



NetApp Digital Advisor API documentation

Digital Advisor API

NetApp
April 27, 2026

Table of Contents

- NetApp Digital Advisor API documentation 1
- Get started 2
 - Learn about Digital Advisor APIs 2
 - Supported API platforms 2
 - AutoSupport 2
 - Benefits of using Digital Advisor APIs 2
 - Learn about the GraphQL API data query language 3
 - Learn about GraphQL API data object relationships 3
 - Data queries and mutations in the GraphQL API for Digital Advisor 4
 - Queries 4
 - Mutations 4
 - Variables 5
 - Example query 6
 - Example mutation 6
 - Pagination in the GraphQL API for Digital Advisor 7
 - Request a cursor in your query 7
 - Change the number of records per page 8
 - Navigate the data set using pagination 8
 - Rate and query limits in the GraphQL API for Digital Advisor 8
 - Request rate limit 9
 - Timeouts 9
 - Error handling in the GraphQL API for Digital Advisor 9
 - Error codes 10
 - Example error responses 10
- Use the Digital Advisor GraphQL API 14
 - Generate your access and refresh tokens to use the Digital Advisor APIs 14
 - Generate your tokens 14
 - Use your access token 15
 - Refresh your tokens 15
 - Create a Digital Advisor GraphQL API request 15
 - Use Digital Advisor GraphQL API example queries 17
 - NetApp inventory management 17
 - Support case management 27
 - Risk management 28
 - Sustainability and energy management 31
 - Upgrade recommendations and history 32
- Migrate from REST to GraphQL 35
 - Learn about migrating Digital Advisor REST API endpoints to GraphQL 35
 - Deprecation of REST APIs 35
 - Benefits of GraphQL's data query language 35
 - Benefits of GraphQL data object relationships 35
 - Comparing GraphQL and REST API operations 36
 - Migrate deprecated Digital Advisor REST API endpoints to GraphQL 36

Step 1: Learn how to replace REST API calls with GraphQL	36
Step 2: Identify deprecated REST API endpoints and their GraphQL replacement.....	37
Step 3: Test out the GraphQL API	37
Legal notices	38
Copyright	38
Trademarks	38
Patents	38
Privacy policy	38
Open source.....	38

NetApp Digital Advisor API documentation

Get started

Learn about Digital Advisor APIs

NetApp Digital Advisor uses automation and artificial intelligence to analyze telemetry and product data. It provides predictive analytics for NetApp environments through API access to telemetry-driven insights and install base details. You can integrate these analytics into your workflows, dashboards, and NetApp inventory management systems.

Supported API platforms

Digital Advisor currently supports both GraphQL and REST APIs. However, Digital Advisor is transitioning from REST to GraphQL as its primary API platform to improve performance, flexibility, and scalability. Digital Advisor REST APIs will eventually be deprecated.

The transition will be phased, with a period of overlap between REST APIs and GraphQL APIs. A deprecation notice is published in the Digital Advisor API catalog for each deprecated REST API endpoint. You need to migrate that specific endpoint to GraphQL within six months.

To identify the REST APIs scheduled for deprecation from the Digital Advisor API catalog, navigate to **API services > Browse > Deprecated APIs**.



Due to the phased deprecation of Digital Advisor REST APIs, NetApp strongly recommends that all new API integrations use the GraphQL-based API.

AutoSupport

AutoSupport is the telemetry service for NetApp products, periodically collecting health, configuration, status, and performance data from your storage environment. This data is transmitted to NetApp and forms the primary data source for the Digital Advisor.

AutoSupport data is system-generated and enabled by default on ONTAP and most other NetApp storage products. AutoSupport data provides benefits such as proactive identification of configuration issues, automated support case reporting and faster resolution, and support for self-service workflows. For more information on AutoSupport, see the [AutoSupport documentation](#).

Benefits of using Digital Advisor APIs

Digital Advisor brings together business and support system data with configuration and AI-derived insights from AutoSupport telemetry data. Instead of siloed views or manual downloads, you can gain programmatic visibility across your NetApp data infrastructure, enabling faster action and smarter decisions.

You can use the Digital Advisor APIs to perform the following tasks across your NetApp environment:

- NetApp inventory management:
 - Track all purchased systems to identify which systems are sending telemetry data. Automatically detects gaps in telemetry and automatic case generation.
 - View sites and contacts to understand where systems are deployed and identify the key contacts for each location.

- Access the latest configuration details of a system according to its last check-in, this includes a wide range of available attributes, including software versions.
- Analyze current and forecasted capacity and storage efficiency for ONTAP, E-Series, and StorageGRID systems.
- Discover end-of-support dates for systems, shelves, and disks to proactively plan for tech refreshes.
- Monitor policy variance requests (PVRs) for your NetApp inventory and their expiration.
- Review contract details, including hardware and software contract dates and identifiers.
- Support case management:
 - Determine which assets have open support cases and their current status.
 - Integrate support case data into your own systems for streamlined management.
- Risk management:
 - Discover and monitor security, upgrade, best practice, configuration, community, and AI issues.
 - Receive actionable recommendations to prevent issues before they happen.
- Sustainability and energy management:
 - Monitor actual and published power consumption, as well as projected power, heat, and carbon metrics.
- Upgrade recommendations and history:
 - Track upgrade recommendations and history, including which systems were upgraded, when, and to what versions.
 - Retrieve recommendations on the next upgrade and readiness for upgrade across your NetApp inventory.

All of the data listed can be integrated into your in-house tools and processes through API-first workflows to help make managing your NetApp inventory easier.

