# Package 'starburst'

November 10, 2026

**Type** Package

**Title** Seamless AWS Cloud Bursting for Parallel R Workloads

**Version** 0.3.12

**Description** A 'future' backend that enables seamless execution of parallel R workloads on 'Amazon Web Services' ('AWS', <https://aws.amazon.com>), including 'EC2' and 'Fargate'. 'staRburst' handles environment synchronization, data transfer, quota management, and worker orchestration automatically, allowing users to scale from local execution to 100+ cloud workers with a single line of code change.

**License** Apache License 2.0

**Encoding** UTF-8

**Depends** R (>= 4.0.0)

**Imports** future (>= 1.40.0), globals, paws.compute, paws.storage, paws.management, paws.cost.management, paws.security.identity, qs2, uuid, renv, jsonlite, crayon, digest, base64enc, processx

**Suggests** future.apply, testthat (>= 3.0.0), knitr, rmarkdown, withr, mockery

**VignetteBuilder** knitr

**URL** https://starburst.ing, https://github.com/scttfrdmn/starburst

**BugReports** https://github.com/scttfrdmn/starburst/issues

**Config/roxygen2/version** 8.0.0

**NeedsCompilation** no

**Author** Scott Friedman [aut, cre]

**Maintainer** Scott Friedman <help@starburst.ing>

**Repository** CRAN

**Date/Publication** 2026-11-10 22:00:02 UTC

# Contents

future.starburst . . . 2
launchFuture.StarburstBackend . . . 3
listFutures.StarburstBackend . . . 4
nbrOfWorkers.StarburstBackend . . . 5
plan.starburst . . . 5
print.StarburstSessionStatus . . . 6
resolved.StarburstFuture . . . 7
result.StarburstFuture . . . 7
run.StarburstFuture . . . 8
session-api . . . 8
starburst . . . 9
StarburstBackend . . . 9
starburst_check_quota_request . . . 11
starburst_cleanup_ecr . . . 11
starburst_cluster . . . 12
starburst_config . . . 13
starburst_estimate . . . 14
starburst_is_configured . . . 15
starburst_list_sessions . . . 16
starburst_logs . . . 16
starburst_map . . . 17
starburst_quota_status . . . 19
starburst_rebuild_environment . . . 19
starburst_request_quota_increase . . . 20
starburst_session . . . 21
starburst_session_attach . . . 23
starburst_setup . . . 24
starburst_setup_ec2 . . . 25
starburst_status . . . 26
stopWorkers.StarburstBackend . . . 26

**Index** **28**

---

`future.starburst` *Create a Future using Starburst Backend*

---

## Description

This is the entry point called by the Future package when a plan(starburst) is active

## Usage

```
## S3 method for class 'starburst'
future(
  expr,
  envir = parent.frame(),
  substitute = TRUE,
  lazy = FALSE,
  seed = FALSE,
  globals = TRUE,
  packages = NULL,
  stdout = TRUE,
  conditions = "condition",
  label = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| expr | Expression to evaluate |
| envir | Environment for evaluation |
| substitute | Whether to substitute the expression |
| lazy | Whether to lazily evaluate (always FALSE for remote) |
| seed | Random seed |
| globals | Globals to export (TRUE for auto-detection, list for manual) |
| packages | Packages to load |
| stdout | Whether to capture stdout (TRUE, FALSE, or NA) |
| conditions | Character vector of condition classes to capture |
| label | Optional label for the future |
| ... | Additional arguments |

## Value

A StarburstFuture object

---

# launchFuture.StarburstBackend

*Launch a future on the Starburst backend*

---

## Description

Launch a future on the Starburst backend

### Usage

```
## S3 method for class 'StarburstBackend'
launchFuture(backend, future, ...)
```

### Arguments

| | |
|---|---|
| backend | A StarburstBackend object |
| future | The future object to launch |
| ... | Additional arguments |

### Value

The future object (invisibly)

---

## listFutures.StarburstBackend

*List futures for StarburstBackend*

---

### Description

List futures for StarburstBackend

### Usage

```
## S3 method for class 'StarburstBackend'
listFutures(backend, ...)
```

### Arguments

| | |
|---|---|
| backend | A StarburstBackend object |
| ... | Additional arguments |

### Value

List of futures (empty for this backend)

## nbrOfWorkers.StarburstBackend *Number of workers for StarburstBackend*

### Description

Number of workers for StarburstBackend

### Usage

