Armadillo-X2
Node-RED 開発ガイド

Version 1.5.3
2025/01/29

株式会社アットマークテクノ [https://www.atmark-techno.com]

Armadillo サイト [https://armadillo.atmark-techno.com]

Armadillo-X2 Node-RED 開発ガイド

株式会社アットマークテクノ

製作著作 © 2023-2025 Atmark Techno, Inc.

Version 1.5.3
2025/01/29

目次

1. はじめに 9
1.1. Node-RED について 9
1.2. 本書について 9
1.2.1. 本書で扱うこと 9
1.2.2. 本書で扱わないこと 9
1.2.3. 本書で必要となる知識と想定する読者 9
1.2.4. フォント 10
1.2.5. コマンド入力例 10
1.2.6. アイコン 10
1.2.7. ユーザー限定コンテンツ 11
1.2.8. 本書および関連ファイルのバージョンについて 11
2. ユーザー登録 12
2.1. 購入製品登録 12
3. Armadillo のセットアップ 13
3.1. 作業の前に 13
3.1.1. 開発セット内容物の確認 13
3.1.2. 開発に必要なもの 13
3.1.3. 接続方法 13
3.2. Node-RED コンテナをインストールする 15
3.3. インターフェースレイアウト 15
3.4. ネットワークに接続する 16
3.4.1. Armadillo の有線 LAN に固定 IP アドレスを設定する 19
3.5. Node-RED に接続する 21
4. 開発を行う 22
4.1. Node-RED に接続する 22
4.2. Node-RED コンテナのログを表示する 23
4.3. Node-RED で利用可能なノードの一覧 23
4.4. フローを作成する 24
4.4.1. LED を制御する 24
4.4.2. CPU の測定温度を取得する 28
4.4.3. RS-485 modbus RTU 読み出しを行う 30
4.4.4. CPU の測定温度のグラフをダッシュボードに表示する 33
4.4.5. 外部プログラムを実行する 41
4.4.6. Node-RED を終了する 43
4.5. ユーザデータを削除する 46
4.6. AWS へデバイス情報を送信するフローを作成する 46
4.6.1. AWS アカウントの作成 47
4.6.2. IAM ユーザーの作成 47
4.6.3. デバイスデータエンドポイントを取得する 49
4.6.4. デバイス証明書を取得するフローの作成 49
4.6.5. デバイスを登録するフローの作成 53
4.6.6. AWS IoT ポリシーを作成するフローの作成 57
4.6.7. デバイス証明書を登録するフローの作成 63
4.6.8. AWS IoT ポリシーをデバイスにアタッチするフローの作成 66
4.6.9. デバイスシャドウを取得するフローの作成 73
4.6.10. デバイスシャドウを更新するフローの作成 77
4.6.11. デバイスシャドウを削除するフローの作成 82
5. 量産する 87
5.1. 概略 87
5.1.1. リードタイムと在庫 87

5.1.2. Armadillo 納品後の製造・量産作業 88
5.2. BTO サービスを使わない場合と使う場合の違い 88
5.2.1. BTO サービスを利用しない(標準ラインアップ品) 89
5.2.2. BTO サービスを利用する 89
5.3. 量産時のイメージ書き込み手法 89
5.4. 開発したシステムをインストールディスクにする 90
5.4.1. 仮想環境のセットアップ 90
5.4.2. VS Code のセットアップ 97
5.4.3. VS Code を使用した初期設定用 SWU の生成 98
5.4.4. 初期設定用 SWU を ABOS Web からインストール 100
5.4.5. VS Code を使用したインストールディスク作成用 SWU の生成 101
5.4.6. USB メモリ上にインストールディスクイメージを生成 106
5.5. インストールディスクの動作確認を行う 106
5.5.1. インストールディスクを作成する 106
5.5.2. インストールディスクを使用する 107
5.6. アップデート用 SWU の生成手法 108
5.6.1. 更新用フローの取得 108
5.6.2. ATDE 上でアップデート用 SWU を生成 110
5.6.3. アップデート用 SWU の適用 111

図目次

3.1. 初回起動時の Node-RED 画面 13
3.2. Armadillo-X2 の接続例 14
3.3. インターフェースレイアウト (ケース内部) 15
3.4. パスワード登録画面 17
3.5. パスワード登録完了画面 18
3.6. ログイン画面 18
3.7. 状態一覧を選択 19
3.8. LAN 情報 19
3.9. 初回起動時の Node-RED 画面 21
4.1. 初回起動時の Node-RED 画面 22
4.2. Node-RED の画面領域 23
4.3. [inject] ノードのプロパティ内容 25
4.4. [trigger] ノードのプロパティ内容 26
4.5. [write file] ノードのプロパティ内容 27
4.6. LED を 1 秒間隔で点滅するフロー 27
4.7. [inject] ノードのプロパティ内容 28
4.8. [read file] ノードのプロパティ内容 29
4.9. [function] ノードのプロパティ内容 30
4.10. CPU の測定温度を 1 秒間隔で取得するフロー 30
4.11. [modbus-client] ノードのプロパティ内容 31
4.12. [Modbus-Read] ノードのプロパティ内容 32
4.13. RS-485 を使用した Modubus RTU 読み出し用フロー 33
4.14. [ダッシュボード]を選択する 33
4.15. ダッシュボード編集画面 34
4.16. ダッシュボードに [TAb1] を追加 34
4.17. [Tab1] プロパティ内容 35
4.18. ダッシュボード編集画面に [Armadillo] タブが追加 36
4.19. ダッシュボード編集画面に [Group1] グループが追加 37
4.20. [Group1] プロパティ内容 38
4.21. ダッシュボード編集画面に [chart] ノードが追加 39
4.22. [inject] ノードのプロパティ内容 40
4.23. CPU の測定温度のグラフをダッシュボードに表示するフロー 40
4.24. CPU の測定温度のグラフのダッシュボード 41
4.25. [exec] ノードのプロパティ内容 42
4.26. 外部プログラムを実行するフロー 42
4.27. [inject] ノードのプロパティ内容 44
4.28. [exit] ノードのプロパティ内容 45
4.29. Node-RED を任意のタイミングで終了するフロー 45
4.30. ポリシーの例 47
4.31. [デバイス証明書の取得] ノードのプロパティ内容 50
4.32. [ファイルの確認] ノードのプロパティ内容 51
4.33. [リファレンスキーの取得] ノードのプロパティ内容 52
4.34. デバイス証明書を取得するフロー 52
4.35. ファイル一式 53
4.36. [接続情報の登録] ノードのプロパティ内容 54
4.37. [exec queue] ノードのプロパティ内容 55
4.38. [exec queue] ノードのプロパティ内容 56
4.39. [デバイス名の取得] ノードのプロパティ内容 57
4.40. デバイスを登録するフロー 57
4.41. 登録したモノの名前 57

4.42. [接続情報の登録] ノードのプロパティ内容 59
4.43. [ポリシー登録の引数設定] ノードのプロパティ内容 60
4.44. [新しいポリシーの作成] ノードのプロパティ内容 61
4.45. [ポリシー名の取得] ノードのプロパティ内容 62
4.46. ポリシーを作成するフロー 62
4.47. 登録したポリシーの名前 62
4.48. [デバイス証明書登録の引数設定] ノードのプロパティ内容 63
4.49. [デバイス証明書の登録] ノードのプロパティ内容 64
4.50. [証明書 ID の取得] ノードのプロパティ内容 65
4.51. デバイス証明書を登録するフロー 65
4.52. 証明書 ID 名 66
4.53. [接続情報の登録] ノードのプロパティ内容 67
4.54. [ポリシーをアタッチするための引数設定] ノードのプロパティ内容 68
4.55. [ポリシーを証明書にアタッチ] ノードのプロパティ内容 69
4.56. [ポリシーアタッチ結果の取得] ノードのプロパティ内容 70
4.57. [証明書をデバイスにアタッチ] ノードのプロパティ内容 71
4.58. [証明書アタッチ結果の取得] ノードのプロパティ内容 72
4.59. デバイスにポリシーをアタッチするフロー 73
4.60. [接続情報の登録] ノードのプロパティ内容 74
4.61. [デバイスシャドウ取得のための引数設定] ノードのプロパティ内容 75
4.62. [暗号化を使用したデバイスシャドウの取得] ノードのプロパティ内容 76
4.63. [デバイスシャドウ内容の取得] ノードのプロパティ内容 77
4.64. データの暗号化を使用したデバイスシャドウを取得するフロー 77
4.65. [デバイスシャドウ内容] ノードのプロパティ内容 78
4.66. [接続情報の登録] ノードのプロパティ内容 79
4.67. [デバイスシャドウ更新のための引数設定] ノードのプロパティ内容 80
4.68. [暗号化を使用したデバイスシャドウの更新] ノードのプロパティ内容 81
4.69. [デバイスシャドウ内容の取得] ノードのプロパティ内容 81
4.70. データの暗号化を使用したデバイスシャドウを更新するフロー 82
4.71. [接続情報の登録] ノードのプロパティ内容 83
4.72. [デバイスシャドウ削除のための引数設定] ノードのプロパティ内容 84
4.73. [暗号化を使用したデバイスシャドウの削除] ノードのプロパティ内容 85
4.74. [デバイスシャドウ内容の取得] ノードのプロパティ内容 85
4.75. データの暗号化を使用したデバイスシャドウを削除するフロー 86
5.1. Armadillo 量産時の概略図 87
5.2. BTO サービスで対応する範囲 88
5.3. GNOME 端末の起動 93
5.4. GNOME 端末のウィンドウ 93
5.5. ソフトウェアをアップデートする 94
5.6. ATDE にデバイスを接続する 94
5.7. 共有フォルダー設定を開く 95
5.8. 共有フォルダー設定 96
5.9. 共有フォルダーの追加 96
5.10. 「ファイル」に表示される共有フォルダー 97
5.11. VS Code を起動する 97
5.12. VS Code に開発用エクステンションをインストールする 98
5.13. initial_setup.swu を作成する 98
5.14. initial_setup.swu 初回生成時の各種設定 99
5.15. SWU インストール 101
5.16. make-installer.swu を作成する 102
5.17. 対象製品を選択する 102
5.18. make-installer.swu 生成時のログ 102
5.19. SWU インストール 104

Armadillo-X2 Node-RED 開発ガイド

5.20. make-installer.swu インストール時のログ 104
5.21. Win32 Disk Imager Renewal 設定画面 107
5.22. インストールディスクを作成する 107
5.23. フローの書き出し 109
5.24. フローのダウンロード 110
5.25. フローの配置 110
5.26. SWU の生成 110

Armadillo-X2 Node-RED 開発ガイド

表目次

1.1. 使用しているフォント 10
1.2. 表示プロンプトと実行環境の関係 10
1.3. コマンド入力例での省略表記 10
3.1. 開発に必要なもの 13
3.2. インターフェース内容 16
3.3. シリアル通信設定 20
4.1. LED 信号配列 24
4.2. セキュアエレメントが使用する I2C バス 49
5.1. ユーザー名とパスワード 92

1. はじめに

このたびは Armadillo をご利用いただき、ありがとうございます。

本書では、Armadillo-X2 上での Node-RED を用いた開発手法を説明します。

Armadillo-X2 の詳細な説明に関しては、製品マニュアルに記載しておりますので、以下の URL よりダウンロードしてください。

Armadillo サイト - Armadillo-X2 ドキュメントダウンロード

https://armadillo.atmark-techno.com/armadillo-x2/resources/documents

1.1. Node-RED について

Node-RED は、IoT アプリケーション開発に適した、オープンソースのローコード開発ツールです。「ノード」と呼ばれる機能ブロックを繋ぎ合わせてアプリケーションを作ります。ビジュアルプログラミング環境のため、Python や JavaScript といったプログラミングの知識がなくてもアプリケーションを作れます。PLC やリモート I/O などの産業機器からのデータ収集や、それらの制御を行うアプリケーション用のノードもあり、製造業 DX の内製ツールとしても注目されています。Armadillo-X2 ですぐに使えるようにした Node-RED コンテナを提供します。

Node-RED は、 OpenJS Foundation の米国およびその他の国における登録商標または商標です。

1.2. 本書について

1.2.1. 本書で扱うこと

本書では Armadillo-X2 へ Node-RED コンテナをインストールし、Armadillo-X2 上で Node-RED を操作し開発する方法を説明します。

1.2.2. 本書で扱わないこと

Node-RED の基本的な説明は記載しておりません。Node-RED の使用方法は各種書籍・Web サイトを参照ください。

また、Armadillo をご利用になられる際の注意事項は、Armadillo 製品マニュアル「注意事項」の章をご一読ください。

1.2.3. 本書で必要となる知識と想定する読者

Node-RED を使用したことがあり、使い方を把握しているエンジニアを対象読者として想定しております。

1.2.4. フォント

本書では以下のような意味でフォントを使いわけています。

表 1.1 使用しているフォント

フォント例	説明
本文中のフォント	本文
[PC ~]$ ls	プロンプトとユーザ入力文字列
text	編集する文字列や出力される文字列。またはコメント

1.2.5. コマンド入力例

本書に記載されているコマンドの入力例は、表示されているプロンプトによって、それぞれに対応した実行環境を想定して書かれています。「 / 」の部分はカレントディレクトリによって異なります。各ユーザのホームディレクトリは「 ~ 」で表します。

表 1.2 表示プロンプトと実行環境の関係

プロンプト	コマンドの実行環境
[PC /]#	作業用 PC の root ユーザで実行
[PC /]$	作業用 PC の一般ユーザで実行
[ATDE ~/]#	ATDE 上の root ユーザで実行
[ATDE ~/]$	ATDE 上の一般ユーザで実行
[armadillo /]#	Armadillo 上 Linux の root ユーザで実行
[armadillo /]$	Armadillo 上 Linux の一般ユーザで実行
[container /]#	Podman コンテナ内で実行
⇒	Armadillo 上 U-Boot の保守モードで実行

コマンド中で、変更の可能性のあるものや、環境により異なるものに関しては以下のように表記します。適宜読み替えて入力してください。

表 1.3 コマンド入力例での省略表記

表記	説明
[VERSION]	ファイルのバージョン番号

1.2.6. アイコン

本書では以下のようにアイコンを使用しています。

注意事項を記載します。

役に立つ情報を記載します。

用語の説明や補足的な説明を記載します。

1.2.7. ユーザー限定コンテンツ

アットマークテクノ Armadillo サイトで購入製品登録を行うと、製品をご購入いただいたユーザーに限定して公開している限定コンテンツにアクセスできるようになります。主な限定コンテンツには、下記のものがあります。

・ 各種信頼性試験データ・納入仕様書等製造関連情報

限定コンテンツを取得するには、「2. ユーザー登録」を参照してください。

1.2.8. 本書および関連ファイルのバージョンについて

本書を含めた関連マニュアル、ソースファイルやイメージファイルなどの関連ファイルは最新版を使用することをおすすめいたします。本書を読み始める前に、Armadillo サイトで最新版の情報をご確認ください。

Armadillo サイト - Armadillo-X2 ドキュメントダウンロード

https://armadillo.atmark-techno.com/armadillo-x2/resources/documents

Armadillo サイト - Armadillo-X2 ソフトウェアダウンロード

https://armadillo.atmark-techno.com/armadillo-x2/resources/software

2. ユーザー登録

アットマークテクノ製品をご利用のユーザーに対して、購入者向けの限定公開データの提供や大切なお知らせをお届けするサービスなど、ユーザー登録すると様々なサービスを受けることができます。サービスを受けるためには、「アットマークテクノ Armadillo サイト」にユーザー登録をする必要があります。

ユーザー登録すると次のようなサービスを受けることができます。

- 製品仕様や部品などの変更通知の閲覧・配信
- 購入者向けの限定公開データのダウンロード
- 該当製品のバージョンアップに伴う優待販売のお知らせ配信
- 該当製品に関する開発セミナーやイベント等のお知らせ配信

詳しくは、「アットマークテクノ Armadillo サイト」をご覧ください。

アットマークテクノ Armadillo サイト

https://armadillo.atmark-techno.com/

2.1. 購入製品登録

ユーザー登録完了後に、購入製品登録することで、「購入者向けの限定公開データ」をダウンロードすることができるようになります。

購入製品登録の詳しい手順は以下の URL をご参照ください。

Armadillo-X2 購入製品登録

https://armadillo.atmark-techno.com/armadillo-x2/register

3. Armadillo のセットアップ

Armadillo-X2 上で Node-RED を使用するためのセットアップ方法を説明します。Armadillo-X2 に Node-RED 対応用のインストールディスクをインストールし、開発用パソコンと Armadillo-X2 を有線 LAN で繋がる様にします。セットアップが完了すると Node-RED の起動画面が表示されます。

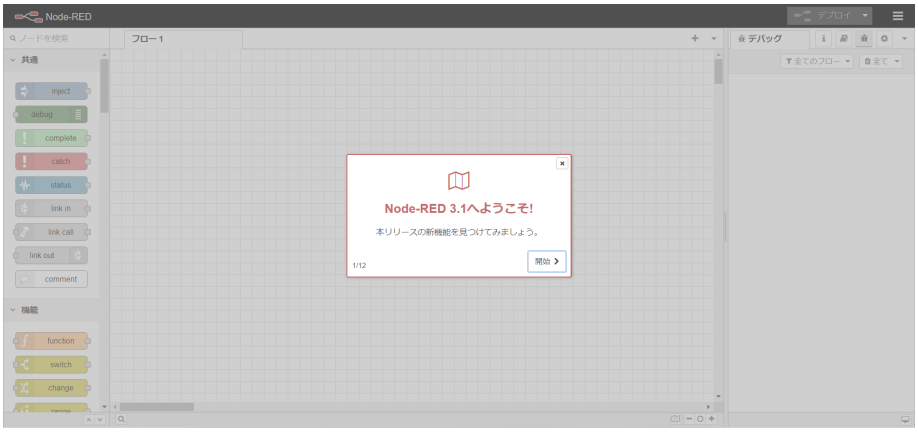


図 3.1 初回起動時の Node-RED 画面

3.1. 作業の前に

3.1.1. 開発セット内容物の確認

開発セットに同梱されている内容物は、同梱の内容物一覧でご確認いただけます。 お使いになる前に、内容物がすべて揃っていることをご確認ください。 万一、内容物の不足または部品の破損等がございましたら、ご購入の販売代理店までご連絡ください。

3.1.2. 開発に必要なもの

Armadillo-X2 上で Node-RED を使用するためには以下のものが必要です。

表 3.1 開発に必要なもの

品目	説明
Armadillo-X2 開発セット	本製品です。
開発用パソコン	Linux または Windows が動作し、ネットワークインターフェースと 1 つ以上の USB ポート、microSD ポートを持つパソコンです。
有線 LAN ケーブル	本ガイドでは Armadillo-X2 と開発用パソコンを有線 LAN で接続する前提で記載しています。
512MB 以上の microSD カード	Node-RED の開発環境をインストールするために使用します。開発用パソコンの SD ポートが microSD ではなく、SD/miniSD ポートの場合、microSD へ変換するアダプターも必要となります。

3.1.3. 接続方法

Armadillo-X2 と周辺装置の接続例を「図 3.2. Armadillo-X2 の接続例」に示します。

開発用パソコンとのファイル転送やインターネットへの接続は、LAN を介して行いますので LAN ケーブルを接続してください。

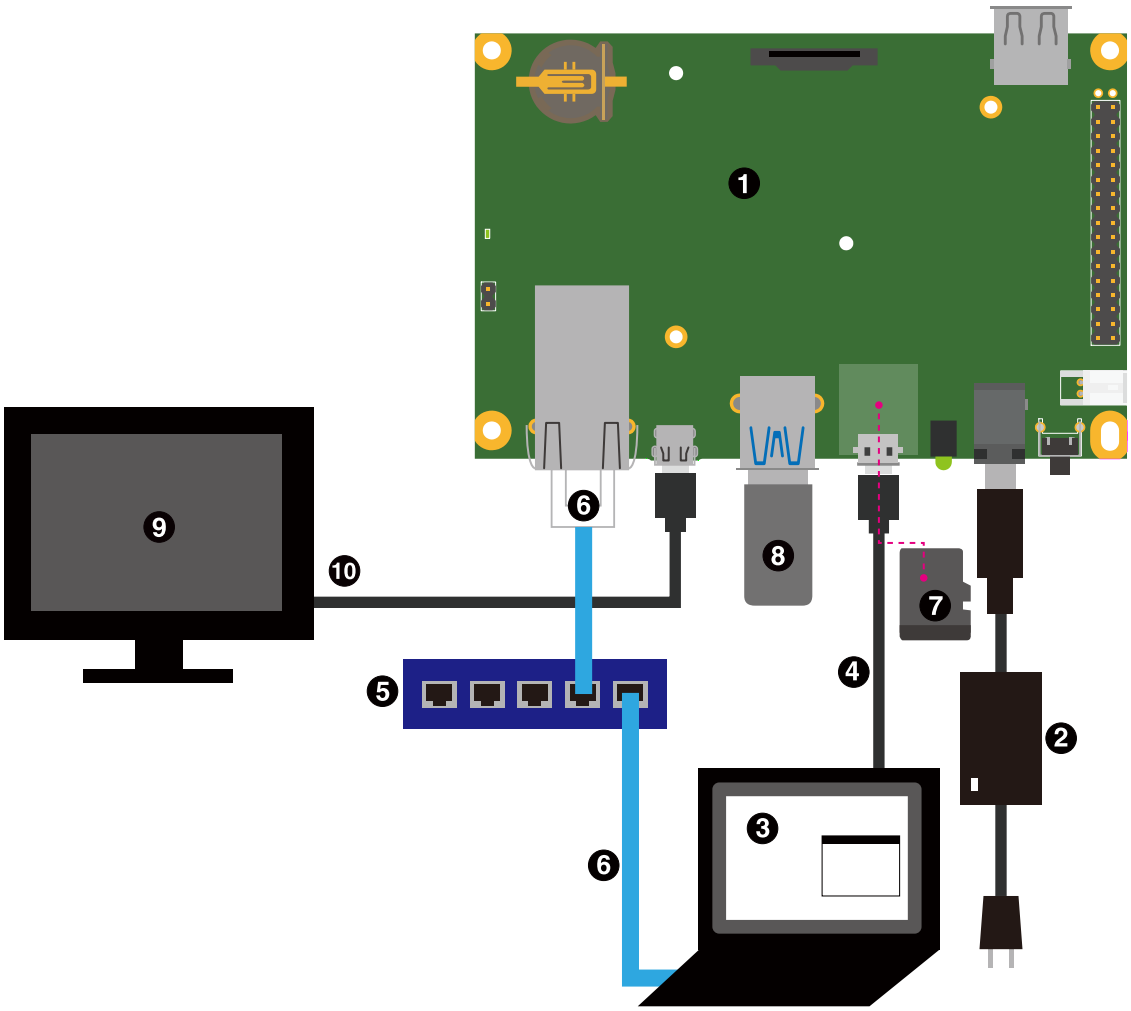


図 3.2 Armadillo-X2 の接続例

1. Armadillo-X2
2. AC アダプタ(12V/3.0A)
3. 作業用 PC
4. シリアル通信用 USB ケーブル(A-microB)
5. LAN HUB
6. Ethernet ケーブル
7. microSD カード
8. USB メモリ
9. ディスプレイ(HDMI 対応)
10. HDMI ケーブル

3.2. Node-RED コンテナをインストールする

SWUpdate の機能を使用して SWU で Armadillo Base OS を最新版にアップデートし、Node-RED コンテナをインストールします。SWUpdate で使用する SWU イメージは以下から取得可能です。

- Armadillo Base OS

  Armadillo-X2 Armadillo Base OS [https://armadillo.atmark-techno.com/resources/software/armadillo-x2/baseos] から「Armadillo-X2 用 SWU メージファイル」をダウンロードしてください。

- Node-RED コンテナ

  Armadillo-X2 Node-RED コンテナ [https://armadillo.atmark-techno.com/resources/software/armadillo-x2/node-red-container] から「Armadillo-X2 用 SWU イメージファイル」をダウンロードしてください。

上記で取得した SWU イメージを USB メモリに配置します。USB メモリを Armadillo-X2 に接続すると自動的にアップデートが始まります。アップデート終了後に Armadillo-X2 は自動で再起動します。

3.3. インターフェースレイアウト

Armadillo-X2 のインターフェースレイアウトです。各インターフェースの配置場所等を確認してください。

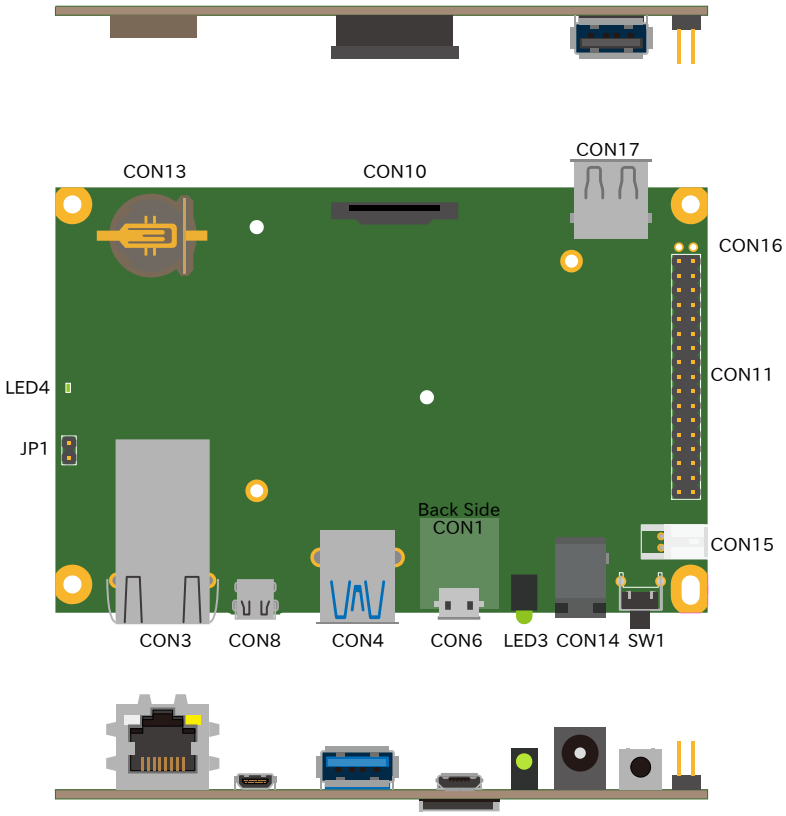


図 3.3 インターフェースレイアウト (ケース内部)

