Armadillo Base OS 開発ガイド

Version 1.4.3
2024/06/26

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 Base OS 開発ガイド

株式会社アットマークテクノ

製作著作 © 2021-2024 Atmark Techno, Inc.

Version 1.4.3
2024/06/26

目次

1. はじめに 10
1.1. 本ドキュメントを読むことで習得できること 10
1.2. アイコンについて 10
1.3. 表記について 11
1.3.1. フォント 11
1.3.2. コマンド入力例 11
1.3.3. アイコン 11
1.4. サンプルソースコード 12
1.5. ライセンス 12
2. 本ドキュメントについて 13
2.1. 本ドキュメントで実施する作業 13
2.2. 本ドキュメントの成果物 15
3. 準備 17
3.1. 必要なもの 17
3.2. 事前準備 17
3.2.1. 開発環境の準備 17
3.2.2. 機械学習向け開発環境の準備 機械学習 17
3.2.3. 接続物の準備 18
4. 基本的な開発の流れ 20
4.1. 機械学習を用いたアプリケーションの開発の流れ 機械学習 20
5. 仕様を検討・決定する 22
5.1. Armadillo Base OS におけるアプリケーション仕様検討 22
5.2. podman コンテナイメージの選定 22
5.2.1. コンテナを用いたシステムの設計 補足情報 22
5.3. ネットワーク構成の検討 補足情報 23
5.4. ハードウェアの拡張 補足情報 23
5.5. 機械学習を用いたアプリケーションについて 機械学習 23
5.5.1. 学習の手間 23
5.5.2. 推論の速度と精度 23
5.5.3. 開発時の技術的負債 24
6. サンプルアプリケーションの作成 25
6.1. サンプルアプリケーションの詳細 25
6.1.1. 詳細な動作 25
6.2. 推論モデルの作成 機械学習 29
6.2.1. 教師データの用意 29
6.2.2. 推論モデルの生成 39
6.3. podman コンテナを作成する 39
6.3.1. 開発前の準備 39
6.3.2. サンプルコンテナをビルド 40
6.3.3. コンテナ内に入る 40
6.4. アプリケーション実行の準備をする 41
6.5. アプリケーションを作成する 41
6.5.1. ファイル構成 42
6.5.2. コンテナ内のファイルの編集 42
6.5.3. アプリケーションの動作確認 43
6.5.4. podman コンテナの保存 44
6.5.5. podman コンテナのエクスポート 45
6.5.6. podman コンテナとアプリケーションの自動実行 46
6.6. 動作確認 47

6.7. USB カメラから映像を取得するように変更 49
6.7.1. 撮影対象の parameter.json を作成 50
6.7.2. パラメータファイルの配置 53
6.7.3. .conf ファイルの設定 53
6.7.4. 自動実行の確認 54
7. 量産時のイメージ書き込み手法 56
8. 量産用インストールディスクの作成 57
8.1. インストールディスクとは 57
8.2. インストールディスクを作成する準備を行なう 57
8.2.1. Armadillo Base OS の更新 57
8.2.2. SWUpdate の初期設定 58
8.2.3. パスワードの確認と変更 58
8.2.4. 開発中のみ使用していたコンテナイメージの削除 59
8.2.5. 開発したコンテナイメージを tmpfs に移行する 60
8.3. 開発したシステムをインストールディスクにする 61
8.3.1. インストール時に任意のシェルスクリプトを実行する 63
8.4. インストールディスクの動作確認 64
9. 量産用 swu イメージの作成 66
9.1. swu イメージとは 66
9.2. SWUpdate で量産を行なう際の流れ 66
9.3. 初回アップデート用 swu イメージの作成 66
9.4. 書き込むイメージの準備 67
9.5. desc ファイルの記述 67
9.5.1. イメージ書き込み時にログを残したい場合 68
9.6. swu イメージの作成 69
9.7. SWUpdate の実行とテスト 70
9.7.1. SWUpdate の実行 70
9.7.2. SWUpdate 後のテスト 71
10. 製造・量産する 72
10.1. ユーザ製品の製造・量産の流れ 72
10.2. クローンインストールディスクを用いてイメージ書き込みを行なう 73
10.2.1. インストール時に任意の処理を行なう 73
10.2.2. インストールを実行する 77
10.3. SWUpdate を用いてイメージ書き込みを行なう 78
10.3.1. SWUpdate の処理内容を変更したい場合 78
10.3.2. インストールを実行する 78
11. Appendix 79
11.1. SWUpdate を用いてソフトウェアをアップデートする 79
11.1.1. SWUpdate による初回アップデート 79
11.2. 作成したコンテナを他の Armadillo に組み込む 81
11.2.1. podman load でコンテナイメージを組み込む 81
11.2.2. SWUpdate でコンテナイメージを組み込む 82
11.3. Device Tree を変更しハードウェアを拡張する 84
11.4. コンテナデザインパターン 84
11.4.1. 複数コンテナで処理を行いコンテナ間でデータを共有する 84
11.4.2. 複数コンテナ間でデータを共有し、クラウドにアップロードする 86
11.4.3. コンテナを増やす 87
11.4.4. ホストコマンドを実行する 88
11.5. 機械学習と NPU の使いどころ 92
11.5.1. 機械学習を活用できる範囲 92
11.5.2. NPU でできること 92
11.5.3. 機械学習と NPU を使うことの課題 93
11.5.4. 機械学習以外の方法 93

11.6. ネットワーク 95
11.6.1. Armadillo-IoT ゲートウェイ G4 のネットワーク概要 95
11.6.2. podman のネットワークの仕組み 97
11.6.3. ネットワーク構成例とその設定方法 101
11.6.4. 量産製造時のネットワーク設定 113
11.6.5. 運用開始後に設定を変更する 114

図目次

1.1. 機械学習関連情報アイコン 11
1.2. 補足情報アイコン 11
1.3. クリエイティブコモンズライセンス 12
2.1. Armadillo Base OS 上でのアプリケーション開発の流れ 13
2.2. 詳細なアプリケーション開発の流れ 14
2.3. サンプルアプリケーションの動作 15
3.1. Armadillo-IoT ゲートウェイ G4 への周辺機器の接続 18
3.2. 接続後のイメージ 19
4.1. 開発時のイメージ 20
6.1. サンプルアプリケーションの詳細な動作の流れ 26
6.2. 元画像 27
6.3. 物体検出した箇所をもとに画像を切り出し 27
6.4. 円を検出 28
6.5. 画像の枠が円に外接するようにリサイズ 28
6.6. 直線を検出 29
6.7. Google Images Download のインストール 30
6.8. Google Images Download の実行 31
6.9. Google Images Download でダウンロードしたファイル 31
6.10. 画像ファイルを連番でリネーム 31
6.11. labelImg をインストールする 32
6.12. Linux 環境における labelImg 実行例 32
6.13. labelImg の起動 32
6.14. 出力形式の変更 34
6.15. アノテーション元の選択 35
6.16. アノテーション情報の保存先の選択 35
6.17. 作業効率化のための設定 1 36
6.18. 作業効率化のための設定 2 37
6.19. ラベル付け例 38
6.20. アノテーション情報ファイルを確認 38
6.21. label_map.pbtxt 作成例 39
6.22. podman 関連ファイルを eMMC 上に保存する 40
6.23. Dockerfile のダウンロード 40
6.24. サンプルコンテナのイメージのビルド 40
6.25. コンテナ内に入る 40
6.26. コンテナへパッケージをインストール 41
6.27. サンプルアプリケーションのダウンロードと展開 41
6.28. サンプルアプリケーションを構成するファイル 42
6.29. ATDE から Armadillo へファイルを送信 42
6.30. コンテナ内にエディタをインストールする 43
6.31. サンプルアプリケーションの動作確認 43
6.32. Armadillo-IoT ゲートウェイ G4 への周辺機器の接続 44
6.33. weston を終了 44
6.34. podman コンテナから抜ける 44
6.35. 起動中の podman コンテナを確認する 44
6.36. 停止中の podman コンテナを確認する 45
6.37. podman コンテナの変更を保存する 45
6.38. podman コンテナイメージを外部ストレージに出力する 46
6.39. sample_container.conf 作成例 46
6.40. sample_container.conf のダウンロード 46
6.41. podman_start の実行例 47

