Overview

The Nexa Platform API is a read-only GraphQL endpoint for querying projects, teams, tasks, and workspace metadata. All requests are served over HTTPS from a single endpoint. Authentication uses bearer tokens scoped to a workspace with the permissions of the issuing user.

```
[title=Endpoint] POST https://api.nexaplatform.com/v2/graphql
```

Authorization
`Bearer nx_pat_a1b2c3d4e5f6g7h8i9j0`

Rate Limit
2000 points/hour per token

Core Types

Project and Task

```
[title=Project & Task Type Definitions] type Project {
id: ID!
name: String!
slug: String!
description: String
team: Team!
tasks(
first: Int
after: String
status: TaskStatus
): TaskConnection!
createdAt: DateTime!
updatedAt: DateTime!
archivedAt: DateTime
}
enum TaskStatus { BACKLOG, TODO, IN_PROGRESS, IN_REVIEW, DONE, CANCELLED }
enum TaskPriority { P0, P1, P2, P3 }
type Task {
id: ID!
title: String!
description: String
status: TaskStatus!
priority: TaskPriority!
assignee: User
project: Project!
labels: [String!]!
dueDate: Date
createdAt: DateTime!
updatedAt: DateTime!
}
```

User, Team, and Connections

```
[title=Supporting Types] type User { id: ID!, username: String!, email: String!, avatarUrl: String, role: Role! }
enum Role { ADMIN, MEMBER, VIEWER }
type Team { id: ID!, name: String!, description: String, members(first: Int, after: String): UserConnection!,
projects(first: Int, after: String): ProjectConnection!, createdAt: DateTime! }
type PageInfo { endCursor: String, hasNextPage: Boolean!, hasPreviousPage: Boolean!, startCursor: String }
type TaskConnection { edges: [TaskEdge!]!, pageInfo: PageInfo!, totalCount: Int! }
type TaskEdge { node: Task!, cursor: String! }
type UserConnection { edges: [UserEdge!]!, pageInfo: PageInfo!, totalCount: Int! }
type UserEdge { node: User!, cursor: String! }
type ProjectConnection { edges: [ProjectEdge!]!, pageInfo: PageInfo!, totalCount: Int! }
type ProjectEdge { node: Project!, cursor: String! }
```

Queries

```
[title=Root Query Fields] type Query {
project(slug: String!): Project
projects(first: Int, after: String, teamId: ID, query: String): ProjectConnection!
task(id: ID!): Task
tasks(first: Int, after: String, projectId: ID, status: TaskStatus, priority: TaskPriority, assigneeId: ID, query:
String): TaskConnection!
team(id: ID!): Team
teams(first: Int, after: String, query: String): TeamConnection!
viewer: User!
}
```

Fetch a Project by Slug

Retrieve a project with its team name and the first five in-progress tasks.

```
[title=Query: Get Project] query GetProject($slug: String!) {
project(slug: $slug) {
id
name
description
team { name }
tasks(first: 5, status: IN_PROGRESS) {
edges { node { id, title, priority, assignee { username }, dueDate } }
pageInfo { endCursor, hasNextPage }
totalCount
}
createdAt
}
}
Variables: { "slug": "meridian-migration" }
```

```
[title=Response] {
"data": {
"project": {
"id": "UHJvamVjdC04MjE=",
"name": "Meridian Migration",
"description": "Migrate legacy Meridian services to Nimbus infra.",
"team": { "name": "Platform Engineering" },
"tasks": {
"edges": [
{ "node": { "id": "VGFzay0zMDE=", "title": "Provision Nimbus staging cluster", "priority": "P0", "assignee": {
"username": "clara.zhou" }, "dueDate": "2026-08-14" } },
{ "node": { "id": "VGFzay0zMDI=", "title": "Drain Meridian ingress routes", "priority": "P1", "assignee": { "username":
"david.park" }, "dueDate": "2026-08-21" } }
],
"pageInfo": { "endCursor": "Y3Vyc29yOjMwMg==", "hasNextPage": true },
"totalCount": 8
},
"createdAt": "2026-07-01T09:15:00Z"
}
}
}
```

