Package 'crew.aws.batch'

June 9, 2025

Title A Crew Launcher Plugin for AWS Batch

Description In computationally demanding analysis projects,
statisticians and data scientists asynchronously
deploy long-running tasks to distributed systems,
ranging from traditional clusters to cloud services.
The 'crew.aws.batch' package extends the 'mirai'-powered
'crew' package with a worker launcher plugin for AWS Batch.
Inspiration also comes from packages 'mirai' by Gao (2023)
<https://github.com/r-lib/mirai>,
'future' by Bengtsson (2021) <doi:10.32614/RJ-2021-048>,
'rrq' by FitzJohn and Ashton (2023) <https://github.com/mrc-ide/rrq>,
'clustermq' by Schubert (2019) <doi:10.1093/bioinformatics/btz284>),
and 'batchtools' by Lang, Bischl, and Surmann (2017).
<doi:10.21105/joss.00135>.

Version 0.0.11

License MIT + file LICENSE

URL https://wlandau.github.io/crew.aws.batch/,
https://github.com/wlandau/crew.aws.batch

BugReports https://github.com/wlandau/crew.aws.batch/issues

Depends R (>= 4.0.0)

Imports cli (>= 3.1.0), crew (>= 1.2.0), paws.common (>= 0.7.0),
paws.compute, paws.management, R6, rlang, tibble, utils

Suggests knitr (>= 1.30), markdown (>= 1.1), rmarkdown (>= 2.4),
testthat (>= 3.0.0)

Encoding UTF-8

Language en-US

Config/testthat/edition 3

RoxygenNote 7.3.2

NeedsCompilation no

Author William Michael Landau [aut, cre] (ORCID:
<https://orcid.org/0000-0003-1878-3253>),
Eli Lilly and Company [cph, fnd]

Maintainer William Michael Landau <will.landau.oss@gmail.com>

Repository CRAN

Date/Publication 2025-06-09 14:40:06 UTC

Contents

`crew.aws.batch-package`

crew.aws.batch: a crew launcher plugin for AWS Batch

Description

In computationally demanding analysis projects, statisticians and data scientists asynchronously deploy long-running tasks to distributed systems, ranging from traditional clusters to cloud services. The `crew.aws.batch` package extends the `mirai`-powered `crew` package with worker launcher plugins for AWS Batch. Inspiration also comes from packages `mirai`, `future`, `rrq`, `clustermq`, and `batchtools`.

`crew_class_definition_aws_batch`

AWS Batch definition class

Description

AWS Batch definition `R6` class

Details

See `crew_definition_aws_batch()`.

IAM policies

In order for the AWS Batch `crew` job definition class to function properly, your IAM policy needs permission to perform the `RegisterJobDefinition`, `DeregisterJobDefinition`, and `DescribeJobDefinitions` AWS Batch API calls. For more information on AWS policies and permissions, please visit https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies.html.

Active bindings

`job_queue` See crew_definition_aws_batch().

`job_definition` See crew_definition_aws_batch().

`log_group` See crew_definition_aws_batch().

`config` See crew_definition_aws_batch().

`credentials` See crew_definition_aws_batch().

`endpoint` See crew_definition_aws_batch().

`region` See crew_definition_aws_batch().

Methods

Public methods:

- crew_class_definition_aws_batch$new()
- crew_class_definition_aws_batch$validate()
- crew_class_definition_aws_batch$register()
- crew_class_definition_aws_batch$deregister()
- crew_class_definition_aws_batch$describe()
- crew_class_definition_aws_batch$submit()

Method `new()`**:** AWS Batch job definition constructor.

Usage:

```
crew_class_definition_aws_batch$new(
  job_queue = NULL,
  job_definition = NULL,
  log_group = NULL,
  config = NULL,
  credentials = NULL,
  endpoint = NULL,
  region = NULL
)
```

Arguments:

`job_queue` See crew_definition_aws_batch().

`job_definition` See crew_definition_aws_batch().

`log_group` See crew_definition_aws_batch().

`config` See crew_definition_aws_batch().

`credentials` See crew_definition_aws_batch().

`endpoint` See crew_definition_aws_batch().

`region` See `crew_definition_aws_batch()`.

Returns: AWS Batch job definition object.

Method `validate()`**:** Validate the object.

Usage:

```
crew_class_definition_aws_batch$validate()
```

Returns: `NULL` (invisibly). Throws an error if a field is invalid.

Method `register()`**:** Register a job definition.

Usage:

```
crew_class_definition_aws_batch$register(
  image,
  platform_capabilities = "EC2",
  memory_units = "gigabytes",
  memory = NULL,
  cpus = NULL,
  gpus = NULL,
  seconds_timeout = NULL,
  scheduling_priority = NULL,
  tags = NULL,
  propagate_tags = NULL,
  parameters = NULL,
  job_role_arn = NULL,
  execution_role_arn = NULL
)
```

Arguments:

`image` Character of length 1, Docker image used for each job. You can supply a path to an image in Docker Hub or the full URI of an image in an Amazon ECR repository.

`platform_capabilities` Optional character of length 1, either `"EC2"` to run on EC2 or `"FARGATE"` to run on Fargate.

`memory_units` Character of length 1, either `"gigabytes"` or `"mebibytes"` to set the units of the `memory` argument. `"gigabytes"` is simpler for EC2 jobs, but Fargate has strict requirements about specifying exact amounts of mebibytes (MiB). for details, read https://docs.aws.amazon.com/cli/latest/reference/batch/register-job-definition.html # nolint

`memory` Positive numeric of length 1, amount of memory to request for each job.

`cpus` Positive numeric of length 1, number of virtual CPUs to request for each job.

`gpus` Positive numeric of length 1, number of GPUs to request for each job.

`seconds_timeout` Optional positive numeric of length 1, number of seconds until a job times out.

`scheduling_priority` Optional nonnegative integer of length 1 between `0` and `9999`, priority of jobs. Jobs with higher-valued priorities are scheduled first. The priority only applies if the job queue has a fair share policy. Set to `NULL` to omit.

`tags` Optional character vector of tags.

`propagate_tags` Optional logical of length 1, whether to propagate tags from the job or definition to the ECS task.

`parameters` Optional character vector of key-value pairs designating parameters for job submission.

