The REST API is now versioned. For more information, see "About API versioning."

Deployments

Use the REST API to create and delete deployments and deployment environments.

About deployments

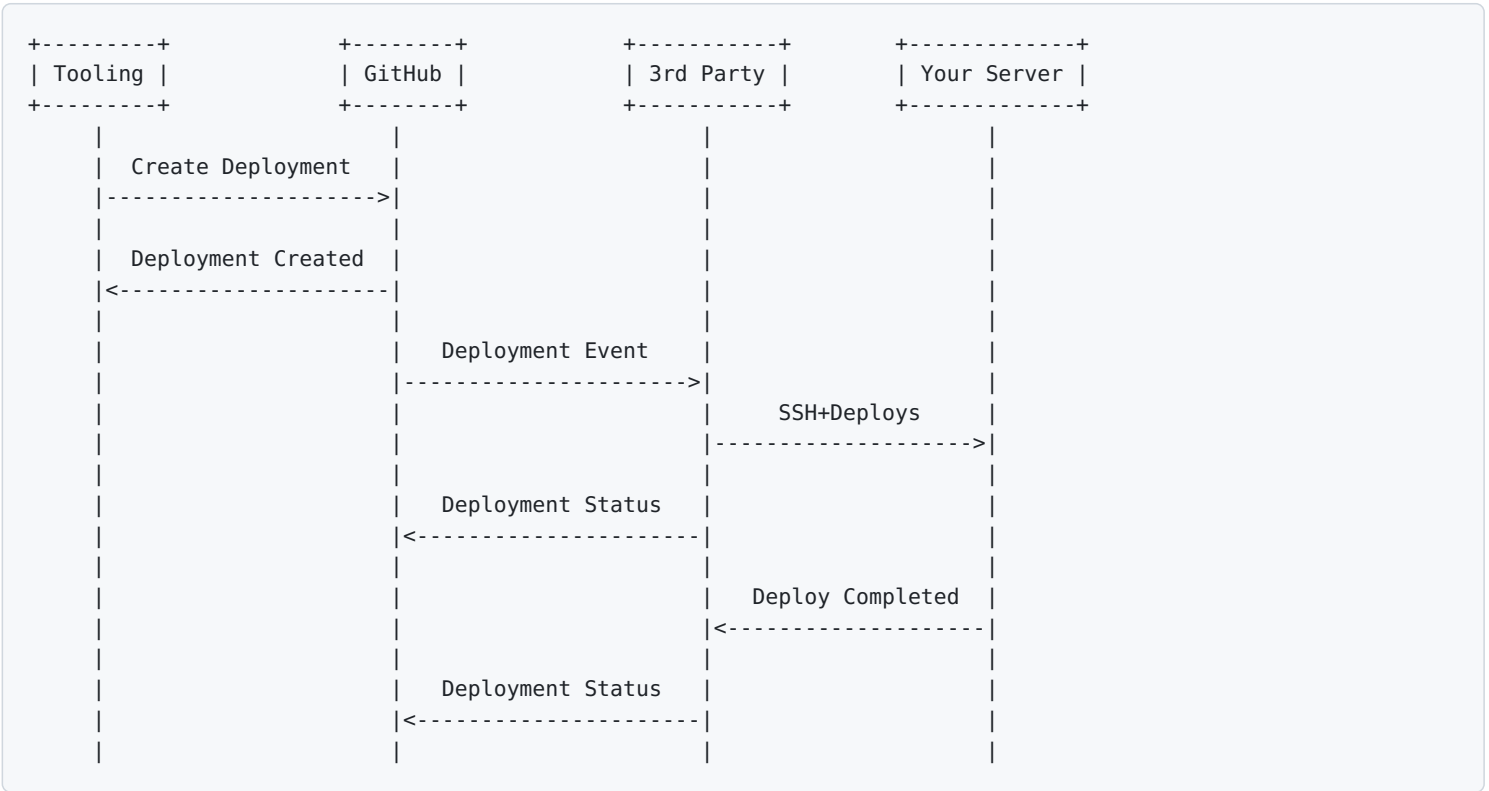
Deployments are requests to deploy a specific ref (branch, SHA, tag). GitHub dispatches a `deployment` event that external services can listen for and act on when new deployments are created. Deployments enable developers and organizations to build loosely coupled tooling around deployments, without having to worry about the implementation details of delivering different types of applications (e.g., web, native).