```
## S3 method for class 'StarburstBackend'
nbrOfWorkers(evaluator)
```

### Arguments

| | |
|---|---|
| evaluator | A StarburstBackend object |

### Value

Number of workers

## plan.starburst *staRburst Future Backend*

### Description

A future backend for running parallel R workloads on AWS (EC2 or Fargate)

### Usage

```
## S3 method for class 'starburst'
plan(
  strategy,
  workers = 10,
  cpu = 4,
  memory = "8GB",
  region = NULL,
  timeout = 3600,
  auto_quota_request = interactive(),
  launch_type = "EC2",
  instance_type = "c7g.xlarge",
  use_spot = TRUE,
  warm_pool_timeout = 600,
  detached = FALSE,
  ...
)
```

### Arguments

| | |
|---|---|
| strategy | The starburst strategy marker (ignored, for S3 dispatch) |
| workers | Number of parallel workers |
| cpu | vCPUs per worker (1, 2, 4, 8, or 16) |
| memory | Memory per worker (supports GB notation, e.g., "8GB") |
| region | AWS region (default: from config or "us-east-1") |
| timeout | Maximum runtime in seconds (default: 3600) |
| auto_quota_request | Automatically request quota increases (default: interactive()) |
| launch_type | Launch type: EC2 or FARGATE (default: EC2) |
| instance_type | EC2 instance type when using EC2 launch type (default: c7g.xlarge) |
| use_spot | Use EC2 Spot instances for cost savings (default: TRUE) |
| warm_pool_timeout | Seconds to keep the EC2 instances running after the plan is shut down (e.g. by `plan(sequential)`), so a following run starts without waiting for new instances (default: 600). The instances keep billing while warm. The release is scheduled on AWS, so it happens even if R exits; `0` releases them when the plan ends. |
| detached | Use detached session mode (deprecated, use starburst_session instead) |
| ... | Additional arguments passed to future backend |

### Value

A future plan object

### Examples

```
if (starburst_is_configured()) {
  future::plan(starburst, workers = 50)
  results <- future.apply::future_lapply(1:100, function(i) i^2)
}
```

---

## print.StarburstSessionStatus

*Print method for session status*

---

### Description

Print method for session status

### Usage

```
## S3 method for class 'StarburstSessionStatus'
print(x, ...)
```

### Arguments

| | |
|---|---|
| x | A StarburstSessionStatus object |
| ... | Additional arguments (ignored) |

### Value

Invisibly returns x.

---

## resolved.StarburstFuture

*Check if StarburstFuture is Resolved*

---

### Description

Checks whether the future task has completed execution

### Usage

```
## S3 method for class 'StarburstFuture'
resolved(x, ...)
```

### Arguments

| | |
|---|---|
| x | A StarburstFuture object |
| ... | Additional arguments |

### Value

Logical indicating if the future is resolved

---

## result.StarburstFuture

*Get Result from StarburstFuture*

---

### Description

Retrieves the result from a resolved future

### Usage

```
## S3 method for class 'StarburstFuture'
result(future, ...)
```

### Arguments

| | |
|---|---|
| future | A StarburstFuture object |
| ... | Additional arguments |

### Value

A FutureResult object

---

## run.StarburstFuture *Run a StarburstFuture*

---

### Description

Submits the future task to AWS for execution (EC2 by default, or Fargate)

### Usage

```
## S3 method for class 'StarburstFuture'
run(future, ...)
```

### Arguments

| | |
|---|---|
| future | A StarburstFuture object |
| ... | Additional arguments |

### Value

The future object (invisibly)

---

## session-api *Detached Session API*

---

### Description

User-facing API for creating and managing detached sessions

## starburst *Starburst strategy marker*

### Description

This function should never be called directly. Use plan(starburst, ...) instead.

### Usage

```
starburst(...)
```

### Arguments

| | |
|---|---|
| ... | Arguments passed to StarburstBackend() |

### Value

Does not return a value; always signals an error if called directly. This object exists as a strategy marker for plan.

### See Also

plan.starburst for the worker, backend, and cost arguments.

### Examples