`job_role_arn` Character of length 1, Amazon resource name (ARN) of the job role.

`execution_role_arn` Character of length 1, Amazon resource name (ARN) of the execution role.

Details: The `register()` method registers a simple job definition using the job definition name and log group originally supplied to `crew_definition_aws_batch()`. Job definitions created with `$register()` are container-based and use the AWS log driver. For more complicated kinds of jobs, we recommend skipping `register()`: first call https://www.paws-r-sdk.com/docs/batch_register_job_definition/ to register the job definition, then supply the job definition name to the `job_definition` argument of `crew_definition_aws_batch()`.

Returns: A one-row `tibble` with the job definition name, ARN, and revision number of the registered job definition.

Method `deregister()`**:** Attempt to deregister a revision of the job definition.

Usage:

```
crew_class_definition_aws_batch$deregister(revision = NULL)
```

Arguments:

`revision` Finite positive integer of length 1, optional revision number to deregister. If `NULL`, then only the highest revision number of the job definition is deregistered, if it exists.

Details: Attempt to deregister the job definition whose name was originally supplied to the `job_definition` argument of `crew_definition_aws_batch()`.

Returns: `NULL` (invisibly).

Method `describe()`**:** Describe the revisions of the job definition.

Usage:

```
crew_class_definition_aws_batch$describe(revision = NULL, active = FALSE)
```

Arguments:

`revision` Positive integer of length 1, optional revision number to describe.

`active` Logical of length 1, whether to filter on just the active job definition.

Returns: A `tibble` with job definition information. There is one row per revision. Some fields may be nested lists.

Method `submit()`**:** Submit an AWS Batch job with the given job definition.

Usage:

```
crew_class_definition_aws_batch$submit(
  command = c("sleep", "300"),
  name = paste0("crew-aws-batch-job-", crew::crew_random_name()),
  cpus = NULL,
  gpus = NULL,
  memory_units = "gigabytes",
  memory = NULL,
  seconds_timeout = NULL,
  share_identifier = NULL,
```

```
  scheduling_priority_override = NULL,
  tags = NULL,
  propagate_tags = NULL,
  parameters = NULL
)
```

Arguments:

`command` Character vector with the command to submit for the job. Usually a Linux shell command with each term in its own character string.

`name` Character of length 1 with the job name.

`cpus` Positive numeric of length 1, number of virtual CPUs to request for each job.

`gpus` Positive numeric of length 1, number of GPUs to request for each job.

`memory_units` Character of length 1, either `"gigabytes"` or `"mebibytes"` to set the units of the `memory` argument. `"gigabytes"` is simpler for EC2 jobs, but Fargate has strict requirements about specifying exact amounts of mebibytes (MiB). for details, read https://docs.aws.amazon.com/cli/latest/reference/batch/register-job-definition.html # nolint

`memory` Positive numeric of length 1, amount of memory to request for each job.

`seconds_timeout` Optional positive numeric of length 1, number of seconds until a job times out.

`share_identifier` Character of length 1 with the share identifier of the job. Only applies if the job queue has a scheduling policy. Read the official AWS Batch documentation for details.

`scheduling_priority_override` Optional nonnegative integer of length between `0` and `9999`, priority of the job. This value overrides the priority in the job definition. Jobs with higher-valued priorities are scheduled first. The priority applies if the job queue has a fair share policy. Set to `NULL` to omit.

`tags` Optional character vector of tags.

`propagate_tags` Optional logical of length 1, whether to propagate tags from the job or definition to the ECS task.

`parameters` Optional character vector of key-value pairs designating parameters for job submission.

Details: This method uses the job queue and job definition that were supplied through `crew_definition_aws_batch()`. Any jobs submitted this way are different from the `crew` workers that the `crew` controller starts automatically using the AWS Batch launcher plugin. You may use the `submit()` method in the definition for different purposes such as testing.

Returns: A one-row `tibble` with the name, ID, and Amazon resource name (ARN) of the job.

See Also

Other definition: `crew_definition_aws_batch()`

crew_class_launcher_aws_batch *AWS Batch launcher class*

Description

AWS Batch launcher R6 class

Details

See crew_launcher_aws_batch().

IAM policies

In order for the AWS Batch crew plugin to function properly, your IAM policy needs permission to perform the SubmitJob and TerminateJob AWS Batch API calls. For more information on AWS policies and permissions, please visit https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies.html.

AWS arguments

The AWS Batch controller and launcher accept many arguments which start with "aws_batch_". These arguments are AWS-Batch-specific parameters forwarded directly to the submit_job() method for the Batch client in the paws.compute R package

For a full description of each argument, including its meaning and format, please visit https://www.paws-r-sdk.com/docs/batch_submit_job/. The upstream API documentation is at https://docs.aws.amazon.com/batch/latest/APIReference/API_SubmitJob.html and the analogous CLI documentation is at https://docs.aws.amazon.com/cli/latest/reference/batch/submit-job.html.

The actual argument names may vary slightly, depending on which : for example, the aws_batch_job_definition argument of the crew AWS Batch launcher/controller corresponds to the jobDefinition argument of the web API and paws.compute::batch()$submit_job(), and both correspond to the --job-definition argument of the CLI.

Verbosity

Control verbosity with the paws.log_level global option in R. Set to 0 for minimum verbosity and 3 for maximum verbosity.

Super class

crew::crew_class_launcher -> crew_class_launcher_aws_batch

Active bindings

options_aws_batch See crew_launcher_aws_batch().

Methods

Public methods:

- crew_class_launcher_aws_batch$new()
- crew_class_launcher_aws_batch$validate()
- crew_class_launcher_aws_batch$launch_worker()
- crew_class_launcher_aws_batch$terminate_worker()

Method `new()`**:** Abstract launcher constructor.

Usage:

```
crew_class_launcher_aws_batch$new(
  name = NULL,
  workers = NULL,
  seconds_interval = NULL,
  seconds_timeout = NULL,
  seconds_launch = NULL,
  seconds_idle = NULL,
  seconds_wall = NULL,
  tasks_max = NULL,
  tasks_timers = NULL,
  reset_globals = NULL,
  reset_packages = NULL,
  reset_options = NULL,
  garbage_collection = NULL,
  tls = NULL,
  processes = NULL,
  r_arguments = NULL,
  options_metrics = NULL,
  options_aws_batch = NULL
)
```

Arguments:

`name` See crew_launcher_aws_batch().