Learn about the GraphQL API data query language

[GraphQL](#) is an open source data query language that allows clients to request exactly the data they need. The GraphQL data query language has the following attributes:

- A [specification](#). The specification determines the validity of the schema on the API server. The schema determines the validity of client calls.
- A strongly typed schema. The schema defines an API's type system and all object relationships.
- An introspective schema. A client can query the schema for details about the schema.
- Hierarchical fields. The shape of a GraphQL call mirrors the shape of the JSON data it returns. Nested fields let you query for and receive only the data you specify in a single round trip.
- An application layer. GraphQL is not a storage model or a database query language. The "graph" refers to graph structures defined in the schema, where nodes define objects and edges define relationships between objects. The API navigates and returns data based on the schema definitions, independent of how the data is stored.

Learn about GraphQL API data object relationships

Learn about the GraphQL API schema to understand how data objects relate and exist within a hierarchy. The schema has the following elements:

- Tailored data retrieval: Queries only the required fields, keeping responses concise and relevant.
- Single endpoint simplicity: Accesses all Digital Advisor intelligence from a unified GraphQL endpoint.
- Improved efficiency: Reduces the number of API calls in dashboards and automation pipelines.
- Future-proof flexibility: Seamlessly adopts new insights with zero disruption as GraphQL schemas evolve.

Data queries and mutations in the GraphQL API for Digital Advisor

The Digital Advisor GraphQL API offers flexibility and the ability to define precisely the data you want to fetch.

The two types of allowed operations in Digital Advisor's GraphQL API are queries and mutations. Queries and mutations share similar features, with some important differences.

Queries

GraphQL queries return only the data you specify. In GraphQL, queries operate like REST GET requests. To create a query, you must specify nested subfields until you return only scalar values. The following example query shows how to get system details by including the system object and nested fields to return scalar values for the `serialNumber` and `systemId`.



Queries that do not specify nested subfields are invalid.

Example query:

```
query getSystems {
  systems {
    cursor
    totalCount
    systems {
      serialNumber
      systemId
    }
  }
}
```

Mutations

In GraphQL, mutations operate like POST, PATCH, and DELETE requests in REST. The mutation name determines which modification is run. To create a mutation, you must specify:

- The mutation name. This is the type of modification you want to perform.
- The input object. This is the data you want to send to the server, composed of input fields. You pass this as an argument to the mutation name.
- The payload object. This is the data you want to return from the server, composed of return fields. You pass this as the body of the mutation name.

Mutation structure example:

The input object in the following example is `MutationNameInput` and the payload object is `MutationNamePayload`.

```
mutation {  
  MUTATION-NAME(input: {MUTATION-NAME-INPUT!}) {  
    MUTATION-NAME-PAYLOAD  
  }  
}
```

Variables

Variables can make queries more dynamic and powerful, and they also can reduce complexity when passing mutation input objects.

Example query with a single variable:

```
query($customerId: String) {  
  system(customerId: $customerId) {  
    systems {  
      systemId  
      serialNumber  
    }  
  }  
}  
  
variables {  
  "customerId": "12345"  
}
```

Use variables in a query

There are three steps required to use variables in your queries.

Steps

1. Define the variable outside the operation in a variables object:

```
{  
  variables: {  
    "customerId": "12345"  
  }  
}
```

The object must be a valid JSON object. This example uses a string variable. You can also define complex types, such as input objects, or multiple variables.

2. Pass the variable to the operation as an argument:

```
query($customerId: String) {
```

The argument is a key-value pair, where the key is the name starting with \$ (such as \$customerId), and the value is the type (such as String). Add a "!" character to indicate whether the type is required. If you've defined multiple variables, include them here as multiple arguments.

3. Use the variable within the operation:

```
system(customerId: $customerId) {
```

In this example, you use the variable for the customer ID you want to retrieve. You specified a type in step 2 because GraphQL enforces strong typing.

This process makes the query argument dynamic. Variables let you update values without changing the query. You can simply change the value in the variables object and keep the rest of the query the same.

Example query

The following query looks up the customer ID "123", fetches the first 10 systems, and returns each system's system ID, serial number, and capacity information:

```
query {
  systems(customerId: "123", pageSize: 10) {
    systems {
      serialNumber
      systemId
      ... on ONTAPSystem {
        capacity {
          logical {
            usedKiB
          }
          physical {
            usedKiB
          }
        }
      }
    }
  }
}
```

Example mutation

Mutations often require information that you can only find by performing a query first. This example shows two operations:

- A query to find risk instances for a specific risk ID.
- A mutation to acknowledge a specific risk on a system.

```

query {
  riskInstances(filter: {
    riskIds: ["123"]
  }) {
    cursor
    totalCount
    riskInstances {
      risk {
        riskId
      }
      system {
        serialNumber
        systemId
      }
    }
  }
}

mutation {
  riskAcknowledge(acknowledgedBy: "test-user", riskAcknowledgementFlag:
true, justification: "test acknowledge", riskId: "123", systemKeys: [{
    serialNumber: "ABC123"}]) {
    message
    status
    updatedCount
  }
}

```

Pagination in the GraphQL API for Digital Advisor

Learn how to navigate data sets using cursor-based pagination with the Digital Advisor GraphQL API.

Digital Advisor's GraphQL API limits the number of items that you can fetch in a single request in order to protect against excessive or malicious requests to Digital Advisor's servers. When you use the GraphQL API, you must supply a `pageSize` or `after` argument on any objects. The GraphQL API returns the number of objects specified by the `pageSize` or `after` argument.

