path monitoring & SLAs."

- **Live** — a **Schedule date** picker (default today), an "Updated HH:mm:ss" poll indicator, and a grid of card tiles with **hierarchical drill-down** (group → children; breadcrumbs back). Status labels: Waiting, In process, Finished, Late to start, Late to finish, Est. late to start, Est. late to finish, Failed, Partial failure, Finished OK, Unknown, Calculating, No schedule date.
- **Operations drill-through** — from a tag card, "View jobs for {card}" deep-links to the runtime **Operations → Job Instances** view for that card's source environment, filtered by tag pattern + schedule date.
- **Cards** — a table (Name, Type, Contains, Day offset, Updated) with the same drill-down + create/edit/delete and the card form (Name, Type, Parent group, Day offset, and for tag cards: Tag pattern, Source environment, Description, Frequency associations with SLA + Trigger sub-editors).

Troubleshooting

Symptom	Likely cause	Resolution
Card shows Unknown	Its source environment is unset / not registered / unhealthy / unreachable this tick	Check the card's source environment binding and that environment's runtime health (Administrator).
Card shows No schedule date	No qualifying frequency day for the anchor date (given day-offset)	Check the card's frequencies and day offset (Builder).
Card status looks stale	The adaptive poll (≥ 5 s), or evaluation is off	Confirm evaluation is enabled for your environment if automated firing is expected (Administrator).
Triggers never fire	Evaluation is off (the default), or the action's event type isn't routable	Enable evaluation; verify the action event type (Administrator).
Drill-through not offered on a card	Card has no tag pattern or no bound source environment	Only tag cards with both offer the Operations link.

Contact support when

- A card evaluates against the wrong environment or the wrong jobs, or its environment status is stuck.
- Evaluation is enabled for your environment but triggers do not fire, or fire twice. Include the card, its frequency/SLA, and the action event type.

Tags: [Reference](#) [Operator](#) [Vision](#)

*Last updated on **Aug 3, 2026***

Last updated: 2026-07-15

Notification Manager (groups, triggers, escalation, channels)

Evolving. This page is accurate for the current build, but the underlying feature is still in active development — details can change between releases.

 **Known gaps in this build.** The notifications interface is new and still changing, and two parts are not yet functional end to end: the **OpCon Event channel** records a notification as "sent" but does not yet deliver anything, and **acknowledging from an email or deep link** has no landing page to arrive at. Some membership kinds also have no interface yet. Configure notifications by all means, but do not depend on those paths in this build.

Task walkthrough: [Manage notifications](#). This page is the full configuration and troubleshooting reference.

Notifications tell people when something happens to their automation — an agent goes offline, a job fails, a schedule finishes — through **notification groups** that watch objects and fire **actions** over channels, with optional **escalation** until someone acknowledges.

Notification group

Top-level, tenant-scoped container. Its **type is set at creation and immutable** — it governs valid member kinds, triggers, and actions.

Setting	Type	Required	Default	Notes
name	string (≤255)	Yes	—	Unique per tenant
type	AGENT RELAY WORKFLOW_INSTANCE JOB_INSTANCE	Yes (create)	—	Immutable; UI labels Agent / Relay / Workflow / Job
description	string	No	null	
enabled	bool	No	true	Whole-group kill switch — nothing fires while disabled

Membership (what the group watches)

Allowed **member kinds** depend on group type:

Group type	Member kinds
Agent / Relay	EXPLICIT only
Workflow	EXPLICIT, NAME_PATTERN
Job	EXPLICIT, NAME_PATTERN, TAG, CUSTOM_FIELD

- **EXPLICIT** — a specific object (Agent/Relay by id; Workflow/Job by **definition id**, since instances are ephemeral).
- **NAME_PATTERN** — a glob (*, ?) on the definition name.

- **TAG** (Job only) — the definition's tags include the tag.
- **CUSTOM_FIELD** (Job only) — a custom-field id + operator (EQUALS/IN/EXISTS) + value.
- Optional per member: `environmentId` — restrict matching to one environment (null = tenant-wide).