6.42. sample_container.conf の永続化 47
6.43. コンテナの停止・削除 47
6.44. サンプル動画に対して物体検出 48
6.45. サンプル動画内のアナログメーターを読む 49
6.46. config_param.py のダウンロードと起動 50
6.47. アナログメーターを正面から捉えたサンプル画像(image.png) 50
6.48. アナログメーターの中心をクリック 51
6.49. 各スケールの位置をクリック 52
6.50. 各スケールの数値を入力 52
6.51. 各スケールの数値を入力 53
6.52. /var/app/rollback/volumes/assets の作成 53
6.53. parameter.json を配置 53
6.54. USB カメラを使用する sample_container.conf 作成例 54
6.55. USB カメラの映像からアナログメーターの値を取得 55
6.56. sample_container.conf の永続化 55
8.1. 任意のファイルパスを/etc/swupdate_preserve_files に追記する 58
8.2. Armadillo Base OS をアップデートする 58
8.3. パスワードを変更する 59
8.4. 作成済みコンテナ一覧を表示する 59
8.5. コンテナを削除する 59
8.6. コンテナイメージ一覧を表示する 60
8.7. R/O が false のイメージを削除する 60
8.8. R/O が true のイメージを削除する 60
8.9. podman のデータの保存先を tmpfs に変更する 61
8.10. 開発完了後のシステムをインストールディスクイメージにする 61
8.11. インストールディスク使用時のログ 64
8.12. インストールに失敗した Armadillo に出力されるメッセージ 64
9.1. Armadillo に書き込みたいソフトウェアを ATDE に配置 67
9.2. desc ファイルの記述例(Armadillo-IoT ゲートウェイ G4 の場合) 67
9.3. desc ファイルの記述例(Armadillo-IoT ゲートウェイ A6E の場合) 68
9.4. ログを残す際の desc ファイルの記述例(Armadillo-IoT ゲートウェイ G4 の場合) 68
9.5. ログを残す際の desc ファイルの記述例(Armadillo-IoT ゲートウェイ A6E の場合) 69
9.6. mkswu コマンドの実行 70
9.7. 各イメージを USB メモリに配置 70
9.8. USB メモリをアンマウント 70
10.1. Windows PC にインストールディスクを接続する 74
10.2. ip_config.txt の内容 75
10.3. IP アドレスの確認 76
10.4. allocated_ips.csv の内容 76
10.5. インストールログを保存する 76
10.6. インストールログの中身 77
11.1. mkswu の取得 79
11.2. mkswu の初期設定を行う 79
11.3. swu イメージの配置 81
11.4. コンテナイメージファイルをインポートする 82
11.5. 作業用ディレクトリの作成と書き込むファイルのダウンロード 82
11.6. usb_container_sample.desc 作成例 82
11.7. swu イメージの作成 83
11.8. アップデート用ファイル群の配置 83
11.9. センサから読み取ったデータを PC に出力するシステム構成図 85
11.10. センサからデータを読み取りクラウドに出力するシステム構成図 86
11.11. 複数のセンサからデータを読み取るシステム構成図 88
11.12. TensorFlow Lite のインストール 93

11.13. ONNX Runtime のインストール 93
11.14. ArmNN のインストール 93
11.15. OpenCV のインストール 94
11.16. Julius のインストール 94
11.17. PIL のインストール 94
11.18. コンテナ内に入る 94
11.19. コンテナへ Julius をインストール 94
11.20. ディクテーションキットのダウンロード 95
11.21. 音声認識を実行 95
11.22. nmcli のコマンド書式 96
11.23. nmtui 起動後の画面 97
11.24. ブリッジモードの構成 98
11.25. ブリッジモード時のネットワークインターフェース一覧 (ホスト OS) 98
11.26. ブリッジモード時のネットワークインターフェース一覧 (コンテナ内) 99
11.27. host モードの構成 100
11.28. host モード時のネットワークインターフェース一覧 (ホスト OS) 100
11.29. host モード時のネットワークインターフェース一覧 (コンテナ内) 100
11.30. ネットワークの構成 102
11.31. 有線 LAN インターフェースの設定を行う (nmcli) 103
11.32. nmtui を起動する 104
11.33. nmtui 起動後の画面 104
11.34. nmtui コネクション選択画面 105
11.35. nmtui コネクション種別選択画面 105
11.36. nmtui コネクション入力画面 106
11.37. nmtui IPv4 設定画面 106
11.38. nmtui Route 設定画面 107
11.39. nmtui デフォルトゲートウェイ無効化 107
11.40. nmtui コネクション Activate 画面 108
11.41. nmtui コネクション Deactivate 後の画面 108
11.42. 有線 LAN インターフェース設定を永続化する 108
11.43. 3G/LTE 回線の接続設定を行う 109
11.44. ファイアーウォールの設定を行う 109
11.45. ファイアーウォールの設定を永続化する 109
11.46. IPv6 の無効化を行う 110
11.47. ネットワークデバイスの状態を確認する 110
11.48. IP アドレスの設定を確認する 110
11.49. ルーティングテーブルの内容を確認する 111
11.50. ファイアーウォールの設定を確認する 111
11.51. .conf ファイル設定例 (IPv4 の場合) 112
11.52. .conf ファイル設定例 (IPv6 の場合) 112
11.53. コンテナ起動方法 (IPv4 の場合) 113
11.54. コンテナ起動方法 (IPv6 の場合) 113
11.55. コンテナ内に入る (IPv4, IPv6 共通) 113
11.56. ATDE 上へのコネクションファイルのコピーとパーミッションの設定を行う 114
11.57. network-setting.desc ファイルを作成する 115
11.58. コネクションファイルを含む swu イメージを作成する 115

表目次

1.1. 使用しているフォント 11
1.2. 表示プロンプトと実行環境の関係 11
6.1. 画像収集に使用できるツール 30
6.2. 物体検出アノテーションに使用できるツール 32
6.3. labelImg でよく使用するショートカットキー一覧 37
7.1. インストールディスクと SWUpdate によるソフトウェア書き込みの比較 56
8.1. インストール中に実行される関数 63
9.1. SWUpdate の進捗と LED3 の状態の対応表 71
11.1. 機械学習の活用範囲 92
11.2. 種別・通信規格とネットワークデバイス 95
11.3. ネットワークに関連する代表的な Capability 101
11.4. ネットワークのアドレス情報 102
11.5. 有線 LAN インターフェース の設定情報(nmcli) 103
11.6. 有線 LAN インターフェース の設定情報(nmtui) 104

1. はじめに

本ドキュメントは、これから Armadillo Base OS を搭載した製品を用いたアプリケーションを設計・開発する方に向けて、Armadillo-IoT ゲートウェイ G4 を例として基本的な開発の流れについての情報を提供します。また、Armadillo-IoT ゲートウェイ G4 の特徴である NPU(Neural Processing Unit)を活かして、実際に機械学習を用いたサンプルアプリケーションを開発しつつ、Armadillo Base OS 搭載製品の開発方法について紹介します。その際に使用する物体検出モデル [1]も、既存のモデルをベースに学習を行う転移学習という手法で作成し、学習したモデルをアプリケーションに組み込みます。

なお、一般的なアプリケーション開発には企画、要件定義、設計、実装、検証、運用保守などといったステップがありますが、本ドキュメントはその全てを網羅するものではありません。本ドキュメントでは、Armadillo Base OS 搭載製品特有の手順や留意点に焦点を当てて説明していきます。説明のない箇所は、Armadillo Base OS であることによる特殊性が影響しない部分であるため、一般的なアプリケーション開発手法や、お客様の会社やグループ内のルールに則って開発を進めてください。

1.1. 本ドキュメントを読むことで習得できること

- Armadillo Base OS 上での開発手法
  - podman イメージ及びコンテナの作成と運用
  - ATDE を用いた各種開発手法
- 機械学習による物体検出アプリケーションを作る際の手法
  - TensorFlow を用いた既存の物体検出モデルの転移学習の方法
  - TensorFlow の SavedModel 形式 [2]のモデルを TFLite 形式 [3]に変換する方法
  - アプリケーションへの物体検出モデル組み込み
- Armadillo Base OS を利用した量産製造の手法
  - 開発完了後のイメージでインストールディスクを作成する方法
  - インストールディスクを用いたイメージ書き換え中に任意のシェルスクリプトを実行する方法

1.2. アイコンについて

具体的な機械学習を使用したサンプルアプリケーションの開発例を通して、Armadillo Base OS の開発手法を紹介します。本ドキュメント内で「図 1.1. 機械学習関連情報アイコン」のアイコンがある箇所は、機械学習について記述した内容であり、それ以外の箇所は機械学習に使用/不使用関係なく Armadillo Base OS に共通した内容になっています。機械学習を用いないアプリケーション開発についての情報だけを見たい方は、「図 1.1. 機械学習関連情報アイコン」のアイコンのない箇所を参照してください。

[1]「機械学習における、入力があったときにそれに何らかの評価を加えて出力値とするもの」を本ドキュメント内では「モデル」及び「推論モデル」と呼称します。
[2]TensorFlow Guide SavedModel 形式の使用: https://www.tensorflow.org/guide/saved_model?hl=ja
[3]TensorFlow Guide TensorFlow Lite: https://www.tensorflow.org/lite/guide?hl=ja

図 1.1 機械学習関連情報アイコン

また、「図 1.2. 補足情報アイコン」に示すアイコンが付いている箇所については、サンプルアプリケーションの開発には利用しませんが Armadillo Base OS を利用した開発においてよく行うことになる手順について紹介していますので、サンプルアプリケーション以外の開発時にお役立てください。

図 1.2 補足情報アイコン

1.3. 表記について

1.3.1. 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

1.3.2. 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
[ATDE ~/]$	ATDE 上の一般ユーザで実行
[armadillo /]#	Armadillo 上 Linux の root ユーザで実行
[container /]#	Podman コンテナ内で実行

1.3.3. アイコン

本ドキュメント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。

1.4. サンプルソースコード

本ドキュメントで紹介するサンプルソースコードは、 https://download.atmark-techno.com/armadillo-iot-g4/example/armadillo-base-os-dev-guide/ からダウンロードできます。

1.5. ライセンス

本ドキュメントで紹介するサンプルソースコードは MIT ライセンス [4]の下に公開します。

ただし、 コンテナイメージ [https://download.atmark-techno.com/armadillo-iot-g4/example/armadillo-base-os-dev-guide/abos-dev-guide-v1.0.0.tar]及び コンテナ内のソフトウェア [https://download.atmark-techno.com/armadillo-iot-g4/example/armadillo-base-os-dev-guide/read_meter.tar.gz]に含まれる、アナログメーターのサンプル動画 [5]及び、本ドキュメント内における当該の動画のスクリーンショットについては、クリエイティブコモンズの表示-継承 4.0 国際ライセンスの下に提供されています。