Request a cursor in your query

When using the GraphQL API, you use cursors to navigate through a paginated data set. The cursor represents a specific position in the data set. For example:

```

query {
  systems (pageSize: 20, after: null) {
    cursor
    totalCount
    systems {
      id
      hostName
      serialNumber
    }
  }
}

```

In this example, the `systems` query requests a page size of 20 items starting from the beginning of the data set (`after: null`). The response includes a `cursor` field that you can use to request the next page of results.

Change the number of records per page

The `pageSize` and `after` arguments control how many items are returned. For more information, see the [rate and query limits](#) for the GraphQL API.

Navigate the data set using pagination

After you return a cursor from a query, you can use the cursor to request the next page of results. To do this, you use the `after` and `pageSize` arguments.

For example, if the cursor value from the previous example is 20, you can use this query to request the next page of results:

```

query {
  systems (pageSize: 20, after: "20") {
    cursor
    totalCount
    systems {
      id
      hostName
      serialNumber
    }
  }
}

```

Rate and query limits in the GraphQL API for Digital Advisor

The Digital Advisor GraphQL API has limitations in place to protect against excessive or malicious calls to Digital Advisor's servers.

Request rate limit

The GraphQL API assigns points to each query and limits the requests that you can use within a specific amount of time. This limit helps prevent abuse and denial-of-service attacks, and ensures that the API remains available for all users.

There is a limit of 300 requests in any five minute period for each originating IP address. If this limit is exceeded, the client receives a "Too Many Requests" error with the 429 status code, as shown in the following example.

```
{
  "errors": [
    {
      "extensions": {
        "service": "ActiveIQ_GraphQL",
        "code": "USER_API_RATE_LIMIT_EXCEEDED",
        "status": 429
      },
      "message": "User API Rate Limit Exceeded."
    }
  ]
}
```

Timeouts

If the GraphQL API takes more than 60 seconds to process an API request, the server terminates the request. You receive a timeout response and a message reporting that the request has timed out, as shown in the following example.

```
{
  "errors": [
    {
      "extensions": {
        "service": "ActiveIQ_GraphQL",
        "code": "REQUEST_TIMEOUT",
        "status": 504
      },
      "message": "The request has timed out."
    }
  ]
}
```

Error handling in the GraphQL API for Digital Advisor

When the GraphQL server encounters errors while processing an Digital Advisor GraphQL API operation, its response to the client includes an array containing each error

that occurred.

Each error in the array has an `extensions` field that provides additional useful information, including an error code and a message. GraphQL allows for a partial response, with omitted entries logged in the errors array showing a reason and a partial success for updates.

The following is an example error response caused by misspelling the `__typename` field in a query:

```
{
  "errors": [
    {
      "message": "Cannot query field \"__typenam\" on type \"Query\".",
      "extensions": {
        "code": "GRAPHQL_VALIDATION_FAILED"
      }
    }
  ]
}
```

Error codes

GraphQL provides built-in error codes to help categorize different types of errors. For a complete reference of all built-in error codes, refer to the [Apollo Server documentation](#).

In addition to the built-in error codes, the following table shows some additional types of errors:

Error Code	HTTP Status Code	Description
BAD_USER_INPUT	200	Invalid value for a field argument
FORBIDDEN	200	Not authorized due to missing or expired role
NOT_FOUND_ERROR	200	No record found for the provided input
UNAUTHENTICATED	200	No token or expired/invalid token
USER_API_RATE_LIMIT_EXCEEDED	429	Request rate limit exceeded

Example error responses

The following examples show error responses with their HTTP status codes. GraphQL returns the HTTP status code of 200 as long as the HTTP request is valid and succeeds. Any execution errors are returned inside the JSON response body as shown in the following examples. For more information, refer to [Apollo Server documentation](#).

BAD_USER_INPUT (HTTP status code - 200)

```
{
  "errors": [
    {
      "extensions": {
        "service": "ActiveIQ_GraphQL",
        "code": "BAD_USER_INPUT",
        "status": 400
      },
      "message": "Either start or end date should be present for the given date range."
    }
  ],
  "data": {
    "systems": null
  }
}
```

FORBIDDEN (HTTP status code - 200)

```
{
  "errors": [
    {
      "extensions": {
        "service": "ActiveIQ_GraphQL",
        "code": "FORBIDDEN",
        "status": 403
      },
      "message": "Access is restricted"
    }
  ],
  "data": null
}
```

NOT_FOUND_ERROR (HTTP status code - 200)

```
{
  "errors": [
    {
      "extensions": {
        "service": "ActiveIQ_GraphQL",
        "code": "NOT_FOUND_ERROR",
        "status": 404
      },
      "message": "No record found!"
    }
  ],
  "data": null
}
```

UNAUTHENTICATED (HTTP status code - 200)

```
{
  "errors": [
    {
      "extensions": {
        "service": "ActiveIQ_GraphQL",
        "code": "UNAUTHENTICATED",
        "status": 401
      },
      "message": "Expected a token!"
    }
  ],
  "data": null
}
```

PARTIAL_SUCCESS (HTTP status code - 200)

```
{
  "data": {
    "riskAcknowledge": {
      "message": "Risk already acknowledged for 1 system(s).",
      "status": "partial",
      "updatedCount": 1
    }
  }
}
```

RATE LIMIT (HTTP status code - 429)

```
{
  "errors": [
    {
      "extensions": {
        "service": "ActiveIQ_GraphQL",
        "code": "USER_API_RATE_LIMIT_EXCEEDED",
        "status": 429
      },
      "message": "User API Rate Limit Exceeded."
    }
  ]
}
```

Use the Digital Advisor GraphQL API

Generate your access and refresh tokens to use the Digital Advisor APIs

Learn how to generate and refresh the required tokens before using the Digital Advisor APIs. The tokens ensure that only authorized persons and systems can access your NetApp information.