`workers` See crew_launcher_aws_batch().

`seconds_interval` See crew_launcher_aws_batch().

`seconds_timeout` See crew_launcher_aws_batch().

`seconds_launch` See crew_launcher_aws_batch().

`seconds_idle` See crew_launcher_aws_batch().

`seconds_wall` See crew_launcher_aws_batch().

`tasks_max` See crew_launcher_aws_batch().

`tasks_timers` See crew_launcher_aws_batch().

`reset_globals` Deprecated. See crew_launcher_aws_batch().

`reset_packages` Deprecated. See crew_launcher_aws_batch().

`reset_options` Deprecated. See crew_launcher_aws_batch().

`garbage_collection` Deprecated. See crew_launcher_aws_batch().

`tls` See crew_launcher_aws_batch().

processes See crew_launcher_aws_batch().

r_arguments See crew_launcher_aws_batch().

options_metrics See crew_launcher_aws_batch().

options_aws_batch See crew_launcher_aws_batch().

Returns: An abstract launcher object.

Method `validate()`**:** Validate the launcher.

Usage:

```
crew_class_launcher_aws_batch$validate()
```

Returns: NULL (invisibly). Throws an error if a field is invalid.

Method `launch_worker()`**:** Launch a local process worker which will dial into a socket.

Usage:

```
crew_class_launcher_aws_batch$launch_worker(call, name, launcher, worker)
```

Arguments:

call Character string, a namespaced call to crew::crew_worker() which will run in the worker and accept tasks.

name Character string, an informative worker name.

launcher Character string, name of the launcher.

worker Character string, name of the worker instance.

Details: The call argument is R code that will run to initiate the worker.

Returns: A handle object to allow the termination of the worker later on.

Method `terminate_worker()`**:** Terminate a local process worker.

Usage:

```
crew_class_launcher_aws_batch$terminate_worker(handle)
```

Arguments:

handle A process handle object previously returned by launch_worker().

Returns: NULL (invisibly).

See Also

Other plugin_aws_batch: crew_controller_aws_batch(), crew_launcher_aws_batch()

crew_class_monitor_aws_batch *AWS Batch monitor class*

Description

AWS Batch monitor R6 class

Details

See crew_monitor_aws_batch().

IAM policies

In order for the AWS Batch crew monitor class to function properly, your IAM policy needs permission to perform the SubmitJob, TerminateJob, ListJobs, and DescribeJobs AWS Batch API calls. In addition, to download CloudWatch logs with the log() method, your IAM policy also needs permission to perform the GetLogEvents CloudWatch logs API call. For more information on AWS policies and permissions, please visit https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies.html.

Active bindings

job_queue See crew_monitor_aws_batch().

job_definition See crew_monitor_aws_batch().

log_group See crew_monitor_aws_batch().

config See crew_monitor_aws_batch().

credentials See crew_monitor_aws_batch().

endpoint See crew_monitor_aws_batch().

region See crew_monitor_aws_batch().

Methods

Public methods:

- crew_class_monitor_aws_batch$new()
- crew_class_monitor_aws_batch$validate()
- crew_class_monitor_aws_batch$terminate()
- crew_class_monitor_aws_batch$status()
- crew_class_monitor_aws_batch$log()
- crew_class_monitor_aws_batch$jobs()
- crew_class_monitor_aws_batch$active()
- crew_class_monitor_aws_batch$inactive()
- crew_class_monitor_aws_batch$submitted()
- crew_class_monitor_aws_batch$pending()

- crew_class_monitor_aws_batch$runnable()
- crew_class_monitor_aws_batch$starting()
- crew_class_monitor_aws_batch$running()
- crew_class_monitor_aws_batch$succeeded()
- crew_class_monitor_aws_batch$failed()

Method `new()`**:** AWS Batch job definition constructor.

Usage:

```
crew_class_monitor_aws_batch$new(
  job_queue = NULL,
  job_definition = NULL,
  log_group = NULL,
  config = NULL,
  credentials = NULL,
  endpoint = NULL,
  region = NULL
)
```

Arguments:

`job_queue` See crew_monitor_aws_batch().

`job_definition` See crew_monitor_aws_batch().

`log_group` See crew_monitor_aws_batch().

`config` See crew_monitor_aws_batch().

`credentials` See crew_monitor_aws_batch().

`endpoint` See crew_monitor_aws_batch().

`region` See crew_monitor_aws_batch().

Returns: AWS Batch job definition object.

Method `validate()`**:** Validate the object.

Usage:

```
crew_class_monitor_aws_batch$validate()
```

Returns: `NULL` (invisibly). Throws an error if a field is invalid.

Method `terminate()`**:** Terminate one or more AWS Batch jobs.

Usage:

```
crew_class_monitor_aws_batch$terminate(
  ids = NULL,
  all = FALSE,
  reason = "cancelled/terminated by crew.aws.batch monitor",
  verbose = TRUE
)
```

Arguments:

`ids` Character vector with the IDs of the AWS Batch jobs to terminate. Leave as `NULL` if `all` is `TRUE`.

`all` `TRUE` to terminate all jobs belonging to the previously specified job definition. `FALSE` to terminate only the job IDs given in the `ids` argument.

`reason` Character of length 1, natural language explaining the reason the job was terminated.

`verbose` Logical of length 1, whether to show a progress bar if the R process is interactive and `length(ids)` is greater than 1.

Returns: `NULL` (invisibly).

Method `status()`**:** Get the status of a single job

Usage:

```
crew_class_monitor_aws_batch$status(id)
```

Arguments:

`id` Character of length 1, job ID. This is different from the user-supplied job name.

Returns: A one-row `tibble` with information about the job.

Method `log()`**:** Get the CloudWatch log of a job.

Usage:

```
crew_class_monitor_aws_batch$log(id, path = stdout(), start_from_head = FALSE)
```

Arguments:

`id` Character of length 1, job ID. This is different from the user-supplied job name.

`path` Character string or stream (e.g. `stdout()`), file path or connection passed to the `con` argument of `writeLines()` to print the log messages. Set to `nullfile()` to suppress output (and use the invisibly returned `tibble` object instead).