表 3.2 インターフェース内容

部品番号	インターフェース名	説明
CON1	SD インターフェース	microSD カード用の SD スロットです。外部ストレージが必要な場合や、microSD カードに配置したブートディスクイメージから起動する場合などに使用します。
CON3	LAN インターフェース	LAN ケーブル接続用の RJ-45 コネクタです。有線 LAN を利用する場合に使用します。
CON4	USB インターフェース	USB メモリや USB 接続デバイス接続用の USB3.0 Type-A コネクタです。外部ストレージが必要な場合や、USB 接続の各種デバイスを使用する場合に使用します。
CON6	USB コンソールインターフェース	USB microB ケーブル接続用の USB micro-B コネクタです。コンソール入出力を利用する場合に使用します。
CON8	HDMI インターフェース	HDMI Type-D ケーブル接続用の HDMI Type-D コネクタです。HDMI 対応ディスプレイ等を利用する場合に使用します。
CON10	MIPI-CSI インターフェース	MIPI CSI-2 対応カメラ接続用の 15 ピン(1mm ピッチ)FFC コネクタです。MIPI CSI-2 対応カメラを利用する場合に使用します。
CON11	拡張インターフェース 1	2 列 17 ピン(2.54mm ピッチ)のピンヘッダが搭載されています。機能拡張する場合に使用します
CON13	RTC バックアップインターフェース	CR2032 等のコイン形電池接続用の電池ボックスです。リアルタイムクロックのバックアップ給電が必要な場合に使用します。
CON14	電源入力インターフェース 1	AC アダプタ接続用 DC ジャックです。付属の AC アダプタから Armadillo-IoT ゲートウェイ G4 へ電源供給する場合に使用します[a]。
CON15	電源入力インターフェース 2	電源入力用 2 ピン(2mm ピッチ)ライトアングルコネクタです。AC アダプタ以外の電源装置からら Armadillo-IoT ゲートウェイ G4 へ電源供給する場合に使用します[a]。
CON16	3.3V 電源出力インターフェース	2 ピン(2.54mm ピッチ)のピンヘッダが搭載されています。機能拡張する場合の 3.3V 電源として使用できます。
CON17	USB インターフェース 2	USB メモリや USB 接続デバイス接続用の USB 2.0 Type-A コネクタです。外部ストレージが必要な場合や、USB 接続の各種デバイスを使用する場合に使用します。
JP1	起動デバイス設定ジャンパ	起動モードを設定するための 2 ピン(2.54mm ピッチ)ピンヘッダです。
SW1	ユーザースイッチ	ユーザーが利用可能なタクトスイッチです。
LED3	ユーザー LED	ユーザーが利用可能な砲弾タイプ(Φ3mm)の緑色 LED です。
LED4	電源 LED	電源の入力状態を表示する表面実装タイプの緑色 LED です。

[a]電源入力インターフェース 1 と電源入力インターフェース 2 の両方から同時に電源供給することはできません。

3.4. ネットワークに接続する

Node-RED の開発をするパソコンと Armadillo のネットワークを接続するために Armadillo-X2 の IP アドレスを取得または設定します。

Armadillo の有線 LAN は、初期状態で DHCP の設定となっておりますので、DHCP が稼働している有線 LAN に接続した場合は、なんらかの IP アドレスが付与された状態になっています。

DHCP で IP アドレスが付与される環境の場合、ABOS Web を使用することで IP アドレスの確認が可能です。

1. Armadillo-X2 を 「3.2. Node-RED コンテナをインストールする」 でセットアップを済ませた場合は ABOS Web が起動しています。再起動がまだの場合は電源を再投入します。
2. 開発用パソコンで Web ブラウザーを起動し https://armadillo.local:58080 にアクセスします。

ABOS Web が動作する Armadillo が、同じ LAN 上に複数ある場合、ABOS Web に接続する URL のホスト名部分

（armadillo.local）は、2 台目は armadillo-2.local、3 台目は armadillo-3.local と、違うものが自動的に割り当てられます。この場合どの URL がどの Armadillo かを判別するのは難しいので固定 IP アドレスを設定し、IP アドレスで指定できるようにする方法があります。固定 IP アドレスの設定方法は「3.4.1. Armadillo の有線 LAN に固定 IP アドレスを設定する」を参照ください。

3. 初回接続時は、ABOS Web のパスワードを設定する必要がありますので、設定します。

図 3.4 パスワード登録画面

4. "初回ログイン"のパスワード登録画面で、"パスワード" フィールドと "パスワード(確認)" フィールドに、登録したいパスワードを入力してから、"登録" ボタンをクリックしてください。パスワード登録完了画面が表示されたら、パスワード登録の完了です。

図 3.5 パスワード登録完了画面

5. ログインします。

図 3.6 ログイン画面

6. 左のメニューから「状態一覧」を選択します。

図 3.7 状態一覧を選択

7. 「LAN 情報」の「IP アドレス」の欄に有線 LAN の IP アドレスが表示されていますのでメモしておいてください。

図 3.8 LAN 情報

3.4.1. Armadillo の有線 LAN に固定 IP アドレスを設定する

DHCP サーバーが存在しない環境で開発を行う場合、Armadillo-X2 に固定の IP アドレスを付与して開発を行います。

Armadillo にシリアルコンソールを接続してコマンドを入力する必要があります。シリアルコンソールの接続方法は 「3.1.3. 接続方法」 を参照ください。

1. Armadillo-X2 と開発用パソコンを開発セットに同梱されています USB(A オス-microB)ケーブルで接続します。Armadillo-X2 側は 「図 3.2. Armadillo-X2 の接続例」 に示す「❹」の microB を、開発用パソコン側は Type-A を接続してください。

2. 開発用パソコンでシリアルコンソールを開きます。対応しているソフトウェアは、Windows OS であれば TeraTerm、Linux であれば minicom などがありますので、インストールしてご利用ください。

3. ソフトウェアのシリアル設定を「表 3.3. シリアル通信設定」に示します。

表 3.3 シリアル通信設定

項目	設定
転送レート	115,200bps
データ長	8bit
ストップビット	1bit
パリティ	なし
フロー制御	なし

1. Armadillo-X2 の AC アダプターをコンセントに繋ぎ電源を投入します。

2. シリアルコンソールに以下のようにログインプロンプトが表示されましたら、root ユーザーでログインします。特に設定していない場合の root ユーザーの初期パスワードは root です。

```
Welcome to Alpine Linux 3.18
Kernel 5.10.197-0-at on an armv7l (/dev/ttymxc2)

armadillo login:
```

3. "Wired connection 1" に固定 IP アドレスを設定します。

```
[armadillo ~]# nmcli connection modify ”Wired connection 1” ¥
ipv4.method manual ipv4.addresses 192.0.2.10/24 ipv4.gateway 192.0.2.1
```

4. 有線 LAN eth0 を再起動します。

```
[armadillo ~]# nmcli connectio down ”Wired connection 1”
[armadillo ~]# nmcli connectio up ”Wired connection 1”
```

5. 設定された IP アドレスは ip addr コマンドで確認できます。

```
[armadillo ~]# ip addr show eth0
2: eth0: ...中略...
    inet 192.0.2.10/24 brd 192.0.2.255 scope global noprefixroute eth0
       valid_lft forever preferred_lft forever
... 後略 ...
```

6. 設定を永続化するために、persist_file コマンドを実行してください。

```
[armadillo ~]# persist_file /etc/NetworkManager/system-connections/"Wired connection
1.nmconnection"
```

3.5. Node-RED に接続する

パソコンの Web ブラウザから、「3.4. ネットワークに接続する」 で取得した IP アドレスを使用して、http://<ip アドレス>:1880/ にアクセスしてください。

Node-RED の起動画面が表示されたらセットアップは終了です。

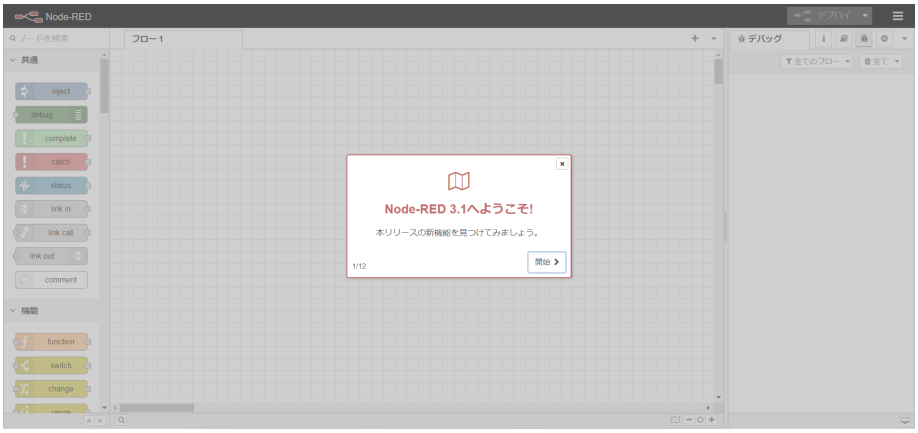


図 3.9 初回起動時の Node-RED 画面

4. 開発を行う

4.1. Node-RED に接続する

「3. Armadillo のセットアップ」 を完了している場合は Armadillo を起動すると、自動的に Node-RED コンテナが起動します。Node-RED への接続には 「3.4. ネットワークに接続する」 で取得した IP アドレスを使用します。

パソコンの Web ブラウザから、`http://<ip アドレス>:1880/` にアクセスしてください。

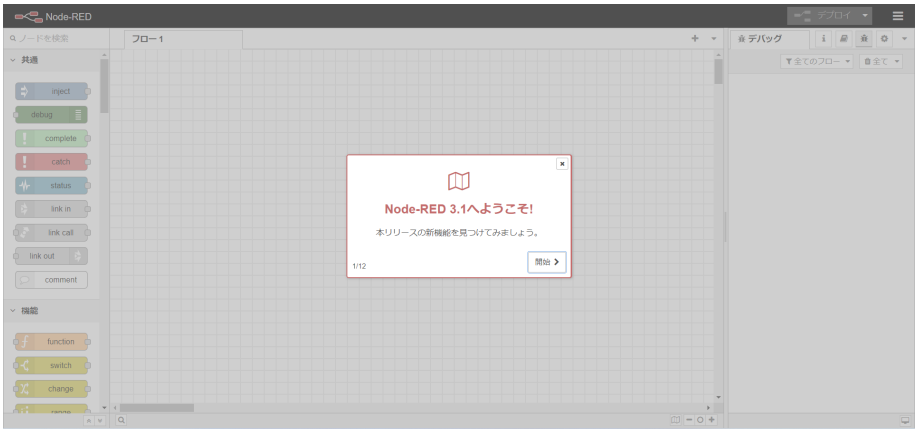


図 4.1 初回起動時の Node-RED 画面

Node-RED には大きく 3 つの領域があります。

- パレット

  使用可能なコアノード、カスタムノードの一覧を表示します。

- ワークスペース

  Node-RED では一つのアプリケーションの流れをフローという単位で表します。ワークスペース上にノードを繋げてフローを作成します。パレットから必要なノードをワークスペースにドラッグ、ドロップで配置します。

- サイドバー

  選択したノードの情報や、デバッグメッセージを表示します。

図 4.2 Node-RED の画面領域

4.2. Node-RED コンテナのログを表示する

パソコンの Web ブラウザから、http://<ip アドレス>:1880/node-red.log にアクセスすると Node-RED コンテナのログが表示されます。

このログファイルは /var/app/rollback/volumes/node-red/log に保存されています。ログのサイズが 5MB を超えると新しいログファイルに移行します。古いログファイルを表示したい場合は、http://<ip アドレス>:1880/node-red.log.1 にアクセスしてください。

4.3. Node-RED で利用可能なノードの一覧

「3. Armadillo のセットアップ」 でインストールした Node-RED コンテナには Node-RED のコアノードの他にインストールされているカスタムノードがあります。インストール済のカスタムノードは以下になります。

- Dashboard ノード

  Dashboard を表示するノードです。

- exit ノード

  Node-RED を終了するためのノードです。

- exec queue ノード

  exec を複数記載できるノードです。

- credentials ノード

  設定内容を暗号化して保持するためのノードです。

- AWS ノード

  AWS サービスを利用するためのノードです。暗号化が不要な場合は、AWS IoTData ノードを使用してデバイスシャドウの操作が可能です。

4.4. フローを作成する

Node-RED では一つのアプリケーションの流れをフローという単位で表します。各ノードを使用したフローの作成方法について紹介します。

4.4.1. LED を制御する

ここでは、LED を 1 秒間隔で点滅するフローを作成します。すべて Node-RED のコアノードを使用します。Armadillo-X2 で使用できる LED としてユーザー LED(緑)があります。ユーザー LED(緑) の場所については 「3.3. インターフェースレイアウト」 をご確認ください。

ユーザー LED(緑) は `/sys/class/leds/led1/brightness` ファイルへ値を書き込むことによって、LED の点灯/消灯を行うことができます。0 を書き込むと消灯、0 以外の値 (1〜255) を書き込むと点灯します。

表 4.1 LED 信号配列

部品番号	名称(色)	説明
LED3	ユーザー LED(緑)	トランジスタを経由して i.MX 8M Plus の GPIO1_IO14 ピンに接続 (Low: 消灯、High: 点灯)

1. パレットから [inject] ノードをワークスペースにドラッグ、ドロップします。
2. [inject] ノードのプロパティを編集します。[inject] は 2 秒間隔で実行します。

```
名前: LED Blink
繰り返し: 指定した時間間隔
時間間隔：2 秒
```

図 4.3 [inject] ノードのプロパティ内容

3. [trigger] ノードをドラッグ、ドロップします。

4. [trigger] ノードのプロパティを編集します。[trigger] は 1 を送信して 1 秒待機後、0 を送信します。これにより点滅動作を実現します。

```
送信データ：1
送信後の処理：指定した時間待機
1 秒
再送信データ：0
```

図 4.4 [trigger] ノードのプロパティ内容

5. [LED Blink] ノードの右側にある端子をクリックし、[trigger] ノードの左側の端子を選択して放します。

6. [write file] ノードをドラッグ、ドロップします。

7. [write file] ノードのプロパティを編集します。/sys/class/leds/led1/brightness ファイルへ [trigger] からの送信データを書き込みます。

```
ファイル名: /sys/class/leds/led1/brightness
動作: ファイルを上書き
文字コード: デフォルト
名前: LED1
```

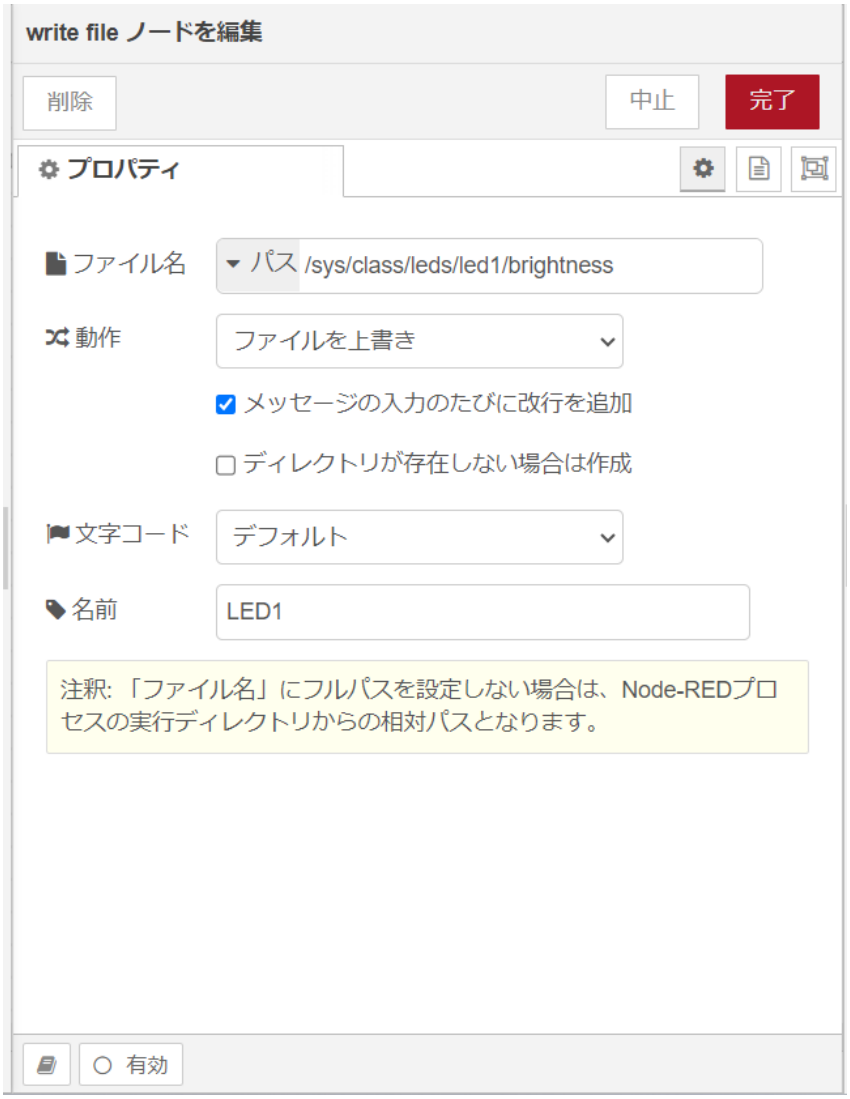


図 4.5 [write file] ノードのプロパティ内容

8. [trigger] ノードの右側にある端子をクリックし、[LED1] ノードの左側の端子を選択して放します。

9. [debug] ノードをドラッグ、ドロップします。

10. [LED1] ノードの右側にある端子をクリックし、[debug] ノードの左側の端子を選択して放します。

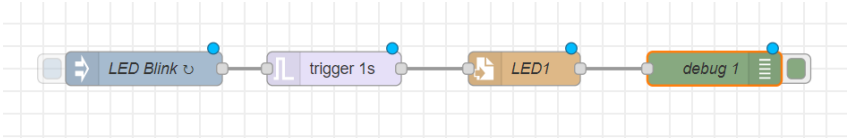


図 4.6 LED を 1 秒間隔で点滅するフロー

11. 画面右上の [デプロイ] を押します。ユーザー LED(緑) が 1 秒毎に点滅を繰り返す動きをすれば成功です。

4.4.2. CPU の測定温度を取得する

ここでは、 Armadillo-X2 の CPU の測定温度を 1 秒間隔で取得するフローを作成します。全て Node-RED のコアノードを使用します。/sys/class/thermal/thermal_zone0/temp ファイルの値を読み出すことによって測定温度を取得することができます。温度はミリ℃ の単位で表示されるため、℃ 単位への変換も行います。

1. パレットから [inject] ノードをワークスペースにドラッグ、ドロップします。
2. [inject] ノードのプロパティを編集します。[inject] は 1 秒間隔で実行します。

> 繰り返し：指定した時間間隔
> 時間間隔：1 秒

図 4.7 [inject] ノードのプロパティ内容

3. [read file] ノードをドラッグ、ドロップします。
4. [read file] ノードのプロパティを編集します。/sys/class/thermal/thermal_zone0/temp ファイルの値を読み出します。

```
ファイル名: /sys/class/thermal/thermal_zone0/temp
出力形式: 文字列
文字コード: デフォルト
名前: CPU temp
```

図 4.8 [read file] ノードのプロパティ内容

5. [Get CPU temp] ノードの右側にある端子をクリックし、[CPU temp] ノードの左側の端子を選択して放します。

6. [function] ノードをドラッグ、ドロップします。

7. [function] ノードのプロパティを編集します。CPU の測定温度がミリ°C の単位のため、°C の単位へ変換します。msg.payload で渡された値を 1000 で割り、msg.payload に戻します。

```
名前: CPU temp calc
コード:
```

```
msg.payload = msg.payload / 1000;
return msg;
```

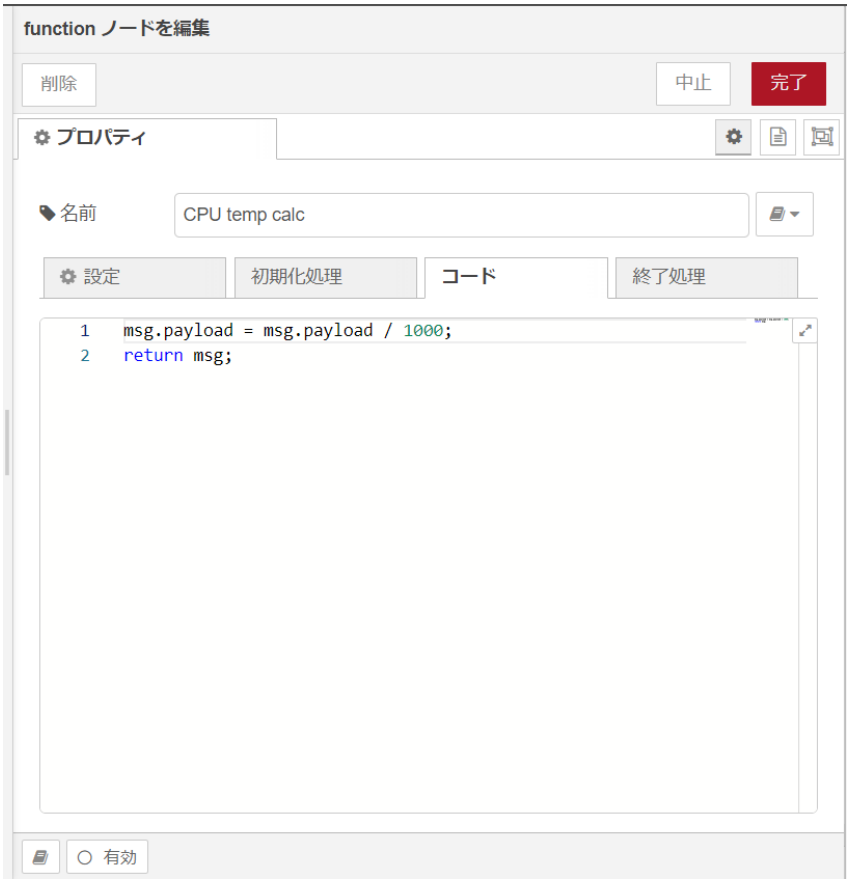


図 4.9 [function] ノードのプロパティ内容

8. [CPU temp] ノードの右側にある端子をクリックし、[CPU temp calc] ノードの左側の端子を選択して放します。
9. [debug] ノードをドラッグ、ドロップします。
10. [CPU temp calc] ノードの右側にある端子をクリックし、[debug] ノードの左側の端子を選択して放します。

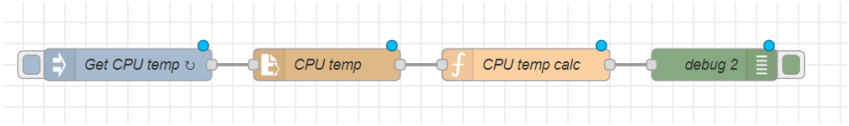


図 4.10 CPU の測定温度を 1 秒間隔で取得するフロー

11. 画面右上の [デプロイ] を押します。デバッグメッセージに 1 秒毎に CPU の測定温度が表示されます。

4.4.3. RS-485 modbus RTU 読み出しを行う

ここでは、 USB RS-485 変換器を使用した Modubus RTU 読み出し用フローを作成します。Node-RED のコアノードのほかに modbus-client ノード、Modbus-Read ノード、 modbus-response ノー

ドを使用します。RS-485 シリアルインターフェースのデバイスファイルは、/dev/ttyUSB0 を使用します。USB の場所については 「3.3. インターフェースレイアウト」 をご確認ください。今回は以下の RS-485 通信対応デバイスを想定した場合の設定内容となります。実際に動作確認する場合は、使用する RS-485 通信対応デバイスの設定に変更してください。

```
ユニット ID: 1
通信プロトコル: Modbus RTU
ボーレート: 9600
読み出しアドレス: 0x00
ファンクションコード: 1
```

1. パレットから [Modbus-Read] ノードをワークスペースにドラッグ、ドロップします。
2. [Modbus-Read] ノードのプロパティを編集します。
3. 先に Server を設定する必要があります。[新規に modbus-client を追加] の右隣の編集ボタンを押して [modbus-client] を作成します。

```
名前: RS485 slave device
Type: Serial
Serial port: /dev/ttyUSB0
Serial type: RTU
Baud rate: 9600
Unit-Id: 1
```

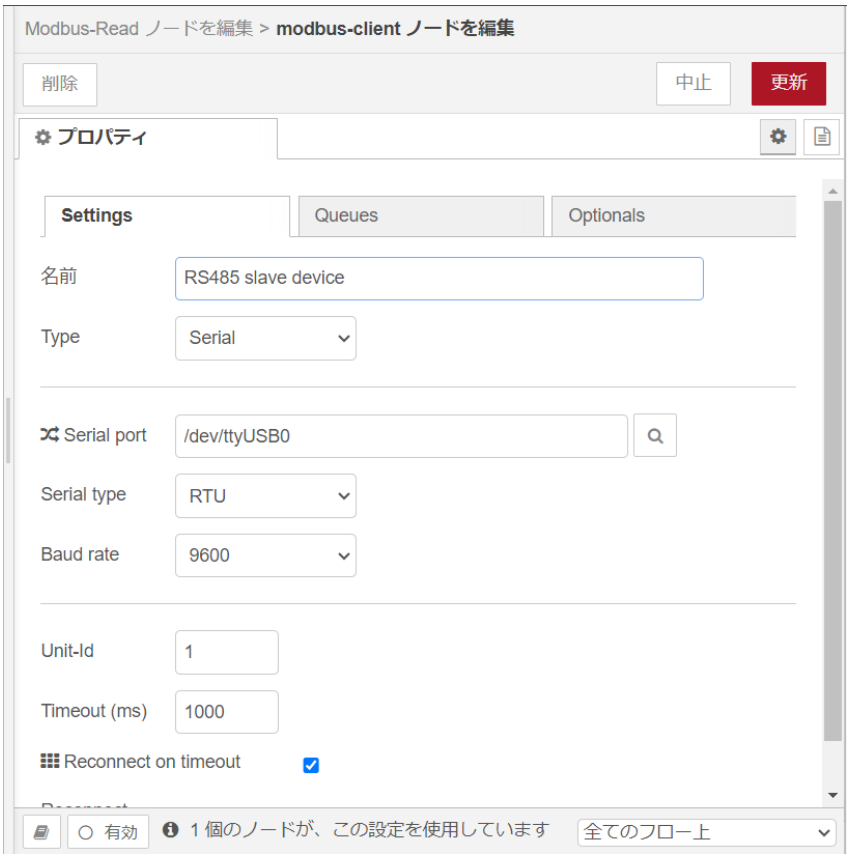


図 4.11 [modbus-client] ノードのプロパティ内容

4. [Modbus-Read] ノードのプロパティを編集します。

```
名前: slave device
トピック:
Unit-Id:
FC: FC 1:Read Coil Status
Address: 0
Quantity: 1
Poll Rate: 1 Second(s)
Server: RS485 slave device
```