UI gap: the member UI does not yet surface CUSTOM_FIELD or a Job **explicit-instance** picker (backend supports both).

Triggers (catalog, opt-in)

Triggers are a **built-in catalog keyed by group type** — users toggle catalog entries (opt-in; default disabled), they don't invent event types.

- **Agent:** Offline, Online, Disabled • **Relay:** Offline, Online
- **Workflow ("Schedule"):** Start, Complete, Placed On Hold, Released From Hold
- **Job:** Wait Machine, Late to Start, Late to Finish, Submitted, Still Attempting to Start, Initialization Error, Prerun Failed, Missed Start Time, Running, To Be Killed, Finished OK, Failed, Marked Finished OK, Marked Failed, Cancelled, Restarted, Skipped, Under Review, Fixed, **Exceeded Max Runtime** (the only *derived* condition).

Per trigger: `enabled`, `escalating` (bool), `escalationPolicyId` (required when escalating). A non-escalating trigger sends its **actions** once; an escalating trigger runs its **escalation policy** instead.

Escalation policy → levels → actions

Named, tenant-scoped. A policy is an **ordered list of levels**; updating `levels` replaces the whole set.

Level field	Notes
ordinal	Level 0 = "initial send," delay 0, fired immediately when escalation begins
delayMinutes	Wait after the previous level
repeatCount / repeatIntervalMinutes	Re-fire this level N times before advancing (interval falls back to delay)
actions	Channel actions, same shapes as trigger actions

Runs level 0 → wait → level 1 → ... A level repeats per its count/interval before advancing. Exhausted with no ack → instance **EXPIRES**. Acknowledgement at any point stops escalation (→ **ACKNOWLEDGED**). Each level fire mints a single-use acknowledgement token, valid for **48 hours** by default.

Channels / actions

An action = **channel + config** (validated per channel). Four channels:

Channel	Config (required)	v1 status
EMAIL	to[], subjectTemplate, bodyTemplate (opt: cc / bcc)	Real (SES API transport, or SMTP, or dev sink)
SMS	to, bodyTemplate	Real (HTTP gateway; dev log-sink if unset)
OPCON_EVENT	eventType (\$PREFIX:ACTION), environmentId (opt payloadTemplate)	Dev log-sink in v1 — records "sent", does not submit a real event yet
IN_APP	one of roles / users, titleTemplate (opt bodyTemplate, severity)	Real (in-app feed)

Templates use `{{key}}` tokens: `objectName`, `objectType`, `status`, `objectId`, `environmentId`, `occurredAt` (unknown tokens render empty; output is **not** HTML-escaped). **Note:** the channel-form UI placeholders show `${name}` style, which does **not** match the `{{...}}` renderer — a UI copy mismatch.

Email is delivered through whichever transport is configured for your environment. Email and SMS retry on transient failures; the OpCon Event channel does not.

How a status event becomes a notification

1. **A transition happens** — a job or workflow changes status, or an agent or relay goes offline or comes back.
2. **Matching** — for each **enabled** group of that type whose **members match**, for each **enabled trigger** the event satisfies, a notification instance is created. Configuration edits take effect within about **5 seconds**, so a change you just saved may not apply to the very next event.

3. **Dedupe** — repeats for the same object and trigger inside a window (default **300 s**) collapse into one notification.
4. **Dispatch** — a non-escalating trigger sends its actions once, within a couple of seconds. An escalating trigger runs its policy instead, issuing a fresh acknowledgement token at each level.
5. **Acknowledge** stops escalation and invalidates the outstanding tokens.

Instance states: `PENDING → SENT` (fire-and-forget) or `PENDING → AWAITING_ACK → (ACKNOWLEDGED | EXPIRED | FAILED)`. (An `ESCALATED` state exists but nothing sets it in this build.)