本ドキュメントは、クリエイティブコモンズの表示-継承 4.0 国際ライセンスの下に公開しています。

本ドキュメント及び、アナログメーターのサンプル動画のクリエイティブコモンズのライセンスの内容は https://creativecommons.org/licenses/by-sa/4.0/ でご確認ください。

図 1.3 クリエイティブコモンズライセンス

[4]http://opensource.org/licenses/mit-license.php
[5]kaggle Pressure Gauge Reader Data: https://www.kaggle.com/juliusgrassme/pressure-gauge-reader-data

2. 本ドキュメントについて

本ドキュメントでは、Armadillo Base OS 搭載製品のライフサイクルの一部(設計、開発、量産製造)について実例を交えながら紹介します。

2.1. 本ドキュメントで実施する作業

本ドキュメントでは、Armadillo Base OS 上で実際にサンプルアプリケーションの作成を行います。

「図 2.1. Armadillo Base OS 上でのアプリケーション開発の流れ」に、Armadillo Base OS 上でのアプリケーション開発の基本的な流れについて示します。本ドキュメントにおいても、この流れに沿って開発を行います。

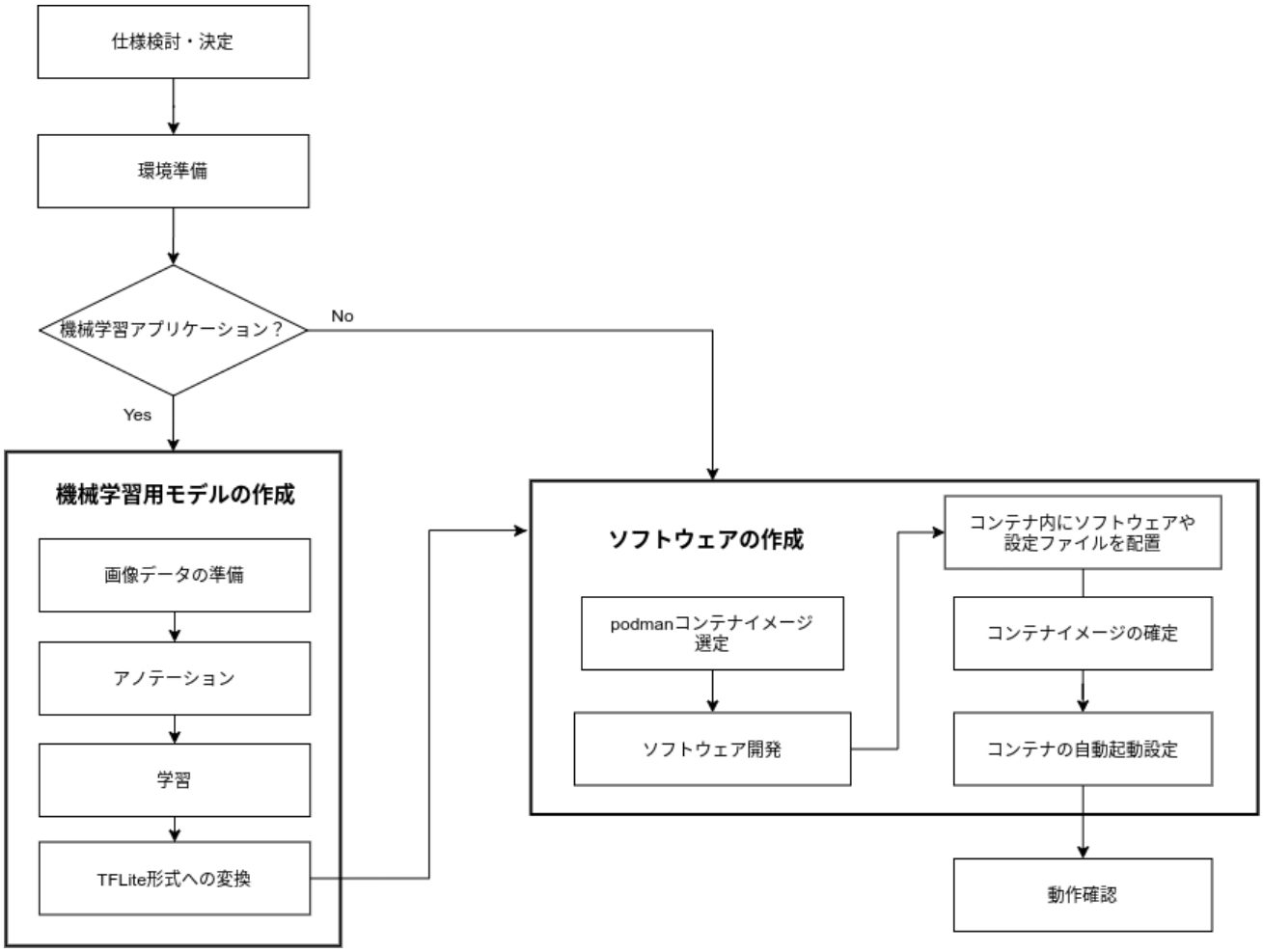


図 2.1 Armadillo Base OS 上でのアプリケーション開発の流れ

Armadillo Base OS における、アプリケーションの開発時の流れをより詳細にした図を「図 2.2. 詳細なアプリケーション開発の流れ」に示します。図中の破線内は、本サンプルアプリケーションのように機械学習をアプリケーションに組み込む場合に必要な手順であり、主に使用する推論モデルのチューニングを行っています。それ以外の機械学習を用いないアプリケーションの場合には、破線内の処理は必要ありません。

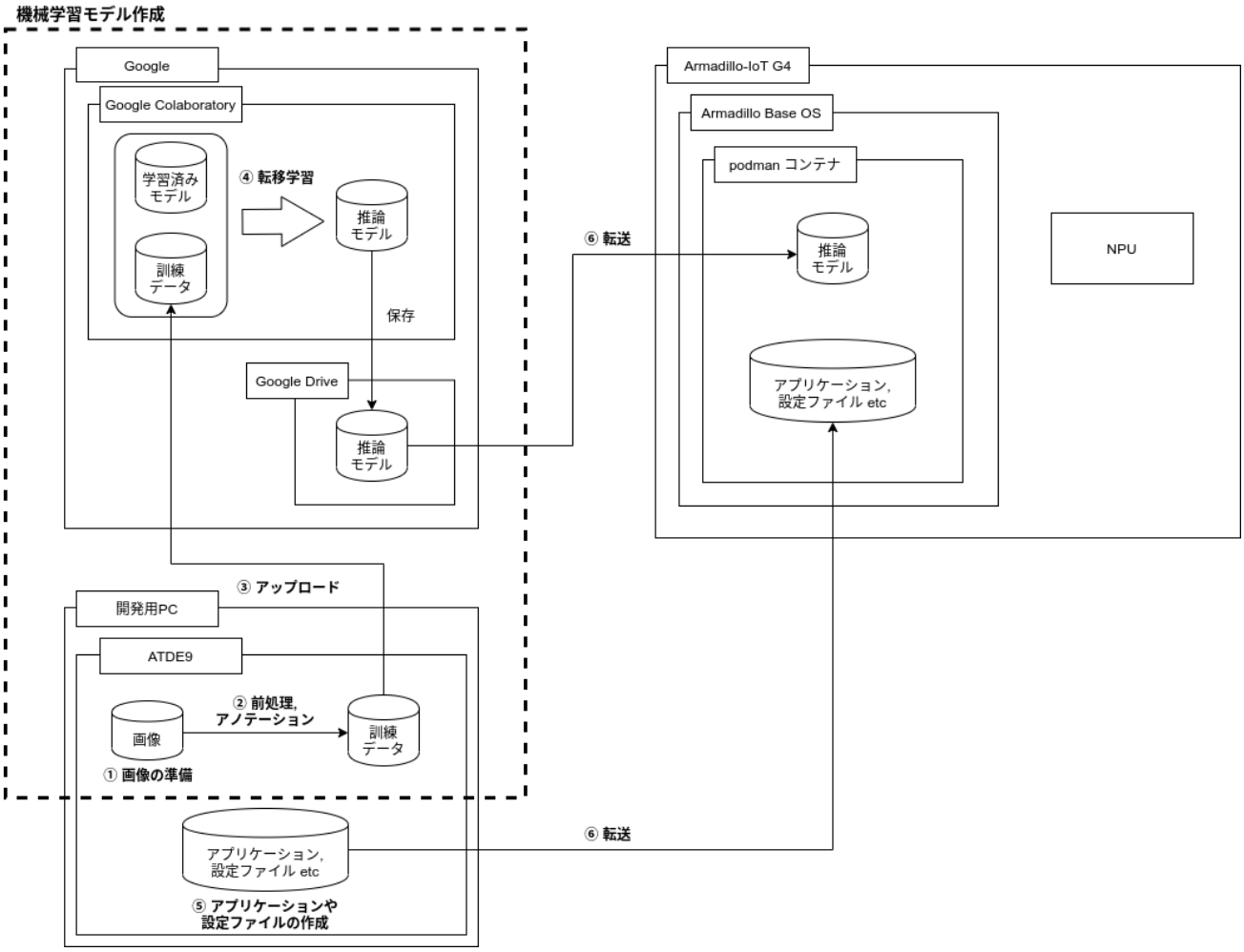


図 2.2 詳細なアプリケーション開発の流れ

本サンプルアプリケーション作成の大まかな流れは、機械学習部分を含むため、以下の通りです。

1. ATDE 上に転移学習用の画像データを用意する
2. 画像データに対してラベル名や、その位置などを含む情報を付与(アノテーション)して訓練データとする
3. 訓練データを Google Colaboratory にアップロードする
4. Google Colaboratory 上で学習済みモデルと組み合わせて転移学習を行う
5. 推論モデルを使用したアプリケーション本体や、付随する設定ファイル等を ATDE 上で作成する
6. 4 で作成した推論モデルと、5 で作成したアプリケーション等を Armadillo に転送する