`start_from_head` Logical of length 1, whether to print earlier log events before later ones.

Details: This method assumes the job has log driver `"awslogs"` (specifying AWS CloudWatch) and that the log group is the one prespecified in the `log_group` argument of `crew_monitor_aws_batch()`. This method cannot use other log drivers such as Splunk, and it will fail if the log group is wrong or missing.

Returns: `log()` invisibly returns a `tibble` with log information and writes the messages to the stream or path given by the `path` argument.

Method `jobs()`**:** List all the jobs in the given job queue with the given job definition.

Usage:

```
crew_class_monitor_aws_batch$jobs(
  status = c("submitted", "pending", "runnable", "starting", "running", "succeeded",
    "failed")
)
```

Arguments:

`status` Character vector of job states. Results are limited to these job states.

Details: The output only includes jobs under the job queue and job definition that were supplied through `crew_monitor_aws_batch()`.

Returns: A `tibble` with one row per job and columns with job information.

Method `active()`**:** List active jobs: submitted, pending, runnable, starting, or running (not succeeded or failed).

Usage:

```
crew_class_monitor_aws_batch$active()
```

Details: The output only includes jobs under the job queue and job definition that were supplied through `crew_monitor_aws_batch()`.

Returns: A `tibble` with one row per job and columns with job information.

Method `inactive()`**:** List inactive jobs: ones whose status is succeeded or failed (not submitted, pending, runnable, starting, or running).

Usage:

```
crew_class_monitor_aws_batch$inactive()
```

Details: The output only includes jobs under the job queue and job definition that were supplied through `crew_monitor_aws_batch()`.

Returns: A `tibble` with one row per job and columns with job information.

Method `submitted()`**:** List jobs whose status is `"submitted"`.

Usage:

```
crew_class_monitor_aws_batch$submitted()
```

Details: The output only includes jobs under the job queue and job definition that were supplied through `crew_monitor_aws_batch()`.

Returns: A `tibble` with one row per job and columns with job information.

Method `pending()`**:** List jobs whose status is `"pending"`.

Usage:

```
crew_class_monitor_aws_batch$pending()
```

Details: The output only includes jobs under the job queue and job definition that were supplied through `crew_monitor_aws_batch()`.

Returns: A `tibble` with one row per job and columns with job information.

Method `runnable()`**:** List jobs whose status is `"runnable"`.

Usage:

```
crew_class_monitor_aws_batch$runnable()
```

Details: The output only includes jobs under the job queue and job definition that were supplied through `crew_monitor_aws_batch()`.

Returns: A `tibble` with one row per job and columns with job information.

Method `starting()`**:** List jobs whose status is `"starting"`.

Usage:

```
crew_class_monitor_aws_batch$starting()
```

Details: The output only includes jobs under the job queue and job definition that were supplied through `crew_monitor_aws_batch()`.

Returns: A `tibble` with one row per job and columns with job information.

Method `running()`**:** List jobs whose status is `"running"`.

Usage:

```
crew_class_monitor_aws_batch$running()
```

Details: The output only includes jobs under the job queue and job definition that were supplied through `crew_monitor_aws_batch()`.

Returns: A `tibble` with one row per job and columns with job information.

Method `succeeded()`**:** List jobs whose status is `"succeeded"`.

Usage:

```
crew_class_monitor_aws_batch$succeeded()
```

Details: The output only includes jobs under the job queue and job definition that were supplied through `crew_monitor_aws_batch()`.

Returns: A `tibble` with one row per job and columns with job information.

Method `failed()`**:** List jobs whose status is `"failed"`.

Usage:

```
crew_class_monitor_aws_batch$failed()
```

Details: The output only includes jobs under the job queue and job definition that were supplied through `crew_monitor_aws_batch()`.

Returns: A `tibble` with one row per job and columns with job information.

See Also

Other monitor: `crew_monitor_aws_batch()`

`crew_controller_aws_batch` *Create a controller with an AWS Batch launcher.*

Description

Create an `R6` object to submit tasks and launch workers on AWS Batch workers.

Usage

```
crew_controller_aws_batch(
  name = NULL,
  workers = 1L,
  host = NULL,
  port = NULL,
  tls = crew::crew_tls(mode = "automatic"),
  tls_enable = NULL,
  tls_config = NULL,
  serialization = NULL,
  seconds_interval = 0.5,
```

```
  seconds_timeout = 60,
  seconds_launch = 1800,
  seconds_idle = 300,
  seconds_wall = Inf,
  retry_tasks = NULL,
  tasks_max = Inf,
  tasks_timers = 0L,
  reset_globals = TRUE,
  reset_packages = FALSE,
  reset_options = FALSE,
  garbage_collection = FALSE,
  crashes_error = NULL,
  processes = NULL,
  r_arguments = c("--no-save", "--no-restore"),
  crashes_max = 5L,
  backup = NULL,
  options_metrics = crew::crew_options_metrics(),
  options_aws_batch = crew.aws.batch::crew_options_aws_batch(),
  aws_batch_config = NULL,
  aws_batch_credentials = NULL,
  aws_batch_endpoint = NULL,
  aws_batch_region = NULL,
  aws_batch_job_definition = NULL,
  aws_batch_job_queue = NULL,
  aws_batch_share_identifier = NULL,
  aws_batch_scheduling_priority_override = NULL,
  aws_batch_parameters = NULL,
  aws_batch_container_overrides = NULL,
  aws_batch_node_overrides = NULL,
  aws_batch_retry_strategy = NULL,
  aws_batch_propagate_tags = NULL,
  aws_batch_timeout = NULL,
  aws_batch_tags = NULL,
  aws_batch_eks_properties_override = NULL
)
```

Arguments

name	Character string, name of the launcher. If the name is NULL, then a name is automatically generated when the launcher starts.
workers	Maximum number of workers to run concurrently when auto-scaling, excluding task retries and manual calls to launch(). Special workers allocated for task retries do not count towards this limit, so the number of workers running at a given time may exceed this maximum. A smaller number of workers may run if the number of executing tasks is smaller than the supplied value of the workers argument.
host	IP address of the mirai client to send and receive tasks. If NULL, the host defaults to nanonext::ip_addr()[1].