In-app feed & acknowledgement

- **Feed** (self-scoped, no admin permission): a user sees a notification when one of their roles is in `target_roles` or their id/email is in `target_users`. Feed + unread count + mark-read + acknowledge endpoints are mounted under `/notifications`.
- **Header bell** (in web-shell): unread badge (polls the count every 60 s, caps at "9"), feed list (latest 10) with severity chip, relative time, an **Ack** button, select-to-read; empty state "You're all caught up."
- **Deep-link ack** (`POST /notifications/ack/{token}`): the single-use token is the credential (mounted before auth). **v1 gap**: POST-only, returns JSON — there is **no HTML landing page**, so selecting a plain email link (GET) does not acknowledge. In-app **Ack** is the working path today.

Permissions

Object `notifications` (global scope):

Permission	Description
<code>notifications.view</code>	View notification configurations
<code>notifications.create</code>	Create new notification configurations
<code>notifications.edit</code>	Modify notification configurations
<code>notifications.delete</code>	Delete notification configurations
<code>notifications.execute</code>	Ingest status events / trigger notifications (S2S producers)

Config routes (groups/members/triggers/actions/escalation) gate on the matching `notifications.<action>`. The in-app feed and deep-link ack carry **no** admin permission (self-scoped / token). **Note:** the status-event intake route enforces auth+tenant (service token) but does **not** appear to check `notifications.execute` in code — treat "producers require that permission" as unverified.

The interface

Mounted at `/notifications/*`. **Notification Manager** page ("Manage notification groups, members, triggers, actions, and escalation policies") with a left sidebar (**Notification Groups**, **Escalation Policies**). Group detail has **Members** and **Triggers & Actions** tabs. Escalation policy create/edit shows the fixed **Initial send** level 0 plus added levels (Delay, Repeat count, Repeat every) and per-level actions. New-group dialog picks one of the four types.

Troubleshooting

Symptom	Likely cause	Resolution
No notification fired	Group disabled (kill switch), trigger not enabled, no member match, or config change <5 s ago	Confirm group <code>enabled</code> , the trigger is on, members match, and wait out the ~5 s snapshot (Administrator).
"Sent" but nothing arrived	Email on the dev sink (no SMTP/SES), or OpCon Event channel (dev-sink in v1)	Use a configured email transport; don't rely on the OpCon Event action yet (Administrator).
Email-link acknowledgement does nothing	No HTML landing page in v1 (POST-only endpoint)	Acknowledge from the in-app bell instead.
Escalation never stops	No acknowledgement received before levels exhaust	Expected — it EXPIRES; acknowledge to stop it earlier (Operator).
A Job group can't target a custom field / one instance	Backend supports it; UI doesn't surface it yet	Use a tag or name pattern for now.

Contact support when

- You need the OpCon Event channel to actually deliver events, or you need a browser landing page for email/deep-link acknowledgement — both are known gaps in this build.
- An escalation policy does not advance through its levels as configured.

Tags: [Reference](#) [Administrator](#) [Notifications](#)

*Last updated on **Aug 3, 2026***

Last updated: 2026-06-24

Job configuration

See [How workflows work](#) for the model.

Task walkthrough: [Build a workflow](#). This page is the full configuration and troubleshooting reference.

A **job** is the unit of work in a workflow. It runs one task type on an agent and carries its own timing, ordering, and outcome handling.

Job fields

Field	Type	Default	Notes
<code>name</code>	string	—	Job name (unique within the workflow).
<code>taskDefinition</code>	object	—	The task type to run: <code>taskTypeName</code> , optional <code>taskTypeVersion</code> , and <code>parameters</code> . See How connectors work .
<code>agentAssignment</code>	object	—	Where the job runs (see modes below).
<code>frequencies</code>	map	—	One or more frequencies with runtime settings. See Scheduling and build .
<code>dependencies</code>	list	—	What must be satisfied before the job runs. See Job dependencies .
<code>events</code>	list	—	Actions fired on certain outcomes. See Workflow events .
<code>properties</code>	list	—	Job instance properties (name/value pairs; a value may be <code>encrypted</code>).
<code>customFields</code>	map	—	Values for admin-defined custom fields, keyed by definition.

Field	Type	Default	Notes
<code>tags</code>	list	—	Labels for organizing/searching jobs.
<code>disabled</code>	boolean	<code>false</code>	A disabled job is not built into the run.
<code>multiInstance</code>	boolean	<code>false</code>	Allows more than one concurrent instance.
<code>documentation</code>	string	empty	Free-text notes on the job.