各手順については本書内で詳しく紹介していきます。

開発後には量産製造に向けて、Armadillo Base OS の機能を利用した Armadillo へのイメージ書き込みの手法について示します。

2.2. 本ドキュメントの成果物

本ドキュメントの手順を踏むことで最終的に、丸型アナログメーター自動読み取りサンプルアプリケーションが作成できます。出来上がるサンプルアプリケーションの動作について「図 2.3. サンプルアプリケーションの動作」に示します。

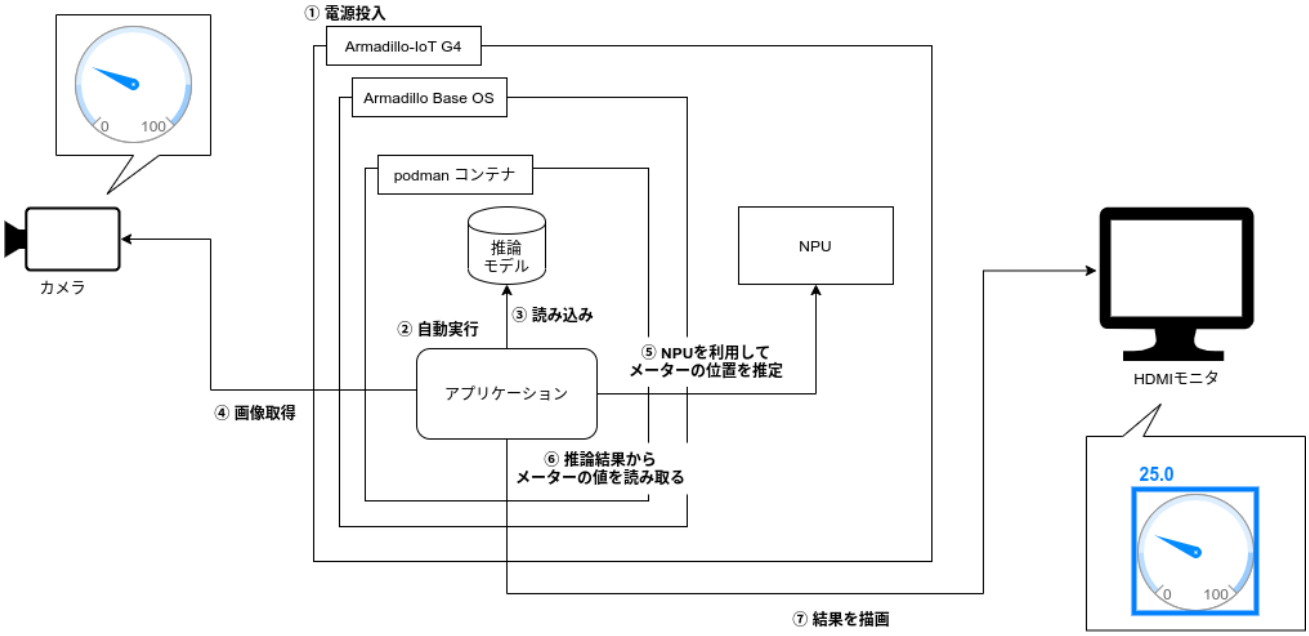


図 2.3 サンプルアプリケーションの動作

アプリケーションの挙動としましては以下の通りです。

1. Armadillo-IoT ゲートウェイ G4 に電源を投入する
2. podman コンテナとアプリケーションが自動的に起動する
3. 初期化処理として推論モデルをロード
4. USB カメラから画像を取得
5. NPU を利用して画像データからアナログメーターの位置推定を行う
6. 5 の結果をもとにアナログメーターが指す値を読み取る
7. 6 で得た結果を画像と共に HDMI モニタに表示
8. 4〜7 を繰り返す

USB カメラから取得した画像内のどこにアナログメーターが存在するかを検知するために、機械学習による物体検出を利用しています。また、アナログメーターの読み取りには OpenCV による画像解析を使用しています。

詳細な仕様については、「6.1. サンプルアプリケーションの詳細」で紹介します。

また、開発完了後の Armadillo を複製できる量産製造用のインストールディスクを作成できます。インストールディスクの実体は microSD カードであり、開発と製造間での受け渡しと製造時のイメージ書き込みが容易に実現できます。

3. 準備

ここからは「2.2. 本ドキュメントの成果物」で紹介したサンプルアプリケーションの作成前に、必要な準備事項について紹介していきます。

3.1. 必要なもの

本ドキュメントでは以下のものを使用します。

- Armadillo-IoT ゲートウェイ G4 開発セット [https://armadillo.atmark-techno.com/armadillo-iot-g4/AGX4500-C00D0]
- 以下を満たす開発用 PC
  - インターネットに接続可能
  - (必要ならば)VMware が動作可能
- USB カメラ(なくても可)
- HDMI モニタ
- microHDMI-HDMI 変換ケーブル
- Google アカウント

サンプルアプリケーションの映像入力は Armadillo 内のサンプル動画でも代用できますので、USB カメラはなくても試すことができます。

3.2. 事前準備

3.2.1. 開発環境の準備

Armadillo-IoT ゲートウェイ G4 製品マニュアルの「 4. Armadillo の電源を入れる前に [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.11.0/ch04.html]」を参考に Armadillo 向け開発環境である ATDE を整備し、シリアルコンソール経由で Armadillo-IoT ゲートウェイ G4 を操作できるように準備してください。

3.2.2. 機械学習向け開発環境の準備 機械学習

本ドキュメントで作成するサンプルアプリケーションでは、既存の学習済みモデルを自分のやりたいこと向けに学習し直す「転移学習」を行います。転移学習を行うことで、学習に必要な教師データ [1]の数が減ったり、学習にかかる時間の短縮につながったりするなどのメリットがあります。転移学習の詳細については TensorFlow の転移学習紹介ページ [https://www.tensorflow.org/js/tutorials/transfer/what_is_transfer_learning?hl=ja]も参照してください。

注意点としまして、Armadillo-IoT ゲートウェイ G4 に搭載されている GPU/NPU は、学習済みモデルを用いて推論を行うことはできますが、学習することには向いていません。そのため、GPU が利用で

[1]教師データとは、機械学習の教師あり学習において、ニューラルネットワークに予め与えられる例題と答えについてのデータを指します。

きる別な環境で学習を行い、その結果得られた推論モデルを Armadillo に送って推論を行うことになります。

お使いの開発用 PC に機械学習開発に十分な GPU が搭載されているならば機械学習の開発環境とすることが可能ですが、搭載されていない場合は、Google が提供している Google Colaboratory [https://colab.research.google.com/notebooks/welcome.ipynb?hl=ja]というサービスで代用できます。

Google Colaboratory は、Google アカウントを持っていれば無料で利用できる、ブラウザ上で Jupyter Notebook [2]形式で Python や Shell コマンドなどを記述・実行できるサービスです。特徴として以下のような点が挙げられます。

- Web ブラウザアプリケーションとして動作し、ローカルに環境構築が不要
- 無料で GPU を利用することができる
- Google Drive を用いたファイルの共有や保存を行うことができる

本ドキュメントでは Google Colaboratory を使用して推論モデルの作成を行います。Google Colaboratory を使用するためには Google アカウントを作成しておく必要があります。Google Colaboratory には無料版と有料版があり、無料版には連続利用可能時間などのいくつかの制約がありますが、本ドキュメントのサンプル開発は無料版で十分です。

3.2.3. 接続物の準備

「図 3.1. Armadillo-IoT ゲートウェイ G4 への周辺機器の接続」、「図 3.2. 接続後のイメージ」に示すようにように、各種接続物を Armadillo-IoT ゲートウェイ G4 に接続してください。