```
if (starburst_is_configured()) {
  # Pass `starburst` to plan(); existing future code then runs on AWS workers
  future::plan(starburst, workers = 4)
  f <- future::future(Sys.info()[["nodename"]])
  future::value(f)
  future::plan(future::sequential)
}
```

## StarburstBackend *Starburst Future Backend*

### Description

A future backend for running parallel R workloads on AWS ECS

## Usage

```
StarburstBackend(
  workers = 10,
  cpu = 4,
  memory = "8GB",
  region = NULL,
  timeout = 3600,
  launch_type = "EC2",
  instance_type = "c6a.large",
  use_spot = FALSE,
  warm_pool_timeout = 600,
  ...
)
```

## Arguments

| | |
|---|---|
| workers | Number of parallel workers |
| cpu | vCPUs per worker (1, 2, 4, 8, or 16) |
| memory | Memory per worker (supports GB notation, e.g., "8GB") |
| region | AWS region (default: from config or "us-east-1") |
| timeout | Maximum runtime in seconds (default: 3600) |
| launch_type | "EC2" or "FARGATE" (default: "EC2") |
| instance_type | EC2 instance type (e.g., "c6a.large") |
| use_spot | Use spot instances (default: FALSE) |
| warm_pool_timeout | Seconds to keep the EC2 pool warm after the plan ends (default: 600) |
| ... | Additional arguments |

## Details

You do not normally call this constructor yourself: `plan(starburst, ...)` creates the backend for you (see `plan.starburst`, whose defaults are the ones users see). It is exported as an extension point for code that builds on the **future** backend API.

## Value

A StarburstBackend object

## See Also

`plan.starburst`

## starburst_check_quota_request *Monitor quota increase request*

### Description

Monitor quota increase request

### Usage

```
starburst_check_quota_request(case_id, region = NULL)
```

### Arguments

| | |
|---|---|
| case_id | Case ID from quota increase request |
| region | AWS region |

### Value

Invisibly returns the quota request details, or NULL on error.

### Examples

```
if (starburst_is_configured()) {
  starburst_check_quota_request("case-12345")
}
```

## starburst_cleanup_ecr *Clean up staRburst ECR images*

### Description

Manually delete Docker images from ECR to save storage costs. Images will be rebuilt on next use (adds 3-5 min delay).

### Usage

```
starburst_cleanup_ecr(force = FALSE, region = NULL)
```

### Arguments

| | |
|---|---|
| force | Delete all images immediately, ignoring TTL |
| region | AWS region (default: from config) |

## Value

Invisibly returns TRUE on success or FALSE if not configured.

## Examples

```
if (starburst_is_configured()) {
  # Delete images past TTL
  starburst_cleanup_ecr()

  # Delete all images immediately (save $0.50/month)
  starburst_cleanup_ecr(force = TRUE)
}
```

---

# starburst_cluster *Create a Starburst Cluster*

---

## Description

Creates a cluster object for managing AWS workers (EC2 by default, or Fargate) using the staRburst Future backend.

## Usage

```
starburst_cluster(
  workers = 10,
  cpu = 4,
  memory = "8GB",
  region = NULL,
  timeout = 3600,
  launch_type = "EC2",
  instance_type = "c7g.xlarge",
  use_spot = TRUE
)
```

## Arguments

| | |
|---|---|
| workers | Number of parallel workers |
| cpu | CPU units per worker |
| memory | Memory per worker |
| region | AWS region |
| timeout | Maximum runtime in seconds |
| launch_type | Compute backend: "EC2" (default) or "FARGATE" |
| instance_type | EC2 instance type when launch_type = "EC2" (default: "c7g.xlarge"). Worker CPU architecture follows the instance type (Graviton *g.* = ARM64, Intel/AMD = x86_64); there is no separate platform argument. |
| use_spot | Use EC2 Spot instances for cost savings (default: TRUE) |

## Value

A starburst_cluster object

## Examples

```
if (starburst_is_configured()) {
  cluster <- starburst_cluster(workers = 20)
  results <- cluster$map(data, function(x) x * 2)

  # Fargate backend instead of the EC2 default
  fg <- starburst_cluster(workers = 20, launch_type = "FARGATE")
}
```

---

# starburst_config *Configure staRburst options*

---

## Description

Reads and updates the persisted staRburst configuration. Call with no arguments to leave settings unchanged (it still returns the current config invisibly); pass one or more of the arguments below to update them.

## Usage