Agent assignment modes

Mode	Behavior
Specific agent (<code>agent</code>)	Runs on a chosen primary agent, with optional secondary / tertiary / quaternary failover agents.
Pool – least-tasked (<code>pool-least-tasked</code>)	Runs on the least-busy agent in the named agent pool.
Pool – run-all (<code>pool-run-all</code>)	Runs on every agent in the named pool.

Things to know

- **Encrypted properties** return no value when read back (the value is null on GET) — that's by design, not data loss.
- A job pointing at a task type that needs a **connection** (e.g. SQL) won't run until that connection is attached — see the connector docs.

- `agentPool` on a job is a **deprecated** field; current jobs use `agentAssignment`. If both appear, treat `agentAssignment` as authoritative.
- A **disabled** job, or a build status of `doNotSchedule` / `toBeSkipped`, is intentional — check before treating a "missing" job as a fault.

Tags: [Reference](#) [Builder](#) [Workflows](#) [LMS-Training](#)

*Last updated on **Aug 3, 2026***

Last updated: 2026-06-24

Connections

For the database field details, see also the connector docs.

Task walkthrough: [Manage connections](#). This page is the full configuration and troubleshooting reference.

A **connection** stores the endpoint and credentials for an external system so jobs can use it without embedding secrets. Connections are **defined by plugins** and are **schema-driven** — each connection type declares its own fields.

How connections work

Concept	Notes
Connection type	Provided by a plugin (<code>displayName</code> , a config schema , and a list of sensitive fields). Defines what a connection of that type needs.
Connection instance	A named connection a customer creates from a type; its config is stored encrypted .
Sensitive fields	Fields like passwords are masked and never displayed when editing — leave them blank to keep the current value.
Test connection	Validates a connection and returns success/failure, a message, and latency.

Connection types available today

Connection types come from installed plugins. Today the **SQL Database Executor** provides the connection types that ship:

Type	For
SQL Server	SQL Server (host, port, database, SQL or Windows auth, certificate trust)
MySQL	MySQL
Oracle	Oracle (connection string)
ODBC	Any ODBC/OleDB database

The field details for each are in [Set up database connections](#) and [SQL Database Executor](#). The framework is **extensible** — new connection types arrive with new plugins; the built-in command/script and internal task types need no connection.

Things to know

- There is **no separate API/SSH/Cloud connection type today** — connection types are whatever the installed plugins provide, which today is the database set.
- On **edit**, credential fields read back blank by design (security), not data loss — re-enter only if changing them.
- **Test connection** failures are the fastest triage for database task failures (certificate, login, driver, reachability) — see [SQL Database Executor](#) troubleshooting.

Troubleshooting

Symptom	Likely cause	Resolution
A task reports its connection type "not found"	No connection of the required type is attached	Attach a matching connection to the task (Builder).
Test connection fails	Bad credentials, certificate, driver, or reachability	See the database connector troubleshooting (Administrator).
A connection type you expect isn't listed	Its plugin isn't installed/enabled	Check plugin installation (Administrator) — see Plugins .

Tags: [Reference](#) [Administrator](#) [Connections](#) [LMS-Training](#)

*Last updated on **Aug 3, 2026***

Last updated: 2026-06-24

Properties and tags

Task walkthrough: [Manage properties, calendars, and tags](#). This page is the full configuration and troubleshooting reference.

Properties (workspace-scoped)

A **property** is a named key/value used to parameterize jobs — for example, a path or an environment name reused across jobs.

Field	Notes
<code>name</code>	Unique property name.
<code>value</code>	The value.
<code>encrypted</code>	When true, the value is stored encrypted and is not returned when read back.
<code>description</code>	Optional.

- Properties are **workspace-scoped** (default to the General workspace).
- **Support note:** an encrypted property's value reads back empty by design — that's not data loss. Use encrypted properties for secrets rather than putting them in job parameters.

Tags (global)

A **tag** is a label for organizing and finding jobs (and other objects).