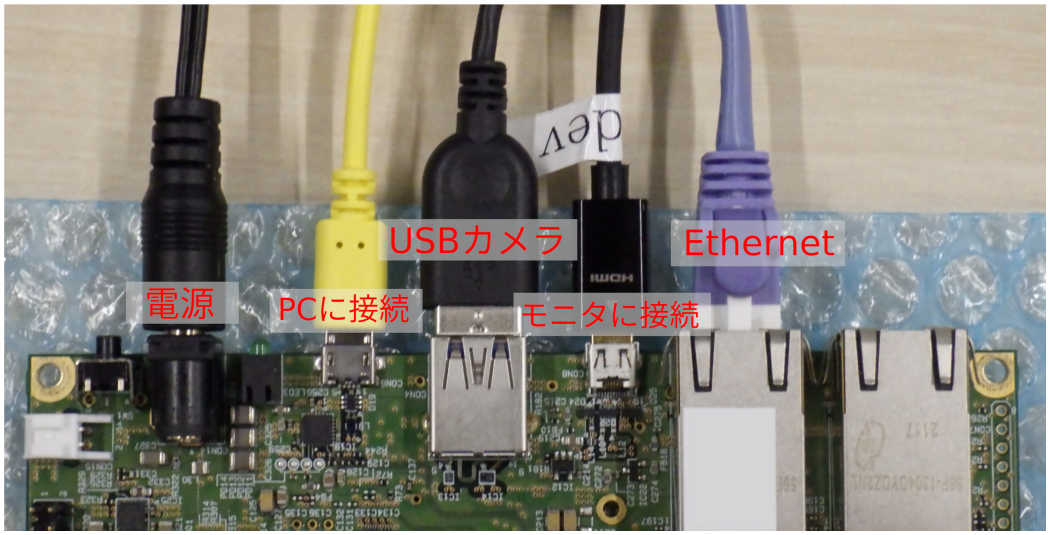


図 3.1 Armadillo-IoT ゲートウェイ G4 への周辺機器の接続

[2]Jupyter Notebook: https://jupyter.org/

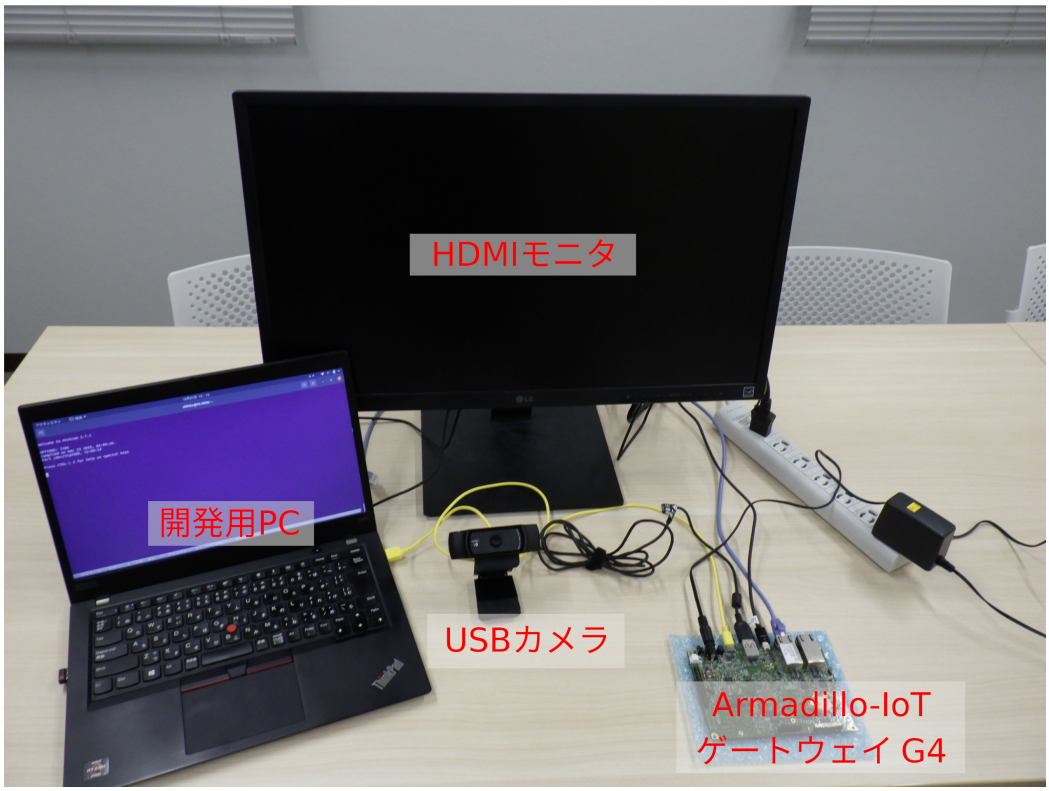


図 3.2 接続後のイメージ

4. 基本的な開発の流れ

Armadillo Base OS 上でのアプリケーション開発の大まかな流れについては、「図 2.1. Armadillo Base OS 上でのアプリケーション開発の流れ」に示した通りです。

Armadillo を用いた開発は、作業用 PC(または Armadillo 向けの開発環境が揃った仮想マシンである ATDE)内でアプリケーションの開発を行い、Armadillo へデプロイする方法と、Armadillo 内で開発を行う 2 パターンがあります。どちらの手法でも問題ありませんが、本ドキュメントでは基本的に作業用 PC 内に ATDE を立ち上げ、その中で開発を行っていきます。ATDE を用いた開発時のイメージを「図 4.1. 開発時のイメージ」に示します。

Armadillo Base OS において、ユーザーアプリケーションは podman コンテナ上で動作させることを前提としています。Armadillo 上で podman コンテナを構築して、そのコンテナ内に開発したアプリケーションが動作する環境を整えることでアプリケーションを実行することができるため、使用時に Armadillo Base OS をあまり意識する必要はありません。

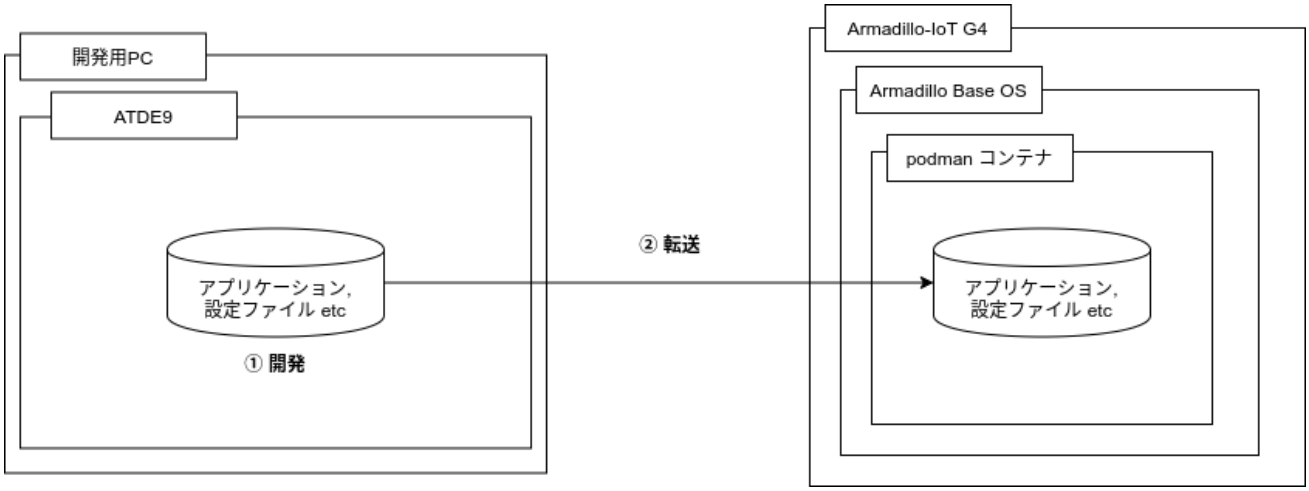


図 4.1 開発時のイメージ

各手順について詳細は「6. サンプルアプリケーションの作成」内で説明します。

4.1. 機械学習を用いたアプリケーションの開発の流れ 機械学習

「図 2.1. Armadillo Base OS 上でのアプリケーション開発の流れ」に示しているように、今回作成するような機械学習を用いたアプリケーション開発においては、仕様検討までは機械学習を用いないアプリケーションと同様の手順ですが、実現したいことに合わせた推論モデルを生成する手順が別途必要になります。

今回は既存の物体検出モデルをベースに転移学習するため、以下のような手順となっています。あくまでも一例ですので、実現したいことによって手順は異なることに注意してください。

1. 教師データの元となる画像を準備
2. アノテーションして教師データを作成
3. 学習

4. TFLite 形式への変換
5. ソフトウェア開発(以降は機械学習を用いないアプリケーションと同様)

各手順についての詳細は「6.2. 推論モデルの作成 機械学習」で説明します。

5. 仕様を検討・決定する

Armadillo Base OS 上での開発に限った話ではありませんが、作成するアプリケーションの仕様の検討を行います。本ドキュメントでは機械学習を用いたアプリケーションを開発するため、この章では主に機械学習に重きを置いて説明します。

5.1. Armadillo Base OS におけるアプリケーション仕様検討

「4. 基本的な開発の流れ」でも説明したとおり、Armadillo Base OS では基本的にユーザーアプリケーションを podman コンテナ上で実行します。そのため、実行環境として Armadillo Base OS を意識する必要はありません。従来の Armadillo 製品を使用した開発を行ったことがある方であれば、その際の製品開発の仕様を引き継ぐことも可能です。

この後の「5.2. podman コンテナイメージの選定」でも紹介していますが、実現したいことに合った実行環境をコンテナイメージとして自由に選択できるため、Armadillo Base OS におけるアプリケーション仕様の検討時の制限は少ないはずです。

5.2. podman コンテナイメージの選定

作成するアプリケーションの実行環境としてどのコンテナイメージをベースとするかは最終的なイメージサイズにも関わりますので、検討しておく必要があります。

podman は、同じくコンテナを扱えるソフトウェアである Docker [https://www.docker.com/]と基本的に互換性があります。そのため、podman で使用するコンテナイメージとして Docker Hub [https://hub.docker.com/]に存在するイメージも使用することができます。

コンテナイメージサイズをなるべく小さくしたいならば alpine、Python で書いたアプリケーションを手軽に実行したいならば Debian ベースの python:3.x-slim-bullseye など、作成するソフトウェア仕様に沿ってベースイメージを決定します。

