【BSMT-R4P4】
API仕様書

Version: v1.5

Rev	更新日	更新内容・理由
v1.3	2023.11.27	初版発行
v1.4	2024.2.19	スケジュール機能の追加、UI調整
v1.5	2025.1.30	バージョンを揃える

目次

1. はじめに
    1-1. APIの利用について
        1-1-1. ローカルAPI
        1-1-2. クラウドAPI
    1-2. クエリを作成する
        1-2-1. GraphQL Playground
        1-2-2. GraphQL リファレンス
2. SMARTIO ローカルAPI
    2-1. ローカルAPIとは
    2-2. ローカルAPIの仕様
        2-2-1. クロスオリジンからのアクセス
    2-3. ローカルAPIの利用手順
        2-3-1. APIが実行可能なユーザの作成
    2-4. ローカルAPI Examples
        2-4-1. 入力ポート状態の取得
        2-4-2. 出力ポートの制御
3. SMARTIO クラウドAPI
    3-1. クラウドAPIとは
    3-2. クラウドAPIの仕様
    3-3. クラウドAPIの利用手順
        3-3-1. 3. IoT Hubのデバイスの作成
        3-3-2. 4. SMARTIOにIoT Hubの接続情報の設定
    3-4. IoT Hubの画面上からクラウドAPIの実行
    3-5. クラウドAPI Examples
        3-5-1. 入力ポート状態の取得
        3-5-2. 出力ポートの制御

1. はじめに

SMARTIOデバイス（BSMT-R4P4）はカスタム操作画面の作成や、外部アプリケーションと連携するためのGraphQL APIを提供しています。
このドキュメントでは、APIの利用方法や、開発に必要な情報をご紹介します。

GraphQLについては、GraphQL 公式サイト をご参照ください。

1-1. APIの利用について

SMARTIOでは、ローカルAPIおよび、クラウドAPIの2種類のAPIを提供しています。
ご利用の環境や目的に応じて、適切なAPIをご利用ください。

1-1-1. ローカルAPI

SMARTIOが接続されたローカルネットワーク内からのみ利用可能です。
同一拠点に設置された複数のSMARTIOをまとめて制御する場合や、設定を一括で行う場合などにご活用ください。

ローカルAPIのご利用方法は、こちら をご覧ください。

1-1-2. クラウドAPI

インターネット経由でAPIを実行します。
遠隔地のSMARTIOを制御する場合や、多拠点に設置されたSMARTIOを一元管理する場合にご活用ください。

ご利用には、Microsoft Azure の契約が必要となります。

クラウドAPIのご利用方法は、こちら をご覧ください。

1-2. クエリを作成する

1-2-1. GraphQL Playground

GraphQL Playgroundでは実際にクエリを実行させて、デバイスの状態取得や制御を行うことができます。
APIで実行するクエリの動作確認や、クエリの作成にご活用ください。

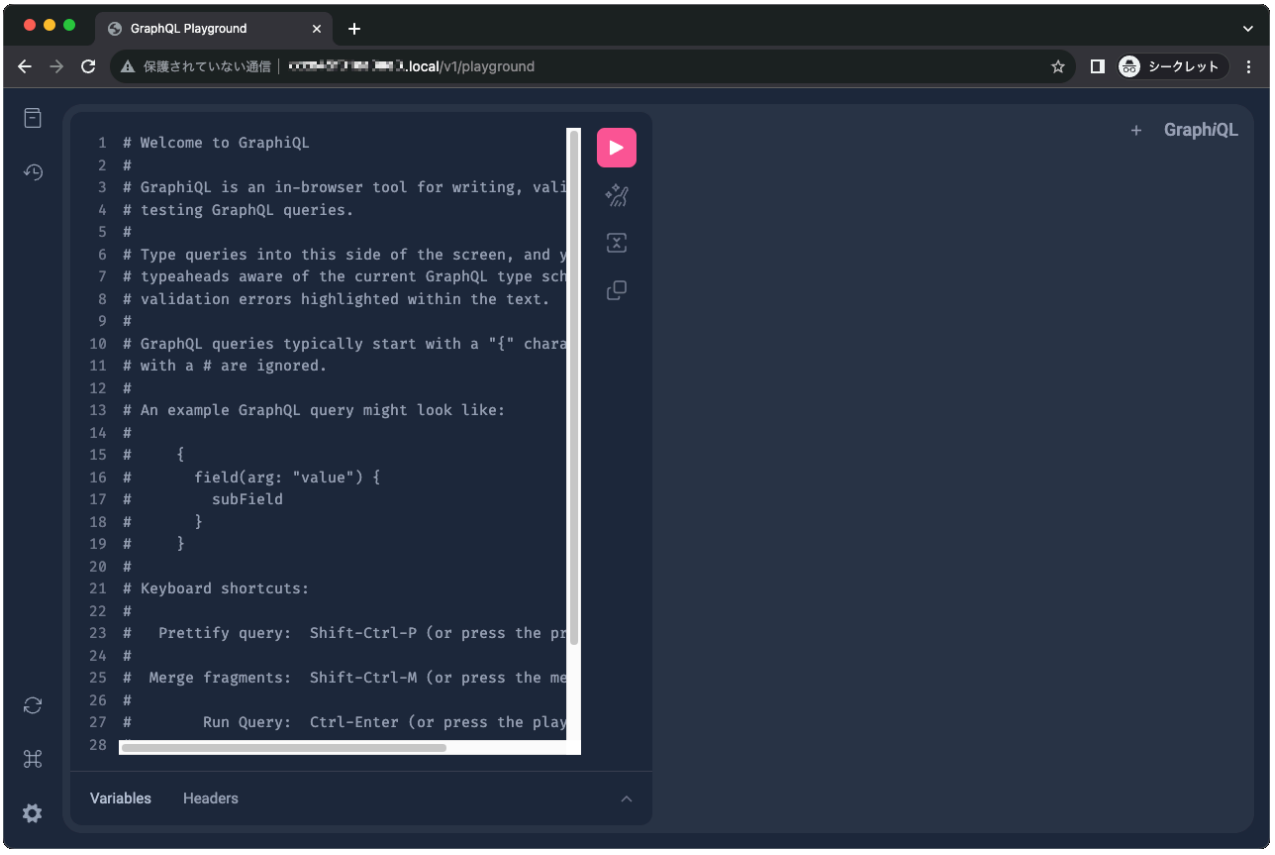
> ⚠ GraphQL Playgroundはクエリのテスト環境ではありません。
> Mutationを実行すると、実際にデバイスの設定が変更されたり、出力ポートの状態が変化しますのでご注意ください。

