YellowDog
SCHEDULER SERVICE

Bringing businesses the best source of compute for every workload

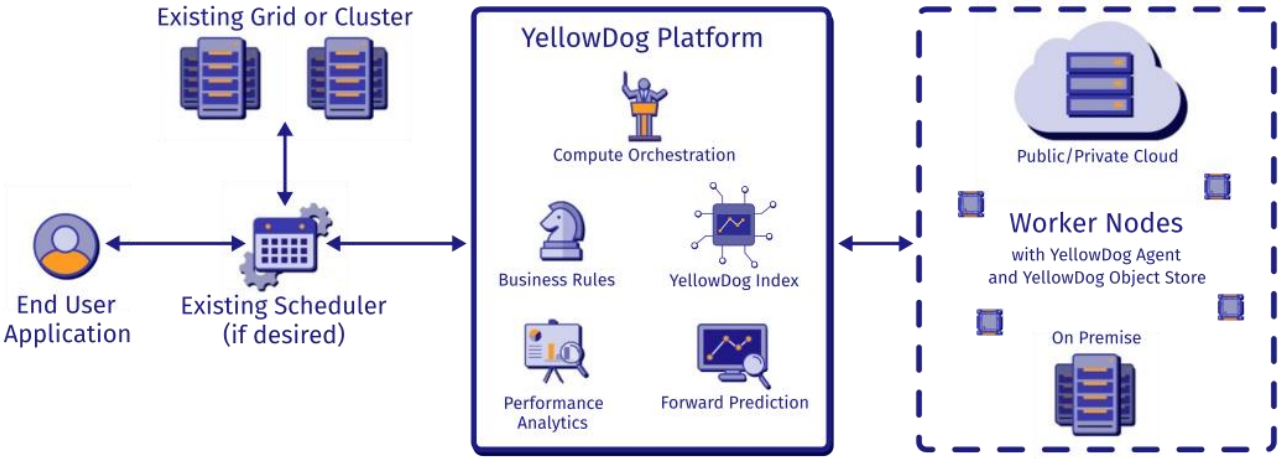
March 2020

TABLE OF CONTENTS

THE SCHEDULER SERVICE

The YellowDog Scheduler Service is the main route for all customers to define their jobs and how they want them to be processed. It supports a configurable set of business rules that determine where jobs will be processed, across both single or multiple compute sources. The processing destinations can be on premise, on cloud, or a combination of both, and can be configured through a simple, flexible and powerful web interface (currently only available for CGI) or via the YellowDog API.

The Scheduler Service is part of a suite of services that make up the YellowDog Platform. It works alongside the YellowDog Compute and Object Store Services, and is accessed via the common platform API. This is a RESTful API that can also be implemented in Java and .NET, and provides a single point of connection to all YellowDog services. The scheduler can be hosted by YellowDog as part of the core hosted YellowDog service, or can be integrated in YellowDog's on premise solutions. It can also be run in a cluster if required to provide high throughput and availability.