Generate your tokens

You need to generate an access token and a refresh token before you can use the Digital Advisor GraphQL APIs.

About this task

- The access and refresh tokens you generate are only valid for a limited time:
 - Access token: Valid for 1 hour. You use this in your API requests.
 - Refresh token: Valid for 7 days. You use this to get a new access token.
- Always save the latest refresh token after each refresh.
- Store your tokens securely (such as in a password manager or secure file).
- Never share your tokens or email them to anyone.
- If possible, automate token refresh in scripts or applications.
- Set reminders to log in and get a new set of tokens every 90 days.
- Systems without valid support contracts will no longer show up in the responses after 90 days.

Before you begin

- You must have valid NetApp Support Site credentials.

Steps

1. Navigate to [Digital Advisor \(Active IQ\)](#) and sign in using your NetApp Support Site credentials.
2. In the top navigation bar, select **Quick Links**, and then select **API Services**.
3. Select **Register for API access**, and then select **Register**.
 - a. Complete the request for the authorization form, and then select **Submit**.



Activation is automatic and should be instantaneous. After you are authorized to use the Digital Advisor APIs, you can generate tokens to use when making programmatic API calls. Tokens always come in sets of two: an access token and a refresh token. The access token must be passed to successfully use the API and the refresh token is used to programmatically obtain a new set of tokens.

4. After your registration is approved, go back to the **API Services** page, and select **Generate Token**.

You can now view and download the access and refresh tokens required to use the Digital Advisor APIs.

5. Save your tokens.

The portal gives you multiple ways to save one or both tokens in the set. You can copy them to the clipboard, download them as a text file, or view them as plain text.



You must download and save the access token and refresh token for later use. Access tokens expire one hour after generation and refresh tokens should be regenerated manually every 7 days and installed in the application. To do this, you do not need to log in to the application. However, after 90 days, you need to log in to the application to obtain a new access and a refresh token.

Use your access token

The following curl example shows how to use your access token in an API request.

```
curl -X POST https://gql.aiq.netapp.com/graphql \
  -H "Authorization: Bearer YOUR_ACCESS_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"query": "query getSystems { systems { totalCount cursor systems {
  serialNumber systemId platformType } } }"}'
```

Refresh your tokens

Your refresh token is valid for 7 days and you must use it to refresh within that timeframe. When you refresh with the API, you receive a new refresh token. You must save the new token and use it the next time you need to refresh your tokens.



You can repeat this process (refreshing and saving the new token) for **90 days** from your original login date. You **must** log in after 90 days and repeat the steps to get a new set of tokens.

The following curl example shows how to refresh your tokens in REST. If you are using a client GraphQL library, you must switch to a standard HTTP request format.

```
curl -X POST https://api.activeiq.netapp.com/v1/tokens/accessToken \
  -H "Content-Type: application/json" \
  -d '{"refreshToken": "YOUR_REFRESH_TOKEN"}'
```

The response includes a new access token and refresh token.

Create a Digital Advisor GraphQL API request

Learn how to create a GraphQL API request to receive data from the GraphQL API for Digital Advisor.

Before you begin

You need to complete the following before you can create a request.

- [Generate access and refresh tokens](#) for access to the Digital Advisor API service.
- Open the [GraphQL API Explorer tool](#) in a separate browser window to test out the API as you perform these steps.

Steps

1. Open the [GraphQL API Explorer tool](#), and select the query you want to run from the left side of the window.
2. Pass your access token in the **Headers** section of the request as shown below:

```
{
  "Authorization": "Bearer <YOUR_ACCESS_TOKEN>"
}
```