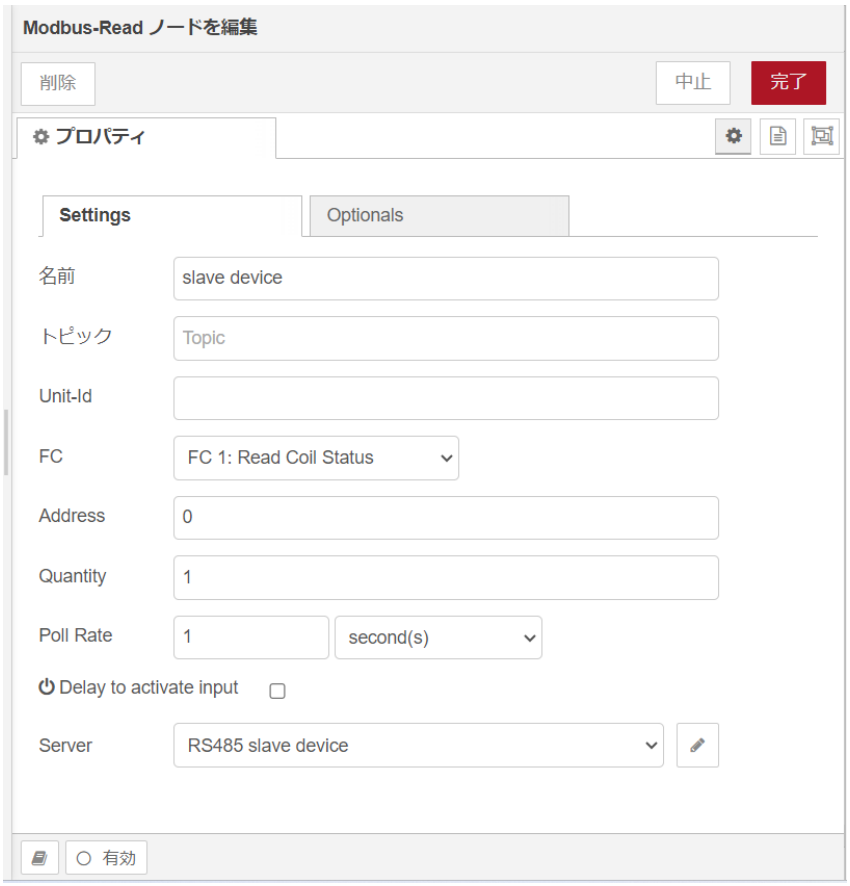


図 4.12 [Modbus-Read] ノードのプロパティ内容

5. [Modbus-Response] ノードをドラッグ、ドロップします。

6. [slave device] ノードの右側にある端子をクリックし、[Modbus-Response] ノードの左側の端子を選択して放します。

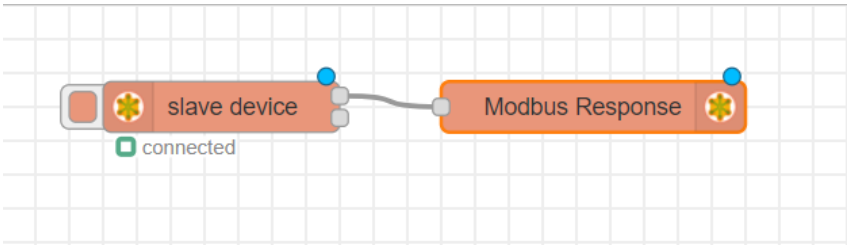


図 4.13 RS-485 を使用した Modubus RTU 読み出し用フロー

7. 画面右上の [デプロイ] を押します。読み出しに成功すると 1 秒毎に読みだした値が更新され [Modbus-Response] ノードの下に表示されます。

4.4.4. CPU の測定温度のグラフをダッシュボードに表示する

「4.4.2. CPU の測定温度を取得する」 で取得した CPU 温度をダッシュボードにグラフとして表示するフローを作成します。

1. サイドバーの右端にある三角形のボタンを押し、[ダッシュボード] を選択します。

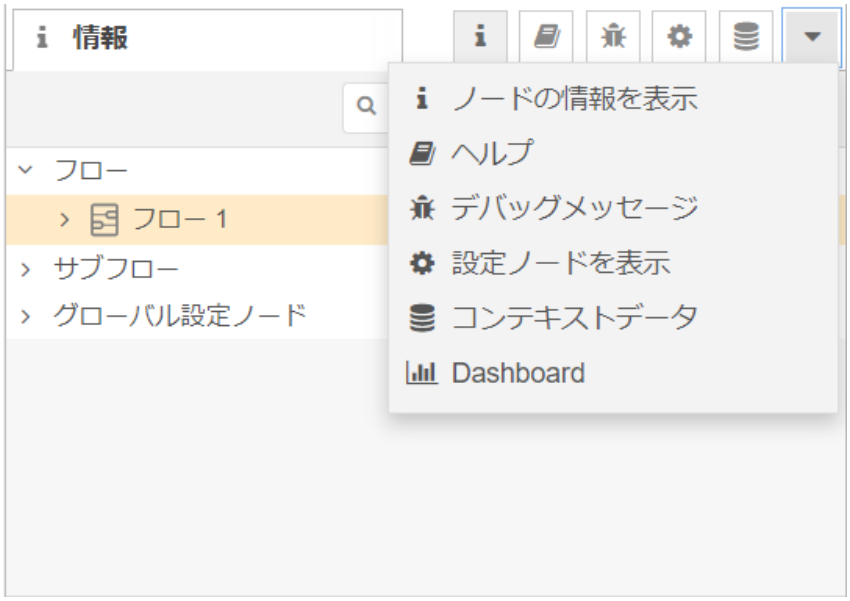


図 4.14 [ダッシュボード]を選択する

2. [ダッシュボード] 編集画面が表示されることを確認します。

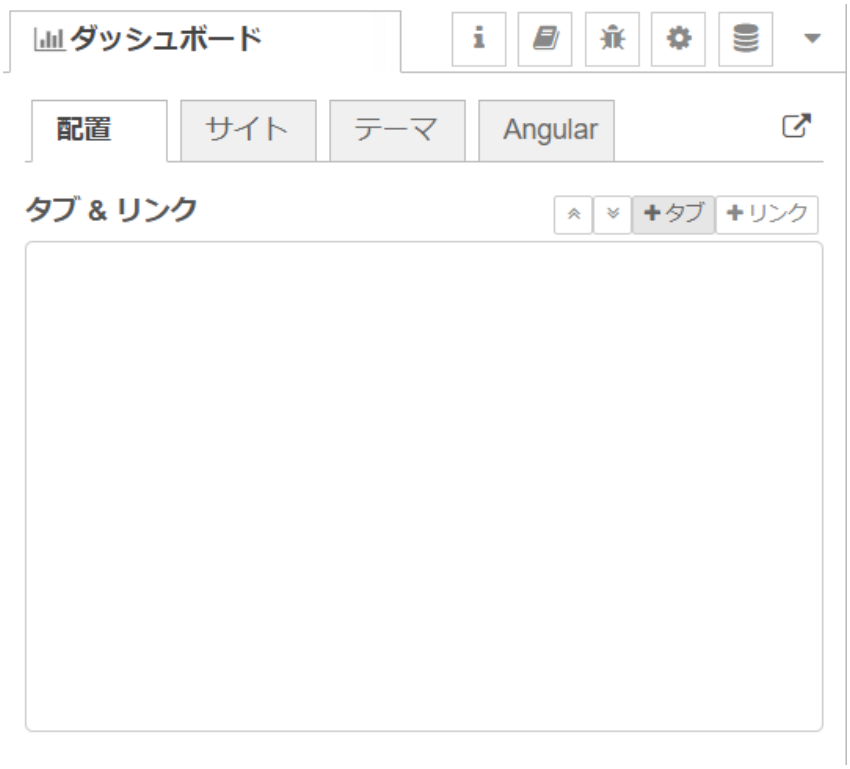


図 4.15 ダッシュボード編集画面

3. [+タブ] ボタンを押して [Tab1] を追加します。

図 4.16 ダッシュボードに [TAb1] を追加

4. [Tab1] プロパティを編集します。

```
名前: Armadillo
アイコン: dashboard
```

図 4.17 [Tab1] プロパティ内容

5. [Armadillo] タブが追加されました。

```
名前: Armadillo
アイコン: dashboard
```

図 4.18 ダッシュボード編集画面に [Armadillo] タブが追加

6. [Armadillo] 横の [+グループ] ボタンを押して [Group1] を追加します。

```
名前: Armadillo
アイコン: dashboard
```

図 4.19 ダッシュボード編集画面に [Group1] グループが追加

7. [Group1] プロパティを編集します。

```
名前: Group 1
タブ: Armadillo
種類:
幅: 15
□グループ名を表示する
```

図 4.20 [Group1] プロパティ内容

8. [chart] ノードをワークスペースにドラッグ、ドロップします。ダッシュボード編集画面の [Group1] グループに追加されたことを確認します。

図 4.21 ダッシュボード編集画面に [chart] ノードが追加

9. [chart] ノードのプロパティを編集します。

```
グループ: [Armadillo] Group 1
サイズ: 自動
ラベル: CPU temp
種類: 折れ線グラフ
X 軸: 直近 1 時間
X 軸ラベル: HH:mm:ss
Y 軸: 最小 20 最大 50
凡例: 非表示 補完: 直線
```

図 4.22 [inject] ノードのプロパティ内容

10. [CPU temp calc] ノードの右側にある端子をクリックし、[CPU temp] ノードの左側の端子を選択して放します。

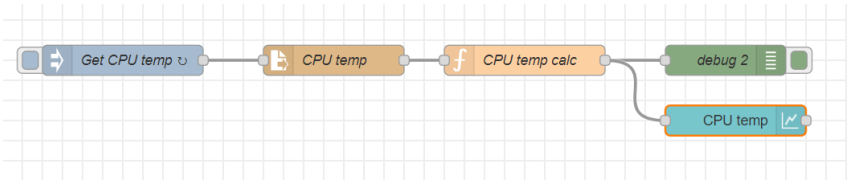


図 4.23 CPU の測定温度のグラフをダッシュボードに表示するフロー

11. 画面右上の [デプロイ] を押します。ダッシュボード編集画面の [テーマ] タブの右側にある四角に矢印が重なったボタンを選択すると、ダッシュボードが表示されます。

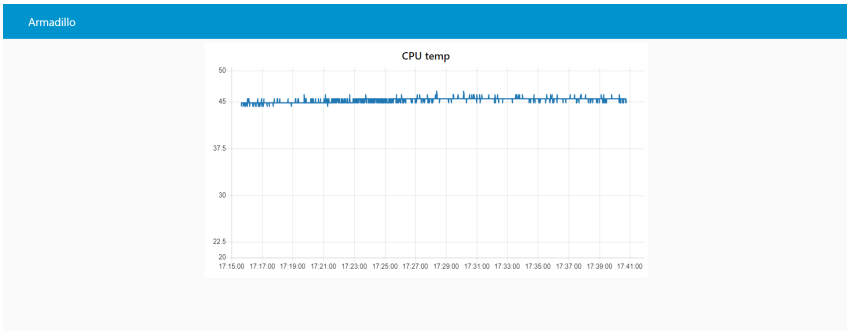


図 4.24 CPU の測定温度のグラフのダッシュボード

12. ダッシュボードの CPU 温度のグラフは一秒毎に更新されます。

4.4.5. 外部プログラムを実行する

ここでは、外部プログラムを実行しその結果を取得するフローを作成します。外部プログラムは [exec] ノードで実行することができます。ここでの外部プログラムとはシステムコマンドやユーザー自身が作成したプログラムのことを指します。

例として `date` コマンドを実行するフローを作成します。

1. パレットから [inject] ノードをワークスペースにドラッグ、ドロップします。プロパティはデフォルトから変更ありません。
2. [exec] ノードをドラッグ、ドロップします。
3. [exec] ノードのプロパティを編集します。

```
コマンド: date
引数: なし
```

図 4.25 [exec] ノードのプロパティ内容

4. [indect] ノードの右側にある端子をクリックし、[exec] ノードの左側の端子を選択して放します。
5. [debug] ノードをドラッグ、ドロップします。
6. [exec] ノードの右側の一番上にある端子をクリックし、[debug] ノードの左側の端子を選択して放します。

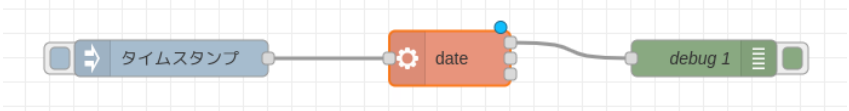


図 4.26 外部プログラムを実行するフロー

7. 画面右上の [デプロイ] を押します。

8. [inject] ノードの左の四角を押すと [exec] ノードに設定した date コマンドが実行され標準出力の結果がデバッグメッセージに表示されます。

[exec] ノードの右の端子は上からそれぞれ「標準出力」「標準エラー出力」「返却コード」となっており、取得したい出力によって使い分けることができます。

4.4.6. Node-RED を終了する

ここでは、 Node-RED を任意のタイミングで終了するフローを作成します。Node-RED のコアノードのほかに exit ノードを使用します。

Node-RED 終了後、Node-RED を再起動するためには Armadillo-X2 の電源を再投入する必要があります。

1. パレットから [inject] ノードをワークスペースにドラッグ、ドロップします
2. [inject] ノードのプロパティを編集します。

```
名前: Finish Node-RED
繰り返し: なし
```

図 4.27 [inject] ノードのプロパティ内容

3. [exit] ノードをドラッグ、ドロップします。
4. [exit] ノードのプロパティを編集します。

```
Name: exit
Exit code: 0
```

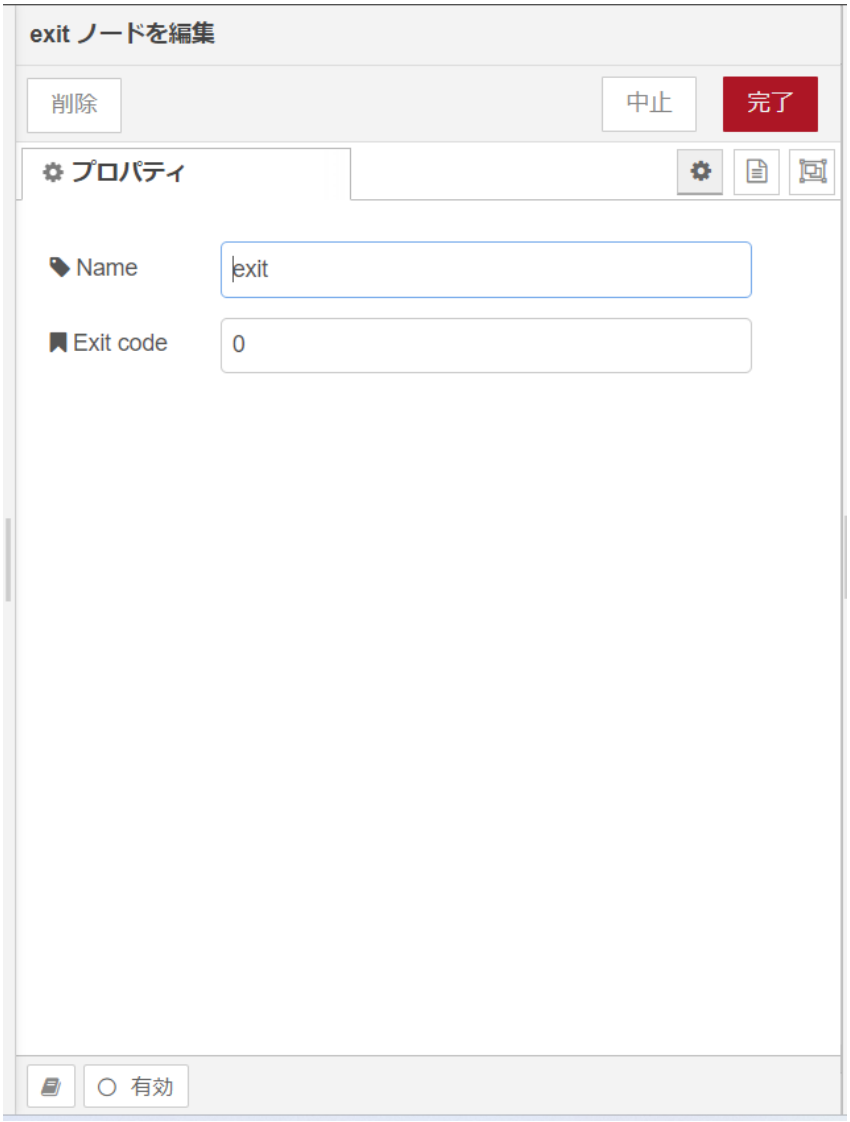


図 4.28 [exit] ノードのプロパティ内容

5. [Finish Node-RED] ノードの右側にある端子をクリックし、[exit] ノードの左側の端子を選択して放します。

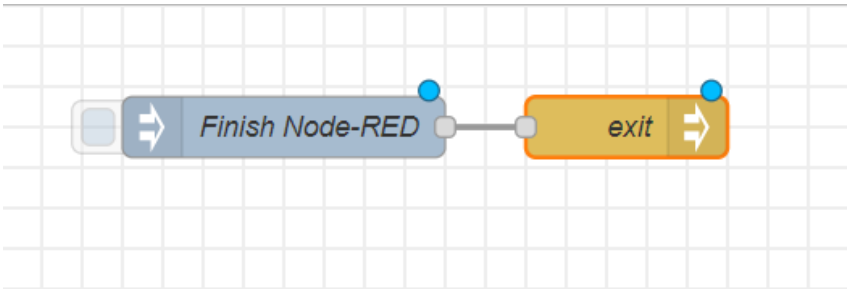


図 4.29 Node-RED を任意のタイミングで終了するフロー

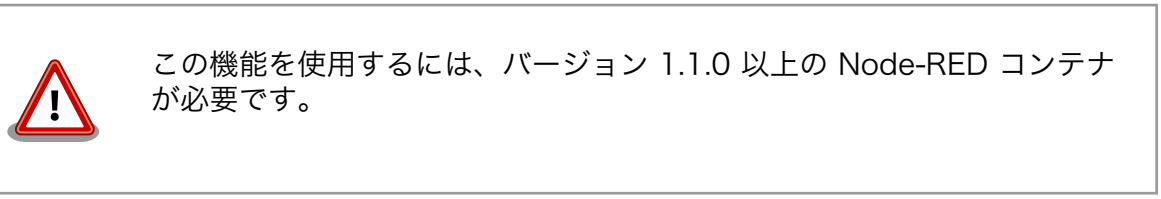
6. 画面右上の [デプロイ] を押します。[Finish Node-RED] ノードの左側にあるボタンを押すと Node-RED が終了します。

4.5. ユーザデータを削除する

Node-RED に関するユーザデータは /var/app/rollback/volumes/node-red/root に保存されます。当ディレクトリを削除した場合はユーザデータは全て削除されます。この場合 Node-RED のブラウザからインストールしたカスタムノードは削除されます。

4.6. AWS へデバイス情報を送信するフローを作成する

Armadillo-X2 から AWS サービスへデバイス情報を送信するための一連のフローについて説明します。デバイス状態の送信には AWS IoT Device Shadow サービスを利用します。



手順を以下に示します。

最初に AWS 上で作業が必要です。

- AWS アカウントの作成
- IAM ユーザーの作成
- デバイスデータエンドポイントの取得

以下は Node-RED 上で操作します。

- デバイス証明書を取得するフローの作成
- デバイスを登録するフローの作成
- AWS IoT ポリシーを作成するフローの作成
- デバイス証明書を登録するフローの作成
- ポリシーをアタッチするフローの作成

ここまでの手順はデバイス登録のために一度のみ実行する必要があります。もし、デバイスを誤って削除してしまった場合や、登録した証明書の期限が切れた場合などがあれば、再度実行する必要があります。

AWS IoT Device Shadow サービスを利用して、デバイスシャドウを制御します。本ドキュメントでは、暗号化したデバイスシャドウを用いることで、デバイス状態についてよりセキュアに通信します。デバイスシャドウを制御するフローについては、以下になります。

- デバイスシャドウを取得するフローの作成
- デバイスシャドウを更新するフローの作成
- デバイスシャドウを削除するフローの作成

4.6.1. AWS アカウントの作成

AWS アカウントの作成方法については、AWS 公式サイトの AWS アカウント作成の流れ [https://aws.amazon.com/jp/register-flow/] を参照してください。

4.6.2. IAM ユーザーの作成

AWS IAM (Identity and Access Management) は、AWS リソースへのアクセスを安全に管理するためのウェブサービスです。IAM により、誰を認証(サインイン)し、誰にリソースの使用を承認する(アクセス許可を持たせる)かを管理することができます。

本章では以下の手順について説明します。

- IAM ユーザーの作成
- IAM ユーザーにポリシーの設定
- IAM ユーザーのアクセスキーの作成

4.6.2.1. IAM ユーザーを作成する

IAM ユーザの作成については 公式のドキュメント [https://docs.aws.amazon.com/ja_jp/IAM/latest/UserGuide/id_users_create.html] を参照してください。

4.6.2.2. IAM ユーザーにポリシーを設定する

作成した AWS IAM ユーザには AWS IoT Device Shadow サービスを利用するために必要な許可を与える必要があります。必要なポリシーについては 公式のドキュメント [https://docs.aws.amazon.com/ja_jp/IAM/latest/UserGuide/access_policies_examples.html] を参照してください。以下はポリシーの例です。ポリシー名については任意の名前を与えてください。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateRole",
                "iam:Get*",
                "iam:PutRolePolicy",
                "iam:DeleteRolePolicy",
                "iam:DeletePolicy",
                "iam:DeleteRole",
                "iam:AttachRolePolicy",
                "iam:List*",
                "iam:Pass*",
                "iot:Connect",
                "iot:Publish",
                "iot:Subscribe",
                "iot:Receive",
                "iot:AcceptCertificateTransfer",
                "iot:AddThingToThingGroup",
                "iot:AssociateTargetsWithJob",
                "iot:Attach*",
                "iot:Cancel*",
                "iot:ClearDefaultAuthorizer",
```

```
                "iot:Create*",
                "iot:Delete*",
                "iot:DeprecateThingType",
                "iot:Describe*",
                "iot:Detach*",
                "iot:DisableTopicRule",
                "iot:EnableTopicRule",
                "iot:Get*",
                "iot:List*",
                "iot:Register*",
                "iot:RejectCertificateTransfer",
                "iot:RemoveThingFromThingGroup",
                "iot:ReplaceTopicRule",
                "iot:SearchIndex",
                "iot:Set*",
                "iot:StartThingRegistrationTask",
                "iot:StopThingRegistrationTask",
                "iot:TransferCertificate",
                "iot:Update*",
                "autoscaling:Describe*",
                "cloudwatch:*",
                "logs:*",
                "s3:PutObject",
                "s3:ListBucket",
                "s3:GetObject",
                "s3:CreateBucket",
                "cloudformation:*"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/events.amazonaws.com/
AWSServiceRoleForCloudWatchEvents*",
            "Condition": {
                "StringLike": {
                    "iam:AWSServiceName": "events.amazonaws.com"
                }
            }
        }
    ]
}
```

図 4.30 ポリシーの例

4.6.2.3. IAM ユーザーのアクセスキーを作成する

作成した AWS IAM ユーザのアクセスキーを作成します。AWS IAM ユーザーのアクセスキーの管理については公式のドキュメント [https://docs.aws.amazon.com/ja_jp/IAM/latest/UserGuide/id_credentials_access-keys.html] を参照してください。作成したアクセスキーとシークレットキーについては、AWS IoT Device Shadow サービスの利用のため Node-RED のノードに設定する必要があるため、無くさないでください。以上で AWS 上で操作が必要な手順については終了です。

4.6.3. デバイスデータエンドポイントを取得する

デバイスシャドウの取得・更新・削除にはデバイスデータエンドポイントを使用します。デバイスデータエンドポイントの取得方法は 公式のドキュメント [https://docs.aws.amazon.com/ja_jp/iot/latest/developerguide/iot-connect-devices.html] を参照してください。

4.6.4. デバイス証明書を取得するフローの作成

Armadillo Base OS 搭載機器は、標準でセキュアエレメントを搭載しており、対応した暗号化方式の認証鍵や証明書を安全に保存・利用することが可能です。EdgeLock SE050 は IoT アプリケーション向けのセキュアエレメントです。 フラッシュメモリを内蔵しており、保存された秘密鍵を外部に露出することなく暗号処理に利用できます。

セキュアエレメントは I2C デバイスとして認識されます。製品によって I2C バスが異なるためご注意ください。Armadillo-X2 で使用する I2C バスを 「表 4.2. セキュアエレメントが使用する I2C バス」に示します。

表 4.2 セキュアエレメントが使用する I2C バス

I2C バス	I2C デバイス	
	アドレス	デバイス名
2(I2C3)	0x48	SE050(セキュアエレメント)

このフローでは秘密鍵とデバイス証明書を EdgeLock SE050 より取得して、/var/app/rollback/volumes/node-red/cert に保存します。

1. パレットから [inject] ノードをワークスペースにドラッグ、ドロップします。プロパティはデフォルトから変更ありません。
2. [exec] ノードをドラッグ、ドロップします。
3. [exec] ノードのプロパティを編集します。

```
コマンド: se05x_getkey 0xF0000111 /cert/device_cert.pem /dev/i2c-2:0x48
引数: チェックなし
出力: コマンド終了時 - exec モード
名前: デバイス証明書の取得
```

図 4.31 [デバイス証明書の取得] ノードのプロパティ内容

4. [inject] ノードの右側にある端子をクリックし、[デバイス証明書の取得] ノードの左側の端子を選択して放します。

5. もう一つ [exec] ノードをドラッグ、ドロップします。

6. [exec] ノードのプロパティを編集します。

```
コマンド: ls /cert
引数: チェックなし
出力: コマンド終了時 - exec モード
名前: ファイルの確認
```

図 4.32 [ファイルの確認] ノードのプロパティ内容

7. [デバイス証明書の取得] ノードの右側にある端子をクリックし、 [ファイルの確認] ノードの左側の端子を選択して放します。

8. もう一つ、[inject] ノードをワークスペースにドラッグ、ドロップします。プロパティはデフォルトから変更ありません。

9. [exec] ノードをドラッグ、ドロップします。

10. [exec] ノードのプロパティを編集します。

```
コマンド: se05x_getkey 0xF0000110 /cert/key.pem /dev/i2c-2:0x48
引数: チェックなし
出力: コマンド終了時 - exec モード
名前: リファレンスキーの取得
```

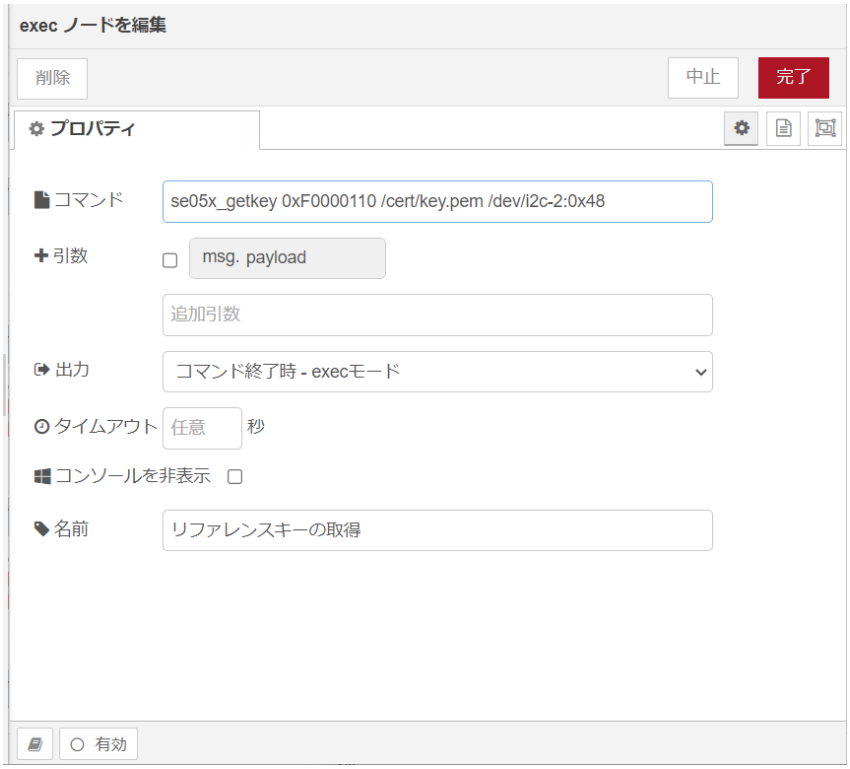


図 4.33 [リファレンスキーの取得] ノードのプロパティ内容

11. 二つ目の [inject] ノードの右側にある端子をクリックし、[リファレンスキーの取得] ノードの左側の端子を選択して放します。

12. [リファレンスキーの取得] ノードの右側にある端子をクリックし、[ファイルの確認] ノードの左側の端子を選択して放します。

13. [debug] ノードをドラッグ、ドロップします。

14. [ファイルの確認] ノードの右側の一番上にある端子をクリックし、[debug] ノードの左側の端子を選択して放します。

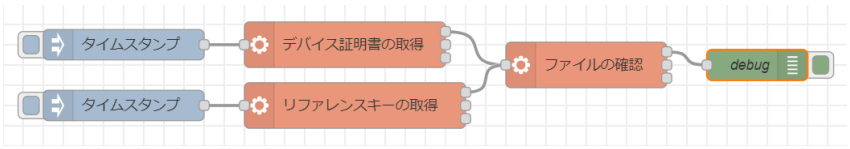


図 4.34 デバイス証明書を取得するフロー

15. 画面右上の [デプロイ] を押します。

16. [inject] ノードの左の四角をそれぞれ押すと、コマンドの実行結果結果がデバッグメッセージに表示されます。下記ファイルが出力されていれば取得に成功しています。