また、Armadillo-IoT ゲートウェイ G4 において映像出力を行う、もしくは NPU を使用して推論を行う場合は、そのために必要なライブラリがありますので、 アットマークテクノが提供している Debian ベースの podman コンテナイメージ [https://armadillo.atmark-techno.com/resources/software/armadillo-iot-g4/container]を使用することをお勧めします。

5.2.1. コンテナを用いたシステムの設計 補足情報

使用するコンテナイメージの選定と共に、コンテナをシステムの中でどのように用いるかを検討しておく必要がある場合があります。本サンプルアプリケーションのように単一のコンテナのみで完結するシステムであれば難しくはないですが、管理・運用のために機能毎にコンテナを複数個に分けてコンテナ間で通信し合う場合であったり、コンテナの外側である Armadillo Base OS のコマンドやリソースへアクセスしながら使用したりする場合などには、どのようなシステム設計をすべきか難しい場合があります。

本ドキュメントでは、Armadillo Base OS 上でコンテナを使用したシステムを構築する際によくあるパターンや注意点について「11.4. コンテナデザインパターン」にまとめてありますので参考にしてください。

5.3. ネットワーク構成の検討 補足情報

作成するアプリケーションがネットワークに接続するものであった場合、Armadillo Base OS や podman、コンテナのネットワークについて把握した上で検討する必要があります。

本ドキュメントでは、 podman のネットワークの概要や具体的なネットワーク構成・その設定方法を「11.6. ネットワーク」に記載しています。構成検討・設定時の参考にしてください。

5.4. ハードウェアの拡張 補足情報

サンプルアプリケーションでは行いませんが、Armadillo の拡張インターフェースに I2C や SPI などのインターフェースを持つ外部デバイスを接続する場合、対象のピンの機能やパラメータを変更しなければならない場合があります。

ハードウェア拡張時のパラメータ変更の方法について、詳しくは「11.3. Device Tree を変更しハードウェアを拡張する」を参照してください。

5.5. 機械学習を用いたアプリケーションについて 機械学習

機械学習を用いたアプリケーションを開発する場合で、今までに機械学習を用いた開発を行ったことがない場合は、少し検討が必要かもしれません。機械学習という言葉が独り歩きして、「機械学習は簡単になんでもできる」など間違った認識で開発が行われる場合があるためです。

機械学習アプリケーションは、それ以外のソフトウェアでは実現できないようなことを実現できる強力なものであり、それだけに魅力的ですが、万能ではありません。当然デメリットも存在することを忘れてはいけません。機械学習アプリケーションのデメリットを知り、そのデメリットをシステムが許容できるかを検討する必要があります。場合によっては機械学習を使わないで、別な手段で問題を解決するという選択をすることも重要になります。

以下に機械学習を用いたアプリケーション開発のデメリットを挙げます。

5.5.1. 学習の手間

自分で一から推論モデルを作成しようとすると、膨大な量の教師データが必要になります。物体検出サンプルならば何千枚といった画像データを用意し、それに手作業でラベル付けと物の位置を指定してはじめて教師データとなります。場合によっては、全ての画像データに対してリサイズや色変換など前処理をする必要がある場合もあり、学習には多くの手間がかることに注意してください。

本ドキュメントの物体検出サンプルでは転移学習を用いることで用意する教師データの数を減らしています。

5.5.2. 推論の速度と精度

推論の速度は、推論モデルと推論を行うハードウェアに依存します。ハードウェアは今回は Armadillo で固定なので問題ではありませんが、推論モデルによっては一度の推論に長い時間がかかる場合もあり、速度を要するシステムでは使えないということもあります。速度は一度推論モデルを作ってみなければわからず、基本的に精度とトレードオフですので、高い精度のモデルほど速度は遅いです。

推論の精度は、推論モデルに依存します。学習時にどれだけの数の訓練データを入力したか、訓練データや特徴量は正しいものだったか、過学習は起こっていないかなど、気にするべき箇所は多々あります。さらに、精度はどこまで学習を積んでも 100%になることはありません。速度や精度は推論モデルのチューニングによっては改善が期待できるかもしれませんが、限界があります。

推論モデルの速度と精度については予め実現したいラインを設定し、そこに近づけられるように試行錯誤と検証を重ねる必要があります。最終的に出来上がる推論モデルの性能に合わせて、システム全体で推論の速度や誤検知を許容できる仕様にしておく必要があります。

5.5.3. 開発時の技術的負債

機械学習はドキュメントやテストコードがなかったり、いきあたりばったりの設計が残ってしまうなど、技術的負債が多くなる傾向があります。それは、以下のような機械学習を用いたシステム構築の難しさが原因としてあります。

- 前述の通り試行錯誤回数が多くなる
- 確率的な処理があるために自動テストが難しい
- 長期運用しているとトレンドの変化などで入力の傾向が変化する
- 実験コードやパラメータが残りやすい

6. サンプルアプリケーションの作成

ここまでで Armadillo Base OS での開発環境の整備や、開発手順がわかったところで、いよいよサンプルアプリケーションを作成していきます。

初めに、USB カメラを使用しないサンプルアプリケーションを含む podman コンテナを作成し、その後それをベースに USB カメラに対応した podman コンテナを作成します。

ここからはサンプルアプリケーション作成手順を各ステップ毎に説明していきますが、 こちら[https://download.atmark-techno.com/armadillo-iot-g4/example/armadillo-base-os-dev-guide/abos-dev-guide-v1.0.0.tar]から完成したアプリケーションを含む podman コンテナイメージをダウンロードできます。

podman コンテナイメージのインポート方法については、「11.2. 作成したコンテナを他の Armadillo に組み込む」を参照してください。

6.1. サンプルアプリケーションの詳細

今回作成するサンプルアプリケーションの大まかな概要は、「2.2. 本ドキュメントの成果物」で示したとおりですが、詳細について紹介します。

6.1.1. 詳細な動作

今回作成するアプリケーションの詳細な動作を「図 6.1. サンプルアプリケーションの詳細な動作の流れ」にまとめます。

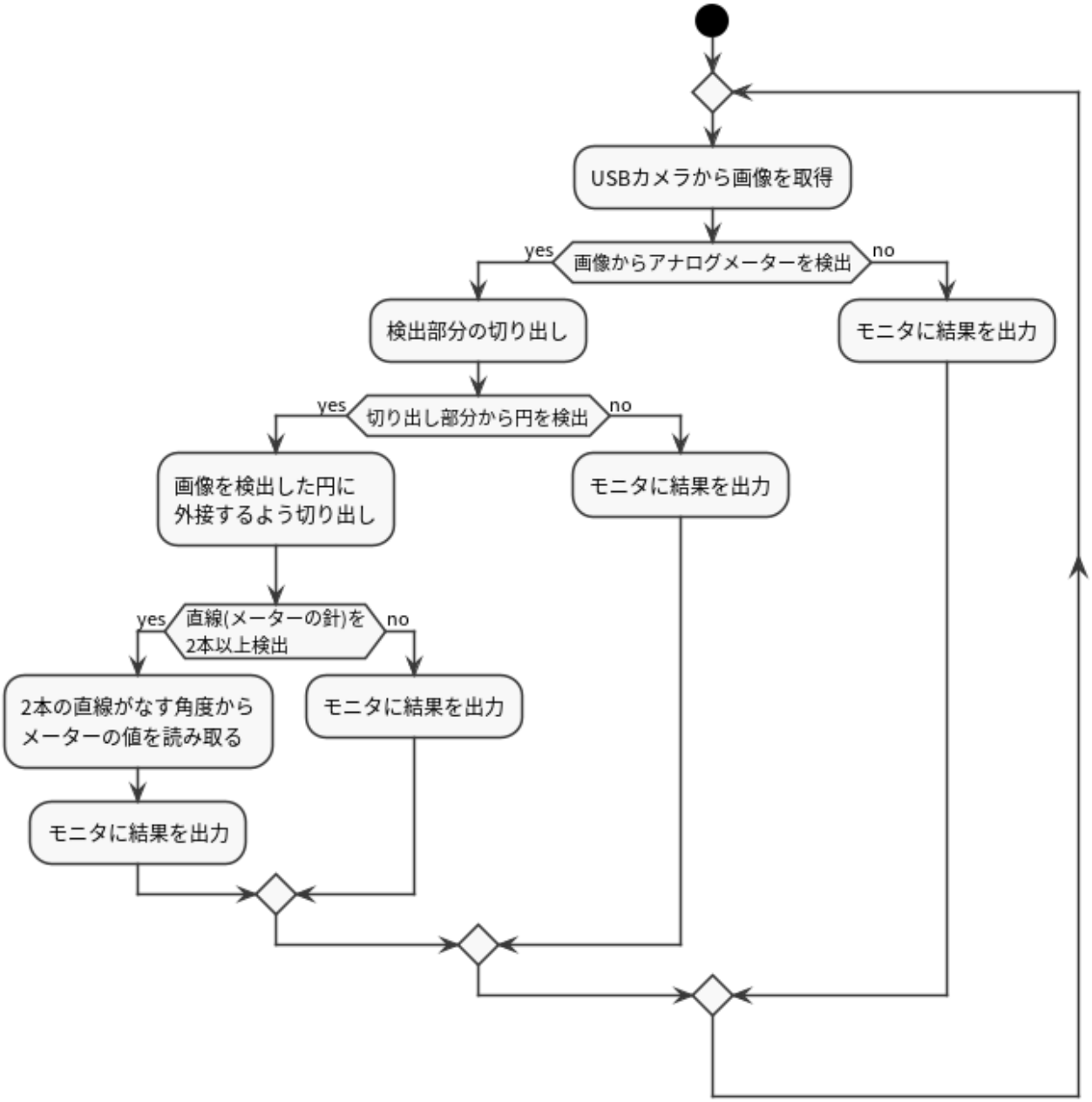


図 6.1 サンプルアプリケーションの詳細な動作の流れ

図中の各処理について説明します。

6.1.1.1. USB カメラから画像を取得

基本的に画像の操作は、Python の OpenCV ライブラリを用いて行っていきます。

初めに、USB カメラまたは任意の動画ファイルから画像を取得します。

図 6.2 元画像

6.1.1.2. 画像からアナログメーターを検出

この部分に機械学習の物体検出を用います。予めアナログメーターの画像を学習させておいた TFLite 形式の推論モデルを用いて、取得した画像に対して推論を実行し、画像内のアナログメーターの有無とその位置を検出します。