List Teams and Viewer

```
[title=Query: List Teams] query ListTeams {
teams(first: 20) {
edges { node {
id
name
members(first: 5) {
edges { node { username, role } }
totalCount
}
} }
pageInfo { endCursor, hasNextPage }
}
}
```

```
[title=Query: Current User] query Viewer {
viewer {
id
username
email
role
avatarUrl
}
}
Response:
{ "viewer": {
"id": "VXNlci00Mg==",
"username": "elena.voss",
"email": "elena@vertexlabs.io",
"role": "ADMIN",
"avatarUrl": null }
}
```

Mutations

Every mutation returns the affected object for single-round-trip reads. Input types follow the naming convention `*Input` with a corresponding `*Payload` return type.

```
[title=Mutation Root & Input Types (Partial)] type Mutation {
createProject(input: CreateProjectInput!): CreateProjectPayload!
updateProject(input: UpdateProjectInput!): UpdateProjectPayload!
archiveProject(id: ID!): ArchiveProjectPayload!
createTask(input: CreateTaskInput!): CreateTaskPayload!
updateTask(input: UpdateTaskInput!): UpdateTaskPayload!
setTaskStatus(id: ID!, status: TaskStatus!): SetTaskStatusPayload!
}
input CreateProjectInput { name: String!, teamId: ID!, description: String }
type CreateProjectPayload { project: Project!, clientMutationId: String }
input CreateTaskInput { title: String!, projectId: ID!, status: TaskStatus, priority: TaskPriority, assigneeId: ID,
labels: [String!], dueDate: Date }
type CreateTaskPayload { task: Task!, clientMutationId: String }
input UpdateTaskInput { id: ID!, title: String, description: String, status: TaskStatus, priority: TaskPriority,
assigneeId: ID, labels: [String!], dueDate: Date }
type UpdateTaskPayload { task: Task!, clientMutationId: String }
```

Create a Task

```
[title=Mutation: Create Task] mutation CreateTask($input: CreateTaskInput!) {
createTask(input: $input) {
task { id, title, status, priority, project { slug }, assignee { username } }
clientMutationId
}
}
Variables: { "input": { "title": "Deploy Vertex auth service to staging", "projectId": "UHJvamVjdC04MjE=", "priority":
"P0", "labels": ["infra","auth"], "dueDate": "2026-08-04" } }
```

```
[title=Response] { "data": { "createTask": { "task": { "id": "VGFzay0zMTk=", "title": "Deploy Vertex auth service
to staging", "status": "TODO", "priority": "P0", "project": { "slug": "meridian-migration" }, "assignee": null },
"clientMutationId": null } } }
```

Bulk Status Update with Aliases

Use GraphQL aliases to run multiple mutations in one request.

```
[title=Mutation: Bulk Status Update via Aliases] mutation BulkStatusUpdate {
t301: setTaskStatus(id: "VGFzay0zMDE=", status: DONE) { task { id, status } }
t302: setTaskStatus(id: "VGFzay0zMDI=", status: IN_REVIEW) { task { id, status } }
t303: setTaskStatus(id: "VGFzay0zMDM=", status: IN_PROGRESS) { task { id, status } }
}
```

Scalars and Errors

Scalar	Format	Description
`DateTime`	ISO 8601	UTC timestamp: `"2026-07-15T14:30:00Z"`
`Date`	ISO 8601	Calendar date: `"2026-08-14"`
`URL`	String	Absolute URL with scheme
`JSON`	Object	Arbitrary JSON, validated server-side

Error Code	HTTP	Meaning
UNAUTHENTICATED	401	Missing or invalid bearer token
FORBIDDEN	403	Token lacks required scope
NOT_FOUND	404	Resource does not exist
VALIDATION_ERROR	400	Input failed server-side validation
RATE_LIMITED	429	Hourly point budget exceeded
INTERNAL	500	Unexpected server error; Nexa on-call alerted

Retry: For RATE_LIMITED and INTERNAL, use exponential backoff starting at 1 s (max 32 s). All other errors are non-retryable without modifying the request.