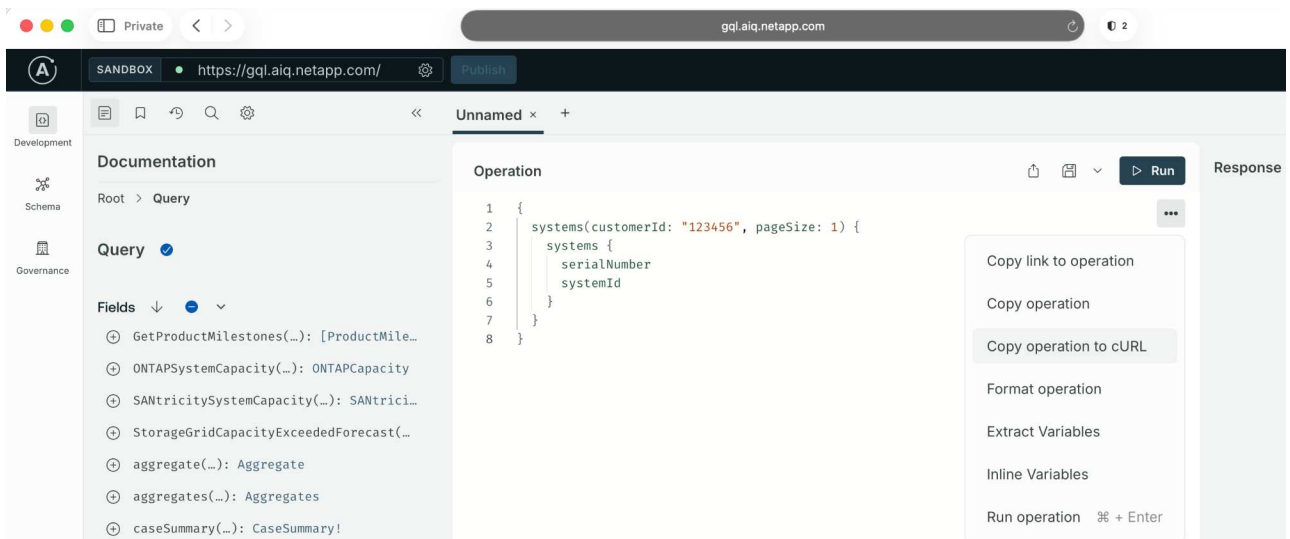


You should have already generated and saved your access token using the steps in [Generate access and refresh tokens](#).

3. Submit the request by selecting **Run** in the upper right corner.

A JSON response is displayed in the **Response Window**.

4. Optionally, copy the curl command to run the query outside of the GraphQL API explorer.
 - a. On the right side of the operation block window, select (...), and then select the **Copy operation to cURL** option from the dropdown menu:



b. The curl command is copied to your clipboard with your pre-configured query, variables, and headers.

You can now paste and run this command in your terminal.

Use Digital Advisor GraphQL API example queries

After you've generated your tokens and learned how to create a GraphQL request, you can use the example queries to retrieve various types of data from the GraphQL API for Digital Advisor.

Before you use the GraphQL query examples, review the following information:

- Replace `$variableName` with actual values or use GraphQL variables.
- Use cursor-based pagination for better performance with large datasets.
- Adjust the `pageSize` based on your application's needs.
- Include only the fields you need to optimize query performance.
- Consider using fragments for reusable field sets.
- Test your queries in the GraphQL Playground or a similar tool before implementation.

For more example GraphQL queries, refer to the [Apollo Studio explorer collections](#).

NetApp inventory management

Learn how to use the example GraphQL queries for NetApp inventory management.

Retrieve a list of sites

You can use this query to fetch a list of site IDs that are to be used in subsequent queries.

Show example

```
query($customerId: String, $after: String) {
  sites(customerId: $customerId, after: $after, pageSize: 10) {
    cursor
    sites {
      id
      name
    }
  }
}
```

Retrieve a list of customers

You can use this query to fetch a list of customer IDs that are to be used in subsequent queries.

Show example

```
query($customerId: String, $after: String) {
  customers(customerId: $customerId, after: $after, pageSize: 10) {
    customers {
      id
      name
    }
  }
}
```

Track all purchased systems

You can use this query to identify which systems are sending telemetry data. This automatically detects gaps in telemetry and for automatic case generation.

Show example

```
query($customerId: String, $after: String) {
  # Get a summary of AutoSupport status across the fleet
  summary(customerId: $customerId) {
    system
    countByField(countBy: AUTOSUPPORT_STATUS) {
      fieldName
      counts {
        system
      }
    }
  }

  # Fetch systems that are not sending telemetry data
  systems(customerId: $customerId, autoSupportStatus: [OFF, NA,
DECLINE, RSS], after: $after, pageSize: 10) {
    cursor
    totalCount
    systems {
      serialNumber
      systemId
      autoSupportConfig {
        autoSupportStatus
      }
    }
  }
}
```

View sites and contacts

You can use this query to understand where systems are deployed and who the key contacts are for each location.

Show example

```
query($customerId: String, $after: String) {
  # Fetch systems along with their site and contact information
  systems(customerId: $customerId, after: $after, pageSize: 10) {
    cursor
    totalCount
    systems {
      serialNumber
      systemId
      site {
        id
        name
        streetAddress
        city
        state
        postalCode
        countryCode
      }
      contactPerson {
        firstName
        lastName
        email
        phone
      }
      sam {
        name
        emailAddress
        managerEmailAddress
      }
      csm {
        name
        emailAddress
        managerEmailAddress
      }
      salesRepresentative {
        name
        emailAddress
        managerEmailAddress
      }
    }
  }
}
```

Access the latest configuration details of the system

You can use this query to access the latest configuration details of the system based on its last check-in. This includes a wide range of available attributes including software versions.

Show example

```
query($customerId: String, $after: String) {  
  # Fetch systems with their autosupport configuration details and OS  
  information  
  systems(customerId: $customerId, after: $after, pageSize: 10) {  
    cursor  
    totalCount  
    systems {  
      serialNumber  
      systemId  
      osType  
      osVersion  
      platformType  
      hardwareModel {  
        name  
      }  
      autoSupportConfig {  
        autoSupportStatus  
        autoSupportTransport  
        isAsupRetransmitEnabled  
        isAutoSupportOnDemandCapable  
        isAutoSupportOnDemandEnabled  
      }  
    }  
  }  
}
```

Analyze current and forecasted capacity and storage efficiency

You can use this query to analyze the current and forecasted capacity and storage efficiency for ONTAP, E-Series, and StorageGRID systems.