GraphQL Playgroundにアクセスするには、SMARTIOが接続されたローカルネットワーク内から以下のURLにアクセスしてください。

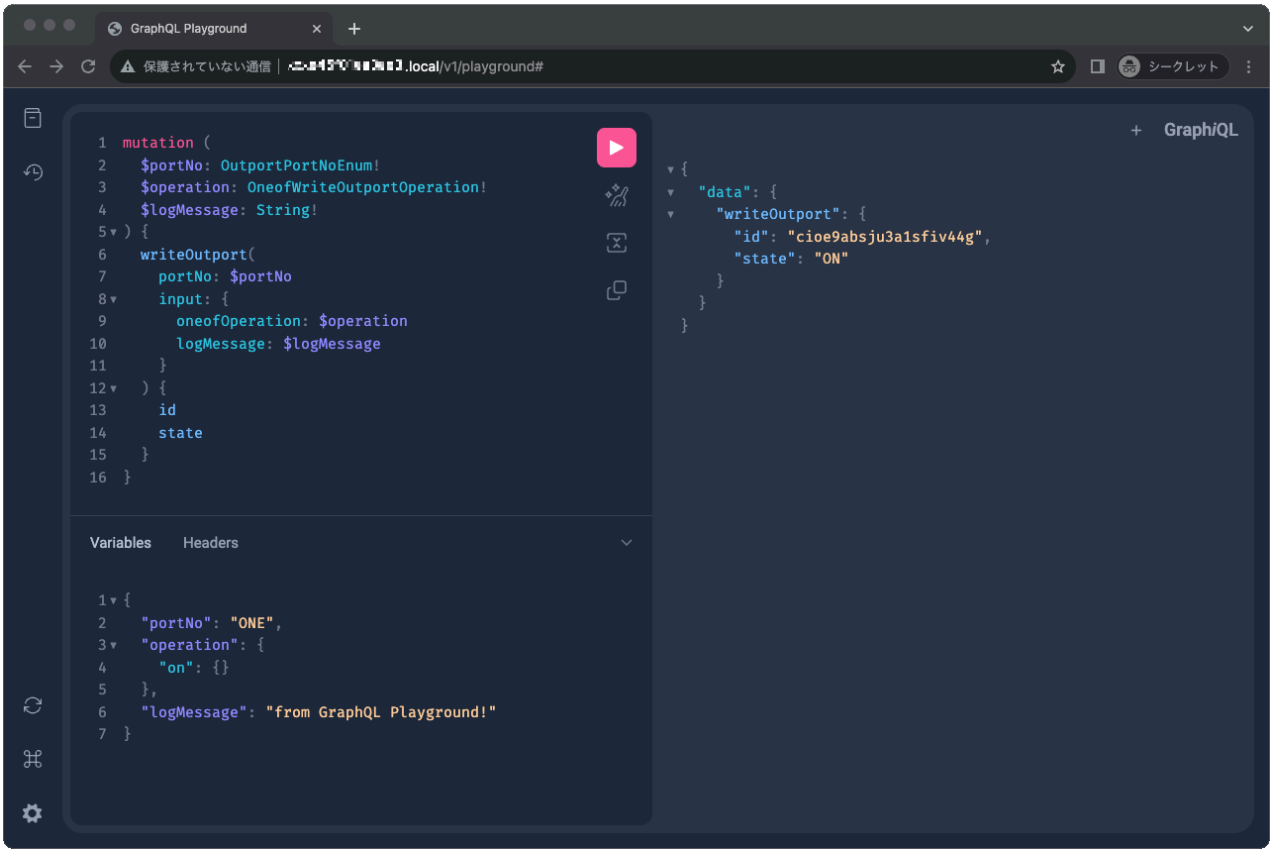
```
http://[SMARTIOのアドレス]/v1/playground
```

`SMARTIOのアドレス` には、対象となるSMARTIOデバイスの `IPアドレス` または `ホスト名 + .local` をご指定ください。

初回アクセス時は認証が求められますので、APIが実行可能なユーザの認証情報 を入力してください。
認証が完了すると、以下のような画面が表示されます。



左側のクエリエディタにクエリを入力し、実行ボタンを押すことで、クエリの動作確認を行うことができます。



1-2-2. GraphQL リファレンス

SMARTIOサポートページ ではSMARTIOが提供しているQueriesおよびMutationsのリファレンスを掲載しています。
クエリの作成時にご活用ください。

2. SMARTIO ローカルAPI

この章では、SMARTIOのローカルAPIのご利用方法についてご紹介します。

2-1. ローカルAPIとは

ローカルAPIでは、ローカルネットワーク内のSMARTIOの制御や設定を行う事ができます。
同一拠点に設置された複数のSMARTIOをまとめて制御する場合や、設定を一括で行う場合などにご利用ください。

ローカルAPIはインターネットを経由した遠隔地からの利用はできません。
遠隔地から利用する場合は、クラウドAPIをご利用ください。

2-2. ローカルAPIの仕様

ローカルAPIはGraphQLの仕様に準じています。

- エンドポイントには `http://[SMARTIOのアドレス]/v1/query` を指定してください。
- Basic認証にてAPIが実行可能なユーザのユーザID・パスワードを指定してください。

`SMARTIOのアドレス` には、対象となるSMARTIOデバイスの `IPアドレス` または `ホスト名 + .local` をご指定ください。

具体的なクエリの実行方法については、ローカルAPI Examplesの他、GraphQL 公式サイトや使用するクライアントライブラリのドキュメントをご参照ください。

2-2-1. クロスオリジンからのアクセス

ローカルAPIはクロスオリジンからのアクセスを許可しており、HTML上からクエリを実行することが可能です。

> ⚠ 認証情報をHTML上に記載する必要があるため、該当のURLにアクセス制限をする等、**セキュリティ対策を十分に行ってください。**

2-3. ローカルAPIの利用手順

ローカルAPIをご利用頂くには、APIが実行可能なユーザの作成が必要です。
APIが実行可能なユーザは、「管理」または「API」の権限が指定されているユーザとなります。

2-3-1. APIが実行可能なユーザの作成

APIが実行可能なユーザの作成は、SMARTIOの設定画面から行います。

1. SMARTIOの管理画面にアクセスし、左メニューの「ユーザ管理」を選択してください。
2. 右上に表示されている「新規登録」ボタンを押してください。
3. 権限で「管理」または「API」を選択し、任意のユーザID・パスワードを入力してください。
4. 「登録する」ボタンを押して、APIが実行可能なユーザの作成を完了してください。

2-4. ローカルAPI Examples

ここでは、ローカルAPIを実行するいくつかの例をご紹介します。

2-4-1. 入力ポート状態の取得

入力ポートの状態を取得するには、`inports query` を使用します。
以下のコードは入力ポートの状態を取得する例です。

`SMARTIOのアドレスおよび`、APIが実行可能なユーザの `ユーザID` `パスワード` はご使用の環境に合わせて変更してください。

Python

サンプルでは gql および aiohttp ライブラリを使用しています。

```
from aiohttp import BasicAuth
from gql import gql, Client
from gql.transport.aiohttp import AIOHTTPTransport

transport = AIOHTTPTransport(
  url="http://[SMARTIOのアドレス]/v1/query", auth=BasicAuth("[APIが実行可能なユーザのユーザID]", "[APIが実行可能なユーザのパスワード]"))
client = Client(transport=transport)

query = gql("""
  query {
    inports {
      edges {
        node {
          portNo
```

```
            state
          }
        }
      }
    }
  }
""")

result = client.execute(query)
print(result)
```