```
AmazonRootCA1.pem
device_cert.pem
key.pem
```

図 4.35 ファイル一式

実行に成功した場合は /var/app/rollback/volumes/node-red/cert/ にデバイス証明書が保存されています。

4.6.5. デバイスを登録するフローの作成

AWS IoT Device Shadow サービスを利用するため、AWS IoT Core にデバイス（モノ）を登録する必要があります。今回モノの名前はデバイスのシリアルナンバー（例: 001234567890）を使用します。

1. パレットから [inject] ノードをワークスペースにドラッグ、ドロップします。プロパティはデフォルトから変更ありません。
2. [credentials] ノードをドラッグ、ドロップします。
3. [credentials] ノードのプロパティを編集します。[追加] ボタンを押して値を追加します。private のステータスが [文字列] のままだと内容が表示されたままのため、確認後は [hidden] に変更するのを推奨します。

```
Name: 接続情報の登録
Values:
private: [hidden] ❶
to: [msg.] aws_access_key
private: [hidden] ❷
to: [msg.] aws_secret_key
private: [文字列] ❸
to: [msg.] aws_iot_region
private: [hidden] ❹
to: [msg.] aws_iot_host
```

❶ 「4.6.2. IAM ユーザーの作成」で取得したアクセスキー

❷ 「4.6.2. IAM ユーザーの作成」で取得したシークレットキー

❸ リージョン(例: ap-northeast-1)

❹ IoT Core REST API エンドポイント(例: iot.ap-northeast-1.amazonaws.com)

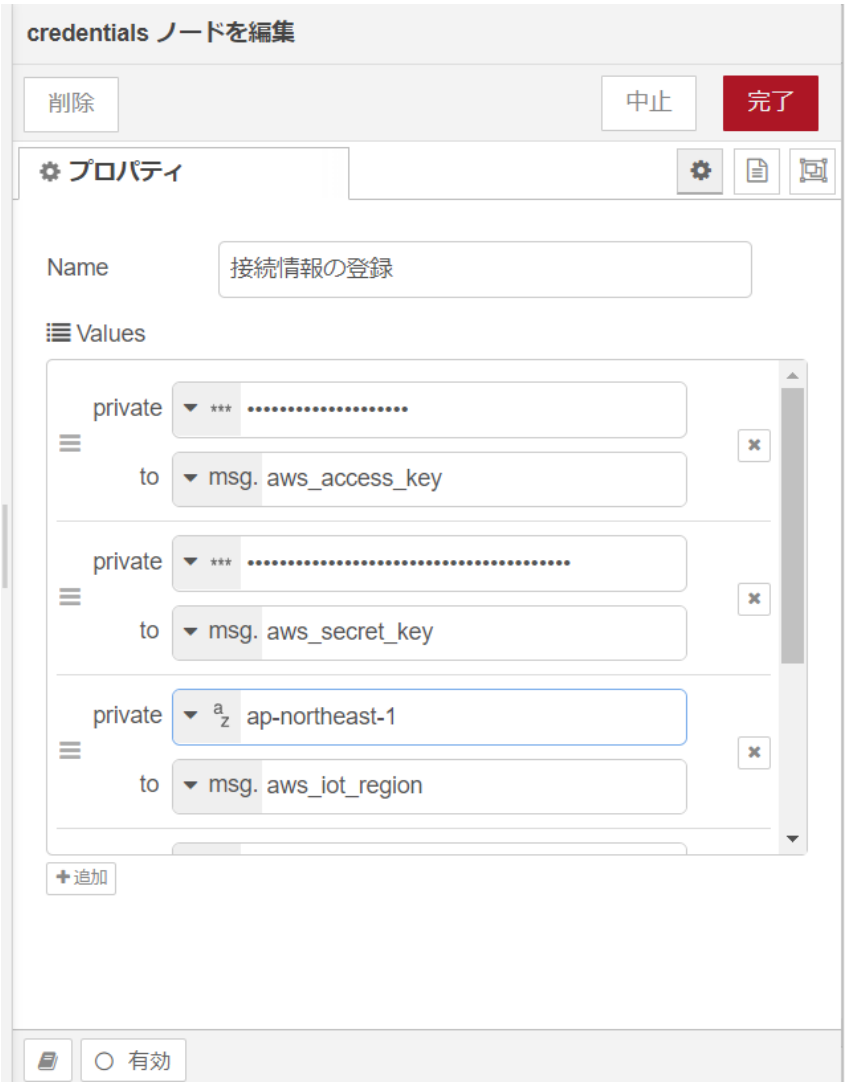


図 4.36 [接続情報の登録] ノードのプロパティ内容

4. [inject] ノードの右側にある端子をクリックし、[接続情報の登録] ノードの左側の端子を選択して放します。

5. [function] ノードをドラッグ、ドロップします。

6. [function] ノードのプロパティを編集します。

```
名前: デバイス登録の引数設定
コード:
msg.payload = msg.aws_access_key + ":" + msg.aws_secret_key
    + " " + msg.aws_iot_region
    + " " + msg.aws_iot_host;
return msg;
```

図 4.37 [exec queue] ノードのプロパティ内容

7. [接続情報の登録] ノードの右側にある端子をクリックし、[デバイス登録の引数設定] ノードの左側の端子を選択して放します。

8. [exec queue] ノードをドラッグ、ドロップします。

9. [exec queue] ノードのプロパティを編集します。

```
名前: デバイス登録
コマンド: bash $file
Queue: 1
Debug Mode: ✓
Append msg.payload: ✓
構文: Shell
内容:
AWS_ACCESS="$1"
AWS_REGION="$2"
ENDPOINT="$3"
URI=/things/${AT_SERIAL_NUMBER}

RESULT=$(curl ¥
        --cacert /cert/AmazonRootCA1.pem ¥
        --user "${AWS_ACCESS}" ¥
        --aws-sigv4 "aws:amz:${AWS_REGION}:execute-api" ¥
        --request POST -v ¥
        -w '¥n%{http_code}' ¥
        -d "{}" ¥
        "https://${ENDPOINT}${URI}")

RESULT_BODY=$(echo "${RESULT}" | sed "$ d")
RESULT_STATUS=$(echo "${RESULT}" | tail -n 1)

if [ "${RESULT_STATUS}" = 200 ]; then
        echo "${RESULT_BODY}"
else
```

```
        echo '{"statusCode": '${RESULT_STATUS}',"responseBody": '${RESULT_BODY}'}'
fi
```

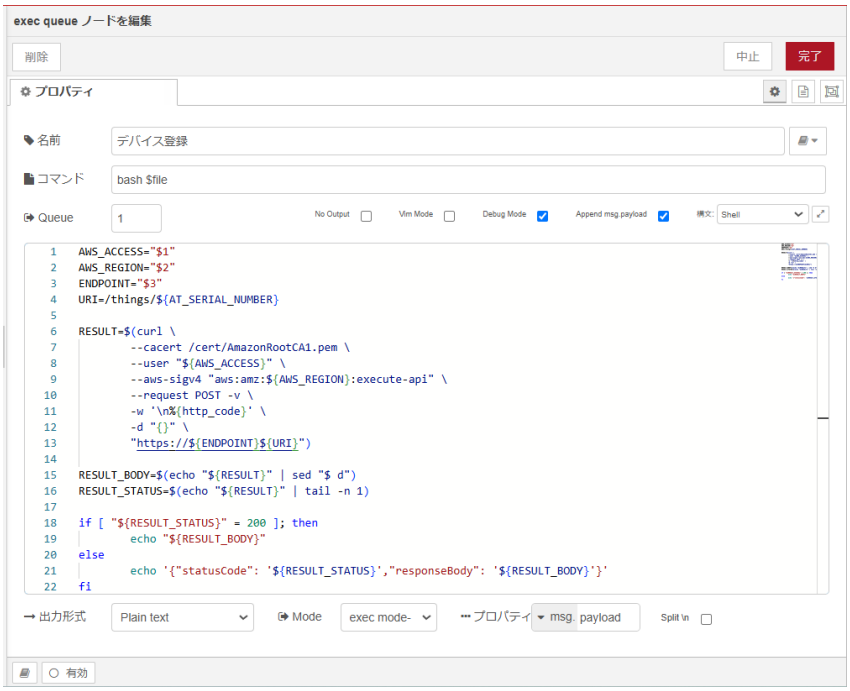


図 4.38 [exec queue] ノードのプロパティ内容

10. [デバイス登録の引数設定] ノードの右側にある端子をクリックし、[デバイス登録] ノードの左側の端子を選択して放します。

11. [function] ノードをドラッグ、ドロップします。

12. [function] ノードのプロパティを編集します。

```
名前: デバイス名の取得
コード:
var str = msg.payload;
var json = str.substr(str.indexOf("{"),str.lastIndexOf("}") + 1);
try {
  var res = JSON.parse(json);
  msg.payload = "thingName:" + res.thingName;
} catch {
  msg.payload = json;
}
return msg;
```

図 4.39 [デバイス名の取得] ノードのプロパティ内容

13. [debug] ノードをドラッグ、ドロップします。
14. [デバイス名の取得] ノードの右側の端子をクリックし、[debug] ノードの左側の端子を選択して放します。

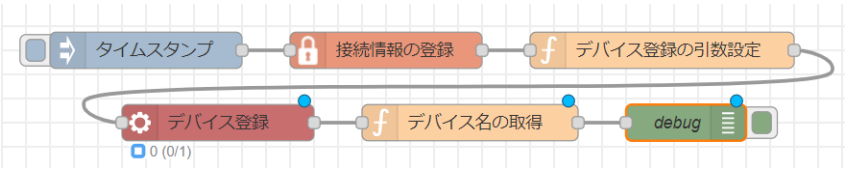


図 4.40 デバイスを登録するフロー

15. 画面右上の [デプロイ] を押します。
16. [inject] ノードの左の四角を押すと登録したモノの名前がデバッグメッセージに表示されます。

```
”thingName:001234567890”
```

図 4.41 登録したモノの名前

表示されたモノの名前がデバイスシャドウの対象になります。もし、デバイスの登録に失敗した場合はエラーに応じたステータスコードを表示します。

4.6.6. AWS IoT ポリシーを作成するフローの作成

登録したモノに対して AWS IoT Device Shadow サービスを利用するための許可を与える必要があります。そのため、必要な AWS IoT ポリシーを作成してアタッチする必要があります。まず、ポリシー

を新しく作成します。「4.6.5. デバイスを登録するフローの作成」 で作成した [接続情報の登録] ノードを使用します。

1. [接続情報の登録] ノードのプロパティに [ポリシー名] を追加します。

```
Name: 接続情報の登録
Values:
private: [hidden] ❶
to: [msg.] aws_access_key
private: hidden: ❷
to: [msg.] aws_secret_key
private: [文字列] ❸
to: [msg.] aws_iot_region
private: [hidden] ❹
to: [msg.] aws_iot_host
private: [hidden] ❺
to: [msg.] aws_policy_name
```

❶ 「4.6.2. IAM ユーザーの作成」で取得したアクセスキー

❷ 「4.6.2. IAM ユーザーの作成」で取得したシークレットキー

❸ リージョン(例: ap-northeast-1

❹ IoT Core REST API エンドポイント(例: iot.ap-northeast-1.amazonaws.com)

❺ ポリシー名(例: node-red-thing-policy)

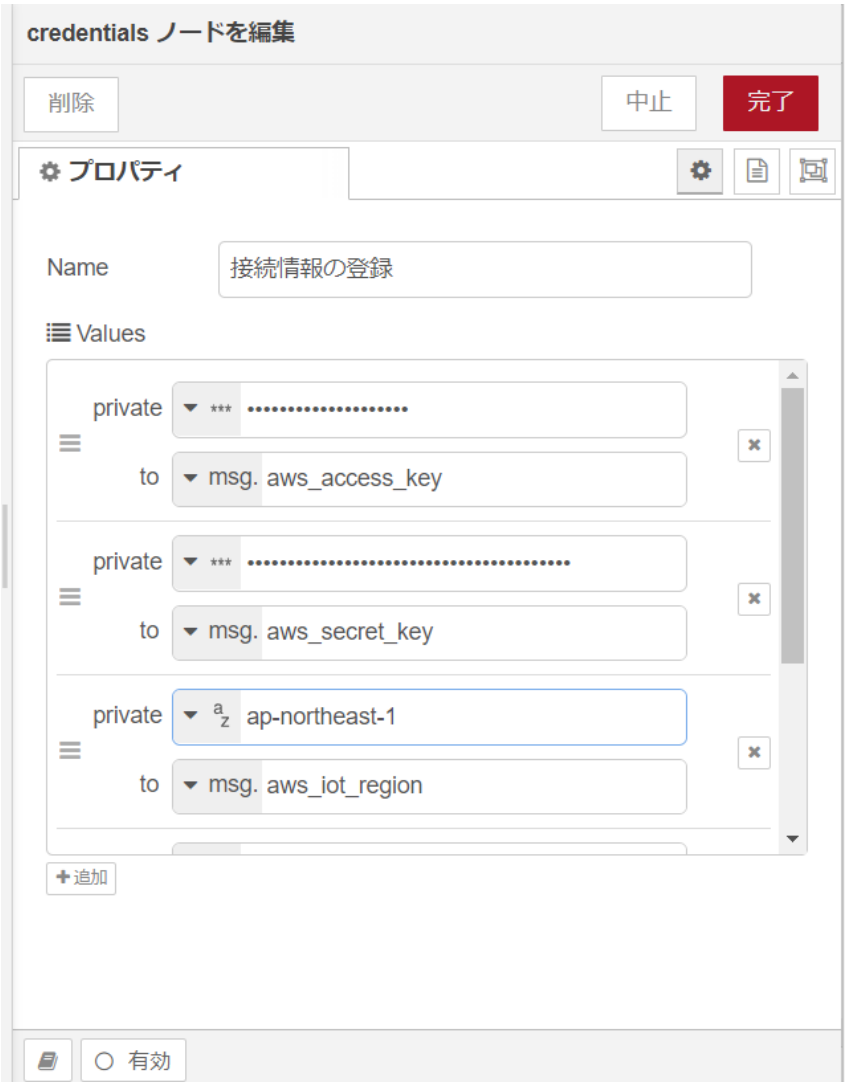


図 4.42 [接続情報の登録] ノードのプロパティ内容

2. [function] ノードをドラッグ、ドロップします。

3. [function] ノードのプロパティを編集します。

```
名前: ポリシー登録の引数設定
コード:
msg.payload = msg.aws_access_key + ":" + msg.aws_secret_key
    + " " + msg.aws_iot_region
    + " " + msg.aws_iot_host
    + " " + msg.aws_policy_name;
return msg;
```

図 4.43 [ポリシー登録の引数設定] ノードのプロパティ内容

4. [接続情報の登録] ノードの右側にある端子をクリックし、[ポリシー登録の引数設定] ノードの左側の端子を選択して放します。

5. [exec queue] ノードをドラッグ、ドロップします。

6. [exec queue] ノードのプロパティを編集します。

```
名前: 新しいポリシーの作成
コマンド: bash $file
Queue: 1
Debug Mode: ✓
Append msg.payload: ✓
構文: Shell
内容:
AWS_ACCESS="$1"
AWS_REGION="$2"
ENDPOINT="$3"
POLICY_NAME="$4"
URI=/policies/${POLICY_NAME}

RESULT=$(curl ¥
        -H "Content-type: application/json" ¥
        --cacert /cert/AmazonRootCA1.pem ¥
        --user "${AWS_ACCESS}" ¥
        --aws-sigv4 "aws:amz:${AWS_REGION}:execute-api" ¥
        -d '{"policyDocument": "{¥"Version¥": ¥"2012-10-17¥",¥"Statement¥": [{¥"Effect¥":
¥"Allow¥",¥"Action¥": ¥"iot:*¥",¥"Resource¥": ¥"*¥"}]}"}' ¥
        --request POST -v ¥
        -w '¥n%{http_code}' ¥
        "https://${ENDPOINT}${URI}")

RESULT_BODY=$(echo "${RESULT}" | sed "$ d")
RESULT_STATUS=$(echo "${RESULT}" | tail -n 1)
```

```
if [ "${RESULT_STATUS}" = 200 ]; then
        echo "${RESULT_BODY}"
else
        echo '{"statusCode": '${RESULT_STATUS}',"responseBody": '${RESULT_BODY}'}'
fi
```

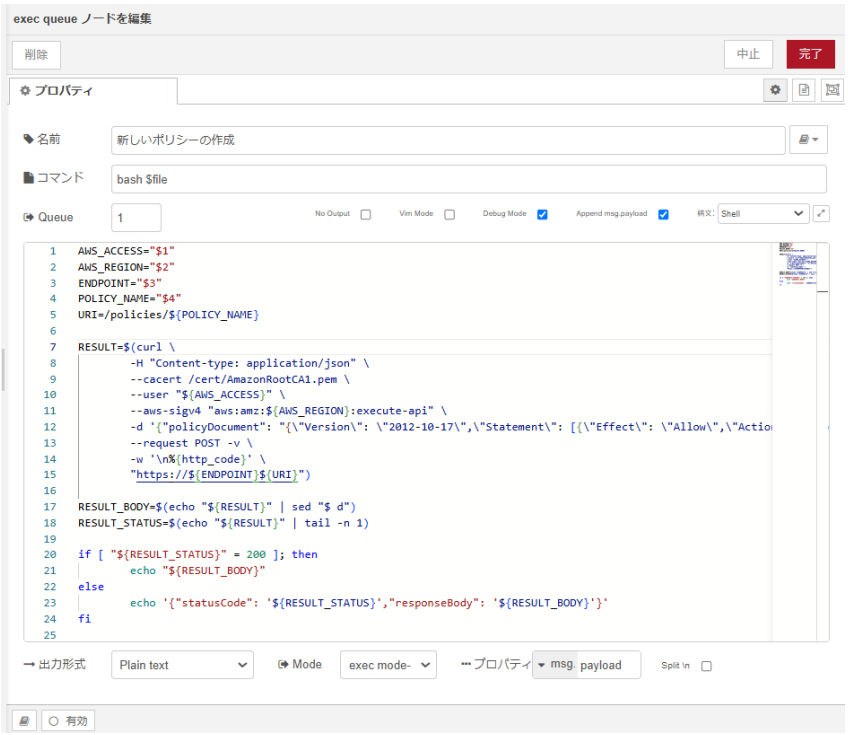


図 4.44 [新しいポリシーの作成] ノードのプロパティ内容

7. [ポリシー登録の引数設定] ノードの右側にある端子をクリックし、[新しいポリシーの作成] ノードの左側の端子を選択して放します。

8. [function] ノードをドラッグ、ドロップします。

9. [function] ノードのプロパティを編集します。

```
名前: ポリシー名の取得
コード:
var str = msg.payload;
var json = str.substr(str.indexOf("{"), str.lastIndexOf("}") + 1);
try {
    var res = JSON.parse(json);
    msg.payload = "policyName:" + res.policyName;
} catch {
    msg.payload = json;
}
return msg;
```

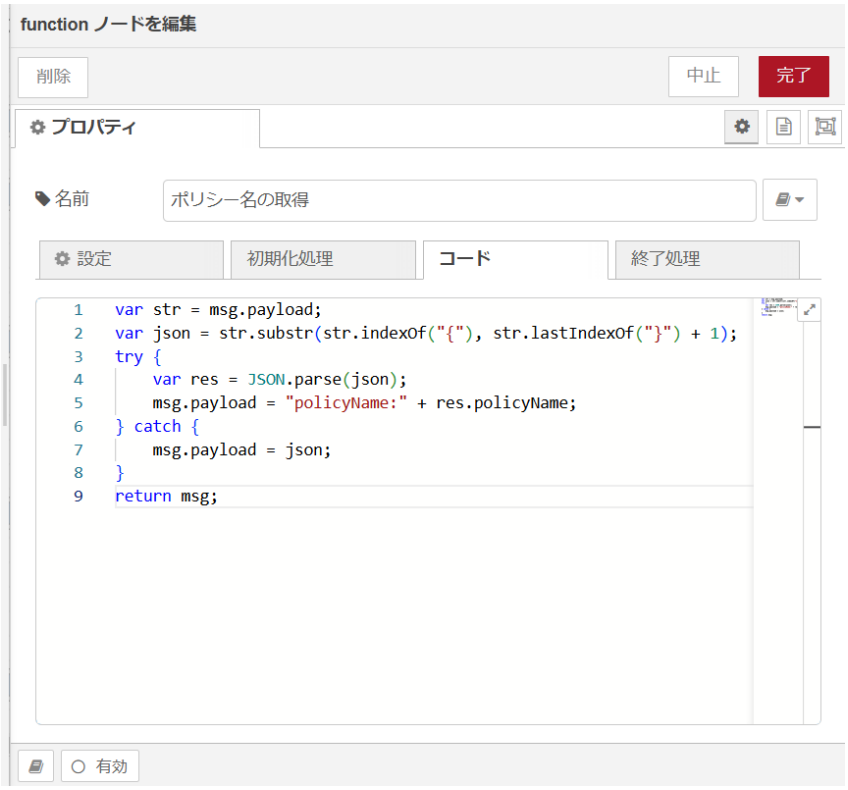


図 4.45 [ポリシー名の取得] ノードのプロパティ内容

10. [新しいポリシーの作成] ノードの右側にある端子をクリックし、[ポリシー名の取得] ノードの左側の端子を選択して放します。

11. [debug] ノードをドラッグ、ドロップします。

12. [ポリシー名の取得] ノードの右側の端子をクリックし、[debug] ノードの左側の端子を選択して放します。

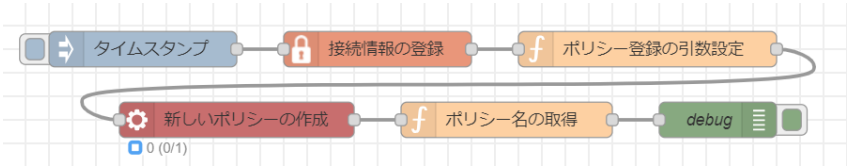


図 4.46 ポリシーを作成するフロー

13. 画面右上の [デプロイ] を押します。

14. [inject] ノードの左の四角を押すとポリシー作成の結果がデバッグメッセージに表示されます。

```
"policyName:node-red-thing-policy"
```

図 4.47 登録したポリシーの名前

すでに登録済みの場合は "Policy cannot be created - name already exists が表示されます。もし、ポリシーの作成に失敗した場合はエラーに応じたステータスコードを表示します。

4.6.7. デバイス証明書を登録するフローの作成

登録したデバイスのデバイス証明書を登録します。「4.6.5. デバイスを登録するフローの作成」 で作成した [接続情報の登録] ノードを使用します。

1. パレットから [function] ノードをワークスペースにドラッグ、ドロップします。
2. [function] ノードのプロパティを編集します。

```
名前: デバイス証明書登録の引数設定
コード:
msg.payload = msg.aws_access_key + ":" + msg.aws_secret_key
    + " " + msg.aws_iot_region
    + " " + msg.aws_iot_host;
return msg;
```

図 4.48 [デバイス証明書登録の引数設定] ノードのプロパティ内容

3. [接続情報の登録] ノードの右側にある端子をクリックし、[デバイス証明書登録の引数設定] ノードの左側の端子を選択して放します。
4. [exec queue] ノードをドラッグ、ドロップします。
5. [exec queue] ノードのプロパティを編集します。

```
名前: デバイス証明書の登録
コマンド: bash $file
Queue: 1
Debug Mode: ✓
Append msg.payload: ✓
構文: Shell
内容:
```

```
AWS_ACCESS="$1"
AWS_REGION="$2"
ENDPOINT="$3"
URI=/certificate/register-no-ca
CERT=$(cat /cert/device_cert.pem | sed -z 's/¥n/¥¥n/g')

RESULT=$(curl ¥
        -H "Content-type: application/json" ¥
        --cacert /cert/AmazonRootCA1.pem ¥
        --user "${AWS_ACCESS}" ¥
        --aws-sigv4 "aws:amz:${AWS_REGION}:execute-api" ¥
        -d "{¥"certificatePem¥":¥"${CERT}¥",¥"status¥":¥"ACTIVE¥"}" ¥
        --request POST -v ¥
        -w '¥n%{http_code}' ¥
        "https://${ENDPOINT}${URI}")

RESULT_BODY=$(echo "${RESULT}" | sed "$ d")
RESULT_STATUS=$(echo "${RESULT}" | tail -n 1)

if [ "${RESULT_STATUS}" = 200 ]; then
        echo "${RESULT_BODY}"
else
        echo '{"statusCode": '${RESULT_STATUS}',"responseBody": '${RESULT_BODY}'}'
fi
```

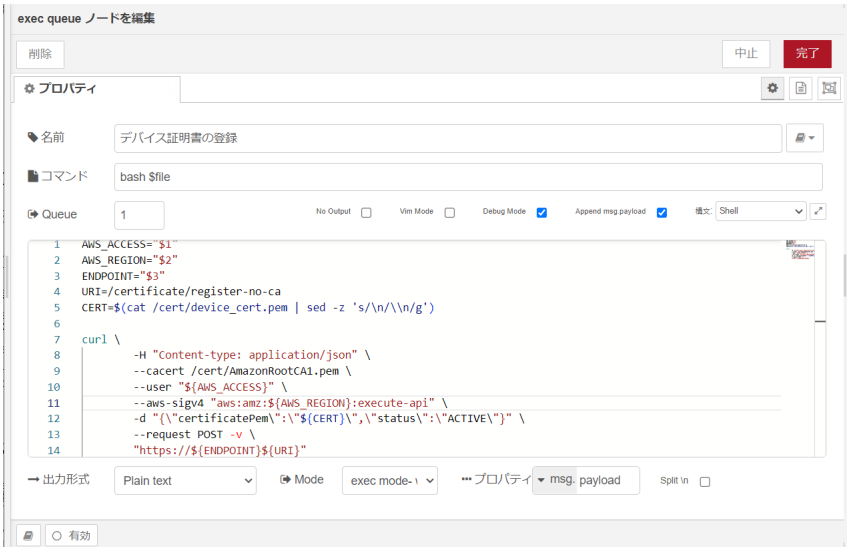


図 4.49 [デバイス証明書の登録] ノードのプロパティ内容

6. [デバイス証明書登録の引数設定] ノードの右側にある端子をクリックし、[デバイス証明書の登録] ノードの左側の端子を選択して放します。

7. [function] ノードをドラッグ、ドロップします。

8. [function] ノードのプロパティを編集します。