検出できたならば次の処理に移り、検出できなければ取得した画像を HDMI モニタに出力して再度画像の取得を行います。

図 6.3 物体検出した箇所をもとに画像を切り出し

6.1.1.3. アナログメーター検出箇所から円を検出

アナログメーターを検出した部分のみを切り出し、その中でハフ変換を用いて円とその中心座標を検出します。

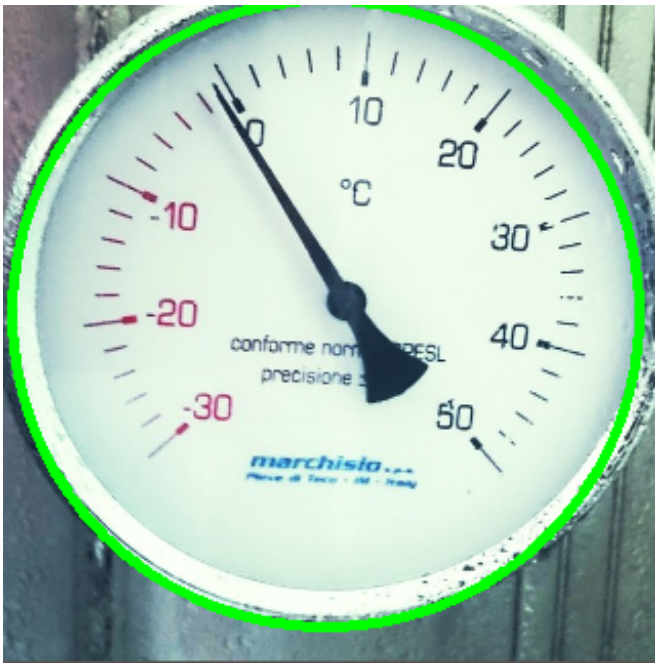


図 6.4 円を検出

検出できたならばその円に外接する四角形で元の画像を切り抜きます。これによって、物体検出による検出位置のズレや誤検出を吸収しつつ、画像内のアナログメーター部分のみを切り抜いています。

図 6.5 画像の枠が円に外接するようにリサイズ

円を検出できなければ、取得した画像を HDMI モニタに出力して再度画像の取得を行います。

6.1.1.4. メーターの読み取り

アナログメーターの針の形をなす 2 本の直線をハフ変換を用いて検出します。その 2 本の直線から針の角度を算出して、予め設定した parameter_sample.json の情報と照らし合わせてアナログメーターが示す値を求めています。

図 6.6 直線を検出

求めた値を元画像に描画した上で HDMI モニタに出力します。

直線が検出できなかった場合や、求めた値がそのアナログメーターでは表示できない値であった場合は、読み取った値は画像に描画されずに HDMI モニタに出力します。

その後、再度画像の取得に戻ります。

以上が、本サンプルアプリケーションの動作です。

6.2. 推論モデルの作成 機械学習

ここからはサンプルアプリケーションの作成手順を紹介していきます。

まずはじめに、アプリケーション本体の前に物体検出モデルを作成します。以下の作業を本ドキュメントでは説明のため ATDE 上で行っていますが、他の PC 上などでも問題ありません。ATDE 以外の環境で実行する際には、適宜各ソフトウェアの使用方法などを調べた上でご使用ください。

6.2.1. 教師データの用意

今回行う物体検出における転移学習では、既存の物体検出用の推論モデルである MobileNet SSD v2 をベースに、新たに検出させたい物が写った画像を含む教師データを用意して学習させることで、対象物を検出するモデルを効率よく作成します。

物体検出における教師データは、検出させたい物が写った画像と、それらの画像内のどこに何があるかをラベル付けしたアノテーション情報の 2 つをまとめたものを指します。

機械学習用に教師データをデータセットという形で公開/販売しているところも存在しているので、使用上のライセンスをよく確認した上でそちらを使用しても問題ありません。

本ドキュメントでは既存のデータセットは使用せず、Web 上から画像を収集して実際にアノテーションを行います。

6.2.1.1. 画像の用意

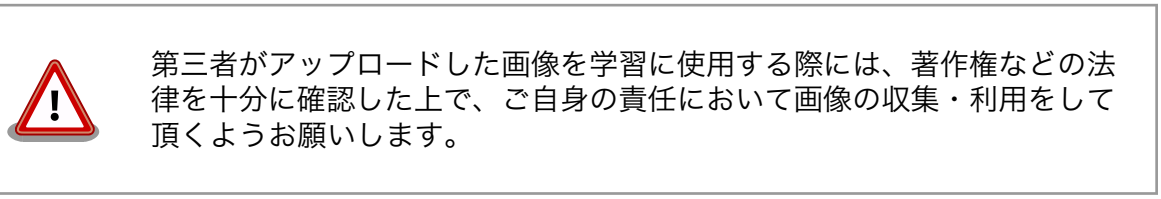
まずは教師データの元となる画像を収集します。一から推論モデルを学習するのであれば数千枚ほどのデータがほしいところですが、今回は転移学習を用いるため、最低 100 枚程度からある程度の精度を出せます。

「表 6.1. 画像収集に使用できるツール」に示すように、様々な機械学習向けに画像を収集できるツールが公開されているので、そちらを利用することで効率よく画像を集めることができます。

表 6.1 画像収集に使用できるツール

ツール名	概要
icrawler [https://github.com/hellock/icrawler]	Python で動作するウェブクローラのフレームワーク
Bing Image Search API [https://docs.microsoft.com/ja-jp/azure/cognitive-services/bing-image-search/overview]	Microsoft 社が無料提供している Bing 画像検索 API
Google Images Download [https://github.com/hardikvasa/google-images-download]	Google の画像検索から画像を収集できる Python スクリプト

上記の他にも、機械学習などで使用できる画像を収集できるツールや Web サイトなどが存在しているので、興味のある方は調べてみてください。



第三者がアップロードした画像を学習に使用する際には、著作権などの法律を十分に確認した上で、ご自身の責任において画像の収集・利用をして頂くようお願いします。

今回は Google Images Download を使用して、画像データを収集していきます。

まずインストールについてですが、Google Images Download は pip でインストールできます。しかし執筆時点では最新の Google 画像検索 API に対応できていませんので、有志がパッチを適用したものをインストールします。Google Images Download のインストール時に必要なパッケージも合わせてインストールします。

```
[ATDE ~]$ sudo apt update && sudo apt upgrade -y
[ATDE ~]$ sudo apt install -y python3-pip
[ATDE ~]$ git clone https://github.com/Joeclinton1/google-images-download.git gid-joeclinton
[ATDE ~]$ cd gid-joeclinton && sudo python3 setup.py install
```

図 6.7 Google Images Download のインストール

Google Images Download を使用して、gauge(アナログメーター)の画像を収集します。Google Images Download は、ダウンロードする画像の拡張子と縦横のサイズ、ダウンロードする数を指定することができます。

ここでは、Google で"gauge"と画像検索した際に得られる画像を jpg 形式で 100 枚ダウンロードします。

```
[ATDE ~/gid-joeclinton]$ cd
[ATDE ~]$ mkdir -p dataset_gauge/annotations && cd dataset_gauge ❶
[ATDE ~/dataset_gauge]$ googleimagesdownload -k "gauge" -l 100 -f jpg ❷
Item no.: 1 --> Item name = gauge
Evaluating...
Starting Download...
: (省略)
Everything downloaded!
Total errors: 2 ❸
Total time taken: 54.75878143310547 Seconds
```

図 6.8 Google Images Download の実行

❶ 作業ディレクトリを作成し、移動します。

❷ Google Images Download を実行します。

❸ ファイルによってはエラーとなるものがあり、必ずしも指定した枚数の画像を収集できるとは限りません。

Google Images Download を用いて 100 枚より多くの画像を収集する際には、別途 Google Chrome と chromedriver をインストールする必要があります。詳しくは 公式ドキュメント [https://google-images-download.readthedocs.io/en/latest/troubleshooting.html#installing-the-chromedriver-with-selenium]を参照してください。

「図 6.8. Google Images Download の実行」の実行例ならば 98 枚の gauge で画像を検索した際の結果が得られました。

```
[ATDE ~/dataset_gauge]$ ls downloads/gauge/ | wc -l
98
```