port — TCP port to listen for the workers. If NULL, then an available ephemeral port is automatically chosen. Controllers running simultaneously on the same computer (as in a controller group) must not share the same TCP port.

tls — A TLS configuration object from crew_tls().

tls_enable — Deprecated on 2023-09-15 in version 0.4.1. Use argument tls instead.

tls_config — Deprecated on 2023-09-15 in version 0.4.1. Use argument tls instead.

serialization — Either NULL (default) or an object produced by mirai::serial_config() to control the serialization of data sent to workers. This can help with either more efficient data transfers or to preserve attributes of otherwise non-exportable objects (such as torch tensors or arrow tables). See ?mirai::serial_config for details.

seconds_interval — Number of seconds between polling intervals waiting for certain internal synchronous operations to complete. In certain cases, exponential backoff is used with this argument passed to seconds_max in a crew_throttle() object.

seconds_timeout — Number of seconds until timing out while waiting for certain synchronous operations to complete, such as checking mirai::status().

seconds_launch — Seconds of startup time to allow. A worker is unconditionally assumed to be alive from the moment of its launch until seconds_launch seconds later. After seconds_launch seconds, the worker is only considered alive if it is actively connected to its assign websocket.

seconds_idle — Maximum number of seconds that a worker can idle since the completion of the last task. If exceeded, the worker exits. But the timer does not launch until tasks_timers tasks have completed. See the idletime argument of mirai::daemon(). crew does not excel with perfectly transient workers because it does not micromanage the assignment of tasks to workers, so please allow enough idle time for a new worker to be delegated a new task.

seconds_wall — Soft wall time in seconds. The timer does not launch until tasks_timers tasks have completed. See the walltime argument of mirai::daemon().

retry_tasks — Deprecated on 2025-01-13 (crew version 0.10.2.9002).

tasks_max — Maximum number of tasks that a worker will do before exiting. See the maxtasks argument of mirai::daemon(). crew does not excel with perfectly transient workers because it does not micromanage the assignment of tasks to workers, it is recommended to set tasks_max to a value greater than 1.

tasks_timers — Number of tasks to do before activating the timers for seconds_idle and seconds_wall. See the timerstart argument of mirai::daemon().

reset_globals — TRUE to reset global environment variables between tasks, FALSE to leave them alone.

reset_packages — TRUE to detach any packages loaded during a task (runs between each task), FALSE to leave packages alone. In either case, the namespaces are not detached.

reset_options — TRUE to reset global options to their original state between each task, FALSE otherwise. It is recommended to only set reset_options = TRUE if reset_packages is also TRUE because packages sometimes rely on options they set at loading

time. for this and other reasons, reset_options only resets options that were nonempty at the beginning of the task. If your task sets an entirely new option not already in options(), then reset_options = TRUE does not delete the option.

garbage_collection
: TRUE to run garbage collection after each task task, FALSE to skip.

crashes_error
: Deprecated on 2025-01-13 (crew version 0.10.2.9002).

processes
: NULL or positive integer of length 1, number of local processes to launch to allow worker launches to happen asynchronously. If NULL, then no local processes are launched. If 1 or greater, then the launcher starts the processes on start() and ends them on terminate(). Plugins that may use these processes should run asynchronous calls using launcher$async$eval() and expect a mirai task object as the return value.

r_arguments
: Optional character vector of command line arguments to pass to Rscript (non-Windows) or Rscript.exe (Windows) when starting a worker. Example: r_arguments = c("--vanilla", "--max-connections=32").

crashes_max
: In rare cases, a worker may exit unexpectedly before it completes its current task. If this happens, pop() returns a status of "crash" instead of "error" for the task. The controller does not automatically retry the task, but you can retry it manually by calling push() again and using the same task name as before. (However, targets pipelines running crew do automatically retry tasks whose workers crashed.)

  crashes_max is a non-negative integer, and it sets the maximum number of allowable consecutive crashes for a given task. If a task's worker crashes more than crashes_max times in a row, then pop() throws an error when it tries to return the results of the task.

backup
: An optional crew controller object, or NULL to omit. If supplied, the backup controller runs any pushed tasks that have already reached crashes_max consecutive crashes. Using backup, you can create a chain of controllers with different levels of resources (such as worker memory and CPUs) so that a task that fails on one controller can retry using incrementally more powerful workers. All controllers in a backup chain should be part of the same controller group (see crew_controller_group()) so you can call the group-level pop() and collect() methods to make sure you get results regardless of which controller actually ended up running the task.

  Limitations of backup: * crashes_max needs to be positive in order for backup to be used. Otherwise, every task would always skip the current controller and go to backup. * backup cannot be a controller group. It must be an ordinary controller.

options_metrics
: Either NULL to opt out of resource metric logging for workers, or an object from crew_options_metrics() to enable and configure resource metric logging for workers. For resource logging to run, the autometric R package version 0.1.0 or higher must be installed.

options_aws_batch
: List of options from crew_options_aws_batch(). The job definition and job

queue must be specified in crew_options_aws_batch(). crew_options_aws_batch() also allows you to request vCPUs, GPUs, and memory for the jobs.

aws_batch_config
: Deprecated. Use options_aws_batch instead.

aws_batch_credentials
: Deprecated. Use options_aws_batch instead.

aws_batch_endpoint
: Deprecated. Use options_aws_batch instead.

aws_batch_region
: Deprecated. Use options_aws_batch instead.

aws_batch_job_definition
: Deprecated. Use options_aws_batch instead.

aws_batch_job_queue
: Deprecated. Use options_aws_batch instead.

aws_batch_share_identifier
: Deprecated. Use options_aws_batch instead.

aws_batch_scheduling_priority_override
: Deprecated. Use options_aws_batch instead.

aws_batch_parameters
: Deprecated. Use options_aws_batch instead.

aws_batch_container_overrides
: Deprecated. Use options_aws_batch instead.

aws_batch_node_overrides
: Deprecated. Use options_aws_batch instead.

aws_batch_retry_strategy
: Deprecated. Use options_aws_batch instead.

aws_batch_propagate_tags
: Deprecated. Use options_aws_batch instead.

aws_batch_timeout
: Deprecated. Use options_aws_batch instead.

aws_batch_tags
: Deprecated. Use options_aws_batch instead.

aws_batch_eks_properties_override
: Deprecated. Use options_aws_batch instead.