Field	Notes
<code>name</code>	Unique tag name.
<code>description</code>	Optional.

- Tags are **global** (shared across workspaces). A job carries a list of tag names.
- Tags drive filtering and search; a cross-reference view shows which jobs use a given tag.

Tags: [Reference](#) [Builder](#) [Toolkit](#)

*Last updated on **Aug 3, 2026***

Last updated: 2026-06-24

Custom fields

Task walkthrough: [Define custom fields](#). This page is the full configuration and troubleshooting reference.

Custom fields let an administrator add their own metadata fields. Definitions are **tenant-global**, and today they apply to **jobs**.

Definition fields

Field	Notes
<code>name</code>	Unique per tenant.
<code>inputType</code>	One of: <code>text</code> , <code>number</code> , <code>boolean</code> , <code>select</code> , <code>date</code> , <code>textarea</code> .
<code>isRequired</code>	Whether a value is required (default off).
<code>defaultValue</code>	Optional default.
<code>options</code>	The choices — required for <code>select</code> , ignored otherwise.
<code>description</code>	Optional.
<code>displayOrder</code>	Controls ordering in the UI.

Things to know

- Custom fields currently apply to **jobs**. The model is extensible to other object types, but there is no per-object-type assignment in the UI today, so custom fields are not yet available on other objects.
- A **select** field needs **options**; an empty options list isn't valid.
- Values are stored as strings and interpreted per `inputType` where used.

Tags: [Reference](#) [Administrator](#) [Administration](#)

*Last updated on **Aug 3, 2026***

Last updated: 2026-06-24

Workflow versions and deployments

See [How workflows work](#).

Task walkthrough: [Version and deploy a workflow](#). This page is the full configuration and troubleshooting reference.

A workflow is **versioned** as it's edited, and **deployed** to an environment to run there. These are separate ideas: a version is a saved snapshot of the design; a deployment decides which version runs in which environment.

Versions

Each saved change creates a new **version** (an incrementing number) with an optional **change description**, linked to the version it came from. This gives a full history that can be reviewed and rolled back to.

Field	Notes
<code>version</code>	Incrementing version number.
<code>changeDescription</code>	Optional note describing the change.
<code>previousVersion</code>	The version this one was based on (null for the first).
<code>createdAt</code> / <code>createdBy</code>	When and by whom the version was saved.

Deployments

A **deployment** links a workflow to an **environment** with a version strategy and an optional date window.

Field	Notes
<code>environmentId</code>	The environment the workflow is deployed to.
<code>versionStrategy</code>	LATEST (always run the newest version) or PINNED (run a fixed version).
<code>pinnedVersion</code>	The version to run when the strategy is PINNED .
<code>effectiveDate</code> / <code>expirationDate</code>	Optional window during which the deployment is active.
<code>isActive</code>	Whether the deployment is currently active.

A workflow can be deployed to multiple environments — for example, **PINNED** to a tested version in production while running **LATEST** in a test environment.

Things to know

- **A change isn't showing up in an environment.** Likely the deployment is **PINNED** to an older version, the new version hasn't been deployed, or the deployment's **effective date** is in the future. Check the environment's deployment, not just the workflow.
- **"Rolling back"** means deploying (or pinning) an earlier version — the older version still exists in history.
- **Production vs test drift** is expected when production is **PINNED** and test runs **LATEST**.

Contact support when

- The active deployment points at a version that demonstrably differs from what's running in the environment.

Include the workflow, the environment, the deployment's version strategy/pinned version, and the effective/expiration dates.

Tags: [Reference](#) [Builder](#) [Workflows](#)

*Last updated on **Aug 3, 2026***

Last updated: 2026-06-24

Service requests

The end-user side is [the self-service portal](#).

Task walkthrough: [Build a service request](#). This page is the full configuration and troubleshooting reference.

A **service request** is a form a Builder publishes so that a business user can trigger automation without touching the workflow — for example, "Request a file restore" or "Run an on-demand report." Submitting the form fires one or more **events**.