Node.js

サンプルでは Apollo Client ライブラリを使用しています。

```
import { ApolloClient, InMemoryCache, gql } from "@apollo/client/core/core.cjs";

const userId = "[APIが実行可能なユーザのユーザID]";
const password = "[APIが実行可能なユーザのパスワード]";

const client = new ApolloClient({
 uri: "http://[SMARTIOのアドレス]/v1/query",
 headers: { authorization: "Basic " + btoa(userId + ":" + password) },
 cache: new InMemoryCache(),
});

(async () => {
 const result = await client.query({
  query: gql`
   query {
    inports {
     edges {
      node {
       portNo
       state
      }
     }
    }
   }
  `,
 });

 console.log(result);
})();
```

HTML

```
<pre id="result"></pre>
<script>
 const userId = "[APIが実行可能なユーザのユーザID]";
 const password = "[APIが実行可能なユーザのパスワード]";

 async function api(portNo, operation) {
  const response = await fetch("http://[SMARTIOのアドレス]/v1/query", {
```

```
    method: "POST",
    headers: {
     authorization: "Basic " + btoa(userId + ":" + password),
     "Content-Type": "application/json",
    },
    body: JSON.stringify({
     query: `
      query {
       inports {
        edges {
         node {
          portNo
          state
         }
        }
       }
      }
     `,
    }),
   });
   const result = await response.json();
   console.log(result);

   return result;
  }

  api().then((result) => {
   document.getElementById("result").textContent = JSON.stringify(
    result,
    null,
    2
   );
  });
 </script>
```

cURL

```
$ QUERY='query {
 inports {
  edges {
   node {
    portNo
    state
   }
  }
 }
}'

$ curl -X POST http://[SMARTIOのアドレス]/v1/query -4 \
-u "[APIが実行可能なユーザのユーザID]:[APIが実行可能なユーザのパスワード]" \
-H "Content-Type: application/json" \
-d "{ \"query\": \"$(echo $QUERY | tr -d '\n')\" }"
```

実行結果

```
{
 "data":{
  "inports":{
   "edges":[
    {"node":{"portNo":"ONE","state":"OFF"}},
    {"node":{"portNo":"TWO","state":"OFF"}},
    {"node":{"portNo":"THREE","state":"OFF"}},
    {"node":{"portNo":"FOUR","state":"OFF"}}
   ]
  }
 }
}
```

2-4-2. 出力ポートの制御

出力ポートを制御するには、`writeOutport mutation` を使用します。
以下のコードは出力の1番ポートをONにする例です。

`SMARTIOのアドレスおよび`、APIが実行可能なユーザの `ユーザID` `パスワード` はご使用の環境に合わせて変更してください。

Python

サンプルでは gql および aiohttp ライブラリを使用しています。

```
from aiohttp import BasicAuth
from gql import gql, Client
from gql.transport.aiohttp import AIOHTTPTransport

transport = AIOHTTPTransport(
  url="http://[SMARTIOのアドレス]/v1/query", auth=BasicAuth("[APIが実行可能なユーザのユーザID]", "[APIが実行可能なユーザのパスワード]"))
client = Client(transport=transport)

query = gql("""
  mutation (
    $portNo: OutportPortNoEnum!
    $operation: OneofWriteOutportOperation!
    $logMessage: String!
  ) {
    writeOutport(
      portNo: $portNo
      input: { oneofOperation: $operation, logMessage: $logMessage }
    ) {
      id
    }
  }
""")

result = client.execute(query, variable_values={
  "portNo": "ONE",
  "operation": {
    "on": {},
```

```
    },
    "logMessage": "Local API from python!",
})
print(result)
```

Node.js

サンプルでは Apollo Client ライブラリを使用しています。

```
import { ApolloClient, InMemoryCache, gql } from "@apollo/client/core/core.cjs";

const userId = "[APIが実行可能なユーザのユーザID]";
const password = "[APIが実行可能なユーザのパスワード]";

const client = new ApolloClient({
 uri: "http://[SMARTIOのアドレス]/v1/query",
 headers: { authorization: "Basic " + btoa(userId + ":" + password) },
 cache: new InMemoryCache(),
});

(async () => {
 const result = await client.mutate({
  mutation: gql`
   mutation (
    $portNo: OutportPortNoEnum!
    $operation: OneofWriteOutportOperation!
    $logMessage: String!
   ) {
    writeOutport(
     portNo: $portNo
     input: { oneofOperation: $operation, logMessage: $logMessage }
    ) {
     id
    }
   }
  `,
  variables: {
   portNo: "ONE",
   operation: {
    on: {},
   },
   logMessage: "Local API from javascript!",
  },
 });

 console.log(result);
})();
```

HTML

```
<button onClick="api('ONE', 'ON')">ON</button>
<button onClick="api('ONE', 'OFF')">OFF</button>
<script>
```

```
  const userId = "[APIが実行可能なユーザのユーザID]";
  const password = "[APIが実行可能なユーザのパスワード]";

  async function api(portNo, operation) {
   const response = await fetch("http://[SMARTIOのアドレス]/v1/query", {
    method: "POST",
    headers: {
     authorization: "Basic " + btoa(userId + ":" + password),
     "Content-Type": "application/json",
    },
    body: JSON.stringify({
     query: `
      mutation (
       $portNo: OutportPortNoEnum!
       $operation: OneofWriteOutportOperation!
       $logMessage: String!
      ) {
       writeOutport(
        portNo: $portNo
        input: {
         oneofOperation: $operation
         logMessage: $logMessage
        }
       ) {
        id
       }
      }
     `,
     variables: {
      portNo,
      operation: operation === "ON" ? { on: {} } : { off: {} },
      logMessage: "Local API from HTML!",
     },
    }),
   });
   const result = await response.json();
   console.log(result);
  }
 </script>
```

cURL

```
 $ QUERY='mutation (
  $portNo: OutportPortNoEnum!
  $operation: OneofWriteOutportOperation!
  $logMessage: String!
 ) {
  writeOutport(
   portNo: $portNo
   input: {
    oneofOperation: $operation
    logMessage: $logMessage
   }
  ) {
```

```
    id
  }
}'

$ VARIABLES='{
  "portNo": "ONE",
  "operation": {
    "on": {}
  },
  "logMessage": "Local API from cURL!"
}'

$ curl -X POST http://[SMARTIOのアドレス]/v1/query -4 \
-u "[APIが実行可能なユーザのユーザID]:[APIが実行可能なユーザのパスワード]" \
-H "Content-Type: application/json" \
-d "{ \"query\": \"$(echo $QUERY | tr -d '\n')\", \"variables\": $VARIABLES }"
```