Show example

```
query($customerId: String, $after: String) {
  systems(customerId: $customerId, after: $after, pageSize: 10) {
    cursor
    totalCount
    systems {
      serialNumber
      systemId
      ... on ONTAPSystem {
        capacity {
          logical {
            usedKiB
            usedSnapshotsKiB
            usedWithoutSnapshotsKiB
            usedSnapshotsAndFlexClonesKiB
            usedWithoutSnapshotsAndFlexClonesKiB
          }
          physical {
            usedKiB
            usedSnapshotsKiB
            usedWithoutSnapshotsKiB
            usedSnapshotsAndFlexClonesKiB
            usedWithoutSnapshotsAndFlexClonesKiB
          }
          reportedOn
          lastAsupId
        }
        monthlyCapacity {
          month
          physical {
            rawMarketingKiB
            utilizationPercentage
          }
          reportedOn
          lastAsupId
        }
      }
      ... on SantricitySystem {
        capacity {
          configured {
            freeKiB
            allocatedKiB
          }
          totalKiB
          unconfiguredKiB
        }
      }
    }
  }
}
```

```

        reportedOn: updatedOn
        lastAsupId
    }
}
... on StorageGrid {
  gridCapacity {
    configured {
      usableKiB
      usedDataKiB
      reservedMetadataKiB
      usedMetadataKiB
    }
    overhead {
      reservedMetadataKiB
    }
    physical {
      actualKiB
      rawMarketingKiB
    }
    reportedOn
    lastAsupId
  }
}
}
}
}
}

```

Discover end-of-support dates for systems, shelves, and disks

You can use this query to proactively plan for future tech refreshes by discovering the end-of-support dates for systems, shelves, and disks.

Show example



```

query($customerId: String, $after: String) {
  systems(customerId: $customerId, after: $after, pageSize: 10) {
    cursor
    totalCount
    systems {
      serialNumber
      systemId
      platformType
      hardwareModel {
        name
        endOfSupport
        endOfAvailability
      }
      techRefreshStatus
      ... on ONTAPSystem {
        shelvesSummary {
          count
          shelfModuleCount
          endOfSupportDate
          hardwareModel {
            name
            endOfSwSupport
            endOfHwSupport
            endOfAvailability
          }
          moduleHardwareModel {
            name
            endOfSwSupport
            endOfHwSupport
            endOfAvailability
          }
        }
      }
      drivesSummary {
        count
        model
        driveType
        driveModel
        driveCapacityKiB
        endOfSupportDate
      }
    }
  }
}

```

Monitor policy variance requests (PVRs)

You can use this query to monitor policy variance requests (PVRs) for your systems along with their expiration dates.

Show example

```
query($customerId: String, $after: String) {
  systems(customerId: $customerId, after: $after, pageSize: 10) {
    cursor
    totalCount
    systems {
      serialNumber
      systemId
      hasPvr
      pvr {
        id
        info
        validFrom
        validTo
      }
    }
  }
}
```

Review contract details

You can use this query to view contract details, including hardware and software contract dates and identifiers.

Show example

```
query($customerId: String, $after: String) {
  systems(customerId: $customerId, after: $after, pageSize: 10) {
    cursor
    totalCount
    systems {
      serialNumber
      systemId
      platformType
      contract {
        expiryDate
        isContractActive
        hardwareContractId
        hardwareContractStartDate
        hardwareContractEndDate
        hardwareWarrantyStartDate
        hardwareWarrantyEndDate
        hardwareServiceLevel
        softwareContractId
        softwareContractStartDate
        softwareContractEndDate
        nrdContractId
        nrdContractStartDate
        nrdContractEndDate
        overallContractEndDate
      }
    }
  }
}
```

Support case management

You can use this query to determine which assets have open support cases and their current status.

Show example

```
query($customerId: String, $after: String) {
  cases(customerId: $customerId, after: $after, pageSize: 10) {
    cursor
    cases {
      caseId
      description
      type
      status
      symptom
      created
      closed
      caseReceivedVia
      priority
      reporterContact {
        userId
        name
      }
      rmaParts {
        numberOfRequestsCreated
        partRequestId
        partRequestItemId
        partRequestItemNumber
        partNumber
        partDescription
        partRequestCreatedDate
        partRequestDeliveryDate
        waybillNumber
      }
      resolution
      lastUpdated
    }
  }
}
```

Risk management

Learn how to use the example GraphQL queries for risk management.

Discover and monitor risk

You can use this query to discover and monitor security, upgrade, best practice, configuration, community, and AI issues.

Show example

```
query($customerId: String, $after: String) {  
  # Fetch count of risks for critical and high severity issues  
  risksCount (  
    customerId: $customerId,  
    filter: {  
      severity: [  
        CRITICAL  
        HIGH  
      ]  
    }  
  ) {  
    riskCount  
    systemCount  
    actionCount  
  }  
  # Fetch risks for critical and high severity issues  
  risks(  
    customerId: $customerId,  
    after: $after,  
    pageSize: 10,  
    filter: {  
      severity: [  
        CRITICAL  
        HIGH  
      ]  
    }  
  ) {  
    cursor  
    risks {  
      riskId  
      impactArea  
      severity  
      shortName  
      riskDetail  
      mitigationAction  
      potentialImpact  
      riskInstances {  
        system {  
          serialNumber  
          systemId  
          platformType  
        }  
      }  
    }  
  }  
}
```

```
}  
}
```

Receive recommendations

You can use this query to receive actionable recommendations to prevent issues before they occur.

Show example

```
query($customerId: String, $after: String) {  
  # Fetch risks for critical and high severity issues with playbook and  
  corrective action details  
  risks(  
    customerId: $customerId,  
    after: $after,  
    pageSize: 10,  
    filter: {  
      severity: [  
        CRITICAL  
        HIGH  
      ]  
    }  
  ) {  
    cursor  
    risks {  
      riskId  
      impactArea  
      severity  
      shortName  
      playbook  
      correctiveAction {  
        url  
        displayName  
      }  
      riskInstances {  
        system {  
          serialNumber  
          systemId  
          platformType  
        }  
      }  
    }  
  }  
}
```

Sustainability and energy management

You can use this query to monitor actual and published power consumption, as well as projected power, heat, and carbon metrics.