```
starburst_config(
  max_hourly_cost = NULL,
  cost_alert_threshold = NULL,
  auto_cleanup_s3 = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| max_hourly_cost | Maximum estimated **hourly** cost (USD/hour) for a job. Jobs whose estimated hourly rate exceeds this error before launching. This is a rate limit, not a total-job-cost cap. NULL leaves it unchanged. |
| cost_alert_threshold | Estimated **hourly** cost (USD/hour) at which a warning is emitted. NULL leaves it unchanged. |
| auto_cleanup_s3 | Logical; automatically delete a job's S3 task/result objects after completion. NULL leaves it unchanged. |
| ... | Additional user-settable keys merged into the config. Recognized keys: use_public_base Logical; pull the public base image instead of building a private one (see starburst_setup). |

`ecr_image_ttl_days` Integer; lifecycle age at which cached ECR images are expired.

## Details

Other keys in the stored config are **infrastructure-managed** — written by `starburst_setup`/`starburst_setup_ec2` and not intended to be set by hand: `region`, `bucket`, `cluster`, `ecr_repository`, `aws_account_id`, `execution_role_arn`, `task_role_arn`, `subnets`, and `security_groups`. Use `starburst_status` to inspect the effective configuration read-only.

## Value

Invisibly returns the updated configuration list.

## See Also

`starburst_status` to view config without changing it; `starburst_setup` for initial provisioning.

## Examples

```
if (starburst_is_configured()) {
  # Update cost guardrails (both are hourly rates, USD/hour)
  starburst_config(
    max_hourly_cost = 10,
    cost_alert_threshold = 5
  )

  # Read the current config without changing anything
  cfg <- starburst_config()
}
```

---

# `starburst_estimate` *Estimate Cloud Performance and Cost*

---

## Description

Runs a small sample of tasks locally to estimate cloud execution time and cost. Provides informed prediction before spending money on cloud execution.

## Usage

```
starburst_estimate(
  .x,
  .f,
  workers = 10,
  cpu = 2,
  memory = "8GB",
  platform = "X86_64",
```

```
  sample_size = 10,
  region = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| .x | A vector or list to iterate over |
| .f | A function to apply to each element |
| workers | Number of parallel workers to estimate for |
| cpu | CPU units per worker (1, 2, 4, 8, or 16) |
| memory | Memory per worker (e.g., "8GB") |
| platform | CPU architecture: "X86_64" (default) or "ARM64" (Graviton3) |
| sample_size | Number of items to run locally for estimation (default: 10) |
| region | AWS region |
| ... | Additional arguments passed to .f |

## Value

Invisible list with estimates, prints summary to console

## Examples

```
if (starburst_is_configured()) {
  # Estimate before running
  starburst_estimate(1:1000, expensive_function, workers = 50)

  # Then decide whether to proceed
  results <- starburst_map(1:1000, expensive_function, workers = 50)
}
```

---

# starburst_is_configured

*Check if staRburst is configured*

---

## Description

Returns TRUE if `starburst_setup()` has been run, the configuration file exists, and AWS credentials are available. Useful for guarding example code that requires AWS credentials.

## Usage

```
starburst_is_configured()
```

## Value

TRUE if configured and credentials are available, FALSE otherwise.

## Examples

```
starburst_is_configured()
```

---

`starburst_list_sessions` *List All Sessions*

---

## Description

List all detached sessions in S3

## Usage

```
starburst_list_sessions(region = NULL)
```

## Arguments

| | |
|---|---|
| region | AWS region (default: from config) |

## Value

Data frame with session information

## Examples

```
if (starburst_is_configured()) {
  sessions <- starburst_list_sessions()
  print(sessions)
}
```

---

`starburst_logs` *View worker logs*

---

## Description

View worker logs

## Usage

```
starburst_logs(task_id = NULL, cluster_id = NULL, last_n = 50, region = NULL)
```

### Arguments

| | |
|---|---|
| task_id | Optional task ID to view logs for specific task |
| cluster_id | Optional cluster ID to view logs for specific cluster |
| last_n | Number of last log lines to show (default: 50) |
| region | AWS region (default: from config) |

### Value

Invisibly returns the list of log events, or NULL if no events were found.

### Examples

```
if (starburst_is_configured()) {
  # View recent logs
  starburst_logs()