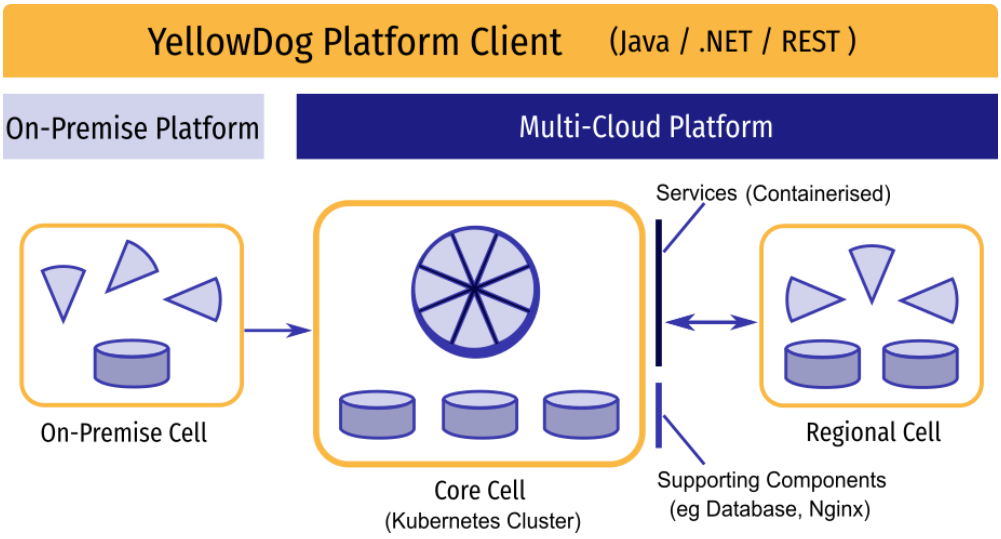
YellowDog Conceptual Architecture

YELLOWDOG SCHEDULER SERVICE FEATURES

- Seamlessly manages any combination of on-premise and cloud-based compute
  - Including cross-region and cross-provider
- Highly available and scalable
  - Internal microservice and event-based architecture allows large service instance clusters
- Decouples the definition and management of work from the provision of resources
  - Provides failure tolerance and recoverability of jobs
  - Allows dynamic provision of compute during running jobs
- Supports multiple concurrent task groups
  - Tasks can be queued individually or made part of a group
  - Can be controlled individually or together
  - Can be assigned to specific compute sources (such as on-premise or a specific provider)
  - Declarative approach to task control (simple document-based definitions)
- Can automatically manage compute resource levels
  - Self-healing compute grid
- Non-polling client updates

IMPLEMENTATION

The YellowDog Platform can be provisioned in three distinct configurations: Core, Regional and on premise. Businesses can access Core or Regional platforms for service, and may subsequently be redirected to another as determined by their location, service level or current load. On premise platforms are provisioned to support the needs of businesses inside the confines of their private networks. On premise platforms may optionally depend upon Core or Regional platforms for some functionality, such as user authentication/authorisation, license validation, and access to public cloud services.



Cloud-based YellowDog platform deployments are managed via Kubernetes. By making use of the Kubernetes model and YellowDog deployment tools, the platform offers high availability and scalability. Businesses who are not yet ready to manage Kubernetes on premise can achieve most of the availability features without the scale using a preconfigured virtual machine.

KEY PRINCIPLES

Each job is implemented in YellowDog as a work requirement. A single work requirements contains task groups and tasks that specify which workers which should execute which parts of the work requirement, how these workers should be claimed, released and shared, and how the number of workers should be maintained as the compute resource available changes. Work requirements are submitted to the YellowDog scheduler via the YellowDog Client API.

The scheduler handles provisioning using the Compute Service. This acquires cloud instances and implements provisioning strategies as specified by the compute requirement (for more information, see Compute Requirement on page 7). It also configures these instances with the correct machine image and, when run by the scheduler, ensures that the YellowDog Agent on each instance is correctly configured to communicate with the Scheduler Service.

The YellowDog Agent is included with the machine image. The agent starts and manages the workers. The number started is defined by the agent configuration. The agent itself runs on configured (on-premise) or provisioned (cloud) instances.

Both the compute requirement and the work requirement define a document that is passed to the system. The YellowDog Platform then shares these documents with the Client, and updates them with any changes. This document model enables both non-polling updates, and dynamic changes to the requirements.

YELLOWDOG SCHEDULER RESOURCES

WORK REQUIREMENTS

A work requirement is an overall programming task that is submitted to the YellowDog Scheduler Service for processing via the YellowDog Client API. It consists of groups of tasks to be executed, and the technical specification of the instances and images that these tasks should be executed on.

Each work requirement exists within a namespace, and has a unique name.

Due to the document model used for the work requirement, tasks can dyamically add new tasks to the work requirement if required. It can also be kept active, held, or restarted via the API.

The progress of a work requirement and of its individual components can be monitored using the status property for each class. This information can also be routed to tools such as Grafana for dashboard-style monitoring.

For more information on work requirement workflow and states, see the Status section of the Appendix on page 9.

TASKS

A task is an individual item of work, run as part of the overall work requirement. Tasks specify the software required to run the task (the task type) and its initialisation data. The YellowDog Scheduler will only allocate tasks to workers that support that task type.

TASK GROUPS

Tasks are grouped into task groups. Task groups help to manage dependencies and information sharing between tasks, as well as enabling a tiered approach to priority setting and progress monitoring. A task group sets worker and scheduler behaviour for its tasks, mainly via the run specification.

RUN SPECIFICATION

The run specification contains a list of machine properties required in order to execute the tasks within the task group. These can relate to the machine configuration (the combination of instance type and machine image), cloud provider settings such as region, or user-defined properties.

The run specification also defines how workers are managed by the task group, via the following properties:

- Task run type This determines the number of workers that the scheduler will attempt to claim for the task group.
- Worker claim strategy Whether the scheduler will continue to attempt to claim more workers after the task group has started processing.
- Worker release strategy When the scheduler releases workers from the task group.
- Share workers Whether workers claimed by this task group can be shared with other task groups and work requirements under the same subscription.
- Maximum task retries How many times a task can be retried after it has failed.