```
名前: 証明書 ID の取得
コード:
var str = msg.payload;
```

```
var json = str.substr(str.indexOf("{"), str.lastIndexOf("}") + 1);
try {
    var res = JSON.parse(json);
    msg.payload = "certificateArn:" + res.certificateArn;
} catch {
    msg.payload = json;
}
return msg;
```

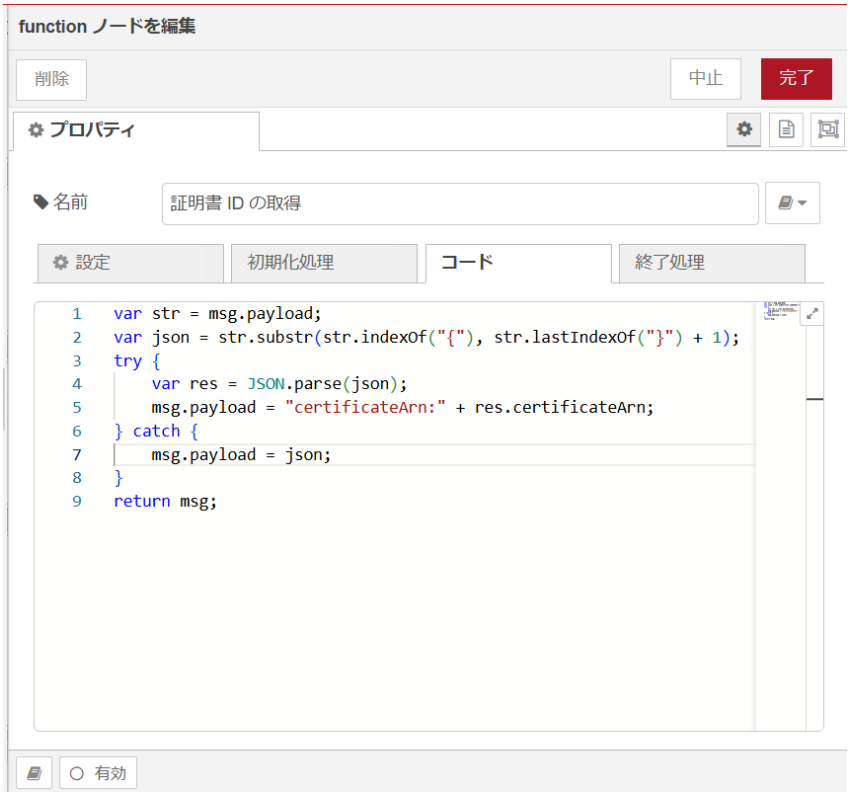


図 4.50 [証明書 ID の取得] ノードのプロパティ内容

9. [デバイス証明書の登録] ノードの右側にある端子をクリックし、[証明書 ID の取得] ノードの左側の端子を選択して放します。

10. [debug] ノードをドラッグ、ドロップします。

11. [証明書 ID の取得] ノードの右側の端子をクリックし、[debug] ノードの左側の端子を選択して放します。

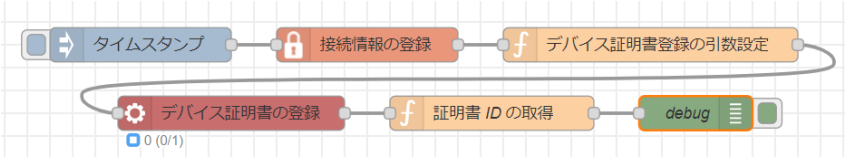


図 4.51 デバイス証明書を登録するフロー

12. 画面右上の [デプロイ] を押します。

13. [inject] ノードの左の四角を押すとデバイス証明書の登録結果がデバッグメッセージに表示されます。実際は使用しているリージョン名と、アカウント ID が含まれます。

```
"certificateArn: arn:aws:iot:ap-northeast-1:00000000:cert/xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
```

図 4.52 証明書 ID 名

すでに登録済みの場合は "The certificate is already provisioned or registered" が表示されます。もし、デバイス証明書の登録に失敗した場合はエラーに応じたステータスコードを表示します。

4.6.8. AWS IoT ポリシーをデバイスにアタッチするフローの作成

「4.6.6. AWS IoT ポリシーを作成するフローの作成」 で作成したポリシーをデバイスにアタッチします。「4.6.5. デバイスを登録するフローの作成」 で作成した [接続情報の登録] ノードを使用します。

1. [接続情報の登録] ノードのプロパティに、「4.6.7. デバイス証明書を登録するフローの作成」 で取得した [証明書 ID] を追加します。

```
Name: 接続情報の登録
Values:
private: [hidden] ❶
to: [msg.] aws_access_key
private: [hidden] ❷
to: [msg.] aws_secret_key
private: [文字列] ❸
to: [msg.] aws_iot_region
private: [hidden] ❹
to: [msg.] aws_iot_host
private: [hidden] ❺
to: [msg.] aws_cert_arn
```

❶ 「4.6.2. IAM ユーザーの作成」で取得したアクセスキー

❷ 「4.6.2. IAM ユーザーの作成」で取得したシークレットキー

❸ リージョン(例: ap-northeast-1)

❹ IoT Core REST API エンドポイント(例: iot.ap-northeast-1.amazonaws.com)

❺ 証明書 ID (例: arn:aws:iot:ap-northeast-1:00000000:cert/xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx)

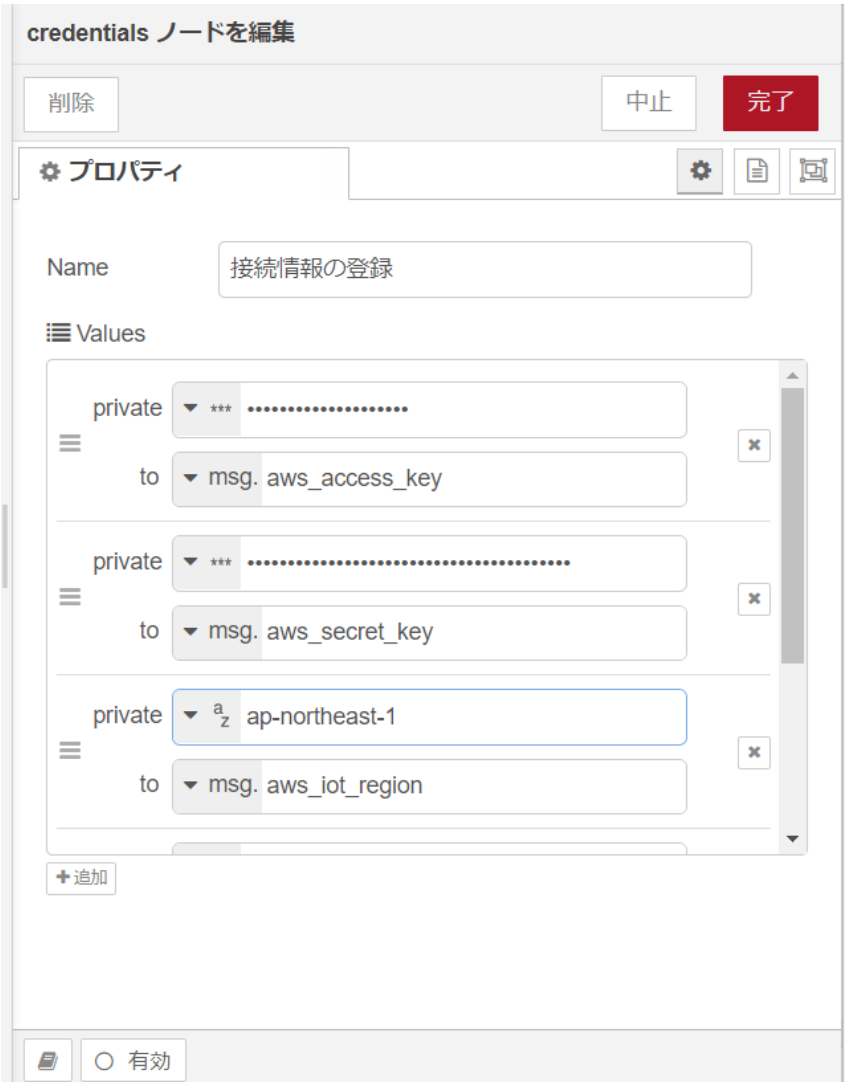


図 4.53 [接続情報の登録] ノードのプロパティ内容

2. パレットから [function] ノードをワークスペースにドラッグ、ドロップします。

3. [function] ノードのプロパティを編集します。

```
名前: ポリシーをアタッチするための引数設定
コード:
msg.payload = msg.aws_access_key + ":" + msg.aws_secret_key
    + " " + msg.aws_iot_region
    + " " + msg.aws_iot_host
    + " " + msg.aws_policy_name
    + " " + msg.aws_cert_arn;
return msg;
```

図 4.54 [ポリシーをアタッチするための引数設定] ノードのプロパティ内容

4. [接続情報の登録] ノードの右側にある端子をクリックし、[ポリシーをアタッチするための引数設定] ノードの左側の端子を選択して放します。

5. [exec queue] ノードをドラッグ、ドロップします。

6. [exec queue] ノードのプロパティを編集します。

```
名前: ポリシーを証明書にアタッチ
コマンド: bash $file
Queue: 1
Debug Mode: ✓
Append msg.payload: ✓
構文: Shell
内容:
AWS_ACCESS="$1"
AWS_REGION="$2"
ENDPOINT="$3"
POLICY_NAME="$4"
CERT_ARN="$5"
URI=/target-policies/${POLICY_NAME}

RESULT=$(curl ¥
        -H "Content-type: application/json" ¥
        --cacert /cert/AmazonRootCA1.pem ¥
        --user "${AWS_ACCESS}" ¥
        --aws-sigv4 "aws:amz:${AWS_REGION}:execute-api" ¥
        -d "{¥"target¥":¥"${CERT_ARN}¥"}" ¥
        --request PUT -v ¥
        -w '¥n%{http_code}' ¥
        "https://${ENDPOINT}${URI}")

RESULT_BODY=$(echo "${RESULT}" | sed "$ d")
RESULT_STATUS=$(echo "${RESULT}" | tail -n 1)
```

```
if [ "${RESULT_STATUS}" = 200 ]; then
        echo "${RESULT_BODY}"
else
        echo '{"statusCode": '${RESULT_STATUS}',"responseBody": '${RESULT_BODY}'}'
fi
```

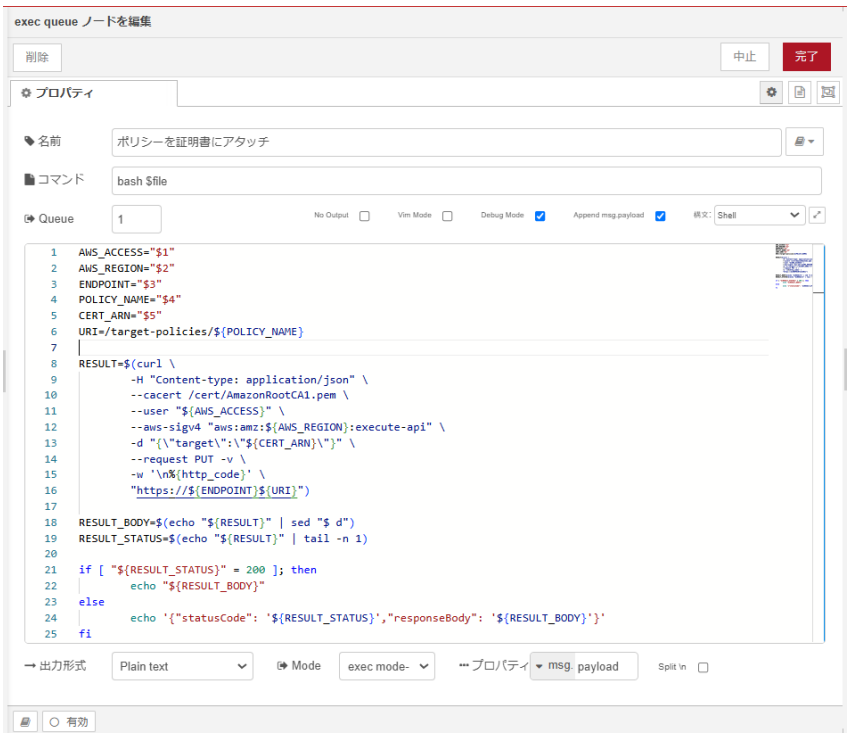


図 4.55 [ポリシーを証明書にアタッチ] ノードのプロパティ内容

7. [ポリシーをアタッチするための引数設定] ノードの右側にある端子をクリックし、[ポリシーを証明書にアタッチ] ノードの左側の端子を選択して放します。

8. [function] ノードをドラッグ、ドロップします。

9. [function] ノードのプロパティを編集します。

```
名前: ポリシーアタッチ結果の取得
コード:
var str = msg.payload;
var json = str.substr(str.indexOf("{"), str.lastIndexOf("}") + 1);
var success_msg = "{ ¥"policyAttach¥" : ¥"success¥" }";

if (json.length === 0) {
    msg.payload = success_msg;
} else {
    try {
        var res = JSON.parse(json);
        if (Object.keys(res).length === 0) {
            msg.payload = success_msg;
        }
    } catch {
        msg.payload = json;
```

```
    }
}

return msg;
```

図 4.56 [ポリシーアタッチ結果の取得] ノードのプロパティ内容

10. [ポリシーを証明書にアタッチ] ノードの右側にある端子をクリックし、[ポリシーアタッチ結果の取得] ノードの左側の端子を選択して放します。

11. [debug] ノードをドラッグ、ドロップします。

12. [ポリシーアタッチ結果の取得] ノードの右側の端子をクリックし、[debug] ノードの左側の端子を選択して放します。

13. [exec queue] ノードをドラッグ、ドロップします。

14. [exec queue] ノードのプロパティを編集します。

```
名前: 証明書をデバイスにアタッチ
コマンド: bash $file
Queue: 1
Debug Mode: ✓
Append msg.payload: ✓
構文: Shell
内容:
AWS_ACCESS="$1"
AWS_REGION="$2"
ENDPOINT="$3"
POLICY_NAME="$4"
```

```
CERT_ARN="$5"
URI=/things/${AT_SERIAL_NUMBER}/principals

RESULT=$(curl ¥
        -H "x-amzn-principal:${CERT_ARN}" ¥
        --cacert /cert/AmazonRootCA1.pem ¥
        --user "${AWS_ACCESS}" ¥
        --aws-sigv4 "aws:amz:${AWS_REGION}:execute-api" ¥
        --request PUT -v ¥
        -w '¥n%{http_code}' ¥
        "https://${ENDPOINT}${URI}")

RESULT_BODY=$(echo "${RESULT}" | sed "$ d")
RESULT_STATUS=$(echo "${RESULT}" | tail -n 1)

if [ "${RESULT_STATUS}" = 200 ]; then
        echo "${RESULT_BODY}"
else
        echo '{"statusCode": '${RESULT_STATUS}',"responseBody": '${RESULT_BODY}'}'
fi
```

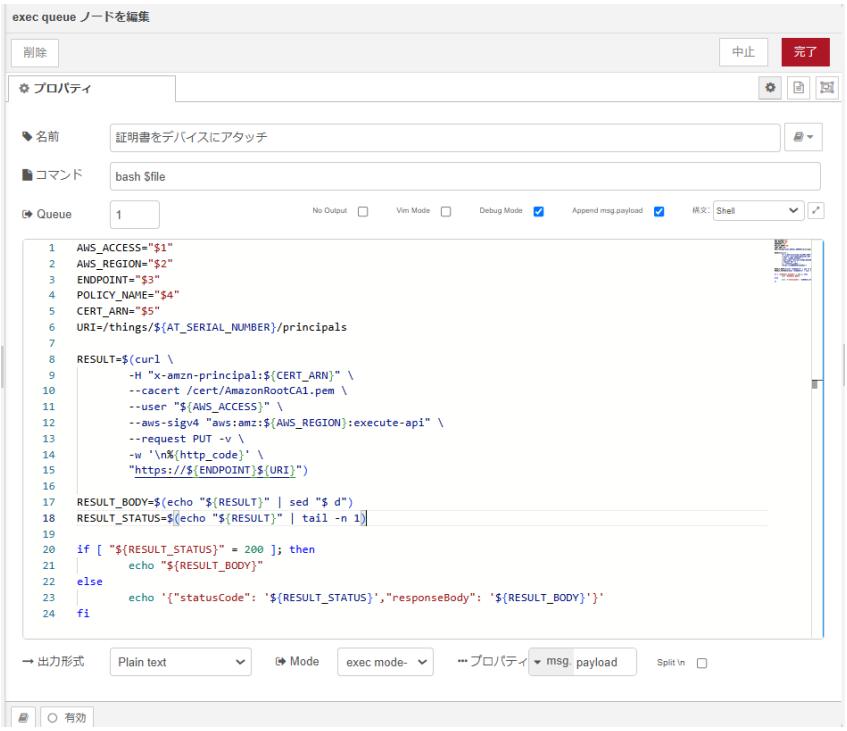


図 4.57 [証明書をデバイスにアタッチ] ノードのプロパティ内容

15. [ポリシーをアタッチするための引数設定] ノードの右側にある端子をクリックし、[証明書をデバイスにアタッチ] ノードの左側の端子を選択して放します。

16. [function] ノードをドラッグ、ドロップします。

17. [function] ノードのプロパティを編集します。

```
名前: 証明書アタッチ結果の取得
コード:
```

```
var str = msg.payload;
var json = str.substr(str.indexOf("{"), str.lastIndexOf("}") + 1);
var success_msg = "{ ¥"deviceCertAttach¥" : ¥"success¥" }";

try {
    var res = JSON.parse(json);
    if (Object.keys(res).length === 0) {
        msg.payload = success_msg;
    }
} catch {
    msg.payload = json;
}

return msg;
```

図 4.58 [証明書アタッチ結果の取得] ノードのプロパティ内容

18. [証明書をデバイスにアタッチ] ノードの右側にある端子をクリックし、[証明書アタッチ結果の取得] ノードの左側の端子を選択して放します。

19. [証明書アタッチ結果の取得] ノードの右側の端子をクリックし、[debug] ノードの左側の端子を選択して放します。

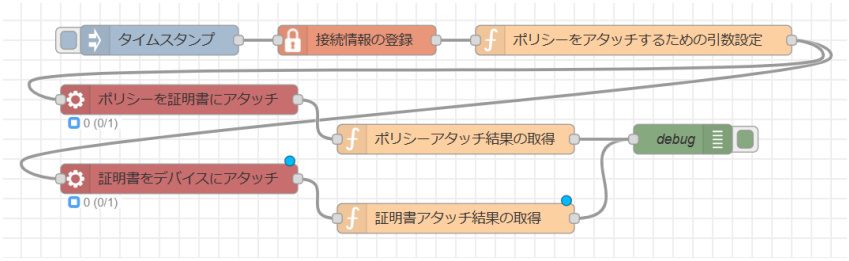


図 4.59 デバイスにポリシーをアタッチするフロー

20. 画面右上の [デプロイ] を押します。

21. [inject] ノードの左の四角を押すとデバイスにポリシーがアタッチされます。

アタッチに成功した場合は "success" が表示されます。もし、失敗した場合はエラーに応じたステータスコードを表示します。

4.6.9. デバイスシャドウを取得するフローの作成

デバイスが登録されている場合、デバイスシャドウが取得可能です。対象とするデバイスは 「4.6.5. デバイスを登録するフローの作成」 で取得した `thingName` を使用します。

1. パレットから [inject] ノードをワークスペースにドラッグ、ドロップします。プロパティはデフォルトから変更ありません。

2. [credentials] ノードをドラッグ、ドロップします。

3. [credentials] ノードのプロパティを編集します。[追加] ボタンを押して値を追加します。private のステータスが [文字列] のままだと内容が表示されたままのため、確認後は [hidden] に変更するのを推奨します。

```
Name: 接続情報の登録
Values:
private: [hidden] ❶
to: [msg.] aws_device_endpoint
```

❶ 「4.6.3. デバイスデータエンドポイントを取得する」 で取得したデバイスデータエンドポイント(例: aaaaaaaa.iot.ap-northeast-1.amazonaws.com)

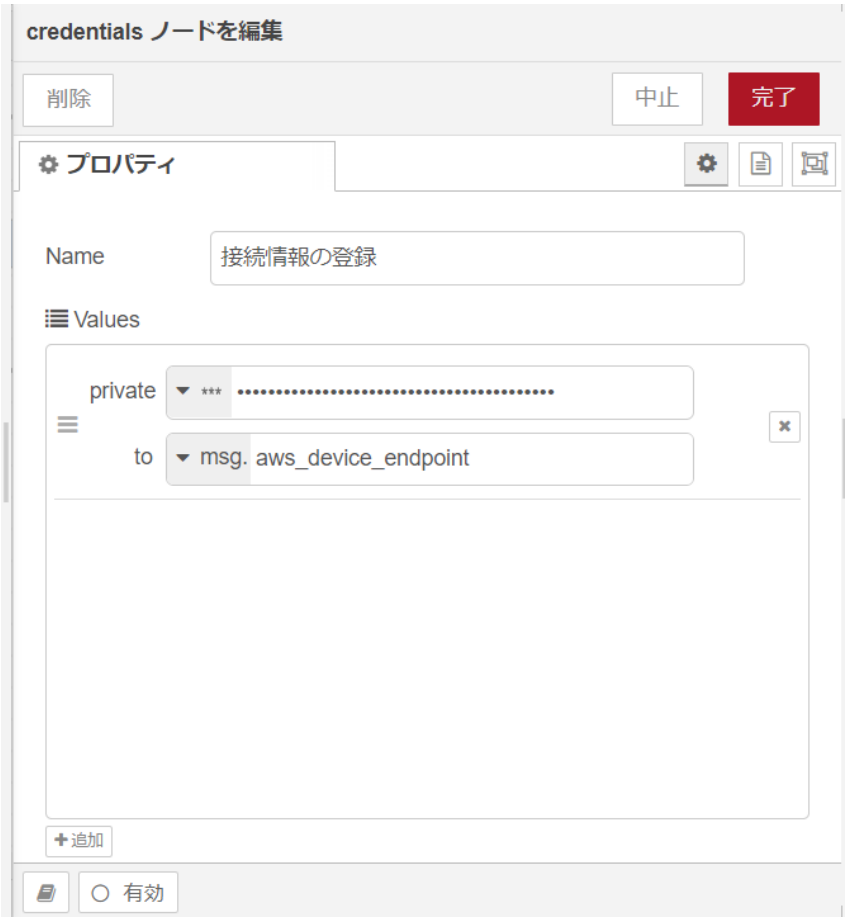


図 4.60 [接続情報の登録] ノードのプロパティ内容

4. [inject] ノードの右側の端子をクリックし、[接続情報の登録] ノードの左側の端子を選択して放します。

5. [function] ノードをドラッグ、ドロップします。

6. [function] ノードのプロパティを編集します。

```
名前: デバイスシャドウ取得のための引数設定
コード:
msg.payload = msg.aws_device_endpoint;
return msg;
```

図 4.61 [デバイスシャドウ取得のための引数設定] ノードのプロパティ内容

7. [接続情報の登録] ノードの右側の端子をクリックし、[デバイスシャドウ取得のための引数設定] ノードの左側の端子を選択して放します。

8. [exec queue] ノードをドラッグ、ドロップします。

9. [exec queue] ノードのプロパティを編集します。

```
名前: 暗号化を使用したデバイスシャドウの取得
コマンド: bash $file
Queue: 1
Debug Mode: ✓
Append msg.payload: ✓
構文: Shell
内容:
ENDPOINT="$1"
URI=/things/${AT_SERIAL_NUMBER}/shadow

curl --tlsv1.2 ¥
        --cacert /cert/AmazonRootCA1.pem ¥
        --key /cert/key.pem ¥
        --cert /cert/device_cert.pem ¥
        --request GET ¥
        "https://${ENDPOINT}:8443${URI}"
```

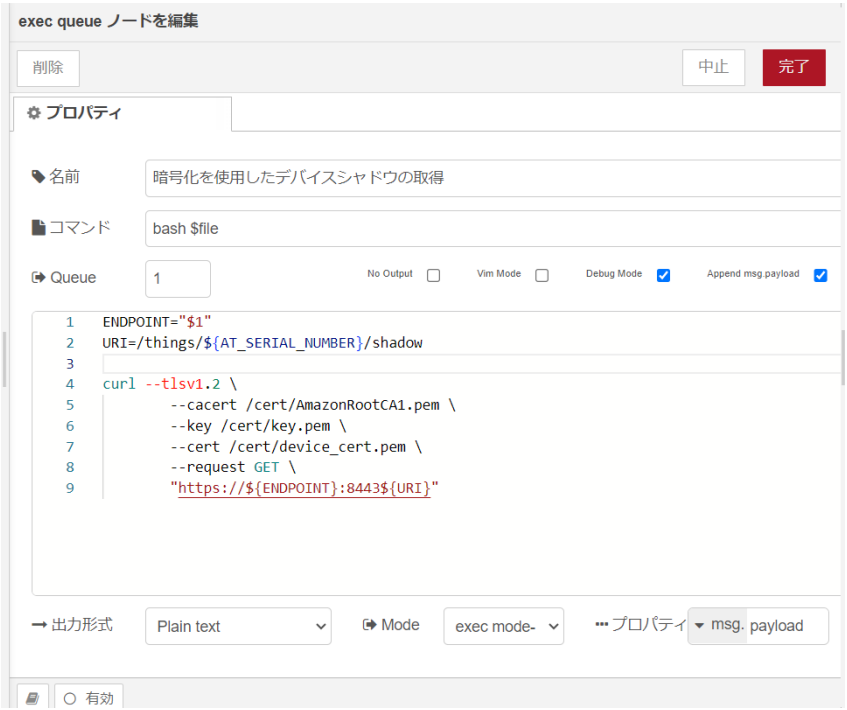


図 4.62 [暗号化を使用したデバイスシャドウの取得] ノードのプロパティ内容

10. [デバイスシャドウ更新のための引数設定] ノードの右側の端子をクリックし、[暗号化を使用したデバイスシャドウの取得] ノードの左側の端子を選択して放します。

11. [function] ノードをドラッグ、ドロップします。

12. [function] ノードのプロパティを編集します。