  # View logs for specific task
  starburst_logs(task_id = "abc-123")

  # View last 100 lines
  starburst_logs(last_n = 100)
}
```

---

## starburst_map *Map a Function Over Data on AWS Workers*

---

### Description

Parallel map function that executes across AWS workers (EC2 by default, or Fargate) using the staRburst Future backend.

### Usage

```
starburst_map(
  .x,
  .f,
  workers = 10,
  cpu = 4,
  memory = "8GB",
  region = NULL,
  timeout = 3600,
  launch_type = "EC2",
  instance_type = "c7g.xlarge",
  use_spot = TRUE,
  .progress = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| .x | A vector or list to iterate over |
| .f | A function to apply to each element |
| workers | Number of parallel workers (default: 10) |
| cpu | CPU units per worker (1, 2, 4, 8, or 16) |
| memory | Memory per worker (e.g., 8GB) |
| region | AWS region |
| timeout | Maximum runtime in seconds per task |
| launch_type | Compute backend: "EC2" (default) or "FARGATE" |
| instance_type | EC2 instance type when launch_type = "EC2" (default: "c7g.xlarge"). The worker CPU architecture follows the instance type — Graviton types (e.g. c7g.*) run ARM64, Intel/AMD types (e.g. c7i.*) run x86_64 — so there is no separate platform argument. |
| use_spot | Use EC2 Spot instances for cost savings (default: TRUE) |
| .progress | Show progress bar (default: TRUE) |
| ... | Additional arguments passed to .f |

## Value

A list of results, one per element of .x

## Examples

```
if (starburst_is_configured()) {
  # Simple parallel computation
  results <- starburst_map(1:100, function(x) x^2, workers = 10)

  # With custom configuration
  results <- starburst_map(
    data_list,
    expensive_function,
    workers = 50,
    cpu = 4,
    memory = "8GB"
  )

  # Use the Fargate backend instead of the EC2 default
  results <- starburst_map(1:100, function(x) x^2,
                           workers = 10, launch_type = "FARGATE")
}
```

## starburst_quota_status *Show quota status*

### Description

Show quota status

### Usage

```
starburst_quota_status(region = NULL)
```

### Arguments

| | |
|---|---|
| region | AWS region (default: from config) |

### Value

Invisibly returns a list with quota information including current limit, usage, and any pending requests.

### Examples

```
if (starburst_is_configured()) {
  starburst_quota_status()
}
```

## starburst_rebuild_environment *Rebuild environment image*

### Description

Rebuild environment image

### Usage

```
starburst_rebuild_environment(region = NULL, force = FALSE)
```

### Arguments

| | |
|---|---|
| region | AWS region (default: from config) |
| force | Force rebuild even if current environment hasn't changed |

## Value

Invisibly returns NULL. Called for its side effect of rebuilding and pushing the Docker environment image.

## Examples

```
if (starburst_is_configured()) {
  starburst_rebuild_environment()
}
```

---

`starburst_request_quota_increase`

*Request quota increase (user-facing)*

---

## Description

Request quota increase (user-facing)

## Usage

```
starburst_request_quota_increase(vcpus = 500, region = NULL)
```

## Arguments

| | |
|---|---|
| vcpus | Desired vCPU quota |
| region | AWS region (default: from config) |

## Value

Invisibly returns TRUE if the increase was requested, FALSE if already sufficient or cancelled.

## Examples

```
if (starburst_is_configured()) {
  starburst_request_quota_increase(vcpus = 500)
}
```

`starburst_session` *Create a Detached Starburst Session*

## Description

Creates a new detached session that can run computations independently of your R session. You can close R and reattach later to collect results.

## Usage

```
starburst_session(
  workers = 10,
  cpu = 4,
  memory = "8GB",
  region = NULL,
  timeout = 3600,
  session_timeout = 3600,
  absolute_timeout = 86400,
  launch_type = "EC2",
  instance_type = "c7g.xlarge",
  use_spot = TRUE,
  warm_pool_timeout = 600
)
```

## Arguments

| | |
|---|---|
| `workers` | Number of parallel workers (default: 10) |
| `cpu` | vCPUs per worker (default: 4) |
| `memory` | Memory per worker, e.g., "8GB" (default: "8GB") |
| `region` | AWS region (default: from config or "us-east-1") |
| `timeout` | Task timeout in seconds (default: 3600) |
| `session_timeout` | Active timeout in seconds (default: 3600) |
| `absolute_timeout` | Maximum session lifetime in seconds (default: 86400) |
| `launch_type` | "EC2" or "FARGATE" (default: "EC2") |
| `instance_type` | EC2 instance type for EC2 launch (default: "c7g.xlarge") |
| `use_spot` | Use spot instances for EC2 (default: TRUE) |
| `warm_pool_timeout` | Seconds to keep the EC2 instances running after `session$cleanup()` (default: 600). They keep billing while warm; the release is scheduled on AWS. `0` releases them at cleanup. |

