Partek Flow Worker Allocator

A worker is a program installed on a single computer within a computer cluster and receives job requests from the Partek Flow server. The program will determine if the computer has the needed resources to complete the requested job, and based on the answer, deny or accept the job and report when completed. Workers can be allocated using the tool below.

Note: This tool requires the Flow rest API license feature

Obtaining the Allocator

$ wget customer.partek.com/flow_worker_allocator

$ chmod 755 flow_worker_allocator

Configuring the Allocator

The configuration file **flow.worker.allocator.json** must exist in the home directory. If this file is not present, an example configuration file will be written by running the allocator like so:

$ ./flow_worker_allocator

The allocator will exit immediately so the example configuration file can be edited. This configuration file is documented in the Configuration File Format section below.

Note: The system's temp folder (e.g. /tmp) must be writable and mounted with execute permissions in order for the allocator to run.

Authenticating the Allocator to Flow

After **flow.worker.allocator.json** has been configured, run the allocator:

$ ./flow_worker_allocator

When prompted, enter the Flow administrator username and password. Upon success, an authentication token is stored in **~/flow.auth.json** which is used for all following authentication attempts. If this token is removed, the allocator will again prompt for a Flow username and password. This token only allows access to the Flow rest API and if compromised can not be used to derive Flow account passwords.

Starting the Allocator as a Background Process

$ nohup ./flow_worker_allocator 2>/dev/null &

The allocator takes no arguments. All configuration is stored in **~/flow.worker.allocator.json**.

Stopping the Allocator

$ killall flow_worker_allocator

If the allocator was run as a foreground process, **CONTROL-C** or **SIGTERM** will cause the process to exit.

Logging

The allocator writes daily rotated log files to **~/flow.worker.allocator.log**. For verbose output, set *DebugMode* to 1 in the configuration file.

Allocation Criteria

Once configured, the allocator will poll the Flow server every *CheckIntervalSec* seconds to ask if there is pending work that would be able to start immediately if another Flow worker was added. If true, *WorkerCounterCmd* is used to query the job scheduler to see how many Flow workers have been allocated. If this is below the *WorkerResourceLimit : MaxWorkers* limit, one worker is allocated using *WorkerAllocatorCmd*.

It is recommended that *WorkerResourceLimit : IdleShutdownMin* be relatively short so that allocations are elastic: large workloads are able to allocate all available resources and quickly return those resources to the job scheduler when Flow workers are idle.

Configuration File Format

FlowAPITimeoutSec : integer

The length of time in seconds the allocator will wait for a response from the Flow server.

CheckIntervalSec : integer

The length of time in seconds the allocator will ask Flow if a worker is needed.

InfrastructureWaitTillUpdateTimeSec : integer

The allocator will communicate with the internal resource allocation infrastructure to see how many workers are running. In most cases, this infrastructure is a job scheduler (e.g. torque, lsf, sge) where there can be a delay between the request of resources and the acknowledgement that the request has been made. This parameter tells the allocator to wait for the job scheduler to update before making any further allocation decisions. *Note: InfrastructureWaitTillUpdateTimeSec should be less than CheckIntervalSec.*

DebugMode : integer

If set to *1*, the allocator will use verbose logging. This includes reporting on all allocation decisions.

FlowExternalServerURL : string

The URL used to log into Flow. For example: *http://flow.server.url:8080*

This URL must be network accessible from the server running the allocator. If the allocator is running on the same server as the Flow server, this URL is likely to be *http://localhost:8080*

FlowServerWorkerConnectionHost : string

The DNS name or IP of the Flow server from the worker node's perspective. In most cases, workers that are launched by a job scheduler run on a private network. This means the name of the Flow server that the worker needs to connect to may be different than the one listed under *FlowExternalServerURL*.

FlowDaemonUser : string

The Linux user under which job allocation requests are made. This is used when communicating with a job scheduler to query the number of running or pending Flow workers.

WorkerResourceLimit : JSON data

Defines resource limits for every allocated worker. These values are used by *RunWorkerCMD* and *WorkerAllocatorCmd* to inform the worker and job scheduler about resource requirements. The following are the resource limit types:

> **MaxWorkers** : string
>
> The maximum number of workers that will be allocated regardless of Flow resource demands. This should not be more than the licensed number of workers or more than the number of jobs the queue will accept for *FlowDaemonUser*.
>
> **MaxCores** : string
>
> This defines the maximum number of CPU cores that a worker will use. This should be consistent with the job queue limits.

MaxMemoryMB : string

The maximum amount of memory a worker is allowed to consume. This should be consistent with the job queue limits.

RuntimeMin : string

Maximum lifetime of a worker. This should be less than or equal to the runtime limits imposed by the job queue.

IdleShutdownMin : string

Flow workers that are idle for this number of minutes will auto-shutdown and release their resources back to the job scheduler.

RunWorkerCMD : JSON data

This is used to build the shell command which starts a Flow worker. This command is submitted to the job scheduler. The parameters are as follows:

Type : string

The only supported method at this time is *SHELL*.

Binary : string

The full path to **partekFlowRemoteWorker.sh**

Options : JSON data

These options must be consistent with those defined in *WorkerResourceLimit* and *FlowServerWorkerConnectionHost*. Each option is appended to the job submission command string in the same order it is defined here. The keys *1 … n* are merely placeholders as is *arg1*. Keys labeled as *@self* refer to fields (referenced above) in this configuration file where their value (encoded as a simple array) denote the json key hierarchy from where to lookup this value. In most cases changes are not necessary unless a new type of worker limit is being added or removed.

WorkerAllocatorCmd : JSON data

This is used to build the shell command that interacts with the job scheduler. These values require modification based on the job scheduler, queue limits, and submission options.

Type : string

Defines the type of job scheduler. This is just a label and has no functional impact. Examples include: SGE, TORQUE, LSF, SWARMKIT.

Binary : string

The executable used to submit jobs. This must be in your path. Examples: bsub, qsub

Options : JSON data

Keys define the command line options (for example: **-x, -q, -M**). The values can be *strings*, *null*, or *@self* to read configuration options from this configuration file. *@self* can contain the key append in order to append static strings to command line values.

WorkerCounterCmd : JSON data

This is used to build the command that asks the job scheduler how many workers have been queued or are running. The output from this command is parsed according to the *OutputParser* definition.

Type : string

Defines the type of job scheduler. This is just a label and has no functional impact. Examples include: SGE, TORQUE, LSF, SWARMKIT

Binary : string

The executable used to query submitted job information. This must be in the user's path. Examples include: qstat, jobstat

Options : JSON data

Keys define the command line options (for example: **-x, -q, -M**). The values can be *strings*, *null*, or *@self* to read configuration options from this configuration file. *@self* can contain the key append in order to append static strings to command line values.

OutputParser: JSON data

Currently the only type is LineGrepCount which returns the number of lines output from *WorkerCounterCmd* that contain the strings defined by *LineGrepCount*.

Additional Assistance

If you need additional assistance, please visit our support page to submit a help ticket or find phone numbers for regional support.



Your Rating: Results: 30 rates