3. SMARTIO クラウドAPI

このページではSMARTIOのクラウドAPIのご利用方法についてご紹介します。

3-1. クラウドAPIとは

クラウドAPIでは、インターネット経由でSMARTIOの制御や設定を行う事ができます。
遠隔地のSMARTIOを制御する場合や、多拠点に設置されたSMARTIOを一元管理する場合にご利用ください。

クラウドAPIのご利用には、Microsoft Azure の契約が必要となります。ローカルネットワーク内でのみAPIを利用する場合は、契約不要で利用頂ける ローカルAPI をご検討ください。

3-2. クラウドAPIの仕様

クラウドAPIは、Azure IoT Hub の ダイレクトメソッド を使用して実装されています。

- ダイレクトメソッドのメソッド名には `query` を指定してください。
- ダイレクトメソッドのリクエストの `payload` にGraphQLのリクエスト文を格納してください。
- ダイレクトメソッドのレスポンスの `payload` にGraphQLのレスポンス文が格納されます。

プログラムからダイレクトメソッドを実行する場合は、Azure IoT Hub Service SDK や こちら をご参照ください。
具体的なクエリの実行方法については、Examples の他、GraphQL 公式サイト や使用するクライアントライブラリのドキュメントをご参照ください。

3-3. クラウドAPIの利用手順

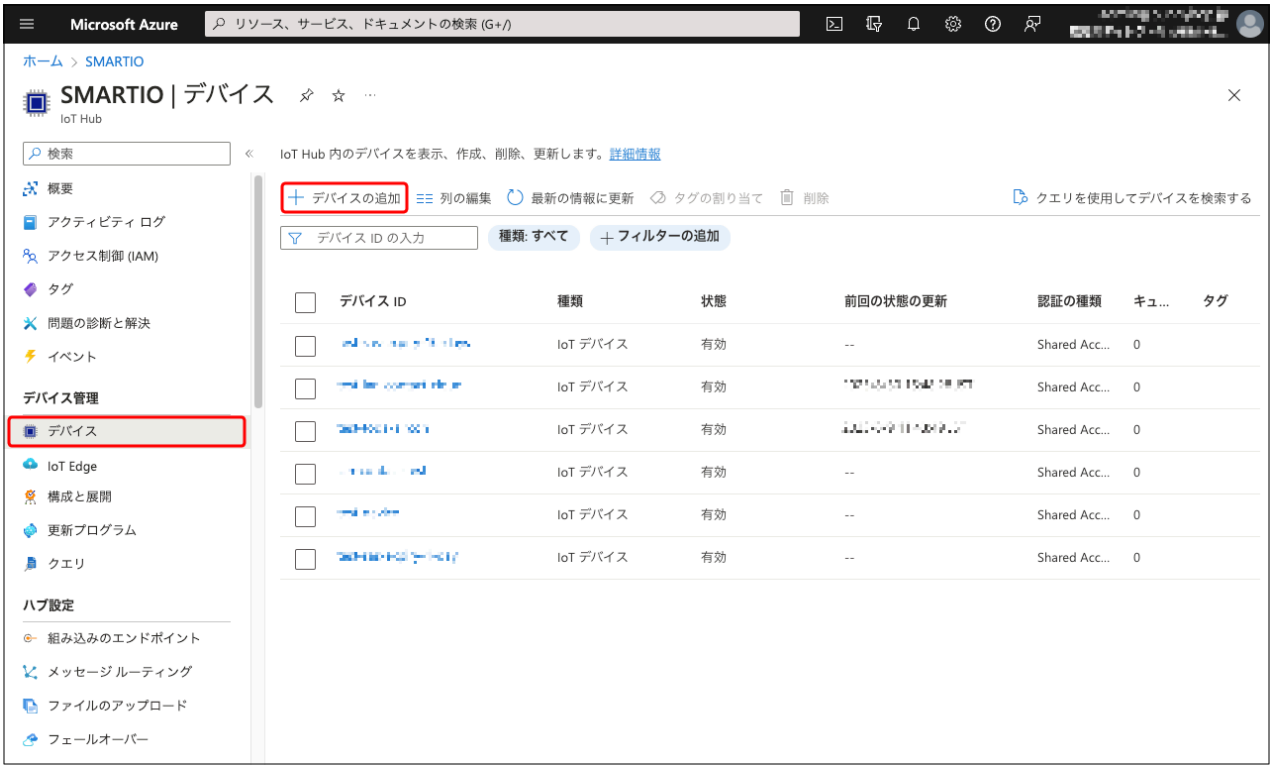
クラウドAPIをご利用頂くには、以下の手順が必要となります。

1. Microsoft Azureの契約
2. IoT Hubの作成
3. IoT Hubのデバイスの作成
4. SMARTIOにIoT Hubの接続情報の設定

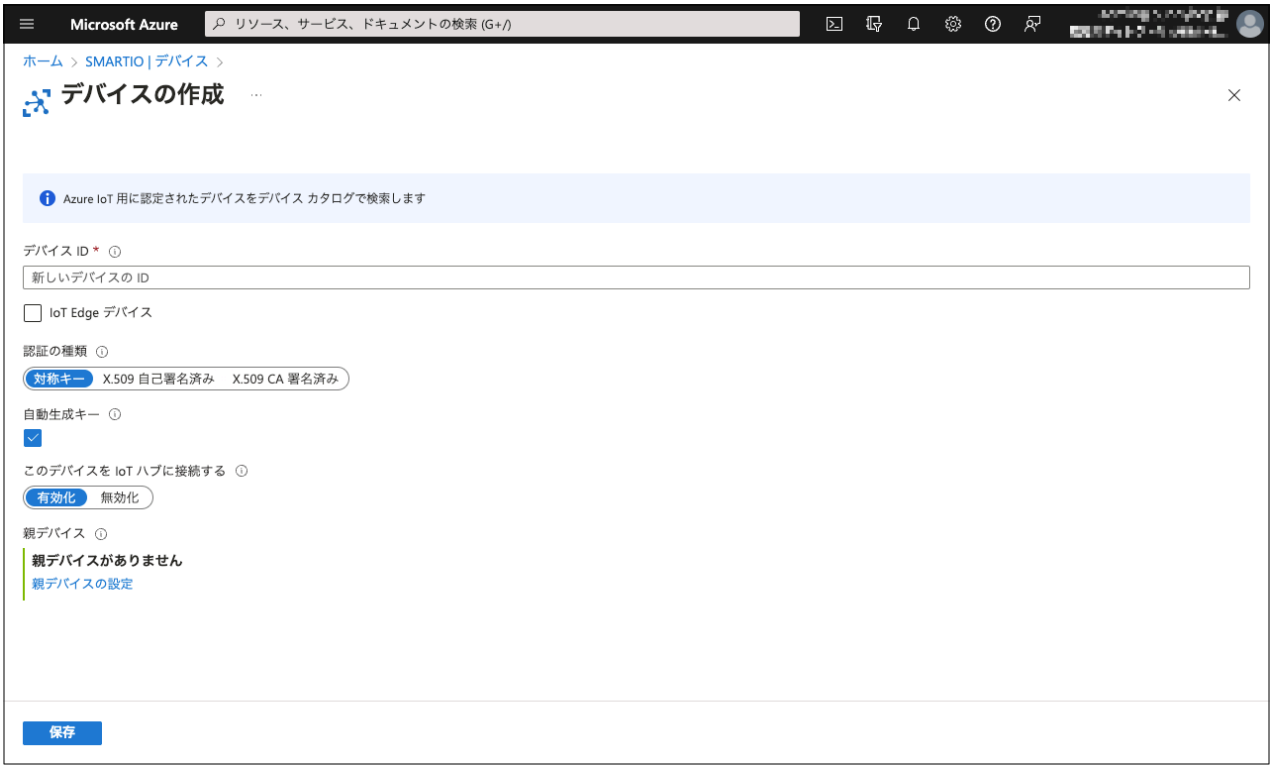
Microsoft Azure や IoT Hub については、各ドキュメントをご参照ください。