IAM policies

In order for the AWS Batch crew plugin to function properly, your IAM policy needs permission to perform the SubmitJob and TerminateJob AWS Batch API calls. For more information on AWS policies and permissions, please visit https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies.html.

AWS arguments

The AWS Batch controller and launcher accept many arguments which start with "aws_batch_". These arguments are AWS-Batch-specific parameters forwarded directly to the submit_job() method for the Batch client in the paws.compute R package

For a full description of each argument, including its meaning and format, please visit https://www.paws-r-sdk.com/docs/batch_submit_job/. The upstream API documentation is at https://docs.aws.amazon.com/batch/latest/APIReference/API_SubmitJob.html and the analogous CLI documentation is at https://docs.aws.amazon.com/cli/latest/reference/batch/submit-job.html.

The actual argument names may vary slightly, depending on which : for example, the `aws_batch_job_definition` argument of the `crew` AWS Batch launcher/controller corresponds to the `jobDefinition` argument of the web API and `paws.compute::batch()$submit_job()`, and both correspond to the `--job-definition` argument of the CLI.

Verbosity

Control verbosity with the `paws.log_level` global option in R. Set to 0 for minimum verbosity and 3 for maximum verbosity.

See Also

Other plugin_aws_batch: `crew_class_launcher_aws_batch`, `crew_launcher_aws_batch()`

Examples

```
if (identical(Sys.getenv("CREW_EXAMPLES"), "true")) {
controller <- crew_controller_aws_batch(
  aws_batch_job_definition = "YOUR_JOB_DEFINITION_NAME",
  aws_batch_job_queue = "YOUR_JOB_QUEUE_NAME"
)
controller$start()
controller$push(name = "task", command = sqrt(4))
controller$wait()
controller$pop()$result
controller$terminate()
}
```

`crew_definition_aws_batch`

Create an AWS Batch job definition object.

Description

Create an `R6` object to manage a job definition for AWS Batch jobs.

Usage

```
crew_definition_aws_batch(
  job_queue,
  job_definition = paste0("crew-aws-batch-job-definition-", crew::crew_random_name()),
  log_group = "/aws/batch/job",
  config = NULL,
```

```
  credentials = NULL,
  endpoint = NULL,
  region = NULL
)
```

Arguments

job_queue	Character vector of names of AWS Batch job queues. As of crew.aws.batch version 0.0.8 and above, you can supply more than one job queue. Methods like jobs() and active() will query all the job queues given.
job_definition	Character of length 1, name of the AWS Batch job definition. The job definition might or might not exist at the time crew_definition_aws_batch() is called. Either way is fine.
log_group	Character of length 1, AWS Batch CloudWatch log group to get job logs. The default log group is often "/aws/batch/job", but not always. It is not easy to get the log group of an active job or job definition, so if you have a non-default log group and you do not know its name, please consult your system administrator.
config	Optional named list, config argument of paws.compute::batch() with optional configuration details.
credentials	Optional named list. credentials argument of paws.compute::batch() with optional credentials (if not already provided through environment variables such as AWS_ACCESS_KEY_ID).
endpoint	Optional character of length 1. endpoint argument of paws.compute::batch() with the endpoint to send HTTP requests.
region	Character of length 1. region argument of paws.compute::batch() with an AWS region string such as "us-east-2". Serves as the region for both AWS Batch and CloudWatch. Tries to default to paws.common::get_config()$region, then to Sys.getenv("AWS_REGION") if unsuccessful, then Sys.getenv("AWS_REGION"), then Sys.getenv("AWS_DEFAULT_REGION").

Value

An R6 job definition object.

IAM policies

In order for the AWS Batch crew job definition class to function properly, your IAM policy needs permission to perform the RegisterJobDefinition, DeregisterJobDefinition, and DescribeJobDefinitions AWS Batch API calls. For more information on AWS policies and permissions, please visit https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies.html.

See Also

Other definition: crew_class_definition_aws_batch

`crew_launcher_aws_batch` *Create an AWS Batch launcher object.*

Description

Create an `R6` AWS Batch launcher object.

Usage

```
crew_launcher_aws_batch(
  name = NULL,
  workers = 1L,
  seconds_interval = 0.5,
  seconds_timeout = 60,
  seconds_launch = 1800,
  seconds_idle = 300,
  seconds_wall = Inf,
  tasks_max = Inf,
  tasks_timers = 0L,
  reset_globals = NULL,
  reset_packages = NULL,
  reset_options = NULL,
  garbage_collection = NULL,
  crashes_error = NULL,
  tls = crew::crew_tls(mode = "automatic"),
  processes = NULL,
  r_arguments = c("--no-save", "--no-restore"),
  options_metrics = crew::crew_options_metrics(),
  options_aws_batch = crew.aws.batch::crew_options_aws_batch(),
  aws_batch_config = NULL,
  aws_batch_credentials = NULL,
  aws_batch_endpoint = NULL,
  aws_batch_region = NULL,
  aws_batch_job_definition = NULL,
  aws_batch_job_queue = NULL,
  aws_batch_share_identifier = NULL,
  aws_batch_scheduling_priority_override = NULL,
  aws_batch_parameters = NULL,
  aws_batch_container_overrides = NULL,
  aws_batch_node_overrides = NULL,
  aws_batch_retry_strategy = NULL,
  aws_batch_propagate_tags = NULL,
  aws_batch_timeout = NULL,
  aws_batch_tags = NULL,
  aws_batch_eks_properties_override = NULL
)
```