Deployment statuses allow external services to mark deployments with an `error`, `failure`, `pending`, `in_progress`, `queued`, or `success` state that systems listening to `deployment_status` events can consume.

Deployment statuses can also include an optional `description` and `log_url`, which are highly recommended because they make deployment statuses more useful. The `log_url` is the full URL to the deployment output, and the `description` is a high-level summary of what happened with the deployment.

GitHub dispatches `deployment` and `deployment_status` events when new deployments and deployment statuses are created. These events allow third-party integrations to receive and respond to deployment requests, and update the status of a deployment as progress is made.

Below is a simple sequence diagram for how these interactions would work.



Keep in mind that GitHub is never actually accessing your servers. It's up to your third-party integration to interact with deployment events. Multiple systems can listen for deployment events, and it's up to each of those systems to decide whether they're responsible for pushing the code out to your servers, building native code, etc.

Note that the `repo_deployment` OAuth scope grants targeted access to deployments and deployment statuses **without** granting access to repository code, while the `public_repo` and `repo` scopes grant permission to code as well.

Inactive deployments

When you set the state of a deployment to `success`, then all prior non-transient, non-production environment deployments in the same repository with the same environment name will become `inactive`. To avoid this, you can set `auto_inactive` to `false` when creating the deployment status.

You can communicate that a transient environment no longer exists by setting its `state` to `inactive`. Setting the `state` to `inactive` shows the deployment as `destroyed` in GitHub and removes access to it.

List deployments

Works with GitHub Apps

Simple filtering of deployments is available via query parameters:

Parameters for "List deployments"

Headers

`accept` string

Setting to `application/vnd.github+json` is recommended.

Path parameters

`owner` string Required

The account owner of the repository. The name is not case sensitive.

`repo` string Required

The name of the repository without the `.git` extension. The name is not case sensitive.

Query parameters

`sha` string

The SHA recorded at creation time.

Default: `none`

`ref` string

The name of the ref. This can be a branch, tag, or SHA.

Default: `none`

`task` string

The name of the task for the deployment (e.g., `deploy` or `deploy:migrations`).

Default: `none`

`environment` string or null

The name of the environment that was deployed to (e.g., `staging` or `production`).

Default: `none`

per_page integer

The number of results per page (max 100).

Default: `30`

page integer

Page number of the results to fetch.

Default: `1`

HTTP response status codes for "List deployments"

Status code	Description
`200`	OK

Code samples for "List deployments"

`GET` `/repos/{owner}/{repo}/deployments`

cURL | JavaScript | GitHub CLI

```
curl -L \ -H "Accept: application/vnd.github+json" \ -H "Authorization: Bearer <YOUR-TOKEN>" \ -H "X-GitHub-Api-Version: 2022-11-28" \ https://api.github.com/repos/OWNER/REPO/deployments
```

Response

Example response | Response schema

Status: 200


```
[ { "url": "https://api.github.com/repos/octocat/example/deployments/1", "id": 1, "node_id": "MDEwOkRlcGxveW1lbnQx", "sha": "a84d88e7554fc1fa21bcbc4efae3c782a70d2b9d", "ref": "topic-branch", "task": "deploy", "payload": {}, "original_environment": "staging", "environment": "production", "description": "Deploy request from hubot", "creator": { "login": "octocat", "id": 1, "node_id": "MDQ6VXNlcjE=", "avatar_url": "https://github.com/images/error/octocat_happy.gif", "gravatar_id": "", "url": "https://api.github.com/users/octocat", "html_url": "https://github.com/octocat", "followers_url": "https://api.github.com/users/octocat/followers", "following_url": "https://api.github.com/users/octocat/following{/other_user}",
```


Create a deployment

Works with GitHub Apps

Deployments offer a few configurable parameters with certain defaults.

The `ref` parameter can be any named branch, tag, or SHA. At GitHub Enterprise Cloud we often deploy branches and verify them before we merge a pull request.