Some of these properties are described more fully below.

TASK RUN TYPE

- Batch This run type allows an ideal and minimum number of workers to be set. The scheduler will try and claim the ideal number of Workers, and will not consider the task group ready for processing until at least the minimum is claimed. This run type is generally used for short-lived tasks that have a specific purpose.
- Services The scheduler will attempt to claim a worker for every task in the task group. The task group will not be considered ready for processing until all the required workers are claimed. This run type is used for tasks that run continually and may continue executing for a very long time.

CLAIM STRATEGIES

- Start-up Only The scheduler will attempt to claim the appropriate number of workers at the start of the work requirement, but will make no attempt to claim additional workers after this point. This is the default claim strategy.
- Maintain Ideal The scheduler checks the number of workers at regular intervals. If the number of workers is below the ideal, it will attempt to claim more workers or re-provision compute.

RELEASE STRATEGIES

- No Waiting Tasks This strategy will release unused workers from a task group when the group no longer has any waiting tasks. This is the default release strategy.
- All Tasks Completed This strategy will only release workers when all the tasks within the group are finished.
- Work Requirement Finished This strategy will only release Workers when the parent work requirement is finished.

DEPENDENCIES

Dependencies can be set between task groups, or between tasks within the same task group. If a dependency does not complete successfully, the dependent task groups or tasks will not start. For example, a task that requires data from another task would be set to be dependent on that task, to ensure that it could not start before the data was available.

OBJECT STORE INTEGRATION

The YellowDog Scheduler Service is integrated with the YellowDog Object Store Service. This integration allows users to set dependencies on uploading and download multiple objects, and to specify inputs and outputs by file name patterns. For example, a task could be configured to upload its task logs to the YellowDog Object Store Service as a text file on completion.

PRIORITIES

Processing priorities can be set on Work Requirements, Task Groups and Tasks, for example prioritising large tasks versus large quantities of small tasks. Priorities are defined as floating point numbers and can be positive or negative, the default priority is zero, allowing a priority setting to prioritize or de-prioritize relative to the default. Priorities are defined “top-down”; Work Requirements priorities are satisfied first, then Task Groups and finally tasks.

WORKER POOLS

Worker pools are collections of workers created using a single compute requirement. This compute may be either configured from invested / on-premise sources, or provisioned from cloud resources.

The workers in a worker pool are used to execute work requirements. A single worker pool can be used for multiple work requirements, or conversely a single work requirement can use workers from multiple worker pools.

A worker pool becomes **active** when at least one worker has been claimed. For provisioned compute, when all workers are released the Scheduler Service shuts down all the workers and uses the Compute Service to terminate the provisioned instances.

YELLOWDOG AGENT

The YellowDog Agent runs on each instance, and is responsible for launching and shutting down workers and providing information to the scheduler. The YellowDog Agent is included on the machine image, and uses a local configuration file to determine how many workers to start and how to communicate with the scheduler. If the agent is installed for an on-premise configured worker pool, the local configuration file is defined as part of setting

up the installation. If the agent is initiated for a cloud-based provisioned worker pool, the configuration is automatically provided by the Scheduler.

AGENT STACK

The agent stack on the Worker Node has three layers:

- Executive Application level program(s) that perform the task allocated to the worker.
- Agent and Workers Communicates with the Scheduler Service to get tasks to perform, and publishes task level stats to the YellowDog Platform.
- Instance Monitor (optional) Publishes low level system stats to the YellowDog Platform.

WORKERS

Workers can be allocated tasks of any task type supported by the worker. For more information, see Tasks on page 5. When workers are started, they register with the Scheduler Service using a token provided during configuration. The scheduler returns an ID for the worker to use when communicating with the scheduler. The worker then requests instructions from the scheduler. These instructions can be to perform a task, to sleep for a period, or to shut down. When a worker completes a task, or wakes from a sleep period, it requests further instructions from the scheduler.

The YellowDog Agent can also unregister workers based on an event such as scheduled down time, or machine shutdown/restart. In this case the agent notifies the scheduler that the task has failed so that the scheduler can resubmit it to other workers.

The worker performs a task by executing a program installed on the instance. The program to be executed, and the data to use when executing it, is defined in the task specification. This data (initData.txt) can be as simple as a set of program arguments, or it can be structured text data such as JSON or XML. The way in which this data is used by the program to be executed is specified in the configuration of the machine image.