```
名前: デバイスシャドウ内容の取得
コード:
var res = msg.payload.match(/¥{.*¥}/);
msg.payload = res[0];
return msg;
```

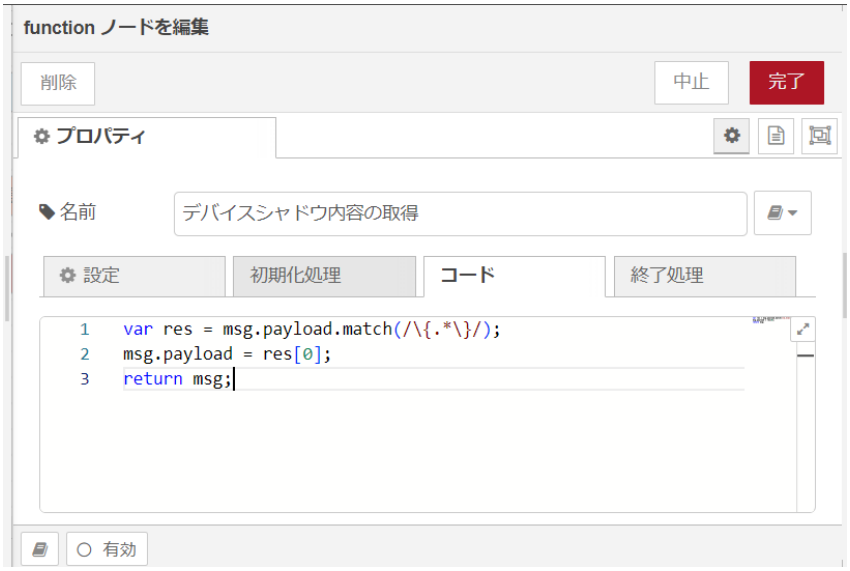


図 4.63 [デバイスシャドウ内容の取得] ノードのプロパティ内容

13. [暗号化を使用したデバイスシャドウの取得] ノードの右側の端子をクリックし、[デバイスシャドウ内容の取得] ノードの左側の端子を選択して放します。

14. [debug] ノードをドラッグ、ドロップします。

15. [デバイスシャドウ内容の取得] ノードの右側の端子をクリックし、[debug] ノードの左側の端子を選択して放します。

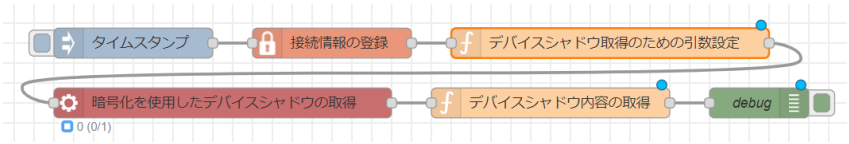


図 4.64 データの暗号化を使用したデバイスシャドウを取得するフロー

16. 画面右上の [デプロイ] を押します。

17. [inject] ノードの左の四角を押すと取得したデバイスシャドウの内容がデバッグメッセージに表示されます。デバイスシャドウが作成されていない場合は "No shadow exists with name: 001234567890" のように表示されます。

4.6.10. デバイスシャドウを更新するフローの作成

デバイスシャドウを更新します。対象とするデバイスは 「4.6.5. デバイスを登録するフローの作成」で取得した `thingName` を使用します。

1. パレットから [inject] ノードをワークスペースにドラッグ、ドロップします。

2. [inject] ノードのプロパティを編集します。

```
名前: デバイスシャドウ内容
msg.payload: { "state": {"desired": {"message": "success" }}}
```

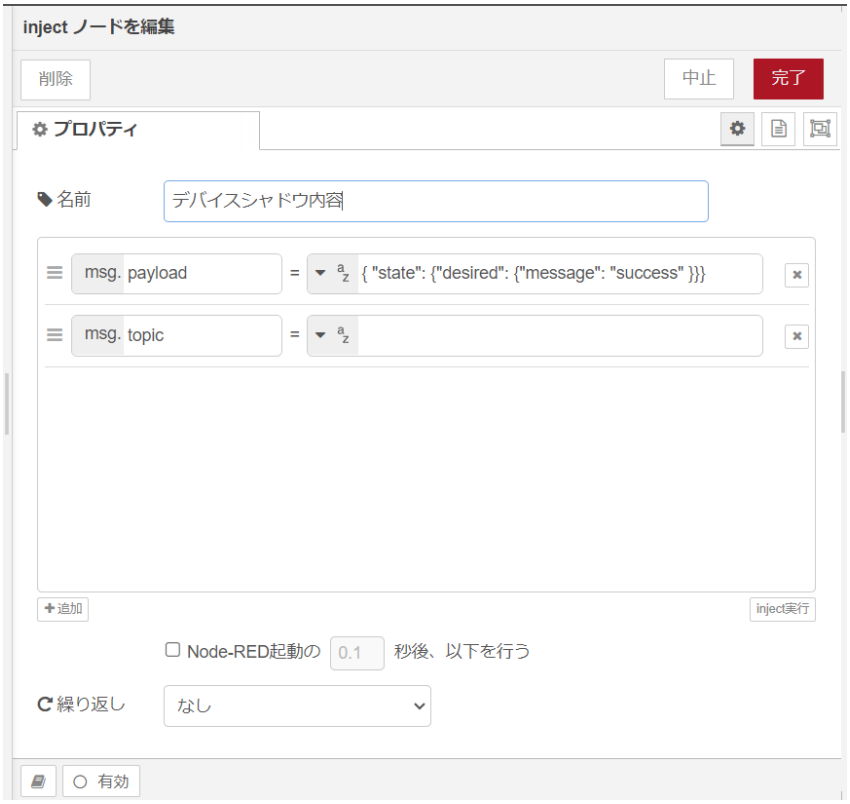


図 4.65 [デバイスシャドウ内容] ノードのプロパティ内容

3. [credentials] ノードをドラッグ、ドロップします。

4. [credentials] ノードのプロパティを編集します。[追加] ボタンを押して値を追加します。private のステータスが [文字列] のままだと内容が表示されたままのため、確認後は [hidden] に変更するのを推奨します。

```
Name: 接続情報の登録
Values:
private: [hidden] ❶
to: [msg.] aws_device_endpoint
```

❶ 「4.6.3. デバイスデータエンドポイントを取得する」 で取得したデバイスデータエンドポイント(例: aaaaaaaa.iot.ap-northeast-1.amazonaws.com)

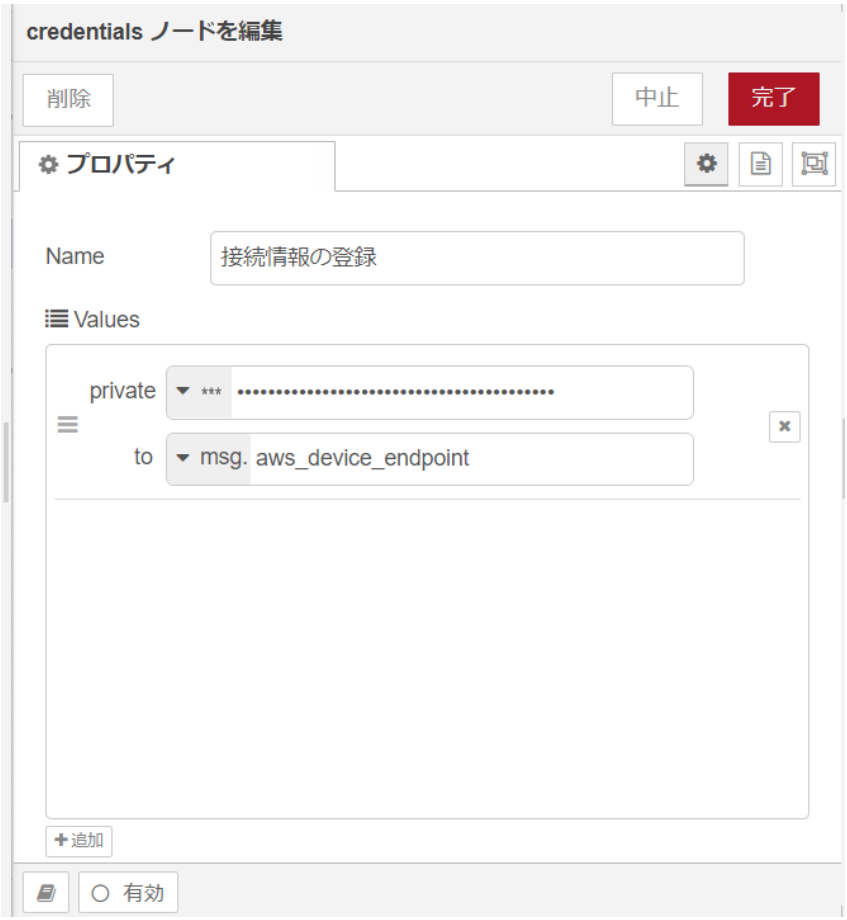


図 4.66 [接続情報の登録] ノードのプロパティ内容

5. [inject] ノードの右側の端子をクリックし、[接続情報の登録] ノードの左側の端子を選択して放します。

6. [function] ノードをドラッグ、ドロップします。

7. [function] ノードのプロパティを編集します。

```
名前：デバイスシャドウ更新のための引数設定
コード:
msg.payload = "¥'" + msg.payload + "¥'"
    + " " + msg.aws_device_endpoint;
return msg;
```

図 4.67 [デバイスシャドウ更新のための引数設定] ノードのプロパティ内容

8. [接続情報の登録] ノードの右側の端子をクリックし、[デバイスシャドウ取得のための引数設定] ノードの左側の端子を選択して放します。

9. [exec queue] ノードをドラッグ、ドロップします。

10. [exec queue] ノードのプロパティを編集します。

```
名前: 暗号化を使用したデバイスシャドウの更新
コマンド: bash $file
Queue: 1
Debug Mode: ✓
Append msg.payload: ✓
構文: Shell
内容:
MESSAGES="$1"
ENDPOINT="$2"
URI=/things/${AT_SERIAL_NUMBER}/shadow

curl --tlsv1.2 ¥
        --cacert /cert/AmazonRootCA1.pem ¥
        --key /cert/key.pem ¥
        --cert /cert/device_cert.pem ¥
        --request POST -v ¥
        -d "${MESSAGES}" ¥
        "https://${ENDPOINT}:8443${URI}"
```

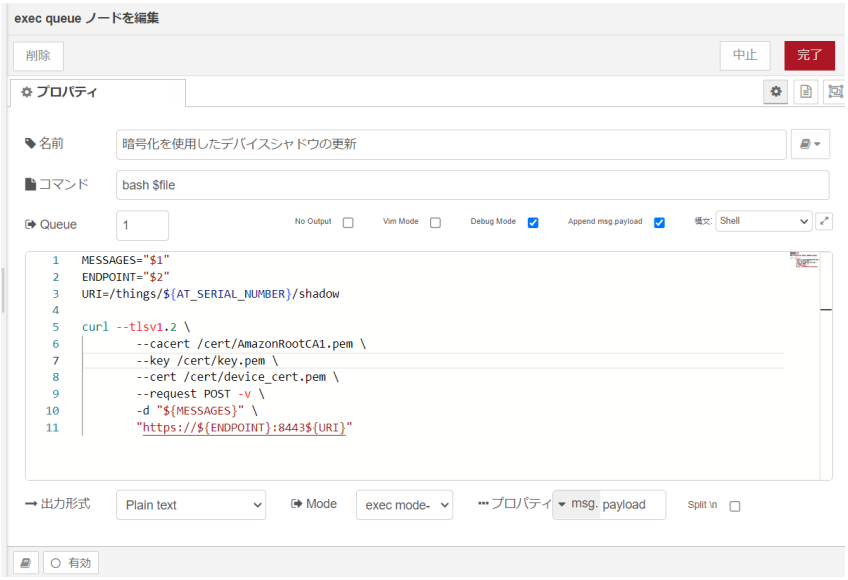


図 4.68 [暗号化を使用したデバイスシャドウの更新] ノードのプロパティ内容

11. [デバイスシャドウ更新のための引数設定] ノードの右側の端子をクリックし、[暗号化を使用したデバイスシャドウの更新] ノードの左側の端子を選択して放します。

12. [function] ノードをドラッグ、ドロップします。

13. [function] ノードのプロパティを編集します。

```
名前: デバイスシャドウ内容の取得
コード:
var res = msg.payload.match(/¥{.*¥}/);
msg.payload = res[0];
return msg;
```

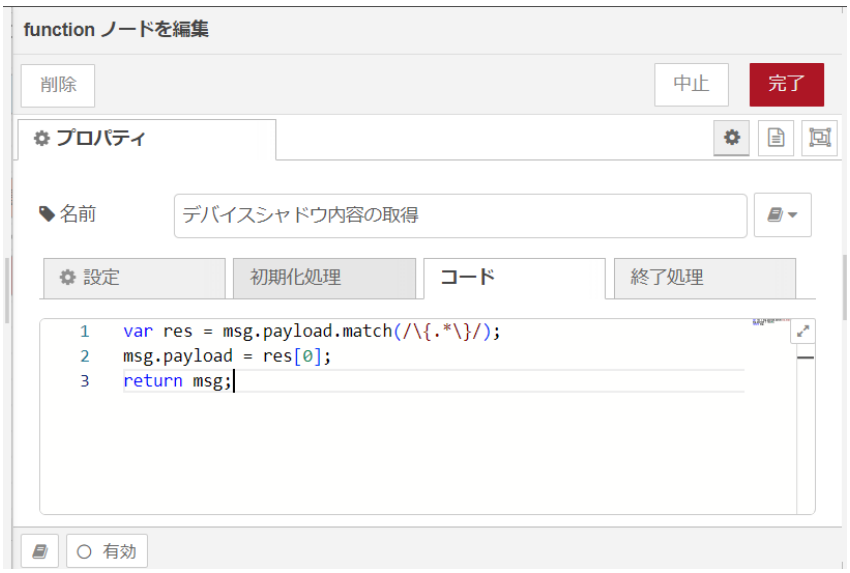


図 4.69 [デバイスシャドウ内容の取得] ノードのプロパティ内容

14. [暗号化を使用したデバイスシャドウの更新] ノードの右側の端子をクリックし、[デバイスシャドウ内容の取得] ノードの左側の端子を選択して放します。

15. [debug] ノードをドラッグ、ドロップします。

16. [デバイスシャドウ内容の取得] ノードの右側の端子をクリックし、[debug] ノードの左側の端子を選択して放します。

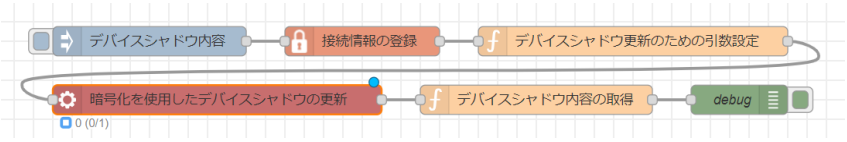


図 4.70 データの暗号化を使用したデバイスシャドウを更新するフロー

17. 画面右上の [デプロイ] を押します。

18. [inject] ノードの左の四角を押すと更新したデバイスシャドウの内容がデバッグメッセージに表示されます。

4.6.11. デバイスシャドウを削除するフローの作成

デバイスシャドウを削除します。対象とするデバイスは 「4.6.5. デバイスを登録するフローの作成」で取得した `thingName` を使用します。

1. パレットから [inject] ノードをワークスペースにドラッグ、ドロップします。プロパティはデフォルトから変更ありません。

2. [credentials] ノードをドラッグ、ドロップします。

3. [credentials] ノードのプロパティを編集します。[追加] ボタンを押して値を追加します。private のステータスが [文字列] のままだと内容が表示されたままのため、確認後は [hidden] に変更するのを推奨します。

```
Name: 接続情報の登録
Values:
private: [hidden] ❶
to: [msg.] aws_device_endpoint
```

❶ 「4.6.3. デバイスデータエンドポイントを取得する」 で取得したデバイスデータエンドポイント (例: aaaaaaaa.iot.ap-northeast-1.amazonaws.com)

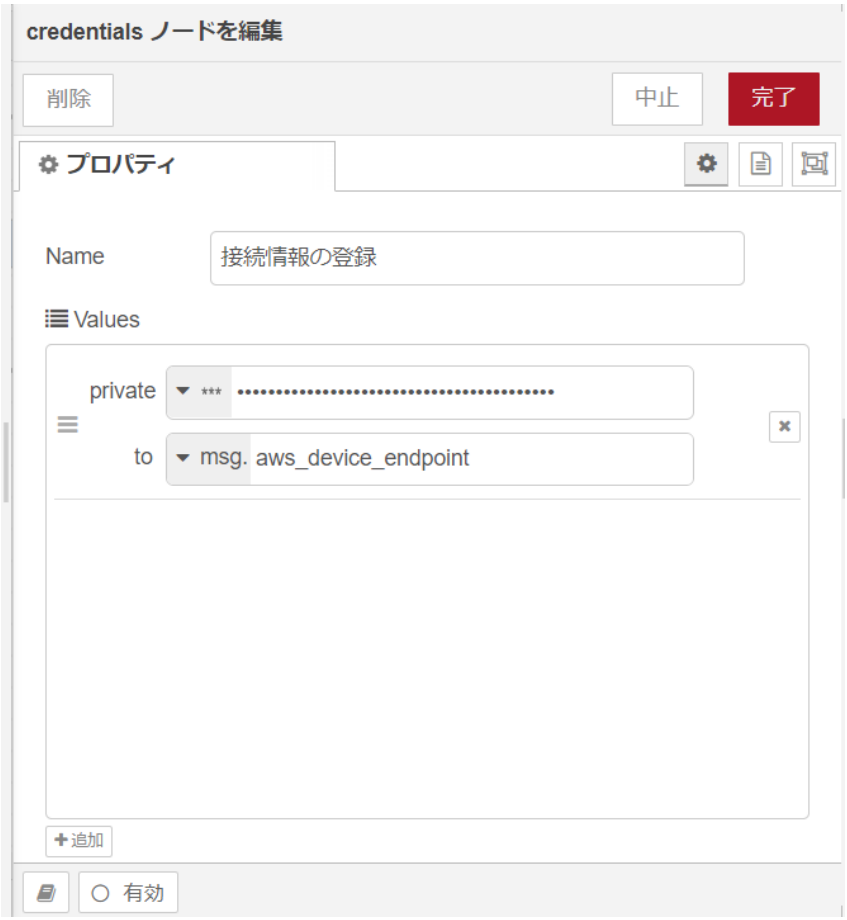


図 4.71 [接続情報の登録] ノードのプロパティ内容

4. [inject] ノードの右側の端子をクリックし、[接続情報の登録] ノードの左側の端子を選択して放します。

5. [function] ノードをドラッグ、ドロップします。

6. [function] ノードのプロパティを編集します。

```
名前: デバイスシャドウ削除のための引数設定
コード:
msg.payload = msg.aws_device_endpoint;
return msg;
```

図 4.72 [デバイスシャドウ削除のための引数設定] ノードのプロパティ内容

7. [接続情報の登録] ノードの右側の端子をクリックし、[デバイスシャドウ削除のための引数設定] ノードの左側の端子を選択して放します。

8. [exec queue] ノードをドラッグ、ドロップします。

9. [exec queue] ノードのプロパティを編集します。

```
名前: 暗号化を使用したデバイスシャドウの削除
コマンド: bash $file
Queue: 1
Debug Mode: ✓
Append msg.payload: ✓
構文: Shell
内容:
ENDPOINT="$1"
URI=/things/${AT_SERIAL_NUMBER}/shadow

curl --tlsv1.2 ¥
        --cacert /cert/AmazonRootCA1.pem ¥
        --key /cert/key.pem ¥
        --cert /cert/device_cert.pem ¥
        --request POST -v ¥
        -d "${MESSAGES}" ¥
        "https://${ENDPOINT}:8443${URI}"
```

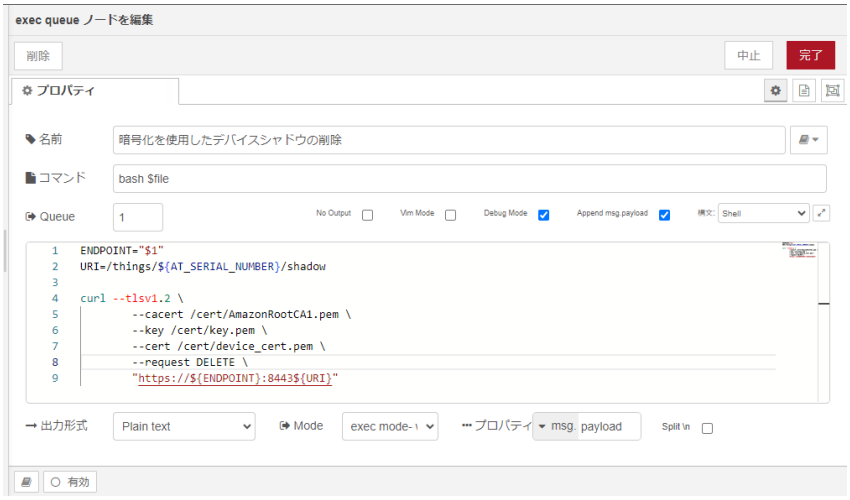


図 4.73 [暗号化を使用したデバイスシャドウの削除] ノードのプロパティ内容

10. [デバイスシャドウ削除のための引数設定] ノードの右側の端子をクリックし、[暗号化を使用したデバイスシャドウの削除] ノードの左側の端子を選択して放します。

11. [function] ノードをドラッグ、ドロップします。

12. [function] ノードのプロパティを編集します。

```
名前：デバイスシャドウ内容の取得
コード:
var res = msg.payload.match(/¥{.*¥}/);
msg.payload = res[0];
return msg;
```

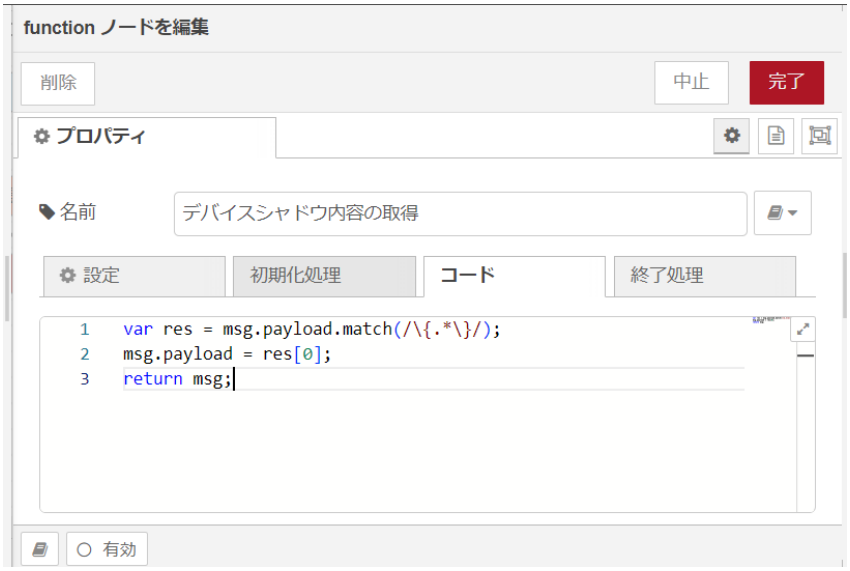


図 4.74 [デバイスシャドウ内容の取得] ノードのプロパティ内容

13. [暗号化を使用したデバイスシャドウの更新] ノードの右側の端子をクリックし、[デバイスシャドウ内容の取得] ノードの左側の端子を選択して放します。

14. [debug] ノードをドラッグ、ドロップします。

15. [デバイスシャドウ内容の取得] ノードの右側の端子をクリックし、[debug] ノードの左側の端子を選択して放します。

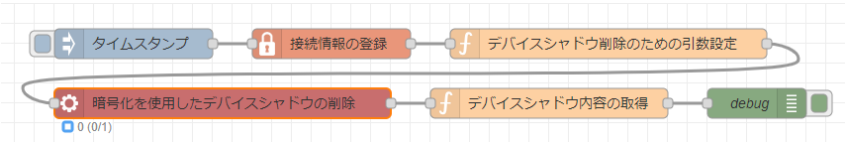


図 4.75 データの暗号化を使用したデバイスシャドウを削除するフロー

16. 画面右上の [デプロイ] を押します。

17. [inject] ノードの左の四角を押すと削除したデバイスシャドウの内容がデバッグメッセージに表示されます。デバイスシャドウがすでに削除されている場合は "No shadow exists with name: 001234567890" のように表示されます。

5. 量産する

本章では Armadillo を組み込んだ最終製品をお客様が製造・量産するうえで、必要となる情報や作業について記載します。

- 「5.1. 概略」では、量産の進め方に関する概略を記載します。
- 「5.2. BTO サービスを使わない場合と使う場合の違い」では、BTO(Build To Order) サービスに関する説明をします。
- 「5.3. 量産時のイメージ書き込み手法」では、開発を完了したソフトウェアの量産用イメージ作成・書き込み方法を説明します。
- 「5.6. アップデート用 SWU の生成手法」では、開発完了したイメージを書き込んだ製品に対して、アップデート用の SWU を生成する方法を説明します。

5.1. 概略

量産の進め方の概略図を「図 5.1. Armadillo 量産時の概略図」に示します。お客様の製品仕様や製造工程の要件によってはこの例とは違った工程順となる場合や、工程の追加・削除がある可能性があります。

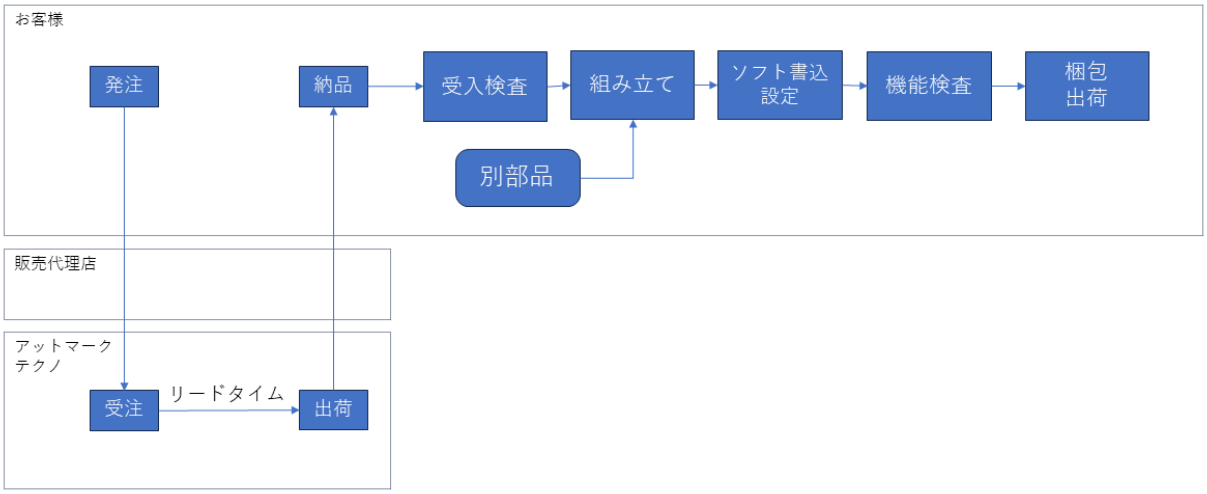


図 5.1 Armadillo 量産時の概略図

5.1.1. リードタイムと在庫

量産モデルを発注後、お客様に納品されるまでにリードタイムが発生します。開発セットや少量の量産モデル購入の場合、アットマークテクノや代理店在庫によって、短期間で納品できることもあります。しかし、まとまった数量の量産モデルの場合、納品までにお時間をいただくことがあります。新規に製品を量産・出荷する場合はリードタイムを考慮したスケジューリングをお願いします。また、リピート製造をする場合でも、欠品を起こさないよう適切な在庫の確保をお願いいたします。

リードタイムは状況・タイミングによって異なりますので、都度、弊社営業、ご利用の販売代理店にお問い合わせください。

5.1.2. Armadillo 納品後の製造・量産作業

お客様が Armadillo を納品後に次に示すようなキッティング作業、組み立て、検査を実施し出荷を行います。

- ソフトウェア書き込み
  - Armadillo Base OS やアプリケーションコンテナイメージの書き込み
  - 設定ファイルの書き込み
- 別部品の組み立て
  - SD カード/ SIM カード/ RTC バックアップ電池等の接続
  - 拡張基板接続やセンサー・外部機器の接続
  - お客様専用筐体への組み込み
- 検査
  - Armadillo の受け入れ検査
  - 組み立て後の通電電検・機能検査
  - 目視検査
- 梱包作業
- 出荷作業

有償の BTO サービスを利用することで、これらの作業の一部をアットマークテクノへ委託・実施済みの状態で Armadillo を納品することも可能です。費用はいただきますがお客様による工程立ち上げ、場所の確保、作業者の教育、品質管理等のトータルコストを考えると委託した方が安く済むケースが多いです。

また、 BTO サービスではお受けできないようなキッティング、検査、作業については、実施可能な業者をご紹介する等、個別の対応をすることで解決できる場合もございます。詳しくは弊社担当の営業、またはご利用の販売代理店にご相談ください。

5.2. BTO サービスを使わない場合と使う場合の違い

図 5.2 BTO サービスで対応する範囲

5.2.1. BTO サービスを利用しない(標準ラインアップ品)

有償の量産サービスを利用しない場合、標準ラインアップ仕様での納品となります。大きく分けて試作開発用途で使う「開発セット」と量産向けの「量産モデル」の 2 種類があります。量産用途では「量産モデル」をご利用ください。

「量産モデル」には AC アダプタ等のオプション品が付属されておりませんので、内容物を確認の上、発注をお願いいたします。量産モデルのラインアップや付属品については、各製品の製品マニュアルを参照してください。

設計開発時には、開発に必要なものを一式含んだ「開発セット」、量産時には、必要最小限のセット内容に絞った「量産用」もしくは「量産ボード」をご購入ください。「量産用」はケース有、「量産ボード」はケース無となります。

5.2.1.1. 標準ラインアップ品に書き込まれているソフトウェア

標準ラインアップ品に書き込まれるソフトウェアイメージ(Armadillo Base OS)は、アットマークテクノで公開している標準イメージとなります。また、ソフトウェアバージョンは指定することができず、ランニングチェンジで随時最新版を適用していきます。このため、納品後の Armadillo 個体では、開発段階で評価した Armadillo Base OS と異なるバージョンが書き込まれている可能性があります。

また、アプリケーションコンテナについては何も書き込まれていない状態となります。

納品後、お客様の量産工程でソフトウェアの書き込み作業が必要となります。詳しくは「5.3. 量産時のイメージ書き込み手法」をご確認ください。

5.2.2. BTO サービスを利用する

BTO サービスは、セミオーダー式メニューから選択して Armadillo の量産品を一括手配いただける有償サービスです。標準ラインアップ品の仕様をベースとして、搭載するモジュールの種類やケース、 AC アダプタの有無、お客様支給品の SD カードや SIM カードの接続、お客様ご指定のソフトウェアイメージ書き込みなど、メニュー内から指定可能なキッティング項目を選択・指定することが可能です。

販売代理店またはアットマークテクノの窓口からお申し込みいただけます。

製品ごとに、対応できる作業とできない作業がございます。また、販売直後の製品の場合など BTO サービスに未対応である場合もあります。詳しくは Armadillo サイトの BTO サービス [https://armadillo.atmark-techno.com/services/customize/bto] をご確認ください。

5.3. 量産時のイメージ書き込み手法

量産時に必要な手順は最終製品によって異なりますが、開発したソフトウェアを Armadillo に書き込む手順は必ず実施することになります。Armadillo Base OS 搭載製品において、量産時に任意のソフトウェアを書き込む際には、以下の 2 つの手法のどちらかを用いると実現できます。

- インストールディスクを用いてソフトウェアを書き込む
- SWUpdate を用いてソフトウェアを書き込む

ただし、SWUpdate は運用中の Armadillo のアップデート機能であり、量産時のイメージ書き込みは本来の用途でないため、基本的にはイメージ書き込みに特化しているインストールディスクを用いた方法を選択してください。そのため、インストールディスクイメージの作成、書き込み方法について「5.4. 開発したシステムをインストールディスクにする」に記します。

5.4. 開発したシステムをインストールディスクにする

ATDE と Visual Studio Code (以降 VS Code と記載します) を使用して、開発したシステムのインストールディスクイメージを生成します。「5.4.2. VS Code のセットアップ」 を参考に、 ATDE に VS Code 開発用エクステンションをインストールしてください。VS Code を使用してインストールディスクを生成する場合は以下の手順になります。

- 仮想環境のセットアップ
- VS Code のセットアップ
- VS Code を使用した初期設定用 SWU の生成
- 初期設定用 SWU を ABOS Web からインストール
- VS Code を使用したインストールディスク作成用 SWU の生成
- Armadillo に USB メモリを挿入
- インストールディスク作成用 SWU を ABOS Web からインストール
- USB メモリ上にインストールディスクイメージを生成

この機能を使用するには、以下に示すバージョンのソフトウェアが必要です。

- ABOSDE 1.6.0 以上
- mkswu 5.3 以上
- abos-base 2.3 以上

Visual Studio Code および VS Code は、Microsoft Corporation の米国およびその他の国における登録商標または商標です。

5.4.1. 仮想環境のセットアップ

作業用 PC をセットアップします。アットマークテクノでは、製品のソフトウェア開発や動作確認を簡単に行うために、Oracle VM VirtualBox 仮想マシンのデータイメージを提供しています。このデータイメージを ATDE(Atmark Techno Development Environment) と呼びます。ATDE の起動には仮想化ソフトウェアである VirtualBox を使用します。

Oracle VM VirtualBox には以下の特徴があります。

- GPL v2(General Public License version 2)で提供されている [1]
- VMware 形式の仮想ディスク(.vmdk)ファイルに対応している

5.4.1.1. VirtualBox のインストール

ATDE を使用するために、作業用 PC に VirtualBox をインストールします。VirtualBox の Web ページ(https://www.virtualbox.org/) を参照してインストールしてください。

また、ホスト OS が Linux の場合、デフォルトでは VirtualBox で USB デバイスを使用することができません。ホスト OS（Linux）で以下のコマンドを実行後、ホスト OS を再起動してください。