The `environment` parameter allows deployments to be issued to different runtime environments. Teams often have multiple environments for verifying their applications, such as `production`, `staging`, and `qa`. This parameter makes it easier to track which environments have requested deployments. The default environment is `production`.

The `auto_merge` parameter is used to ensure that the requested ref is not behind the repository's default branch. If the

ref *is* behind the default branch for the repository, we will attempt to merge it for you. If the merge succeeds, the API will return a successful merge commit. If merge conflicts prevent the merge from succeeding, the API will return a failure response.

By default, commit statuses for every submitted context must be in a `success` state. The `required_contexts` parameter allows you to specify a subset of contexts that must be `success`, or to specify contexts that have not yet been submitted. You are not required to use commit statuses to deploy. If you do not require any contexts or create any commit statuses, the deployment will always succeed.

The `payload` parameter is available for any extra information that a deployment system might need. It is a JSON text field that will be passed on when a deployment event is dispatched.

The `task` parameter is used by the deployment system to allow different execution paths. In the web world this might be `deploy:migrations` to run schema changes on the system. In the compiled world this could be a flag to compile an application with debugging enabled.

Users with `repo` or `repo_deployment` scopes can create a deployment for a given ref.

Merged branch response:

You will see this response when GitHub automatically merges the base branch into the topic branch instead of creating a deployment. This auto-merge happens when:

- Auto-merge option is enabled in the repository
- Topic branch does not include the latest changes on the base branch, which is `master` in the response example
- There are no merge conflicts

If there are no new commits in the base branch, a new request to create a deployment should give a successful response.

Merge conflict response:

This error happens when the `auto_merge` option is enabled and when the default branch (in this case `master`), can't be merged into the branch that's being deployed (in this case `topic-branch`), due to merge conflicts.

Failed commit status checks:

This error happens when the `required_contexts` parameter indicates that one or more contexts need to have a `success` status for the commit to be deployed, but one or more of the required contexts do not have a state of `success`.

Parameters for "Create a deployment"

Headers

`accept` string

Setting to `application/vnd.github+json` is recommended.

Path parameters

`owner` string Required

The account owner of the repository. The name is not case sensitive.

`repo` string Required

The name of the repository without the `.git` extension. The name is not case sensitive.

Body parameters

`ref` string Required

The ref to deploy. This can be a branch, tag, or SHA.

`task` string

Specifies a task to execute (e.g., `deploy` or `deploy:migrations`).

Default: `deploy`

`auto_merge` boolean

Attempts to automatically merge the default branch into the requested ref, if it's behind the default branch.

Default: `true`

`required_contexts` array of strings

The status contexts to verify against commit status checks. If you omit this parameter, GitHub verifies all unique contexts before creating a deployment. To bypass checking entirely, pass an empty array. Defaults to all unique contexts.

`payload` object or string

JSON payload with extra information about the deployment.

`environment` string

Name for the target deployment environment (e.g., `production`, `staging`, `qa`).

Default: `production`

`description` string or null

Short description of the deployment.

Default: `""`

`transient_environment` boolean

Specifies if the given environment is specific to the deployment and will no longer exist at some point in the future. Default: `false`

Default: `false`

`production_environment` boolean

Specifies if the given environment is one that end-users directly interact with. Default: `true` when `environment` is `production` and `false` otherwise.

HTTP response status codes for "Create a deployment"

Status code	Description
`201`	Created
`202`	Merged branch response
`409`	Conflict when there is a merge conflict or the commit's status checks failed
`422`	Validation failed, or the endpoint has been spammed.

Code samples for "Create a deployment"

`POST` `/repos/{owner}/{repo}/deployments`

cURL | JavaScript | GitHub CLI

```
curl -L \ -X POST \ -H "Accept: application/vnd.github+json" \ -H "Authorization: Bearer <YOUR-TOKEN>" \ -H "X-GitHub-Api-Version: 2022-11-28" \ https://api.github.com/repos/OWNER/REPO/deployments \ -d '{"ref":"topic-branch","payload":"{ \"deploy\": \"migrate\"
```

}","description":"Deploy request from hubot"}'