Arguments

name	Character string, name of the launcher. If the name is NULL, then a name is automatically generated when the launcher starts.
workers	Maximum number of workers to run concurrently when auto-scaling, excluding task retries and manual calls to launch(). Special workers allocated for task retries do not count towards this limit, so the number of workers running at a given time may exceed this maximum. A smaller number of workers may run if the number of executing tasks is smaller than the supplied value of the workers argument.
seconds_interval	Number of seconds between polling intervals waiting for certain internal synchronous operations to complete. In certain cases, exponential backoff is used with this argument passed to seconds_max in a crew_throttle() object.
seconds_timeout	Number of seconds until timing out while waiting for certain synchronous operations to complete, such as checking mirai::status().
seconds_launch	Seconds of startup time to allow. A worker is unconditionally assumed to be alive from the moment of its launch until seconds_launch seconds later. After seconds_launch seconds, the worker is only considered alive if it is actively connected to its assign websocket.
seconds_idle	Maximum number of seconds that a worker can idle since the completion of the last task. If exceeded, the worker exits. But the timer does not launch until tasks_timers tasks have completed. See the idletime argument of mirai::daemon(). crew does not excel with perfectly transient workers because it does not micromanage the assignment of tasks to workers, so please allow enough idle time for a new worker to be delegated a new task.
seconds_wall	Soft wall time in seconds. The timer does not launch until tasks_timers tasks have completed. See the walltime argument of mirai::daemon().
tasks_max	Maximum number of tasks that a worker will do before exiting. See the maxtasks argument of mirai::daemon(). crew does not excel with perfectly transient workers because it does not micromanage the assignment of tasks to workers, it is recommended to set tasks_max to a value greater than 1.
tasks_timers	Number of tasks to do before activating the timers for seconds_idle and seconds_wall. See the timerstart argument of mirai::daemon().
reset_globals	Deprecated on 2025-05-30 (crew version 1.1.2.9004). Please use the reset_globals option of crew_controller() instead.
reset_packages	Deprecated on 2025-05-30 (crew version 1.1.2.9004). Please use the reset_packages option of crew_controller() instead.
reset_options	Deprecated on 2025-05-30 (crew version 1.1.2.9004). Please use the reset_options option of crew_controller() instead.
garbage_collection	Deprecated on 2025-05-30 (crew version 1.1.2.9004). Please use the garbage_collection option of crew_controller() instead.
crashes_error	Deprecated on 2025-01-13 (crew version 0.10.2.9002).

`tls` A TLS configuration object from `crew_tls()`.

`processes` `NULL` or positive integer of length 1, number of local processes to launch to allow worker launches to happen asynchronously. If `NULL`, then no local processes are launched. If 1 or greater, then the launcher starts the processes on `start()` and ends them on `terminate()`. Plugins that may use these processes should run asynchronous calls using `launcher$async$eval()` and expect a `mirai` task object as the return value.

`r_arguments` Optional character vector of command line arguments to pass to `Rscript` (non-Windows) or `Rscript.exe` (Windows) when starting a worker. Example: `r_arguments = c("--vanilla", "--max-connections=32")`.

`options_metrics` Either `NULL` to opt out of resource metric logging for workers, or an object from `crew_options_metrics()` to enable and configure resource metric logging for workers. For resource logging to run, the `autometric` R package version 0.1.0 or higher must be installed.

`options_aws_batch` List of options from `crew_options_aws_batch()`. The job definition and job queue must be specified in `crew_options_aws_batch()`. `crew_options_aws_batch()` also allows you to request vCPUs, GPUs, and memory for the jobs.

`aws_batch_config` Deprecated. Use `options_aws_batch` instead.

`aws_batch_credentials` Deprecated. Use `options_aws_batch` instead.

`aws_batch_endpoint` Deprecated. Use `options_aws_batch` instead.

`aws_batch_region` Deprecated. Use `options_aws_batch` instead.

`aws_batch_job_definition` Deprecated. Use `options_aws_batch` instead.

`aws_batch_job_queue` Deprecated. Use `options_aws_batch` instead.

`aws_batch_share_identifier` Deprecated. Use `options_aws_batch` instead.

`aws_batch_scheduling_priority_override` Deprecated. Use `options_aws_batch` instead.

`aws_batch_parameters` Deprecated. Use `options_aws_batch` instead.

`aws_batch_container_overrides` Deprecated. Use `options_aws_batch` instead.

`aws_batch_node_overrides` Deprecated. Use `options_aws_batch` instead.

`aws_batch_retry_strategy` Deprecated. Use `options_aws_batch` instead.

`aws_batch_propagate_tags` Deprecated. Use `options_aws_batch` instead.

`aws_batch_timeout`
: Deprecated. Use `options_aws_batch` instead.

`aws_batch_tags`
: Deprecated. Use `options_aws_batch` instead.

`aws_batch_eks_properties_override`
: Deprecated. Use `options_aws_batch` instead.

Value

An `R6` AWS Batch launcher object.

IAM policies

In order for the AWS Batch `crew` plugin to function properly, your IAM policy needs permission to perform the `SubmitJob` and `TerminateJob` AWS Batch API calls. For more information on AWS policies and permissions, please visit https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies.html.

AWS arguments

The AWS Batch controller and launcher accept many arguments which start with `"aws_batch_"`. These arguments are AWS-Batch-specific parameters forwarded directly to the `submit_job()` method for the Batch client in the `paws.compute` R package

For a full description of each argument, including its meaning and format, please visit https://www.paws-r-sdk.com/docs/batch_submit_job/. The upstream API documentation is at https://docs.aws.amazon.com/batch/latest/APIReference/API_SubmitJob.html and the analogous CLI documentation is at https://docs.aws.amazon.com/cli/latest/reference/batch/submit-job.html.

The actual argument names may vary slightly, depending on which : for example, the `aws_batch_job_definition` argument of the `crew` AWS Batch launcher/controller corresponds to the `jobDefinition` argument of the web API and `paws.compute::batch()$submit_job()`, and both correspond to the `--job-definition` argument of the CLI.

Verbosity

Control verbosity with the `paws.log_level` global option in R. Set to 0 for minimum verbosity and 3 for maximum verbosity.

See Also

Other plugin_aws_batch: `crew_class_launcher_aws_batch`, `crew_controller_aws_batch()`

crew_monitor_aws_batch *Create an AWS Batch monitor object.*

Description

Create an R6 object to list, inspect, and terminate AWS Batch jobs.