3-3-1. 3. IoT Hubのデバイスの作成

IoT Hubの画面にアクセスし、左メニューから「デバイス」を選択してデバイスの一覧画面にアクセスします。



「デバイスの追加」を押して、デバイスの作成画面にアクセスします。

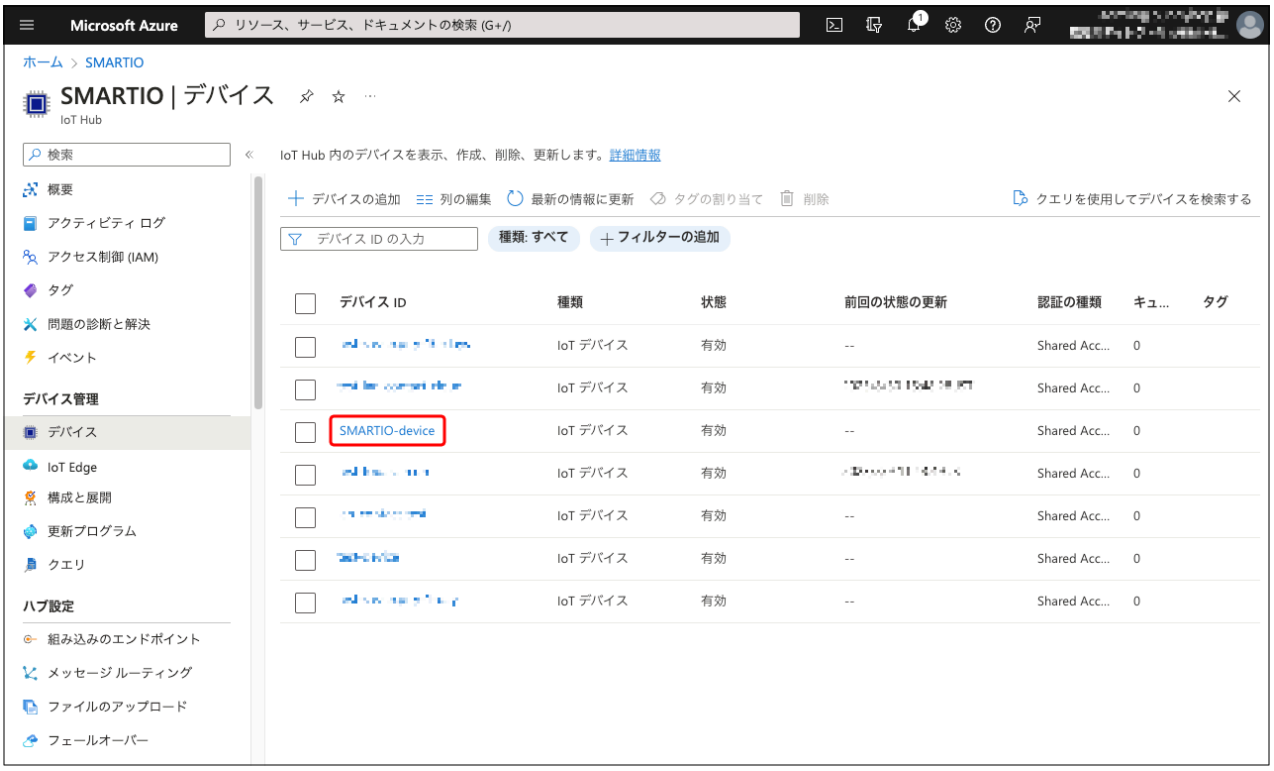


任意のデバイスIDを入力し、「保存」を押してデバイスの作成を完了します。

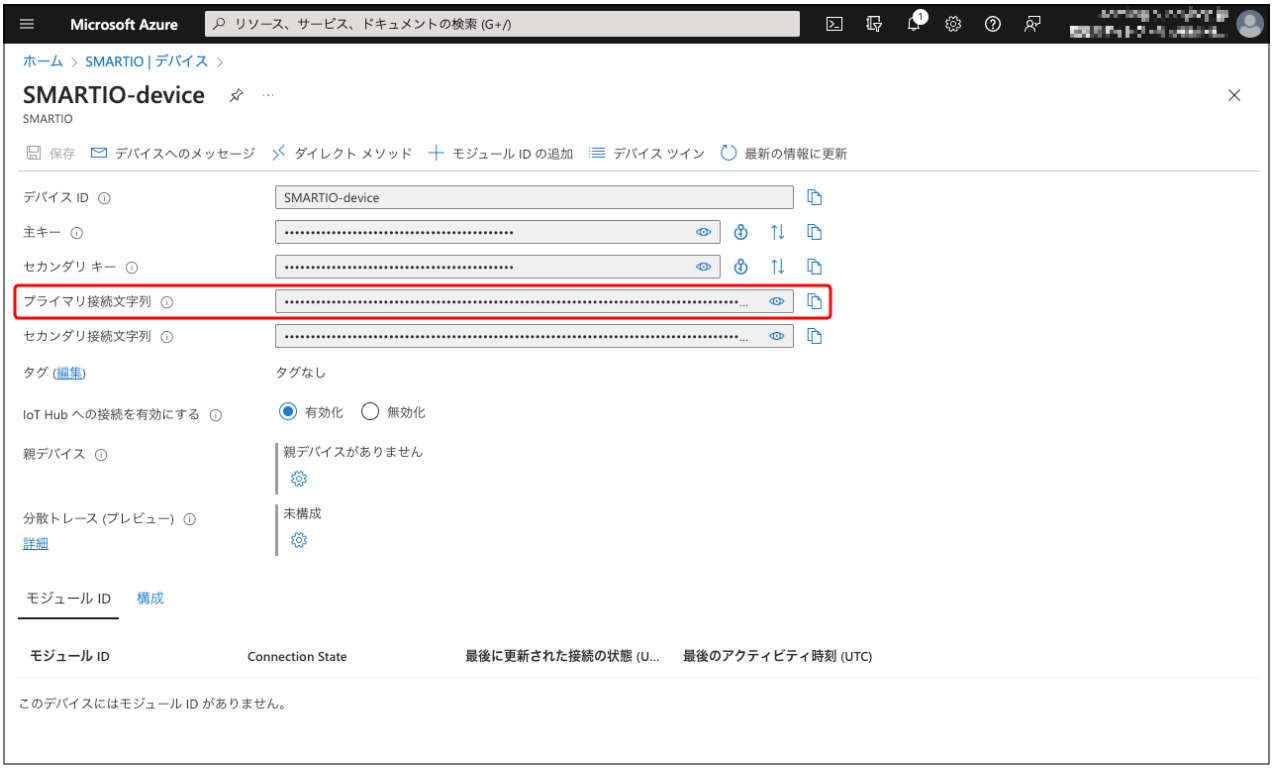
- IoT Edge デバイス: チェック不要
- 認証の種類: 対称キー
- 自動生成キー: チェック
- このデバイスをIoTハブに接続する: 有効化
- 親デバイス: 指定無し