Simple example

Example response | Response schema

Status: 201

{ "url": "https://api.github.com/repos/octocat/example/deployments/1", "id": 1, "node_id": "MDEwOkRlcGxveW1lbnQx", "sha": "a84d88e7554fc1fa21bcbc4efae3c782a70d2b9d", "ref": "topic-branch", "task": "deploy", "payload": {}, "original_environment": "staging", "environment": "production", "description": "Deploy request from hubot", "creator": { "login": "octocat", "id": 1, "node_id": "MDQ6VXNlcjE=", "avatar_url": "https://github.com/images/error/octocat_happy.gif", "gravatar_id": "", "url": "https://api.github.com/users/octocat", "html_url": "https://github.com/octocat", "followers_url": "https://api.github.com/users/octocat/followers", "following_url": "https://api.github.com/users/octocat/following{/other_user}",

Get a deployment

Works with GitHub Apps

Parameters for "Get a deployment"

Headers

accept string

Setting to `application/vnd.github+json` is recommended.

Path parameters

owner string Required

The account owner of the repository. The name is not case sensitive.

repo string Required

The name of the repository without the `.git` extension. The name is not case sensitive.

deployment_id integer Required

deployment_id parameter

HTTP response status codes for "Get a deployment"

Status code	Description
200	OK
404	Resource not found

Code samples for "Get a deployment"

GET /repos/{owner}/{repo}/deployments/{deployment_id}

GET /repos/{owner}/{repo}/deployments/{deployment_id}

cURL | JavaScript | GitHub CLI

```
curl -L \ -H "Accept: application/vnd.github+json" \ -H "Authorization: Bearer <YOUR-TOKEN>" \ -H "X-GitHub-Api-Version: 2022-11-28"
\ https://api.github.com/repos/OWNER/REPO/deployments/DEPLOYMENT_ID
```

Response

Example response | Response schema

```
Status: 200
```


```
{ "url": "https://api.github.com/repos/octocat/example/deployments/1", "id": 1, "node_id": "MDEwOkRlcGxveW1lbnQx", "sha":
"a84d88e7554fc1fa21bcbc4efae3c782a70d2b9d", "ref": "topic-branch", "task": "deploy", "payload": {}, "original_environment":
"staging", "environment": "production", "description": "Deploy request from hubot", "creator": { "login": "octocat", "id": 1,
"node_id": "MDQ6VXNlcjE=", "avatar_url": "https://github.com/images/error/octocat_happy.gif", "gravatar_id": "", "url":
"https://api.github.com/users/octocat", "html_url": "https://github.com/octocat", "followers_url":
"https://api.github.com/users/octocat/followers", "following_url": "https://api.github.com/users/octocat/following{/other_user}",
```


Delete a deployment

Works with GitHub Apps

If the repository only has one deployment, you can delete the deployment regardless of its status. If the repository has more than one deployment, you can only delete inactive deployments. This ensures that repositories with multiple deployments will always have an active deployment. Anyone with `repo` or `repo_deployment` scopes can delete a deployment.

To set a deployment as inactive, you must:

- Create a new deployment that is active so that the system has a record of the current state, then delete the previously active deployment.
- Mark the active deployment as inactive by adding any non-successful deployment status.

For more information, see "Create a deployment" and "Create a deployment status."

Parameters for "Delete a deployment"

Headers

accept string

Setting to `application/vnd.github+json` is recommended.

Path parameters

owner string Required

The account owner of the repository. The name is not case sensitive.

repo string Required

The name of the repository without the `.git` extension. The name is not case sensitive.

deployment_id integer Required

deployment_id parameter

HTTP response status codes for "Delete a deployment"

Status code	Description
204	No Content
404	Resource not found
422	Validation failed, or the endpoint has been spammed.

Code samples for "Delete a deployment"

DELETE /repos/{owner}/{repo}/deployments/{deployment_id}

cURL JavaScript GitHub CLI

```
curl -L \ -X DELETE \ -H "Accept: application/vnd.github+json" \ -H "Authorization: Bearer <YOUR-TOKEN>" \ -H "X-GitHub-Api-Version: 2022-11-28" \ https://api.github.com/repos/OWNER/REPO/deployments/DEPLOYMENT_ID
```

Response

```
Status: 204
```