What a service request contains

Part	Notes
Inputs	The form fields the user fills in.
Events	The actions fired on submission, in order.
Documentation / confirmation message	Guidance shown above the form / before submitting.
Tags	For organizing and for filtering in the portal.
Versions	Each save creates a new version with an optional change description.
Deployments	Which environments the request is available in, and at which version.
Transformations	Per-environment overrides applied to the request (see below).

Input types

Type	Status
Text, Password, Number, Date, Choice (list)	Implemented
Workflow / Job / Workflow-instance / Job-instance selection	Declared but not yet active — these render as a plain text field today

Each input can be **required**, have a **default**, **placeholder**, **help text**, and type-specific **validation** (min/max, length, pattern, date range).

Events and variable substitution

Each event has a **type**, structured **parameters**, and an **order**. Parameters can include placeholders like `${input.<field>}` that are **replaced with the user's form values** when the request is submitted. Multiple events are submitted together; a partial failure is reported back.

Deployments

Like workflows, a service request is **deployed** to an environment with a version strategy:

Field	Notes
<code>versionStrategy</code>	LATEST or PINNED .
<code>pinnedVersion</code>	Used when PINNED.
<code>effectiveDate</code> / <code>expirationDate</code>	Optional active window.
<code>isActive</code>	Whether the deployment is active.

Transformations

A **transformation rule** overrides a value in the request for a specific environment or deployment — e.g. swap a dev value for a production value so the same request behaves correctly in each environment. Rules are applied when the portal serves the request. (The builder UI currently exposes the **replace** transform; other transform types exist at the API level.)

Things to know

- The four **selection** input types are not active yet — they appear as text fields, so a request form cannot offer a workflow or job picker.
- A request not appearing in an environment usually means it isn't **deployed** there (or the deployment is pinned/expired/inactive) — check the deployment, like with workflows.
- Encrypted/secret handling: sensitive values should come from connections/properties, not from a plain text input.

Related topics

- [Self-service portal](#)
- [Workflow versions and deployments](#)

Tags: [Reference](#) [Builder](#) [Service Requests](#) [LMS-Training](#)

*Last updated on **Aug 3, 2026***

Last updated: 2026-06-29

Roles and permissions

Evolving. This page is accurate for the current build, but the underlying feature is still in active development — details can change between releases.

 **Enforcement is not switched on yet.** In the current build every permission check passes and row-level scoping returns open results. The roles UI, the permission catalog, scopes, and simulation are all fully built and configurable — but the access you configure is **not actually restricting anything yet**. Read the roles you set up as *intended* access rather than *enforced* access, and do not rely on them to protect anything in this build.

Task walkthrough: [Manage roles and permissions](#). This page is the full configuration and troubleshooting reference.

Roles

Field	Notes
<code>name</code>	Unique per tenant.
<code>description</code>	Optional.
<code>isSystem</code>	System roles (e.g. Administrator) are immutable.
<code>permissions</code>	Catalog permission IDs (<code><objectType>.<action></code>).
<code>scopes</code>	Where the role applies (workspace × environment tuples).

The **Administrator** role is seeded per tenant with **all** permissions and a wildcard scope; it can't be modified. Roles can be duplicated.

Permission catalog

A permission is `<objectType>.<action>`.

- **Actions:** `view`, `create`, `edit`, `delete`, `execute`.
- **Scope models:**
 - **Workspace (W)** — checked against the object's workspace (e.g. workflows, scripts, properties, thresholds, resources, service-requests, agents, agent-pools, connections, **vision-card**).
 - **Global (G)** — tenant-wide (e.g. workspaces, frequencies, calendars, reports, custom-fields, tags, settings, roles, audit-history, **vision-card-category**).
 - **Workspace × Environment (WE)** — both axes checked (e.g. workflow-instances, job-instances, workflow-deployments, service-request-deployments, self-service-portal).
- **Implied view:** granting `create`/`edit`/`delete`/`execute` automatically includes that object's `view` ("you can't manage what you can't see").

Scope tuples

Each role scope is a `(workspaceId, environmentId)` pair where `null` = **any**:

Scope	Applies
<code>(null, null)</code>	Everywhere.
<code>(workspace, null)</code>	That workspace, all environments.
<code>(null, environment)</code>	That environment, all workspaces.
<code>(workspace, environment)</code>	Only that combination.