3-3-2. 4. SMARTIOにIoT Hubの接続情報の設定

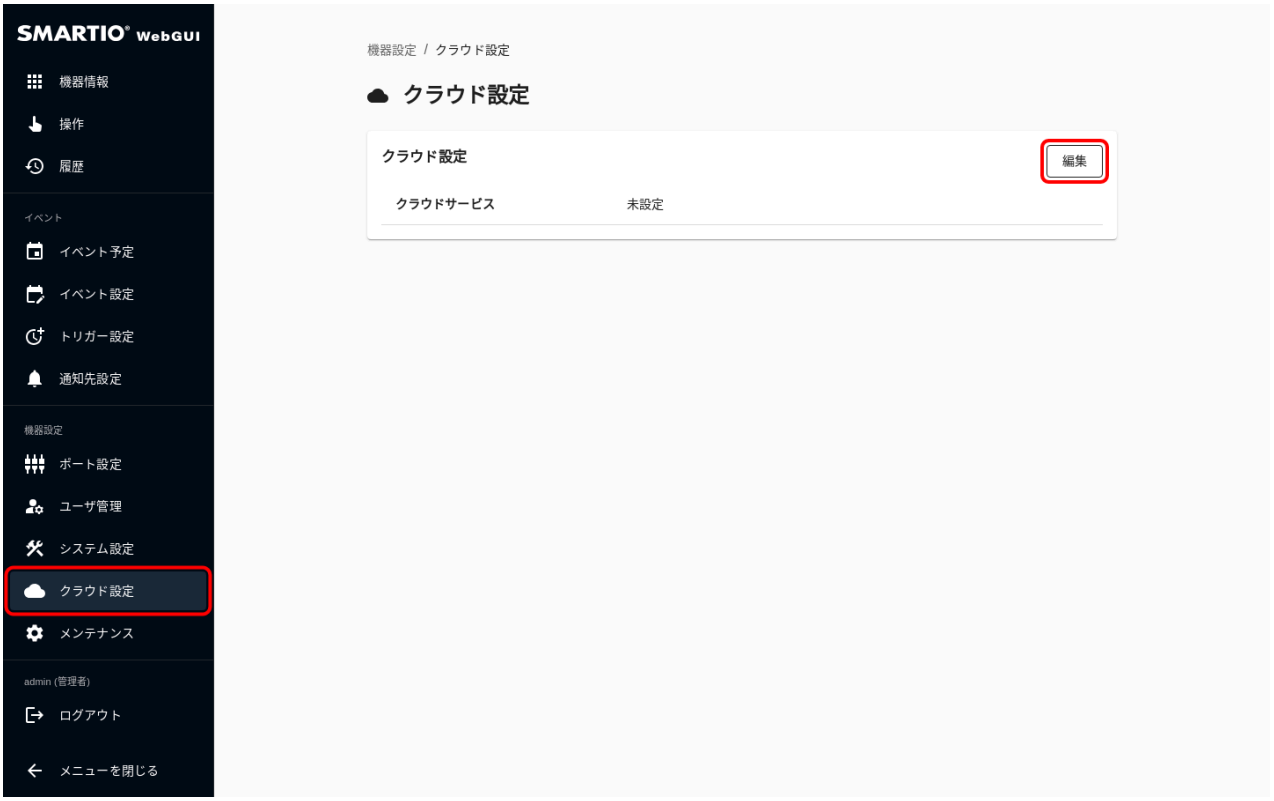
IoT Hubのデバイスの一覧画面にアクセスし、3.で作成したデバイスを選択してデバイスの詳細画面にアクセスします。