The YellowDog Scheduler actively monitors workers by means of a heartbeat mechanism. If a worker does not respond within a specified timeframe, the Scheduler will automatically retry the task as defined by the run specification for that task's task group. For more information, see Run Specification on page 5.

COMPUTE REQUIREMENT

The compute requirement is a document that defines the strategy to use when provisioning compute, and with the sources of compute to use. When it is submitted to the Compute Service, the Compute Service updates the document with data describing the state of the requirement and the state of any provisioned computer machine instances. The latest state of the compute requirement and its associated instances can be retrieved at any point from the Compute Service.

COMPUTE PROVISIONING STRATEGY

The provisioning strategy determines how the Compute Service selects from specified sources of compute to meet the number of instances required. Available provisioning strategies are:

- Single Source Provision Strategy Uses a single compute provision source for the compute requirement. If this source cannot supply the entire requirement, the request will fail.
- Waterfall Provision Strategy Acquires the maximum number of instances available from each compute provision source, in order, until the required number of instances is reached.

- Split Provision Strategy Attempts to split the provision of instances as evenly as possible across the specified sources.

The sources that can be used are specified as part of the provisioning strategy. It is possible to set the same source multiple times, with different properties. For example, this could be used to provision a provider's cheaper instances first, but to complete the requirement with more expensive instances if not enough cheaper instances are available.

COMPUTE SHUTDOWN BEHAVIOURS

The following compute shutdown behaviours are available:

- NoRegisteredWorkersShutdownCondition Shuts down a worker pool if no workers have registered with it within the worker boot time.
- UnusedAfterStartupShutdownCondition Shuts down a worker pool if no workers have been claimed from the pool between the time the first worker registered and the specified time period.
- AllWorkersReleasedShutdownCondition Shuts down a worker pool if no workers from it are currently claimed, but at least one worker has been claimed and released. A delay can be specified such that the worker pool will only be shut down if no further workers are claimed.

If no automatic shutdown behaviours are specified, then the system default behaviour is to apply:

- NoRegisteredWorkersShutdownCondition
- AllWorkersReleasedShutdownCondition (with a zero-time delay)

It is also possible to require an explicit API call to shut down the worker pool. This can be done by specifying an empty list of automatic shutdown behaviours.

APPENDIX

STATUS

Each of the key items of work managed by the scheduler - task, task group and work requirement – has its own status.

The task status indicates the current activity being undertaken by the system relating to that task. The task group status provides a summary status across the tasks in that group. The work requirement status generally provides a summary status across its task groups.

TASK STATUS

Status	Active	Finished	Description
NEW	-	-	The task group has been created but not yet submitted to YellowDog Scheduler.
PENDING	-	-	The task group has been submitted and YellowDog Scheduler is in the process of claiming workers to start the task group. **The task group will remain in PENDING state until the minimum required number of workers has been claimed.**
READY	-	-	The task group has claimed at least the minimum required number of workers and is ready to start working. **The task group will remain in READY state until the Work Requirement is started.**
UNFULFILLED	-	-	The task group (or parent work requirement) was unable to claim the minimum required number of workers to start working and fulfilOnSubmit was specified for the work requirement.
WAITING	Y	-	The task group has been started but is waiting for a dependency task group to complete before executing any tasks.
WORKING	Y	Y	The task group is running, and tasks are being executed by Workers.
STARVED	Y	Y	The task group was in progress, but it has lost all its workers and none of its tasks are being executed.
HELD	Y	-	The task group has been started and may have been running but the parent work requirement has been held by the user such that no further tasks are executed. **The task group will remain in HELD state until the user reactivates the parent work requirement.**
COMPLETED	Y	-	All tasks within the task group have been completed.
FAILED	Y	-	All tasks within the task group have been finished and most may be completed but at least one has failed.
CANCELLING	Y	-	The parent work requirement is in the process of being cancelled, no further tasks will be executed.

Status	Active	Finished	Description
CANCELLED	Y	-	The parent work requirement has been cancelled, no tasks are currently being executed or will be executed.