The object type's scope model decides which axes matter.

Simulation and catalog views

- **Permission simulation** — pick a role (+ optional workspace/environment) and see each permission as **granted** or **denied**, with a reason (`no_role_grants_action`, `no_role_scope_matches`, `missing_workspace_context`, `missing_environment_context`). It shows what a role is **intended** to grant, so it's the right tool for explaining *why* a role would or wouldn't allow something, regardless of whether enforcement is switched on in your release.
- **Permission catalog** — a read-only, searchable list of every permission grouped by object type. The catalog currently holds **104** permissions; the newest object types are **vision-card** (workspace-scoped) and **vision-card-category** (global), for the upcoming Vision feature.

Things to know

- **The most important thing to know:** access is **not enforced** in the current build. If the permissions you configure are not restricting anything, that is expected today rather than a fault.
- A non-system role named "Administrator" blocks the seeded one (the seed is skipped and logged) — a possible cause of unexpected Administrator behavior.
- Use **simulation** to answer "would this role grant X?" questions even while enforcement is off.

Contact support when

- You need to know when access enforcement will be switched on for your environment.
- Simulation results contradict the permission catalog or the role's scopes.

Tags: [Reference](#) [Administrator](#) [Administration](#) [LMS-Training](#)

Last updated on **Aug 3, 2026**

Last updated: 2026-06-24

Reports

Task walkthrough: [Build and run reports](#). This page is the full configuration and troubleshooting reference.

Reports let users query the platform's configuration and history and export the results. A report is built by choosing columns, sorting, and filters against a chosen object; reports can be saved and re-run.

Report catalog

Category	Objects you can report on
Configuration & reference	Agents, Calendars, Frequencies, Jobs, Scripts, Tags, Properties, Resources, Thresholds
Operational objects	Job History, Event Details (by date), Properties, Resources, Thresholds
Governance & access	Audit History

Building a report

Each report supports **columns** (selected and ordered), **sort rules**, and **filters**.
Filter types:

Filter type	Operators
Text	Contains, Equals, Starts with, Is empty, Is not empty
Number	Equals, Greater than, Less than
Single-select	Is, Is not
Date/time	On, Before, After, Between

For example, **Job History** offers filters for environment (required), date range (required), workflow/job/agent name, termination status, exit code, run time, and duration.

Saved reports, results, and export

- A report can be **saved** (name, description, columns, column order, sort rules, filters) and re-run later.
- Results are **paginated** (25 rows per page by default), with a truncation warning for large sets; large exports stream.
- **Export formats:** CSV and Excel (`xlsx`).

Things to know

- **Reports are read-only** — they don't change anything.
- **Known stub:** the **Agent Type** filter currently has no options to choose from.
- For ongoing/scheduled delivery of a result set, see saved **searches** ([Global search](#)), which support scheduled email; reports themselves are run on demand.

Tags: [Reference](#) [Administrator](#) [Reports](#) [LMS-Training](#)

Last updated on Aug 3, 2026

Last updated: 2026-06-24

Global search

Task walkthrough: [Search across OpCon Continuum](#). This page is the full configuration and troubleshooting reference.

Global search finds objects across the platform in one place, with bulk actions, export, and saveable (optionally scheduled) searches.

What's searchable

Ten entity types: **Workflows, Jobs, Calendars, Frequencies, Thresholds, Resources, Tags, Properties, Scripts, Service Requests**. Any combination can be searched at once.

Searching

- Full-text query across the selected types (minimum 2 characters to save a search).
- Per-type filters, pagination (10/20/50/100 per page).
- Multi-entity result sets are capped (≈5,000 rows) with a warning past the threshold.

Saved searches

A search can be **saved** (entity types, query, filters, sort) and re-run. Saved searches can be **scheduled for email delivery** — daily, weekly, or monthly, with day/time and timezone.

Bulk row actions

Action	Notes
Go to object / Open in new tab	Go to the object.
Copy	Supported for workflows and scripts only.
Delete	Available for most types; not for jobs.