デバイスの詳細画面に表示されている「プライマリ接続文字列」をコピーします。



SMARTIOの管理画面にアクセスし、左メニューの「クラウド設定」を選択し、クラウド設定画面にアクセスします。



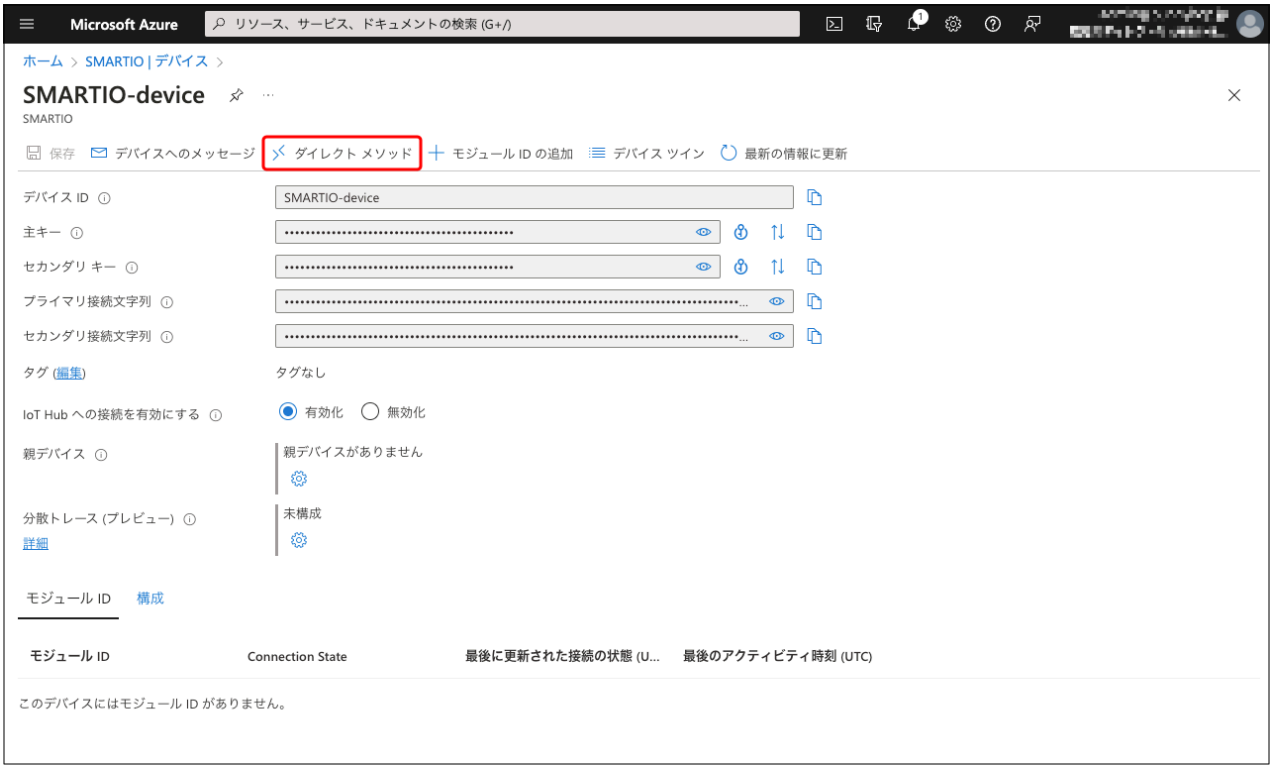
「編集」ボタンを押して「クラウド設定ダイアログ」を表示し、クラウドサービスで「Azure IoT Hub」を選択します。
「Azure IoT Hub 接続文字列」の入力欄に、先ほどコピーした「プライマリ接続文字列」を入力し、「更新する」ボタンを押して設定を保存します。

SMARTIOのダッシュボード画面にアクセスし、システム情報のクラウドに「Azure IoT Hub 接続済み」が表示されていることを確認します。
以上でクラウドAPIの実行準備は完了となります。

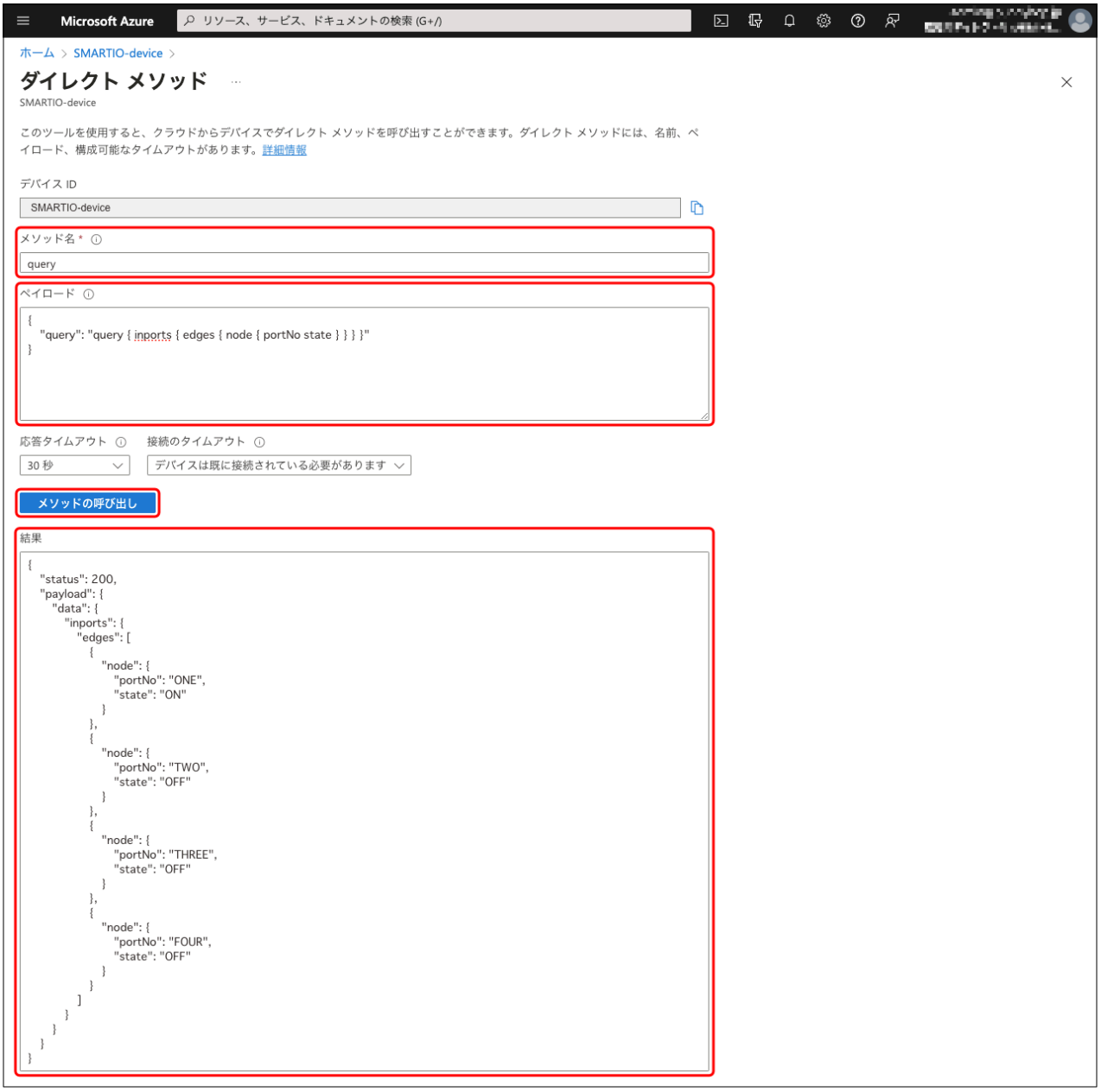
3-4. IoT Hubの画面上からクラウドAPIの実行

IoT Hubでは、ダイレクトメソッドを実行するためのツールが用意されています。
ツールを使用することで簡単にクラウドAPIを体験頂く事が可能です。

ツールを使用するには、IoT Hubのデバイスの詳細画面にアクセスし、「ダイレクトメソッド」を選択してダイレクトメソッドの画面にアクセスします。



メソッド名に`query`を入力し、ペイロードにGraphQLのリクエスト文を入力します。
「メソッドの呼び出し」ボタンを押すと、クエリが実行されて結果が表示されます。