```
[PC ~]$ sudo usermod -a -G vboxusers $USER
```

ホスト OS が Windows の場合はこの操作は必要ありません。

5.4.1.2. ATDE のアーカイブを取得

ATDE のアーカイブ(.ova ファイル)を Armadillo サイト(https://armadillo.atmark-techno.com/resources/software/atde/atde-v9)からダウンロードします。

アットマークテクノ製品の種類ごとに対応している ATDE のバージョンが異なります。本製品に対応している ATDE のバージョンは以下のとおりです。

VirtualBox	ATDE9 v20240925 以降
VMware	ATDE9 v20230123 から ATDE9 v20240826

ATDE9 は Debian GNU/Linux 11 (コードネーム bullseye) をベースに、Armadillo-X2 のソフトウェア開発を行うために必要なクロス開発ツールや、Armadillo-X2 の動作確認を行うために必要なツールが事前にインストールされています。

作業用 PC の動作環境(ハードウェア、VirtualBox、ATDE の対応アーキテクチャなど)により、 ATDE が正常に動作しない可能性があります。VirtualBox の Web ページ(https://www.virtualbox.org/) から、使用している VirtualBox のドキュメントなどを参照して動作環境を確認してください。

5.4.1.3. ATDE のインポート

1. VirtualBox を起動し、[ファイル]-[仮想アプライアンスのインポート]を選択します。
2. [ソース]の項目で、ダウンロードした ATDE のアーカイブ(.ova ファイル)を選択します。

[1]バージョン 3.x までは PUEL(VirtualBox Personal Use and Evaluation License)が適用されている場合があります。

3. [設定]の項目で、[ハードドライブを VDI としてインポート]のチェックを外します。
4. [完了]をクリックします。インポートの処理が完了するまで数分程要します。
5. インポートの処理が完了したら、ホスト OS の環境に合わせた設定に更新するため仮想マシンを選択して[設定]をクリックした後に[OK]をクリックします。

ATDE に割り当てるメモリおよびプロセッサ数を増やすことで、 ATDE をより快適に使用することができます。仮想マシンのハードウェア設定の変更方法については、VirtualBox の Web ページ (https://www.virtualbox.org/) から、VirtualBox のドキュメントなどを参照してください。

5.4.1.4. ATDE の起動

1. 仮想マシンを選択して[起動]をクリックしてください。
2. ATDE のログイン画面が表示されます。

ATDE にログイン可能なユーザーを、「表 5.1. ユーザー名とパスワード」に示します [2]。

表 5.1 ユーザー名とパスワード

ユーザー名	パスワード	権限
atmark	atmark	一般ユーザー
root	root	特権ユーザー

5.4.1.5. コマンドライン端末(GNOME 端末)の起動

Armadillo を利用した開発では、 CUI (Character-based User Interface)環境を提供するコマンドライン端末を通じて、 Armadillo や ATDE に対して操作を行う場面が多々あります。コマンドライン端末にはいくつかの種類がありますが、ここでは GNOME デスクトップ環境に標準インストールされている GNOME 端末を起動します。

GNOME 端末を起動するには、「図 5.3. GNOME 端末の起動」のようにデスクトップ左上のアプリケーションの「ユーティリティ」カテゴリから「端末」を選択してください。

[2]特権ユーザーで GUI ログインを行うことはできません

図 5.3 GNOME 端末の起動

「図 5.4. GNOME 端末のウィンドウ」のようにウィンドウが開きます。

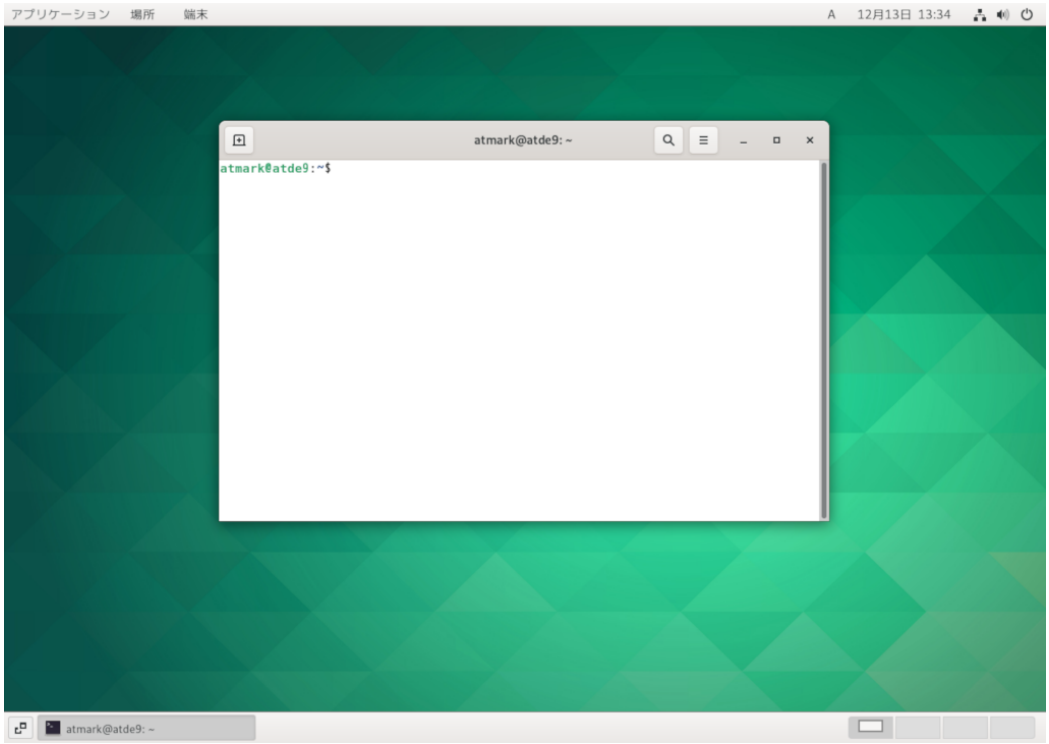


図 5.4 GNOME 端末のウィンドウ

5.4.1.6. ソフトウェアのアップデート

コマンドライン端末から次の操作を行い、ソフトウェアを最新版へアップデートしてください。

```
[ATDE ~]$ sudo apt update
[ATDE ~]$ sudo apt upgrade
```

図 5.5 ソフトウェアをアップデートする

5.4.1.7. 取り外し可能デバイスの使用

VirtualBox は、ゲスト OS (ATDE)による取り外し可能デバイス(USB デバイスや DVD など)の使用をサポートしています。デバイスによっては、ホスト OS (VirtualBox を起動している OS)と ATDE で同時に使用することができません。そのようなデバイスを ATDE で使用するためには、ATDE にデバイスを接続する 「図 5.6. ATDE にデバイスを接続する」 の操作が必要になります。

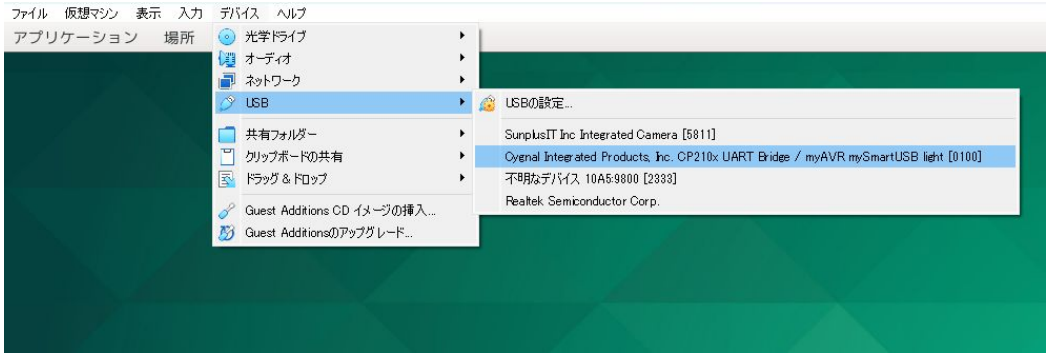


図 5.6 ATDE にデバイスを接続する

5.4.1.8. VirtualBox Guest Additions の再インストール

ATDE は VirtualBox 仮想マシン用ソフトである VirtualBox Guest Additions があらかじめインストールされた状態で配布されています。

Guest Additions のバージョンは VirtualBox 自体のバージョンと連動しているため、お使いの VirtualBox のバージョンと ATDE にインストール済みの Guest Additions のバージョンが異なる場合があります。

VirtualBox と Guest Additions のバージョンが異なることによって問題が起こる可能性は低いですが、これに起因すると思われる不具合（ATDE の画面・共有フォルダー・クリップボード等の不調）が発生した場合は、以下の手順を参考に Guest Additions を再インストールしてください。(実行前に ATDE のスナップショットを作成しておくことを推奨します)

1. ATDE を起動後、上部バーの[ツール]-[Guest Additions CD イメージの挿入]を選択してください。
2. お使いの VirtualBox と同じバージョンの VBox_GAs_[VERSION] が「ファイル」上に表示されます。
3. VBox_GAs_[VERSION] をマウントするために、「ファイル」から VBox_GAs_[VERSION] を押下してください。
4. インストールする前に、以下のコマンドで既にインストール済みの Guest Additions をアンインストールします。

```
sudo /opt/VBoxGuestAdditions-[VERSION]/uninstall.sh
```

5. 以下のコマンドでお使いの VirtualBox のバージョンに合った Guest Additions がインストールされます。

```
cd /media/cdrom0
sudo sh ./autorun.sh
```

5.4.1.9. 共有フォルダーの作成

ホスト OS と ATDE 間でファイルを受け渡す手段として、共有フォルダーがあると大変便利です。ここでは、ホスト OS と ATDE 間の共有フォルダを作成する手順を紹介しますが、不要な方はこの手順をスキップしてください。

1. VirtualBox の上部バーから[デバイス]-[共有フォルダー]-[共有フォルダー設定]を選択します。(「図 5.7. 共有フォルダー設定を開く」)
2. 「図 5.8. 共有フォルダー設定」 の赤枠で示したアイコンをクリックします。
3. 「図 5.9. 共有フォルダーの追加」 のように、[フォルダーのパス]-[その他]を選択して、共有フォルダーに設定したいホスト OS 上のフォルダーを選択します。
4. 「図 5.9. 共有フォルダーの追加」 のように、[自動マウント]と[永続化する]にチェックを入れます。
5. [OK]をクリックして共有フォルダーを追加します。

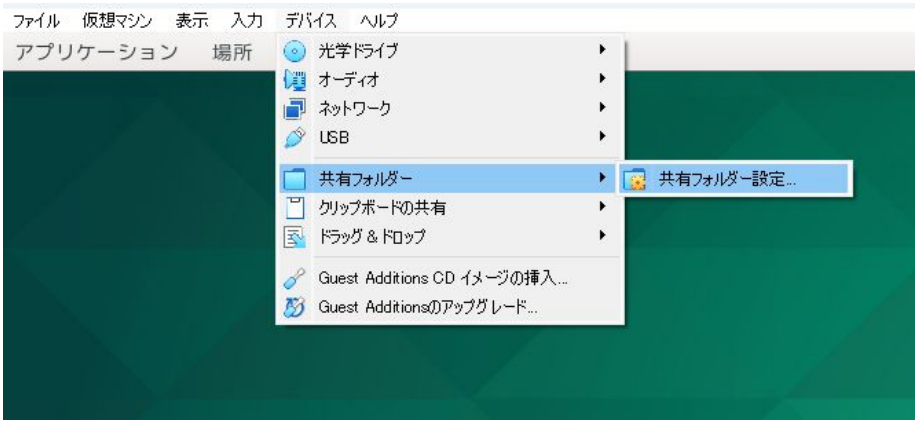


図 5.7 共有フォルダー設定を開く

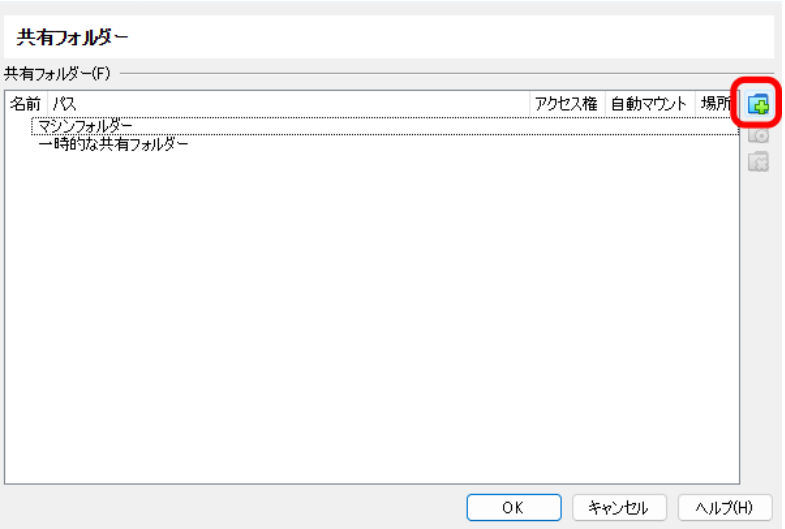


図 5.8 共有フォルダー設定

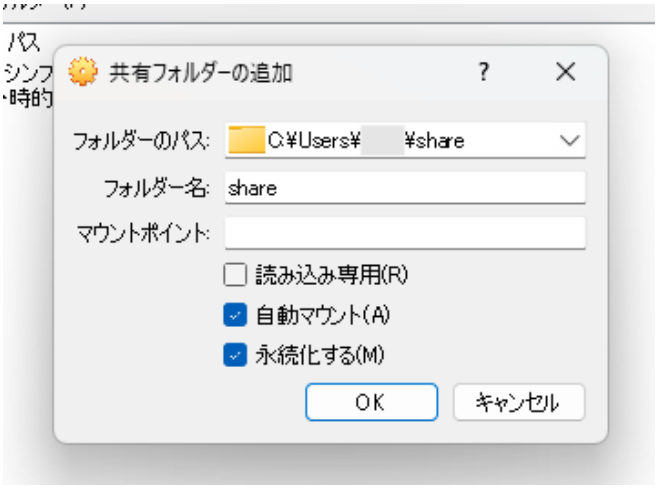


図 5.9 共有フォルダーの追加

図 5.10 「ファイル」に表示される共有フォルダー

追加した共有フォルダーは、「図 5.10. 「ファイル」に表示される共有フォルダー」のように「ファイル」からアクセスするか、または /media/sf_share（共有フォルダー名） からアクセスできます。(share というフォルダー名で作成すると、ATDE 上では sf_share として表示されます。)

5.4.2. VS Code のセットアップ

ATDE に VS Code 及び、開発用エクステンションとクロスコンパイル用ライブラリをインストールしてください。

以下の手順は全て ATDE 上で実施します。

5.4.2.1. ソフトウェアのアップデート

ATDE のバージョン v20230123 以上には、 VS Code がインストール済みのため新規にインストールする必要はありませんが、使用する前には 「図 5.5. ソフトウェアをアップデートする」 を参照して最新版へのアップデートを行ってください。

VS Code を起動するには code コマンドを実行します。

```
[ATDE ~]$ code
```

図 5.11 VS Code を起動する

VS Code を起動すると、日本語化エクステンションのインストールを提案してくることがあります。その時に表示されるダイアログに従ってインストールを行うと VS Code を日本語化できます。

5.4.2.2. VS Code に開発用エクステンションをインストールする

VS Code 上でアプリケーションを開発するためのエクステンションをインストールします。

エクステンションはマーケットプレイスからインストールすることができます。VS Code を起動し、左サイドバーのエクステンションを選択して、検索フォームに「abos」と入力してください。

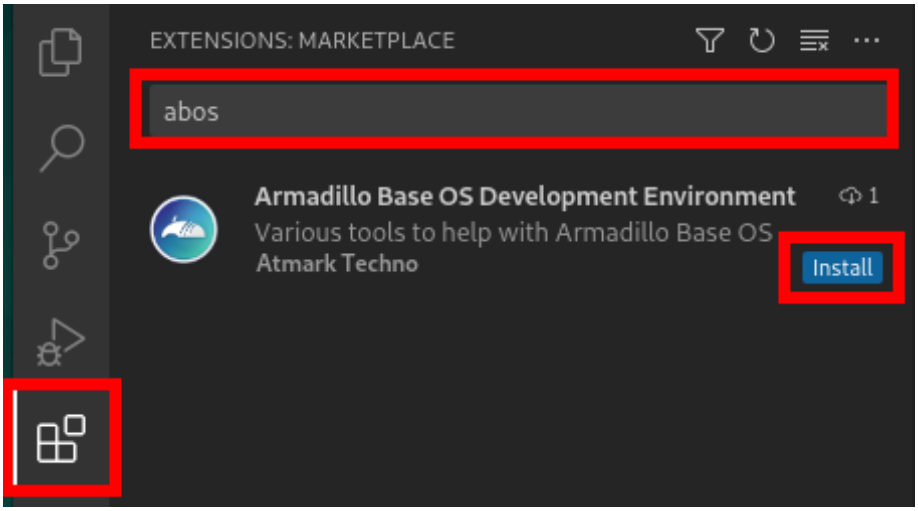


図 5.12 VS Code に開発用エクステンションをインストールする

表示された「Armadillo Base OS Development Environment」の 「Install」ボタンを押すとインストールは完了します。

5.4.3. VS Code を使用した初期設定用 SWU の生成

`initial_setup.swu` はログインパスワードやユーザー固有の証明書などの初期設定を Armadillo にインストールするためのファイルです。`initial_setup.swu` でインストールされるユーザー固有の証明書がない場合、ユーザーが開発したアプリケーションをインストール、またはアップデートすることができません。このため、開発開始時に `initial_setup.swu` のインストールを行う必要があります。

「図 5.13. `initial_setup.swu` を作成する」 に示すように、VS Code の左ペインの [COMMON PROJECT COMMAND] から [Generate Initial Setup Swu] を実行してください。

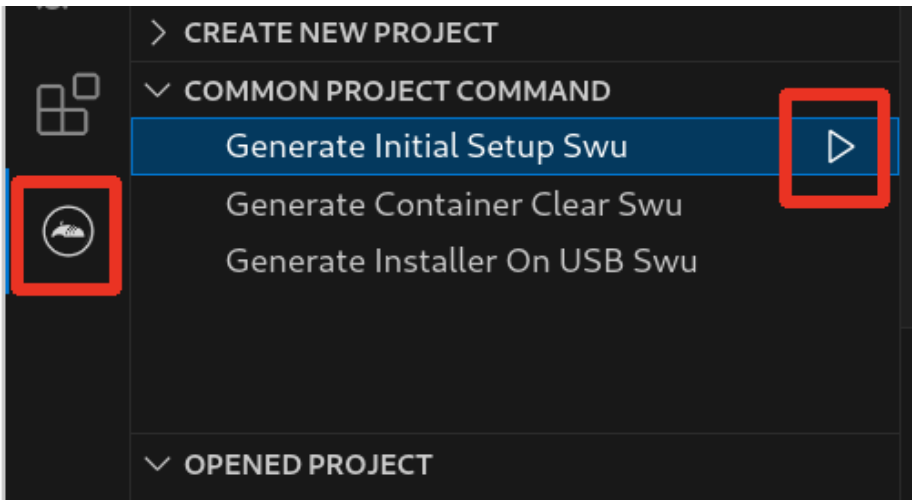


図 5.13 `initial_setup.swu` を作成する

初回実行時には各種設定の入力を求められます。入力する設定の内容を 「図 5.14. initial_setup.swu 初回生成時の各種設定」 に示します。

```
Executing task: ./scripts/generate_initial_setup_swu.sh

mkdir: ディレクトリ '/home/atmark/mkswu' を作成しました
設定ファイルを更新しました：/home/atmark/mkswu/mkswu.conf
証明書のコモンネーム(一般名)を入力してください: [COMMON_NAME] ❶
証明書の鍵のパスワードを入力ください（4-1024 文字）❷
証明書の鍵のパスワード（確認）：
Generating an EC private key
writing new private key to '/home/atmark/mkswu/swupdate.key.tmp'
 -----
アップデートイメージを暗号化しますか？（N/y）❸
アットマークテクノが作成したイメージをインストール可能にしますか？（Y/n）❹
root パスワード: ❺
root のパスワード（確認）：
atmark ユーザのパスワード（空の場合はアカウントをロックします）：❻
atmark のパスワード（確認）：
BaseOS/プリインストールコンテナを armadillo.atmark-techno.com サーバーから自動アップデートしますか？ ⏎
(N/y) ❼
abos-web のパスワードを設定してください。
abos-web のパスワード（空の場合はサービスを無効にします）：❽
abos-web のパスワード（確認）：
/home/atmark/mkswu/initial_setup.swu を作成しました。

"/home/atmark/mkswu/initial_setup.swu" をそのまま使うことができますが、
モジュールを追加してイメージを再構築する場合は次のコマンドで作成してください：
  mkswu "/home/atmark/mkswu/initial_setup.desc" [他の.desc ファイル]

インストール後は、このディレクトリを削除しないように注意してください。
鍵を失うと新たなアップデートはデバイスの /etc/swupdate.pem
を修正しないとインストールできなくなります。
 *  Terminal will be reused by tasks, press any key to close it.