## Value

A `StarburstSession` object (also carrying `$session_id`, the handle you pass to `starburst_session_attach`) with methods:

`submit(expr, globals = NULL, packages = NULL)` Submit one task (a quoted expression). Returns the task id. Call repeatedly to fan out work.

`status()` Return a progress summary (counts of pending / running / completed / failed tasks). Safe to call from a fresh R session after reattaching.

`collect(wait = FALSE)` Retrieve results, keyed by task id, in submission order. With `wait = FALSE` returns whatever has finished so far; `wait = TRUE` blocks until every submitted task is terminal. Every terminal task appears: a success carries its return value; a **failed** task carries a structured failure `list(error = TRUE, message = ..., value = NULL, task_id = ...)` so failures are visible rather than silently dropped.

`extend(seconds = 3600)` Extend the active/absolute timeout of a still-running session.

`cleanup(stop_workers = TRUE, force = FALSE)` Stop the session's workers and mark it terminated. By default **S3 objects are preserved** (so you can still inspect/collect); pass `force = TRUE` to also delete the session's S3 task/result objects. Otherwise the session self-terminates at `absolute_timeout`.

## Lifecycle

`starburst_session()` launches workers immediately and returns a handle. Submit tasks, then either poll `status()`/`collect()` in the same session, or record `session$session_id`, close R, and later `starburst_session_attach(session_id)` to reconnect and collect. A session ends when you call `cleanup()`, when `session_timeout` elapses with no activity, or at `absolute_timeout` — whichever comes first.

## Failure behavior

A failed task is recorded (surfaced via `status()`) and does not abort the others; `collect()` returns it as a structured failure entry (`error = TRUE` with a `message`) alongside the successful results, rather than dropping it or raising. If the client dies, workers keep running against S3 until a timeout, which is what makes reattaching possible. `cleanup()` is the only thing that frees resources early — sessions do not auto-clean on garbage collection.

## See Also

`starburst_session_attach`, `starburst_list_sessions`; `starburst_map` for ephemeral (non-detached) fan-out.

## Examples

```
if (starburst_is_configured()) {
  # Create detached session
  session <- starburst_session(workers = 10)

  # Submit tasks
  task_ids <- lapply(1:100, function(i) {
    session$submit(quote(expensive_computation(i)))
```

```
  })

  # Close R and come back later...
  session_id <- session$session_id

  # Reattach
  session <- starburst_session_attach(session_id)

  # Collect results
  results <- session$collect(wait = TRUE)
}
```

---

`starburst_session_attach` *Reattach to Existing Session*

---

## Description

Reattach to a previously created detached session

## Usage

```
starburst_session_attach(session_id, region = NULL)
```

## Arguments

| | |
|---|---|
| `session_id` | Session identifier |
| `region` | AWS region (default: from config) |

## Value

A StarburstSession object

## Examples

```
if (starburst_is_configured()) {
  session <- starburst_session_attach("session-abc123")
  status <- session$status()
  results <- session$collect()
}
```

`starburst_setup` *Setup staRburst*

## Description

One-time configuration to set up AWS resources for staRburst

## Usage

```
starburst_setup(
  region = "us-east-1",
  force = FALSE,
  use_public_base = TRUE,
  ecr_image_ttl_days = NULL,
  build_image = TRUE,
  setup_ec2 = TRUE
)
```