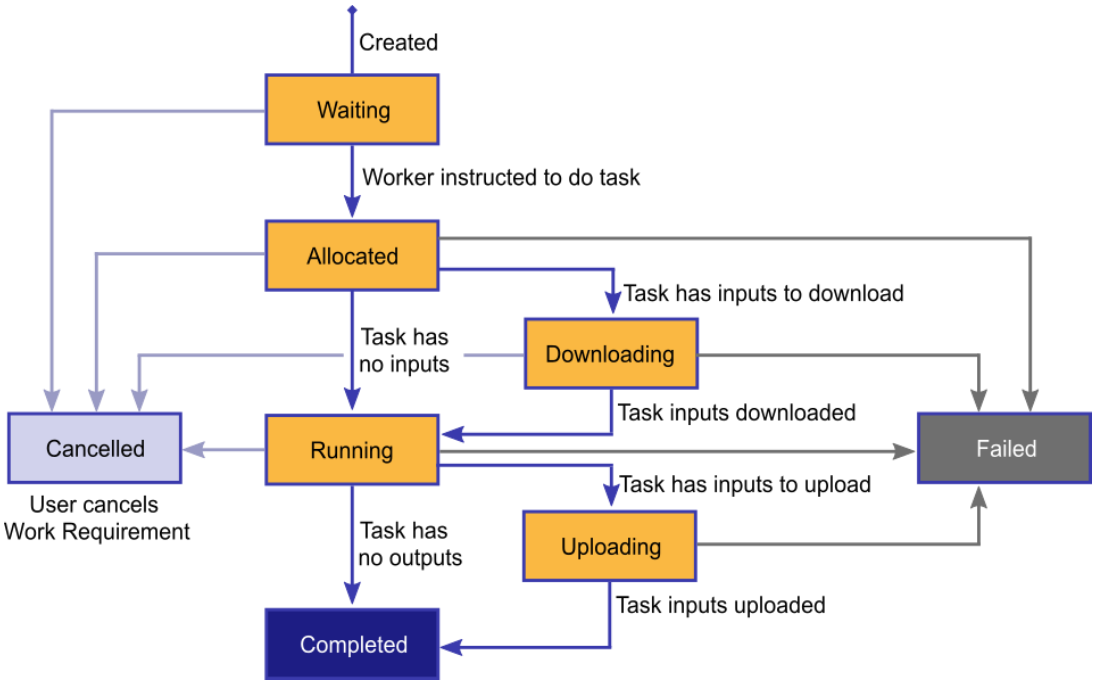


Diagram of a task's flow through the possible task statuses

TASK GROUP STATUS

Status	Active	Finished	Description
NEW	-	-	The task group has been created but not yet submitted to YellowDog Scheduler.
PENDING	-	-	The task group has been submitted and YellowDog Scheduler is in the process of claiming workers to start the task group. **The task group will remain in PENDING state until the minimum required number of workers has been claimed.**
READY	-	-	The task group has claimed at least the minimum required number of workers and is ready to start working. **The task group will remain in READY state until the Work Requirement is started.**
UNFULFILLED	-	-	The task group (or parent work requirement) was unable to claim the minimum required number of workers to start working and fulfilOnSubmit was specified for the work requirement.
WAITING	Y	-	The task group has been started but is waiting for a dependency task group to complete before executing any tasks.

Status	Active	Finished	Description
WORKING	Y	Y	The task group is running, and tasks are being executed by Workers.
STARVED	Y	Y	The task group was in progress, but it has lost all its workers and none of its tasks are being executed.
HELD	Y	-	The task group has been started and may have been running but the parent work requirement has been held by the user such that no further tasks are executed. **The task group will remain in HELD state until the user reactivates the parent work requirement.**
COMPLETED	Y	-	All tasks within the task group have been completed.
FAILED	Y	-	All tasks within the task group have been finished and most may be completed but at least one has failed.
CANCELLING	Y	-	The parent work requirement is in the process of being cancelled, no further tasks will be executed.
CANCELLED	Y	-	The parent work requirement has been cancelled, no tasks are currently being executed or will be executed.

WORK REQUIREMENT STATUS

Status	Active	Finished	Description
NEW	-	-	The work requirement has been created but not yet submitted to YellowDog Scheduler.
PENDING	-	-	The work requirement is waiting to be started once enough workers have been claimed.
UNFULFILLED	-	Y	The work requirement was unable to claim enough workers on submit and has been finished.
WORKING	Y	-	The work requirement is in progress and its tasks are being executed.
STARVED	Y	-	The work requirement was in progress, but it has lost all its workers and none of its tasks are being executed.
HELD	-	-	The work requirement has been held by the user and no further tasks will be executed until it is resumed.
COMPLETED	-	Y	All task groups in the work requirement have been successfully completed.
FAILED	-	Y	All task groups in the work requirement have been finished and most may be completed but at least one has failed.
CANCELLING	-	-	The work requirement is in the process of cancelling, once no task groups are currently executing the status will become CANCELLED.
CANCELLED	-	Y	The work requirement has been explicitly cancelled by the user.

OBJECT MODEL