[ATDE ~]$ ls ~/mkswu
initial_setup.desc  initial_setup.swu  mkswu.conf
swupdate.aes-key    swupdate.key       swupdate.pem ❾
```

図 5.14 initial_setup.swu 初回生成時の各種設定

❶ COMMON_NAME には証明鍵の「common name」として会社や製品が分かるような任意の名称を入力してください。

❷ 証明鍵を保護するパスフレーズを 2 回入力します。

❸ SWU イメージ自体を暗号化する場合に「y」を入力します。

❹ アットマークテクノのアップデートをインストールしない場合は「n」を入力します。

❺ root のパスワードを 2 回入力します。使用するパスワードは以下のルールに従ってください。

- 辞書に載っている言葉を使用しない
- 単調な文字列を使用しない

・8 文字以上のパスワード長にする

❻ atmark ユーザーのパスワードを 2 回入力します。何も入力しない場合はユーザーをロックします。使用できるパスワードの制限は root と同様です。

❼ 自動アップデートを無効のままで進みます。ここで「y」を入れると、定期的にアットマークテクノのサーバーからアップデートの有無を確認し、自動的にインストールします。

❽ abos-web を使用する場合はパスワードを設定してください。ここで設定したパスワードは abos-web から変更できます。使用できるパスワードの制限は root と同様です。

❾ 作成したファイルを確認します。「swupdate.aes-key」は暗号化の場合にのみ作成されます。

ファイルは ~/mkswu/initial_setup.swu に保存されます。この SWU イメージを 「5.4.4. 初期設定用 SWU を ABOS Web からインストール」 を参照して Armadillo へインストールしてください。

インストール後に ~/mkswu ディレクトリ以下にある mkswu.conf と、鍵ファイルの swupdate.* をなくさないようにしてください。

5.4.4. 初期設定用 SWU を ABOS Web からインストール

ABOS Web を使用して、PC 上の SWU イメージや HTTP サーバー上の SWU イメージを Armadillo にインストールすることができます。生成した initial_setup.swu をインストールします。

ABOS Web のトップページから、"SWU インストール"をクリックすると、「図 5.15. SWU インストール」の画面に遷移します。

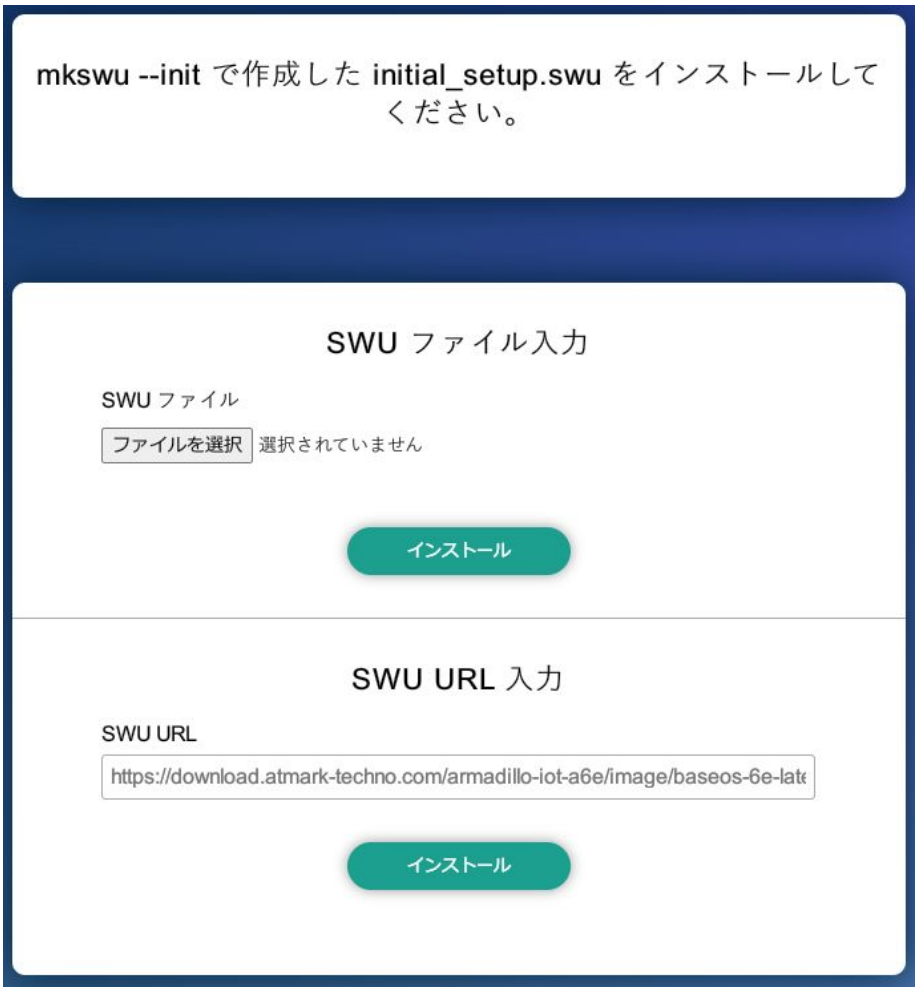


図 5.15 SWU インストール

この画面では、PC 上の SWU イメージファイルまたは、HTTP サーバー上の SWU イメージファイルの URL を指定して、Armadillo にインストールすることができます。Armadillo のソフトウェアのアップデート用に最初に行う設定で作成する initial_setup.swu が、まだ Armadillo にインストールされていなければ、"mkswu --init で作成した initial_setup.swu をインストールしてください。" というメッセージを画面上部に表示します。

生成した `initial_setup.swu` を選択します。実行中のログが ABOS Web 上に表示されます。インストールに成功したら完了です。

5.4.5. VS Code を使用したインストールディスク作成用 SWU の生成

VS Code の左ペインの [COMMON PROJECT COMMAND] から [Generate Installer On USB Swu] を実行します。

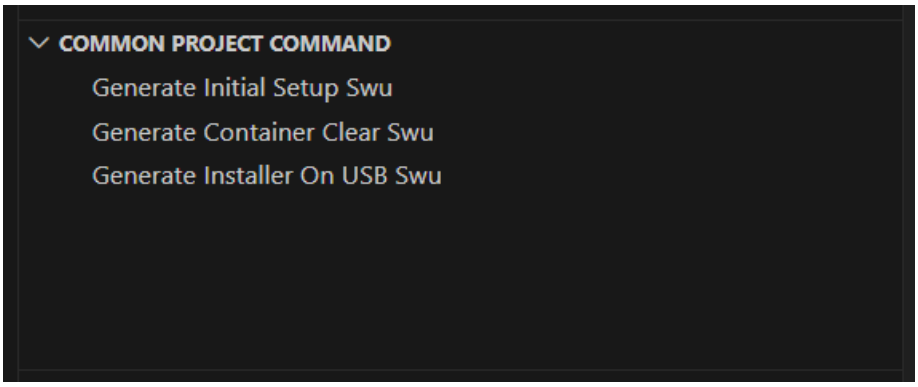


図 5.16 make-installer.swu を作成する

次に、対象製品を選択します。

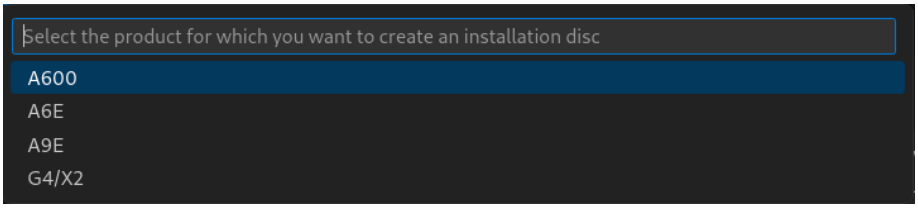


図 5.17 対象製品を選択する

無事に生成された場合、コンソールに以下のログが出力されます。

```
/home/atmark/.vscode/extensions/atmark-techno.armadillo-base-os-development-environment-1.6.0/
shell/desc/make_installer_usb.desc のバージョンを 1 から 2 に変更しました。
Enter pass phrase for /home/atmark/mkswu/swupdate.key: ❶
/home/atmark/mkswu/make_installer_usb.swu を作成しました。
To create Armadillo installer on USB memory install /home/atmark/mkswu/make_installer_usb.swu in
Armadillo
 *  Terminal will be reused by tasks, press any key to close it.
```

図 5.18 make-installer.swu 生成時のログ

❶ パスワードの入力を求められますので、初期化用 swu を生成したときと同じパスワードを入力します

/home/atmark/mkswu ディレクトリ内に make-installer.swu が作成されます。

セキュリティーの観点から、インストールディスクによるインストール実行時に以下のファイルを再生成しております：

- /etc/machine-id (ランダムな個体識別番号)
- /etc/abos_web/tls (ABOS-Web の https 鍵)
- /etc/ssh/ssh_host_*key* (ssh サーバーの鍵。なければ生成しません）ABOS 3.19.1-at.3 以降

Node-RED ではポート番号 1880 を使用して、ネットワーク上からアクセスして開発します。しかし、量産時にポートを開けておくと外部からアクセスできてしまう危険性があります。そのため、`make-installer.swu` を使用してインストールディスクイメージを生成する場合は、このポートを閉じます。

5.4.5.1. Armadillo に USB メモリを挿入

Armadillo に電源を投入し、インストールディスクを保存するために USB メモリを挿入してください。

USB メモリは vfat もしくは ext4 形式でフォーマットし、空き容量が 10GB 以上のものを使用してください。Armadillo-X2 への USB メモリのマウントは不要です。

インストールディスクイメージは `installer.img` という名前で保存します。すでに同名のファイルが存在する場合は上書きされます。

5.4.5.2. インストールディスク作成用 SWU を ABOS Web からインストール

「5.4.4. 初期設定用 SWU を ABOS Web からインストール」 で実行したように、今度は生成した `make-installer.swu` をインストールします。

ABOS Web のトップページから、"SWU インストール"をクリックすると、「図 5.19. SWU インストール」の画面に遷移します。実行時は ABOS Web 上に「図 5.20. make-installer.swu インストール時のログ」ようなログが表示されます。

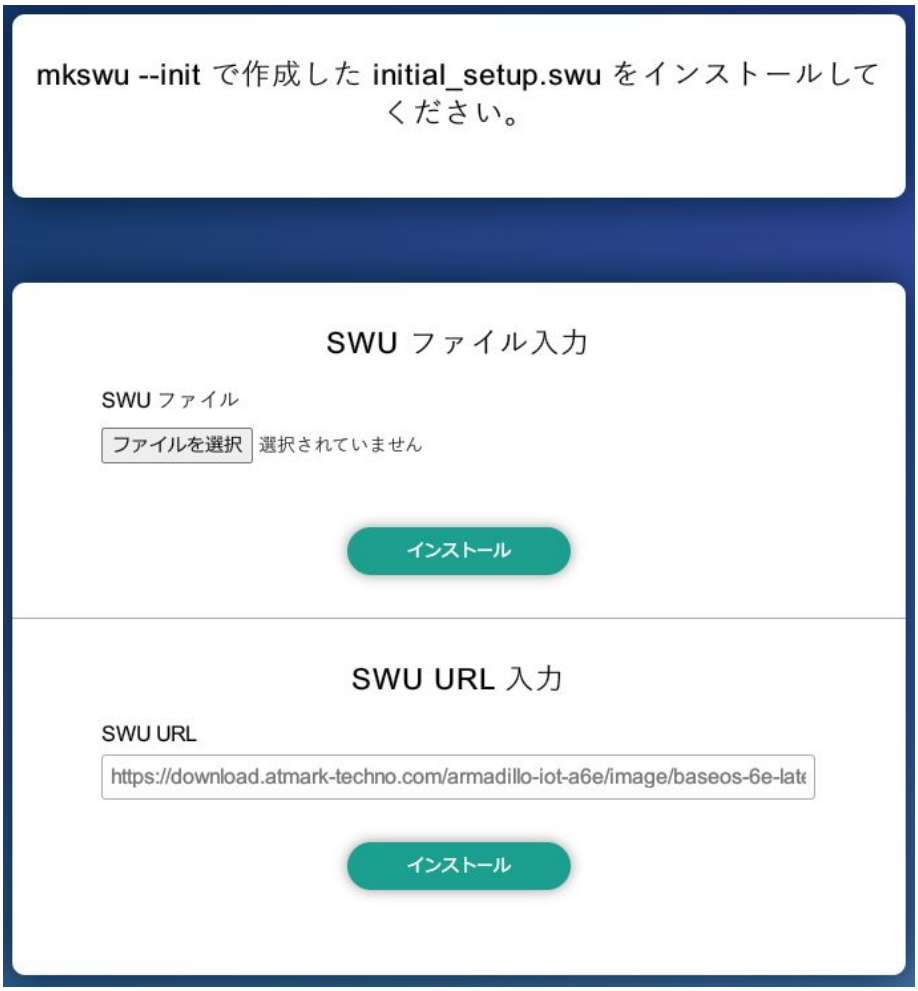


図 5.19 SWU インストール

生成した make-installer.swu を選択します。

```
make_installer_usb.swu をインストールします。
SWU アップロード完了

SWUpdate v2023.05_git20231025-r0

Licensed under GPLv2. See source distribution for detailed copyright notices.

[INFO ] : SWUPDATE running : [main] : Running on iot-a6e Revision at1

[INFO ] : SWUPDATE started : Software Update started !

[INFO ] : SWUPDATE running : [install_single_image] : Installing pre_script

[INFO ] : SWUPDATE running : [read_lines_notify] : No base os update: copying current os over

[INFO ] : SWUPDATE running : [read_lines_notify] : Waiting for btrfs to flush deleted subvolumes

[INFO ] : SWUPDATE running : [install_single_image] : Installing Copying installer to USB device

[INFO ] : SWUPDATE running : [install_single_image] : Installing swdesc_command_nochroot 'podman
```

```
kill -a'

[INFO ] : SWUPDATE running : [install_single_image] : Installing swdesc_command_nochroot --stdout-info 'abos-ctrl make-installer --noprompt --output /target/mnt/installer.img'

[INFO ] : SWUPDATE running : [read_lines_notify] : Using installer image on image file.

[INFO ] : SWUPDATE running : [read_lines_notify] : Would you like to create a windows partition?

[INFO ] : SWUPDATE running : [read_lines_notify] : That partition would only be used for customization script at the end of

[INFO ] : SWUPDATE running : [read_lines_notify] : install, leave at 0 to skip creating it.

[INFO ] : SWUPDATE running : [read_lines_notify] : Custom partition size (MB, [0] or 16 - 364): 0

[INFO ] : SWUPDATE running : [read_lines_notify] : Checking and growing installer main partition

[INFO ] : SWUPDATE running : [read_lines_notify] : Resize device id 1 (/dev/loop0p1) from 513.00MiB to max

[INFO ] : SWUPDATE running : [read_lines_notify] : Copying boot image

[INFO ] : SWUPDATE running : [read_lines_notify] : Copying rootfs

[INFO ] : SWUPDATE running : [read_lines_notify] : Copying appfs

[INFO ] : SWUPDATE running : [read_lines_notify] : At subvol app/snapshots/volumes

[INFO ] : SWUPDATE running : [read_lines_notify] : At subvol app/snapshots/boot_volumes

[INFO ] : SWUPDATE running : [read_lines_notify] : At subvol app/snapshots/boot_containers_storage

[INFO ] : SWUPDATE running : [read_lines_notify] : Trying to shrink the installer partition...

[INFO ] : SWUPDATE running : [read_lines_notify] : Shrinking the installer partition...

[INFO ] : SWUPDATE running : [read_lines_notify] : Cleaning up and syncing changes to disk...

[INFO ] : SWUPDATE running : [read_lines_notify] : Installer updated successfully!

[INFO ] : SWUPDATE running : [read_lines_notify] : -rwxr-xr-x 1 root root 687.0M Jan 23 15:12 /target/mnt/installer.img

[INFO ] : SWUPDATE running : [install_single_image] : Installing post_script

[INFO ] : SWUPDATE running : [read_lines_notify] : Removing unused containers

[INFO ] : SWUPDATE running : [read_lines_notify] : Command 'command podman rm -a -f' output:

[INFO ] : SWUPDATE running : [read_lines_notify] : 9f4f64ec1926d17e75de4060dac4a448e66ca3d9535c408f632e4e2de4bafa4f

[INFO ] : SWUPDATE running : Installation in progress

[INFO ] : SWUPDATE successful ! SWUPDATE successful !

[INFO ] : No SWUPDATE running : Waiting for requests...
```

```
swupdate exited

インストールが成功しました。
```

図 5.20 make-installer.swu インストール時のログ

完了後、USB メモリを抜いてください。もし、エラーが出た場合は Armadillo-X2 の電源を再投入してやり直してください。

5.4.6. USB メモリ上にインストールディスクイメージを生成

無事に生成が完了した場合、USB メモリ上に `installer.img` が保存されています。この `installer.img` を microSD カードに書き込むことでインストールディスクを作成することができます。動作確認については「5.5. インストールディスクの動作確認を行う」をご参照ください。これで、VS Code を使用してインストールディスクを生成する方法については終了です。

5.5. インストールディスクの動作確認を行う

生成したインストールディスクの動作確認を実施するには、開発に使用した Armadillo 以外の個体が必要になります。また、インストール先の Armadillo の eMMC 内のデータは上書きされて消えるため、必要なデータは予めバックアップを取っておいてください。

開発したシステムをインストールディスクにする手順を無事に終了している場合、USB メモリ中に `installer.img` が存在します。microSD カードに`installer.img` を書き込んで、インストールディスクを作成します。

5.5.1. インストールディスクを作成する

5.5.1.1. Windows PC 上でインストールディスクを作成する

Windows PC の場合、[Win32 Disk Imager Renewal] を使用して img ファイルを microSD カードに書き込みます。[Win32 Disk Imager Renewal] は以下からダウンロードしてください。

- Win32 Disk Imager Renewal リリースページ

  Win32 Disk Imager Renewal リリースページ [https://github.com/dnobori/DN-Win32DiskImagerRenewal/releases/]

最新版の EXE ファイルをダウンロードして起動します。以下に img ファイルの書き込み手順について示します。

1. [image File] に installer.img ファイルを指定します。
2. [Device] には microSD が挿入されているスロットを指定します。

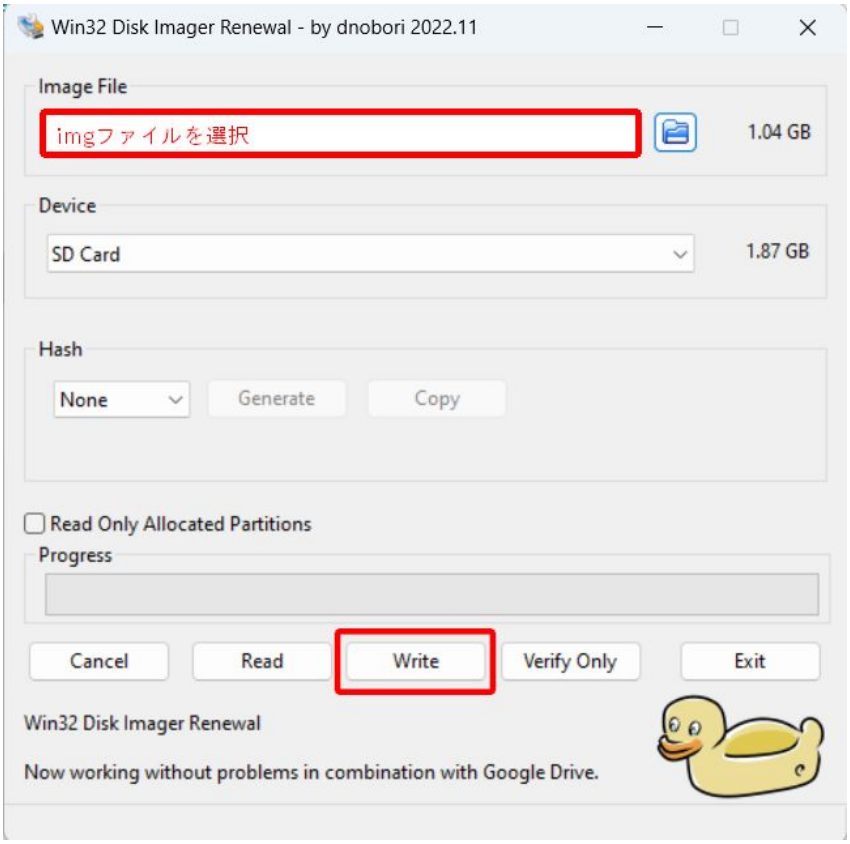


図 5.21 Win32 Disk Imager Renewal 設定画面

3. [write] ボタンを押すと警告が出ますので、問題なければ[はい]を選択して書き込みます。
4. 書き込み終了ダイアログが表示されたらインストールディスクの作成は完了です。

5.5.1.2. Linux PC 上でインストールディスクを作成する

Linux PC の場合、以下のように microSD カードに書き込むことができます。sd[X] の [X] は microSD のドライブを指定してください。

```
[PC ~]$ sudo dd if=installer.img of=/dev/sd[X] bs=1M oflag=direct status=progress
```

図 5.22 インストールディスクを作成する

書き込みが終了したらインストールディスクの作成は完了です。

5.5.2. インストールディスクを使用する

1. インストールディスクを Armadillo-X2 の microSD に挿入します。microSD スロットの場所については「3.3. インターフェースレイアウト」を参照してください。
2. Armadillo-X2 の JP1 ジャンパーをショート(SD ブートに設定)します。JP1 ジャンパーの場所については「3.3. インターフェースレイアウト」を参照してください。
3. Armadillo-X2 の AC アダプターをコンセントに繋ぎ電源を投入すると、20 分程度でインストールが完了します。

4. インストールが完了すると電源が切れて、LED4 が消灯します。
5. Armadillo-X2 の AC アダプターをコンセントから抜き、続いてジャンパーと microSD カードを外してください。
6. 10 秒以上待ってから再び電源を投入すると、Armadillo-X2 が起動します。

5.6. アップデート用 SWU の生成手法

開発終了後の Node-RED について、フローの修正などが必要な場合、修正したフローを製品に適用する必要があります。その方法として SWUpdate を推奨します。SWUpdate を用いると必要なソフトウェアのみをアップデートすることが可能です。以下に SWUpdate で使用する SWU の生成方法について記します。

- 更新用フローの取得
- ATDE 上でアップデート用 SWU の生成
- アップデート用 SWU のインストール

この機能を使用するには、「5.3. 量産時のイメージ書き込み手法」 の内容を完了している必要があります。

5.6.1. 更新用フローの取得

Node-RED からフローを取得します。メニューから [書き出し] を選択します。

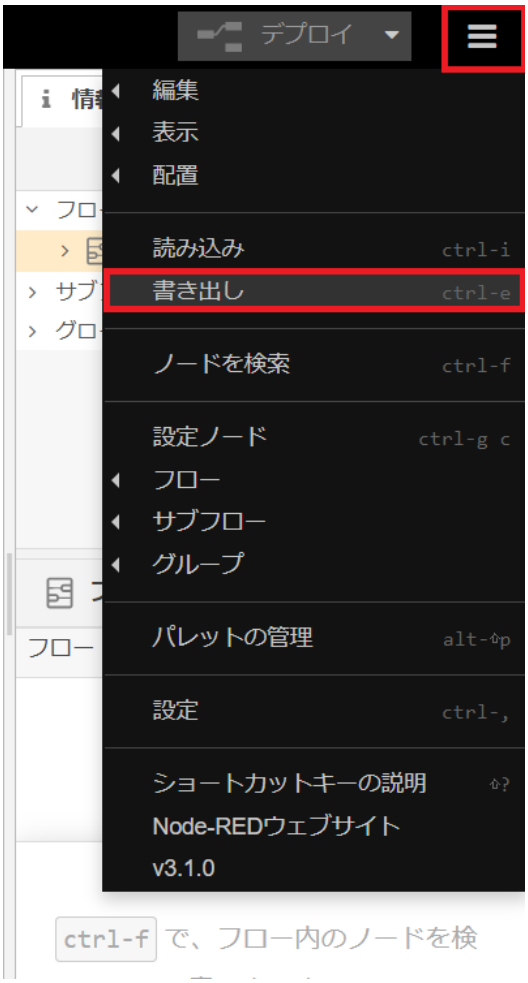


図 5.23 フローの書き出し

書き出し対象が [現在のタブ] の場合、クリップボードに含まれるノードの一覧が表示されます。問題なければ右下にある[ダウンロード]ボタンを押してダウンロードします。

図 5.24 フローのダウンロード

flows.json がダウンロードされたことを確認してください。

5.6.2. ATDE 上でアップデート用 SWU を生成

ATDE を起動します。以下のディレクトリに SWU 生成に必要な desc ファイルが置かれたディレクトリをコピーします。コピーしたディレクトリにダウンロードしたフロー flows.json を配置します。

```
[ATDE ~]$ cp -r /usr/share/mkswu/examples/node-red/ ~/mkswu/node-red/
[ATDE ~]$ cp flows.json ~/mkswu/node-red/
```

図 5.25 フローの配置

以下のコマンドを実行して SWU を生成します。

```
[ATDE ~]$ mkswu --update-version ~/mkswu/node-red/update_flows.desc ❶
/home/atmark/mkswu/node-red/update_flows.desc のバージョンを 1 から 2 に変更しました。
[ATDE ~]$ mkswu ~/mkswu/node-red/update_flows.desc -o update_flows.swu
Enter pass phrase for /home/atmark/mkswu/swupdate.key: ❷
update_flows.swu を作成しました。
```

図 5.26 SWU の生成

❶ SWU のバージョンの更新を行います。

❷ 証明書の鍵のパスワードを入力します。

成功すると現在のディレクトリに update_flows.swu が生成されます。

5.6.3. アップデート用 SWU の適用

「5.6.2. ATDE 上でアップデート用 SWU を生成」 で生成した update_flows.swu を製品に適用します。「5.4.4. 初期設定用 SWU を ABOS Web からインストール」 を参照して Armadillo へインストールしてください。

インストールに成功したら Armadillo は自動的に再起動します。

初期設定用 SWU を適用していない状態で update_flows.swu を適用した場合「FAILURE ERROR : Signature verification failed」というエラーが発生します。その場合は、「5.4.3. VS Code を使用した初期設定用 SWU の生成」 と 「5.4.4. 初期設定用 SWU を ABOS Web からインストール」 を参照して、初期設定用 SWU の生成と Armadillo へのインストールを行ってください。

初期設定用 SWU を適用後、改めて update_flows.swu を適用してください。

Armadillo-X2 Node-RED 開発ガイド

改訂履歴

バージョン	年月日	改訂内容
1.0.0	2023/12/26	・初版発行
1.1.0	2024/01/29	・「4.4.5. 外部プログラムを実行する」 を追加
1.2.0	2024/02/02	・「5. 量産する」 を追加
1.3.0	2024/03/26	・「5.6. アップデート用 SWU の生成手法」 を追加
1.4.0	2024/04/23	・「4.6. AWS へデバイス情報を送信するフローを作成する」 を追加
1.4.1	2024/08/28	・「図　5.14. initial_setup.swu　初回生成時の各種設定」 に initial_setup のパスワード制限を追記 ・誤記・わかりにくい文章の修正
1.5.0	2024/09/25	・「5.4.1. 仮想環境のセットアップ」 を Oracle VM VirtulaBox を使用したものに変更 ・誤記修正
1.5.1	2024/10/30	・誤記修正
1.5.2	2024/11/27	・「4.6.8. AWS IoT ポリシーをデバイスにアタッチするフローの作成」 の「[exec queue] ノードのプロパティを編集します。」に記載されている AmazonRootCA1.pem のパスを修正
1.5.3	2025/01/29	・「4.6. AWS へデバイス情報を送信するフローを作成する」 の各種フローをステータスコードが取得できるように修正

Armadillo-X2 Node-RED 開発ガイド
Version 1.5.3
2025/01/29