## Arguments

| | |
|---|---|
| `region` | AWS region (default: "us-east-1") |
| `force` | Force re-setup even if already configured |
| `use_public_base` | Use public base Docker images (default: TRUE). Set to FALSE to build private base images in your ECR. |
| `ecr_image_ttl_days` | Number of days to keep Docker images in ECR (default: NULL = never delete). AWS will automatically delete images older than this many days. This prevents surprise costs if you stop using staRburst. Recommended: 30 days for regular users, 7 days for occasional users. When images are deleted, they will be rebuilt on next use (adds 3-5 min). |
| `build_image` | Build the worker environment image during setup (default: TRUE). Set to FALSE to provision AWS resources (S3/ECR/ECS/VPC), write config, and check quotas without triggering the multi-minute Docker image build. The image is then built lazily on first worker launch via `ensure_environment()`. Useful for CI / connectivity checks. |
| `setup_ec2` | Provision the EC2 capacity provider and Auto Scaling Group for the default instance type during setup (default: TRUE). This is what makes the default EC2 backend work out of the box — without it, the first `starburst_map()`/`plan(starburst)` run on EC2 would fail because no capacity provider exists. It creates infrastructure at `DesiredCapacity = 0` (no billable instances are launched during setup). Set to FALSE if you only use the Fargate backend, which needs no capacity provider. Provision additional instance types later with `starburst_setup_ec2`. |

## Value

Invisibly returns the configuration list.

## Examples

```
if (starburst_is_configured()) {
  # Default: keep images forever (~$0.50/month idle cost)
  starburst_setup()

  # Auto-delete images after 30 days (saves money if you stop using it)
  starburst_setup(ecr_image_ttl_days = 30)

  # Use private base images with 7-day cleanup
  starburst_setup(use_public_base = FALSE, ecr_image_ttl_days = 7)

  # Provision resources without building the image (fast; CI / connectivity checks)
  starburst_setup(build_image = FALSE)
}
```

---

# starburst_setup_ec2 *Setup EC2 capacity providers for staRburst*

---

## Description

One-time setup for EC2 launch type. Creates IAM roles, instance profiles, and capacity providers for specified instance types.

## Usage

```
starburst_setup_ec2(
  region = "us-east-1",
  instance_types = c("c7g.xlarge", "c7i.xlarge"),
  force = FALSE
)
```

## Arguments

| | |
|---|---|
| region | AWS region (default: "us-east-1") |
| instance_types | Character vector of instance types to setup (default: c("c7g.xlarge", "c7i.xlarge")) |
| force | Force re-setup even if already configured |

## Value

Invisibly returns TRUE on success or FALSE on failure or cancellation.

## Examples

```
if (starburst_is_configured()) {
  # Setup with default instance types (Graviton and Intel)
  starburst_setup_ec2()

  # Setup with custom instance types
  starburst_setup_ec2(instance_types = c("c7g.2xlarge", "r7g.xlarge"))
}
```

---

# starburst_status *Show staRburst status*

---

## Description

Prints the configured region, S3 bucket, and ECR repository, plus current Fargate vCPU quota usage (this makes AWS API calls).

## Usage

```
starburst_status()
```

## Value

Invisibly returns a list with current configuration and quota information.

## Examples

```
if (starburst_is_configured()) {
  starburst_status()
}
```

---

# stopWorkers.StarburstBackend *Stop workers for StarburstBackend*

---

## Description

Called by **future** when the plan changes (e.g. `plan(sequential)`): stops this backend's remaining tasks, cleans up its S3 objects, and holds the EC2 pool for `warm_pool_timeout` before releasing it.

## Usage

```
## S3 method for class 'StarburstBackend'
stopWorkers(backend, ...)
```

## Arguments

| | |
|---|---|
| backend | A StarburstBackend object |
| ... | Additional arguments |

## Value

TRUE (invisibly)

# Index


future.starburst, 2

launchFuture.StarburstBackend, 3
listFutures.StarburstBackend, 4

nbrOfWorkers.StarburstBackend, 5

plan, *9*
plan.starburst, 5, *9, 10*
print.StarburstSessionStatus, 6

resolved.StarburstFuture, 7
result.StarburstFuture, 7
run.StarburstFuture, 8

session-api, 8
starburst, 9
starburst_check_quota_request, 11
starburst_cleanup_ecr, 11
starburst_cluster, 12
starburst_config, 13
starburst_estimate, 14
starburst_is_configured, 15
starburst_list_sessions, 16, *22*
starburst_logs, 16
starburst_map, 17, *22*
starburst_quota_status, 19
starburst_rebuild_environment, 19
starburst_request_quota_increase, 20
starburst_session, 21
starburst_session_attach, *22*, 23
starburst_setup, *13, 14*, 24
starburst_setup_ec2, *14*, *24*, 25
starburst_status, *14*, 26
StarburstBackend, 9
stopWorkers.StarburstBackend, 26