3-5. クラウドAPI Examples

ここでは、クラウドAPIを実行するいくつかの例をご紹介します。

3-5-1. 入力ポート状態の取得

入力ポートの状態を取得するには、`inports query` を使用します。
以下のコードは入力ポートの状態を取得する例です。

`デバイスID` はAzure IoT Hub上で設定したデバイスIDを入力してください。

Python

`共有アクセスポリシーの接続文字列` は、共有アクセスポリシーの `service` のプライマリ接続文字列をご使用ください。

```
from azure.iot.hub import IoTHubRegistryManager
from azure.iot.hub.models import CloudToDeviceMethod
```

```
connection_str = "[共有アクセスポリシーの接続文字列]"
device_id = "[デバイスID]"

registry_manager = IoTHubRegistryManager.from_connection_string(connection_str)

deviceMethod = CloudToDeviceMethod(
  method_name="query",
  payload={
    "query": """
      query {
        inports {
          edges {
            node {
              portNo
              state
            }
          }
        }
      }
    """
  }
)

result = registry_manager.invoke_device_method(device_id, deviceMethod)
print(result)
```

Node.js

`共有アクセスポリシーの接続文字列` は、共有アクセスポリシーの `service` のプライマリ接続文字列をご使用ください。

```
import { Client } from "azure-iothub";

const connectionString = "[共有アクセスポリシーの接続文字列]";
const deviceId = "[デバイスID]";

const serviceClient = Client.fromConnectionString(connectionString);

serviceClient.open(function (err) {
 if (err) {
  console.error("Could not connect: " + err.message);
 } else {
  console.log("Service client connected");

  const methodParams = {
   methodName: "query",
   payload: {
    query: `
     query {
      inports {
       edges {
        node {
         portNo
         state
```

```
          }
         }
        }
       }
      `,
    },
    timeoutInSeconds: 30,
   };

   serviceClient.invokeDeviceMethod(
    deviceId,
    methodParams,
    function (err, result) {
     if (err) {
      console.error("Direct method error: " + err.message);
     } else {
      console.log(JSON.stringify(result));
     }
    }
   );
  }
});
```

cURL

SASトークンの取得にazコマンドを使用しています。
Azure Cloud Shell 環境ですとazコマンド等のインストールが不要で試すことができます。

IoT Hub名 はご使用の環境に合わせて変更してください。

```
$ SAS=$(az iot hub generate-sas-token -n [IoT Hub名] | jq .sas | tr -d '"')

$ QUERY='query {
 inports {
  edges {
   node {
    portNo
    state
   }
  }
 }
}'

$ curl -X POST \
https://[IoT Hub名].azure-devices.net/twins/[デバイスID]/methods?api-version=2021-04-12 \
-H "Authorization: $SAS" \
-H 'Content-Type: application/json' \
-d "{ \
 \"methodName\": \"query\", \
 \"responseTimeoutInSeconds\": 200, \
 \"payload\": { \"query\": \"$(echo $QUERY | tr -d '\n')\" } \
}"
```

実行結果

```
{
 "status":200,
 "payload":{
  "data":{
   "inports":{
    "edges":[
     {"node":{"portNo":"ONE","state":"ON"}},
     {"node":{"portNo":"TWO","state":"OFF"}},
     {"node":{"portNo":"THREE","state":"OFF"}},
     {"node":{"portNo":"FOUR","state":"OFF"}}
    ]
   }
  }
 }
}
```


3-5-2. 出力ポートの制御

出力ポートを制御するには、writeOutport mutation を使用します。
以下のコードは出力の1番ポートをONにする例です。

デバイスID はAzure IoT Hub上で設定したデバイスIDを入力してください。

Python

共有アクセスポリシーの接続文字列 は、共有アクセスポリシーの service のプライマリ接続文字列をご使用ください。

```
from azure.iot.hub import IoTHubRegistryManager
from azure.iot.hub.models import CloudToDeviceMethod

connection_str = "[共有アクセスポリシーの接続文字列]"
device_id = "[デバイスID]"

registry_manager = IoTHubRegistryManager.from_connection_string(connection_str)

deviceMethod = CloudToDeviceMethod(
  method_name="query",
  payload={
    "query": """
      mutation (
        $portNo: OutportPortNoEnum!
        $operation: OneofWriteOutportOperation!
        $logMessage: String!
      ) {
        writeOutport(
          portNo: $portNo
          input: { oneofOperation: $operation, logMessage: $logMessage }
        ) {
          id
        }
      }
    """,
    "variables": {
      "portNo": "ONE",
```

```
        "operation": {
          "on": {},
        },
        "logMessage": "Cloud API from python!",
      },
    }
)

result = registry_manager.invoke_device_method(device_id, deviceMethod)
print(result)
```

Node.js

`共有アクセスポリシーの接続文字列` は、共有アクセスポリシーの `service` のプライマリ接続文字列をご使用ください。

```
import { Client } from "azure-iothub";

const connectionString = "[共有アクセスポリシーの接続文字列]";
const deviceId = "[デバイスID]";

const serviceClient = Client.fromConnectionString(connectionString);

serviceClient.open(function (err) {
 if (err) {
  console.error("Could not connect: " + err.message);
 } else {
  console.log("Service client connected");

  const methodParams = {
   methodName: "query",
   payload: {
    query: `
     mutation (
      $portNo: OutportPortNoEnum!
      $operation: OneofWriteOutportOperation!
      $logMessage: String!
     ) {
      writeOutport(
       portNo: $portNo
       input: { oneofOperation: $operation, logMessage: $logMessage }
      ) {
       id
      }
     }
    `,
    variables: {
     portNo: "ONE",
     operation: {
      on: {},
     },
     logMessage: "Cloud API from javascript!",
    },
   },
   timeoutInSeconds: 30,
```

```
  };

  serviceClient.invokeDeviceMethod(
   deviceId,
   methodParams,
   function (err, result) {
    if (err) {
     console.error("Direct method error: " + err.message);
    } else {
     console.log(JSON.stringify(result));
    }
   }
  );
 }
});
```

cURL

SASトークンの取得にazコマンドを使用しています。
AWS Cloud Shell 環境ですとazコマンド等のインストールが不要で試すことができます。

`IoT Hub名` はご使用の環境に合わせて変更してください。

```
$ SAS=$(az iot hub generate-sas-token -n [IoT Hub名] | jq .sas | tr -d '"')

$ QUERY='mutation (
 $portNo: OutportPortNoEnum!
 $operation: OneofWriteOutportOperation!
 $logMessage: String!
) {
 writeOutport(
  portNo: $portNo
  input: {
   oneofOperation: $operation
   logMessage: $logMessage
  }
 ) {
  id
 }
}'

$ VARIABLES='{
 "portNo": "ONE",
 "operation": {
  "on": {}
 },
 "logMessage": "Cloud API from cURL!"
}'

$ curl -X POST \
https://[IoT Hub名].azure-devices.net/twins/[デバイスID]/methods?api-version=2021-04-12 \
-H "Authorization: $SAS" \
-H 'Content-Type: application/json' \
-d "{ \
```

```
  \"methodName\": \"query\", \
  \"responseTimeoutInSeconds\": 200, \
  \"payload\": { \"query\": \"$(echo $QUERY | tr -d '\n')\", \"variables\": $VARIABLES } \
}"
```