Show example

```
query($siteId: String) {
  # Fetch sites with their carbon emission data
  sites(siteId: $siteId) {
    cursor
    sites {
      id
      name
      carbonEmission {
        emissionFactorPerKwh
        carbonMitigation {
          mitigationPercentage
          modifiedDate
          userModified
        }
      }
      source {
        name
        url
      }
    }
  }
}

# Fetch sustainability scores for a specific site
sustainabilityScore(siteId: $siteId) {
  cursor
  sustainabilityScores {
    changeFactors
    generatedDate
    percentageChange
    scorePercentage
  }
}
```

Upgrade recommendations and history

Learn how to use the example GraphQL queries for upgrade recommendations and history.

Track upgrade recommendations and history

You can use this query to track upgrade recommendations and history, including which systems were upgraded, when, and to which versions.

Show example

```
query($customerId: String, $after: String) {
  clusters(customerId: $customerId, after: $after, pageSize: 10) {
    cursor
    clusters {
      id
      name
      osUpgradeHistory {
        fromVersion
        toVersion
        postUpgradeAsupId
        postUpgradeAsupGenDate
      }
    }
  }
}
```

Retrieve upgrade recommendations

You can use this query to retrieve recommendations on your next upgrade and readiness for upgrade across your NetApp inventory.

Show example

```
query($customerId: String, $after: String) {
  clusters(customerId: $customerId, after: $after, pageSize: 10) {
    cursor
    clusters {
      id
      name
      osRecommendation {
        recommendedVersion
        recommendedLatestPatch
        upgradeImageUrl
        upgradeImageUrls {
          upgradeAutomationEncryptedPath
          upgradeAutomationUnencryptedPath
        }
        customUpgradeVersionStates {
          version
          supportState
        }
      }
    }
  }
}
```

Migrate from REST to GraphQL

Learn about migrating Digital Advisor REST API endpoints to GraphQL

Digital Advisor is transitioning from REST APIs to GraphQL as its primary API platform to improve performance, flexibility, and scalability. Learn why you should migrate your REST API endpoints to GraphQL.



Due to the phased deprecation of Digital Advisor REST APIs, NetApp strongly recommends that all new API integrations use the GraphQL-based API.

Deprecation of REST APIs

Digital Advisor currently supports both GraphQL and REST APIs. However, Digital Advisor is transitioning from REST APIs to GraphQL as its primary API platform to improve performance, flexibility, and scalability. Digital Advisor REST APIs will eventually be deprecated.

The transition will be phased, with a period of overlap between REST APIs and GraphQL APIs. A deprecation notice will be published in the Digital Advisor API catalog for each deprecated REST API endpoint. You need to migrate that specific endpoint to GraphQL within six months. For more information, see [Migrate deprecated Digital Advisor REST API endpoints to GraphQL](#).

Benefits of GraphQL's data query language

The [GraphQL](#) data query language allows clients to request exactly the data they need. The benefits of using the GraphQL data query language include:

- A [specification](#). The specification determines the validity of the schema on the API server. The schema determines the validity of client calls.
- A strongly typed schema. The schema defines an API's type system and all object relationships.
- An introspective schema. A client can query the schema for details about the schema.
- Hierarchical fields. The shape of a GraphQL call mirrors the shape of the JSON data it returns. Nested fields let you query for and receive only the data you specify in a single round trip.
- An application layer. GraphQL is not a storage model or a database query language. The graph refers to graph structures defined in the schema, where nodes define objects and edges define relationships between objects. The API traverses and returns data based on the schema definitions, independent of how the data is stored.

Benefits of GraphQL data object relationships

Compared to REST, GraphQL is organized around objects, not endpoints, which improves predictability and provides the following benefits:

- Tailored data retrieval: Queries only the required fields, keeping responses concise and relevant. When you create a GraphQL request, you request only the fields you need. This is one of GraphQL's key efficiency benefits.
- Single endpoint simplicity: Accesses all Digital Advisor intelligence from a unified GraphQL endpoint.

- Improved efficiency: Reduces the number of API calls in dashboards and automation pipelines.
- Future-proof flexibility: Seamlessly adopts new insights with zero disruption as GraphQL schemas evolve.

Comparing GraphQL and REST API operations

GraphQL uses two types of operations to fetch data. These are called queries and mutations. Queries operate like GET requests and mutations operate like POST, PATCH, and DELETE requests in REST. The following table shows some differences between REST and GraphQL:

Component	REST API example	Example GraphQL equivalent
Path parameters	<code>/v1/system/inventory/details/level/customer/id/123</code>	<code>systems(customerId: "123")</code>
Filters	<code>?filterByPlatform=0NTAP</code>	<code>systems(platformType: [ONTAP])</code>
Pagination (cursor)	<code>?start=0&limit=10</code>	<code>systems(after: "0", pageSize: 10)</code>

To learn more about queries and mutations, refer to [Digital Advisor GraphQL API data queries and mutations](#).

Migrate deprecated Digital Advisor REST API endpoints to GraphQL

The transition from REST APIs to GraphQL APIs will be phased, with a period of overlap between REST APIs and GraphQL APIs. Learn how to identify and migrate deprecated REST API endpoints to GraphQL.