Export

Export to **CSV** or **Excel (xlsx)**. Exports include formula-injection protection and are blocked if any selected entity type failed to load (with a clear error so the user can retry).

Things to know

- **Copy** and **delete** availability vary by type (copy: workflows/scripts only; delete: not for jobs) — that's by design, not a missing feature.
- A blocked export usually means one entity type failed to load (timeout, unreachable, access denied) — refresh and retry.

Tags: [Reference](#) [Operator](#) [Search](#)

*Last updated on **Aug 3, 2026***

Glossary

Key terms used across OpCon Continuum documentation. This page and the inline definitions you see when you hover a term elsewhere in the docs are generated from the same source, so they always agree.

Core concepts

Term	Meaning
OpCon Continuum	The product — the platform that automates your processes and their interactions with your systems, including the mainframe core.
Workflow	An automated process: a set of jobs that run in a defined order, on a schedule. (Represented internally as a <i>schedule</i> .)
Job	A single unit of work in a workflow. It runs one task type on an agent.
Task type	The kind of work a job does (run a command, run a SQL script, wait for a file, etc.).
Connector	What provides task types — the way Continuum works with a kind of system or action.

Roles

Term	Meaning
Builder	The person who designs workflows, jobs, service requests, and Toolkit items (frequencies, calendars, thresholds, resources, properties, tags).
Operator	The person who monitors the daily run and responds to jobs that need attention.
Administrator	The person who manages agents, connections, environments, plugins, workspaces, and roles and permissions.
Non-OpCon User	A business user who finds and submits service requests through self-service, without building automation.
Scope / Scoped	The workspaces and environments a role's permissions apply to. A <i>scoped</i> item belongs to, and is visible within, a particular workspace.

Running work

Term	Meaning
Agent	The software that runs jobs on a machine. The Universal Agent runs the built-in and downloadable connectors.
Agent pool	A group of agents that jobs can be routed to.
Relay / legacy LSAM	The bridge to existing legacy OpCon LSAM machines, which run jobs through the relay.
Build	Turning a workflow into a concrete day's run.
Instance	A specific run of a workflow or job.
Daily run	The set of workflow/job instances running for a given day.

Ordering and scheduling

Term	Meaning
Dependency	A condition a job waits on before it runs — a job , threshold , resource , or expression .
Predecessor	A job that another job depends on (on success , failure , or any result).
Frequency	A reusable scheduling rule that decides which days a job is eligible to run.
Calendar	A named set of dates (e.g. holidays) used to exclude days.
Threshold	A named numeric value used to gate a job.
Resource	A named capacity pool that jobs reserve and release, used to limit concurrency.

Building blocks and organization

Term	Meaning
Property	A named, reusable value (a path, an environment name); can be encrypted for secrets.
Tag	A label for organizing and searching jobs and other objects.
Event	An action (a command) fired in response to a job outcome.
Connection	Stored endpoint and credentials for an external system, so jobs use it without embedding secrets.
Plugin	A package that provides task types (and sometimes connection types); enabled per organization.

Requests and governance

Term	Meaning
Service request	A form a Builder publishes so business users can trigger automation.
Self-service	The portal where business users find and submit deployed service requests.
Workspace	A boundary that scopes and isolates content.
Custom field	An admin-defined attribute added to workflows, jobs, and other objects.
Environment	A target such as test or production that workflows and requests are deployed to.
Version / Deployment	A saved snapshot of a design / which version runs in an environment (Latest or Pinned).

Operator actions

Term	Meaning
Hold / Release	Pause a job so it won't run / take it off hold.
Force Start	Start a waiting job now, regardless of what it's waiting on.
Skip	Mark a job so it doesn't run; dependents stop waiting on it.
Restart	Run a failed or cancelled job again.
Mark Fixed	Treat a failed job as resolved (work done outside the system).
Cancel	Stop a job; dependents stop waiting on it.

Related topics

- [Getting started with OpCon Continuum](#)

Tags: [Reference](#) [Glossary](#)

*Last updated on **Jul 29, 2026***