Usage

```
crew_monitor_aws_batch(
  job_queue,
  job_definition,
  log_group = "/aws/batch/job",
  config = NULL,
  credentials = NULL,
  endpoint = NULL,
  region = NULL
)
```

Arguments

job_queue	Character vector of names of AWS Batch job queues. As of crew.aws.batch version 0.0.8 and above, you can supply more than one job queue. Methods like jobs() and active() will query all the job queues given.
job_definition	Character string, name of the AWS Batch job definition.
log_group	Character of length 1, AWS Batch CloudWatch log group to get job logs. The default log group is often "/aws/batch/job", but not always. It is not easy to get the log group of an active job or job definition, so if you have a non-default log group and you do not know its name, please consult your system administrator.
config	Optional named list, config argument of paws.compute::batch() with optional configuration details.
credentials	Optional named list. credentials argument of paws.compute::batch() with optional credentials (if not already provided through environment variables such as AWS_ACCESS_KEY_ID).
endpoint	Optional character of length 1. endpoint argument of paws.compute::batch() with the endpoint to send HTTP requests.
region	Character of length 1. region argument of paws.compute::batch() with an AWS region string such as "us-east-2". Serves as the region for both AWS Batch and CloudWatch. Tries to default to paws.common::get_config()$region, then to Sys.getenv("AWS_REGION") if unsuccessful, then Sys.getenv("AWS_REGION"), then Sys.getenv("AWS_DEFAULT_REGION").

IAM policies

In order for the AWS Batch `crew` monitor class to function properly, your IAM policy needs permission to perform the `SubmitJob`, `TerminateJob`, `ListJobs`, and `DescribeJobs` AWS Batch API calls. In addition, to download CloudWatch logs with the `log()` method, your IAM policy also needs permission to perform the `GetLogEvents` CloudWatch logs API call. For more information on AWS policies and permissions, please visit https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies.html.

See Also

Other monitor: crew_class_monitor_aws_batch

`crew_options_aws_batch` *AWS Batch options*

Description

Options for the AWS Batch controller.

Usage

```
crew_options_aws_batch(
  job_definition = "example",
  job_queue = "example",
  cpus = NULL,
  gpus = NULL,
  memory = NULL,
  memory_units = "gigabytes",
  config = list(),
  credentials = list(),
  endpoint = NULL,
  region = NULL,
  share_identifier = NULL,
  scheduling_priority_override = NULL,
  parameters = NULL,
  container_overrides = NULL,
  node_overrides = NULL,
  retry_strategy = NULL,
  propagate_tags = NULL,
  timeout = NULL,
  tags = NULL,
  eks_properties_override = NULL,
  verbose = FALSE
)
```

Arguments

job_definition	Character of length 1, name of the AWS Batch job definition to use. There is no default for this argument, and a job definition must be created prior to running the controller. Please see https://docs.aws.amazon.com/batch/ for details. To create a job definition, you will need to create a Docker-compatible image which can run R and crew. You may which to inherit from the images at https://github.com/rocker-org/rocker-versioned2.
job_queue	Character of length 1, name of the AWS Batch job queue to use. There is no default for this argument, and a job queue must be created prior to running the controller. Please see https://docs.aws.amazon.com/batch/ for details.
cpus	Positive numeric scalar, number of virtual CPUs to request per job. Can be NULL to go with the defaults in the job definition. Ignored if container_overrides is not NULL.
gpus	Positive numeric scalar, number of GPUs to request per job. Can be NULL to go with the defaults in the job definition. Ignored if container_overrides is not NULL.
memory	Positive numeric scalar, amount of random access memory (RAM) to request per job. Choose the units of memory with the memory_units argument. Fargate instances can only be certain discrete values of mebibytes, so please choose memory_units = "mebibytes" in that case. The memory argument can be NULL to go with the defaults in the job definition. Ignored if container_overrides is not NULL.
memory_units	Character string, units of memory of the memory argument. Can be "gigabytes" or "mebibytes". Fargate instances can only be certain discrete values of mebibytes, so please choose memory_units = "mebibytes" in that case.
config	Named list, config argument of paws.compute::batch() with optional configuration details.
credentials	Named list. credentials argument of paws.compute::batch() with optional credentials (if not already provided through environment variables such as AWS_ACCESS_KEY_ID).
endpoint	Character of length 1. endpoint argument of paws.compute::batch() with the endpoint to send HTTP requests.
region	Character of length 1. region argument of paws.compute::batch() with an AWS region string such as "us-east-2".
share_identifier	NULL or character of length 1. For details, visit https://www.paws-r-sdk.com/docs/batch_submit_job/ and the "AWS arguments" sections of this help file.
scheduling_priority_override	NULL or integer of length 1. For details, visit https://www.paws-r-sdk.com/docs/batch_submit_job/ and the "AWS arguments" sections of this help file.
parameters	NULL or a nonempty list. For details, visit https://www.paws-r-sdk.com/docs/batch_submit_job/ and the "AWS arguments" sections of this help file.
container_overrides	NULL or a nonempty named list of fields to override in the container specified in the job definition. Any overrides for the command field are ignored because crew.aws.batch needs to override the command to run the crew worker.

For more details, visit https://www.paws-r-sdk.com/docs/batch_submit_job/ and the "AWS arguments" sections of this help file.

`node_overrides` NULL or a nonempty named list. For more details, visit https://www.paws-r-sdk.com/docs/batch_submit_job/ and the "AWS arguments" sections of this help file.

`retry_strategy` NULL or a nonempty named list. For more details, visit https://www.paws-r-sdk.com/docs/batch_submit_job/ and the "AWS arguments" sections of this help file.

`propagate_tags` NULL or a logical of length 1. For more details, visit https://www.paws-r-sdk.com/docs/batch_submit_job/ and the "AWS arguments" sections of this help file.

`timeout` NULL or a nonempty named list. For more details, visit https://www.paws-r-sdk.com/docs/batch_submit_job/ and the "AWS arguments" sections of this help file.

`tags` NULL or a nonempty named list. For more details, visit https://www.paws-r-sdk.com/docs/batch_submit_job/ and the "AWS arguments" sections of this help file.

`eks_properties_override`
NULL or a nonempty named list. For more details, visit https://www.paws-r-sdk.com/docs/batch_submit_job/ and the "AWS arguments" sections of this help file.

`verbose` TRUE to print informative console messages, FALSE otherwise.

Value

A classed list of options for the controller.

Retryable options

Retryable options are deprecated in `crew.aws.batch` as of 2025-01-27 (version `0.0.8`).

Index