A deprecation notice will be published in the Digital Advisor API catalog for each deprecated REST API endpoint. You need to migrate that specific endpoint to GraphQL within six months.



Due to the phased deprecation of Digital Advisor REST APIs, NetApp strongly recommends that all new API integrations use the GraphQL-based API.

About this task

- You should first migrate the endpoints that already have the six-month deprecation notice before you migrate other endpoints. NetApp updates these first and provides confirmed GraphQL replacements.
- GraphQL is organized around objects, not endpoints, which improves predictability. To understand the schema types, refer to the [GraphQL Apollo Studio explorer](#).

Step 1: Learn how to replace REST API calls with GraphQL

To replace the REST API calls with GraphQL, you need to learn how to do the following:

- Use queries instead of endpoints.

GraphQL retrieves exactly the fields you need instead of fixed REST payloads (reducing over and under fetching). For more information, see [Digital Advisor GraphQL API data queries and mutations](#).

- Pass variables instead of URL parameters.

GraphQL uses variables instead of URL parameters. For more information, see [Use variables in a query](#).

- Consolidate multiple REST API calls.

With GraphQL, you can combine several requests into a single query and get all the data you need at once.



Digital Advisor GraphQL APIs use the same authentication methods as the REST API for token generation. For more information, see [Generate your Digital Advisor API access and refresh tokens](#).

Step 2: Identify deprecated REST API endpoints and their GraphQL replacement

Steps

1. Review the Digital Advisor REST APIs that you are currently using in your integration.
2. Navigate to [Digital Advisor \(ActiveIQ\)](#) and sign in using your NetApp Support Site credentials.
3. In the top navigation bar, select **Quick Links**, and then select **API Services**.
4. To review the list of APIs scheduled for deprecation, select **Browse**, and then select **Deprecated APIs**.
5. From the list of deprecated APIs, confirm the following:
 - Whether an endpoint you are using in your integration is listed.
 - The expected retirement date for endpoints you are using.
6. Review the **Migration information** section, which includes the equivalent GraphQL query that corresponds to each deprecated REST API endpoint. To view the equivalent GraphQL query, select **Sample GraphQL Query** in the table.



Most of the REST API endpoints have a direct functional replacement available in GraphQL. Review each deprecated REST API endpoint to find the corresponding migration information.

Step 3: Test out the GraphQL API

You can use the [GraphQL API Explorer tool](#) to perform the following actions:

- Run sample queries
- Validate output
- Adjust your field selection
- Test variables

For more information, see [Use the Digital Advisor GraphQL APIs](#).

After you finish

After you've learned how to use the GraphQL API and validated the GraphQL version of your deprecated endpoint, update your automation code and scripts and discontinue the use of the deprecated REST call before its retirement date.

Legal notices

Legal notices provide access to copyright statements, trademarks, patents, and more.

Copyright

<https://www.netapp.com/company/legal/copyright/>

Trademarks

NETAPP, the NETAPP logo, and the marks listed on the NetApp Trademarks page are trademarks of NetApp, Inc. Other company and product names may be trademarks of their respective owners.

<https://www.netapp.com/company/legal/trademarks/>

Patents

A current list of NetApp owned patents can be found at:

<https://www.netapp.com/pdf.html?item=/media/11887-patentspage.pdf>

Privacy policy

<https://www.netapp.com/company/legal/privacy-policy/>

Open source

Notice files provide information about third-party copyright and licenses used in NetApp software.

Copyright information

Copyright © 2026 NetApp, Inc. All Rights Reserved. Printed in the U.S. No part of this document covered by copyright may be reproduced in any form or by any means—graphic, electronic, or mechanical, including photocopying, recording, taping, or storage in an electronic retrieval system—without prior written permission of the copyright owner.

Software derived from copyrighted NetApp material is subject to the following license and disclaimer:

THIS SOFTWARE IS PROVIDED BY NETAPP “AS IS” AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WHICH ARE HEREBY DISCLAIMED. IN NO EVENT SHALL NETAPP BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

NetApp reserves the right to change any products described herein at any time, and without notice. NetApp assumes no responsibility or liability arising from the use of products described herein, except as expressly agreed to in writing by NetApp. The use or purchase of this product does not convey a license under any patent rights, trademark rights, or any other intellectual property rights of NetApp.

The product described in this manual may be protected by one or more U.S. patents, foreign patents, or pending applications.

LIMITED RIGHTS LEGEND: Use, duplication, or disclosure by the government is subject to restrictions as set forth in subparagraph (b)(3) of the Rights in Technical Data -Noncommercial Items at DFARS 252.227-7013 (FEB 2014) and FAR 52.227-19 (DEC 2007).

Data contained herein pertains to a commercial product and/or commercial service (as defined in FAR 2.101) and is proprietary to NetApp, Inc. All NetApp technical data and computer software provided under this Agreement is commercial in nature and developed solely at private expense. The U.S. Government has a non-exclusive, non-transferrable, nonsublicensable, worldwide, limited irrevocable license to use the Data only in connection with and in support of the U.S. Government contract under which the Data was delivered. Except as provided herein, the Data may not be used, disclosed, reproduced, modified, performed, or displayed without the prior written approval of NetApp, Inc. United States Government license rights for the Department of Defense are limited to those rights identified in DFARS clause 252.227-7015(b) (FEB 2014).

Trademark information

NETAPP, the NETAPP logo, and the marks listed at <http://www.netapp.com/TM> are trademarks of NetApp, Inc. Other company and product names may be trademarks of their respective owners.