図 6.9 Google Images Download でダウンロードしたファイル

また、場合によっては壊れたファイルをダウンロードしてくる場合がありますので、その場合は適宜手動で削除してください。

その後、「図 6.10. 画像ファイルを連番でリネーム」を実行して、ファイル名をリネームしておきます。

```
[ATDE ~/dataset_gauge]$ cd downloads/gauge/
[ATDE ~/dataset_gauge/downloads/gauge/]$ ls | awk '{ printf "mv %s %06d.jpg¥n", $0, NR }' | sh
[ATDE ~/dataset_gauge/downloads/gauge/]$ ls
000001.jpg
000002.jpg
: (省略)
```

図 6.10 画像ファイルを連番でリネーム

6.2.1.2. アノテーション

画像が用意できたら、次はその画像内のどこに何が写っているかを示すラベル付け(アノテーション)を行っていきます。画像データの数によりますが、推論モデル学習の一連の流れの中で一番工数がかかる手順がこのアノテーション作業です。

「表 6.2. 物体検出アノテーションに使用できるツール」にアノテーションする際によく使われるツールをまとめました。どれも Windows/Linux 共に使用可能ですので、お好きなものをご使用ください。

表 6.2 物体検出アノテーションに使用できるツール

ツール名	概要
labellmg [https://github.com/tzutalin/labellmg]	Python+Qt で記述されたオープンソースアノテーションソフトウェア
VoTT [https://github.com/microsoft/VoTT]	Microsoft 社が提供しているオープンソースアノテーションソフトウェア
Labelbox [https://github.com/Labelbox/Labelbox]	リッチな GUI で多機能な Labelbox 社製ツール。基本無料で利用量が一定を超えると課金される点に注意

本ドキュメントでは、labellmg を使用してアノテーションを行っていきます。

labellmg は「図 6.11. labellmg をインストールする」に示すコマンドを実行することでインストールできます。

```
[ATDE ~]$ sudo pip3 install labelImg
```

図 6.11 labellmg をインストールする

labellmg を起動するには、「図 6.12. Linux 環境における labellmg 実行例」に示すコマンドを実行します。

```
[ATDE ~]$ labelImg
```

図 6.12 Linux 環境における labellmg 実行例

実行すると、「図 6.13. labellmg の起動」のような labellmg のアプリケーションウィンドウが立ち上がります。

図 6.13 labellmg の起動

まず、アノテーション情報の保存形式を指定します。今回は PascalVOC 形式で保存します。

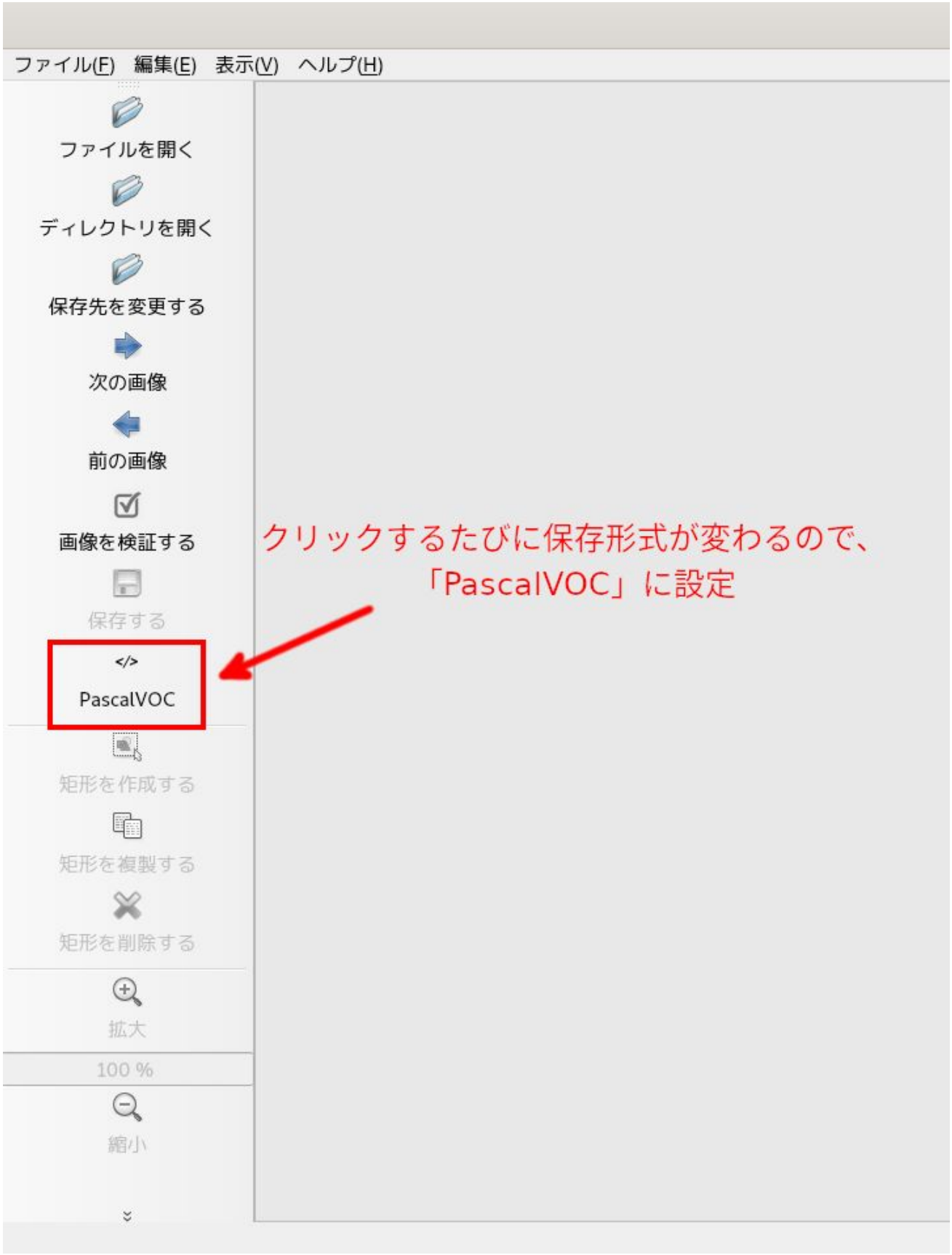


図 6.14 出力形式の変更

アノテーションを開始するために、画面左の「ディレクトリを開く」をクリックして、アノテーションしたい画像が入ったディレクトリ(今回は~/dataset_gauge/downloads/gauge/)を指定します。

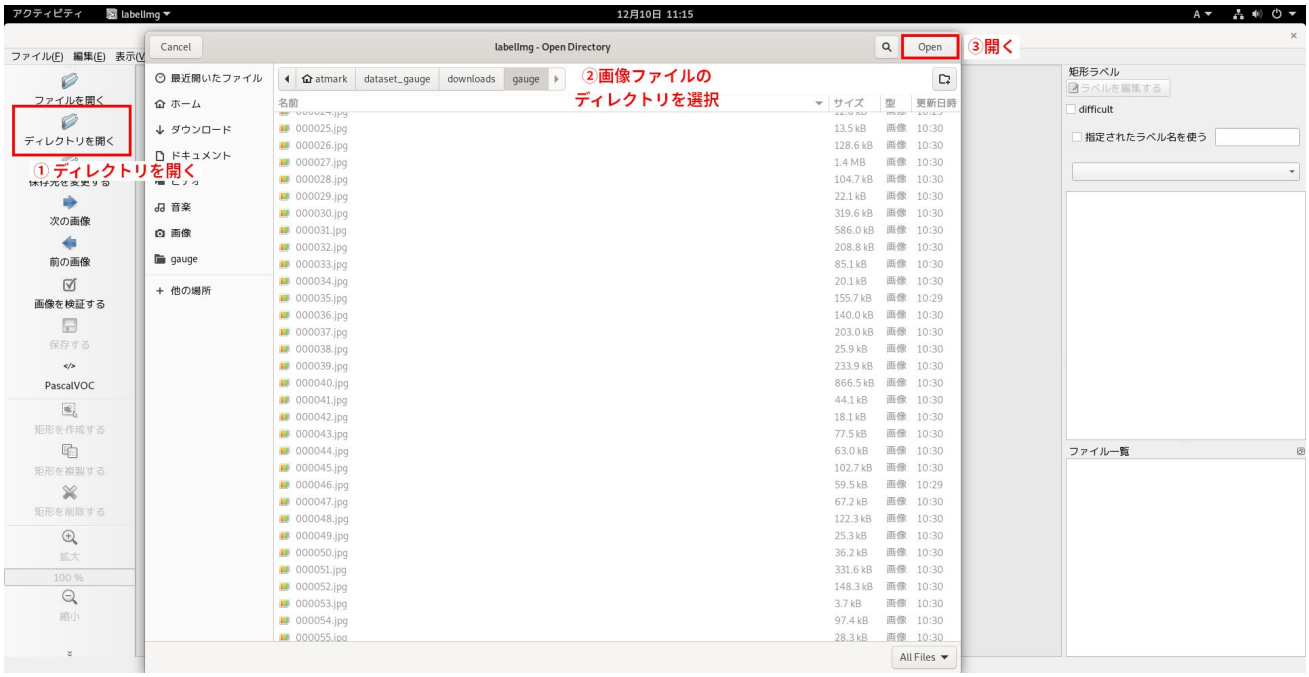


図 6.15 アノテーション元の選択

さらに、アノテーション情報をどこに保存するかを設定します。画面左の「保存先を変更する」をクリックして、アノテーション情報を保存したいディレクトリ(今回は~/dataset_gauge/annotations/)を指定します。

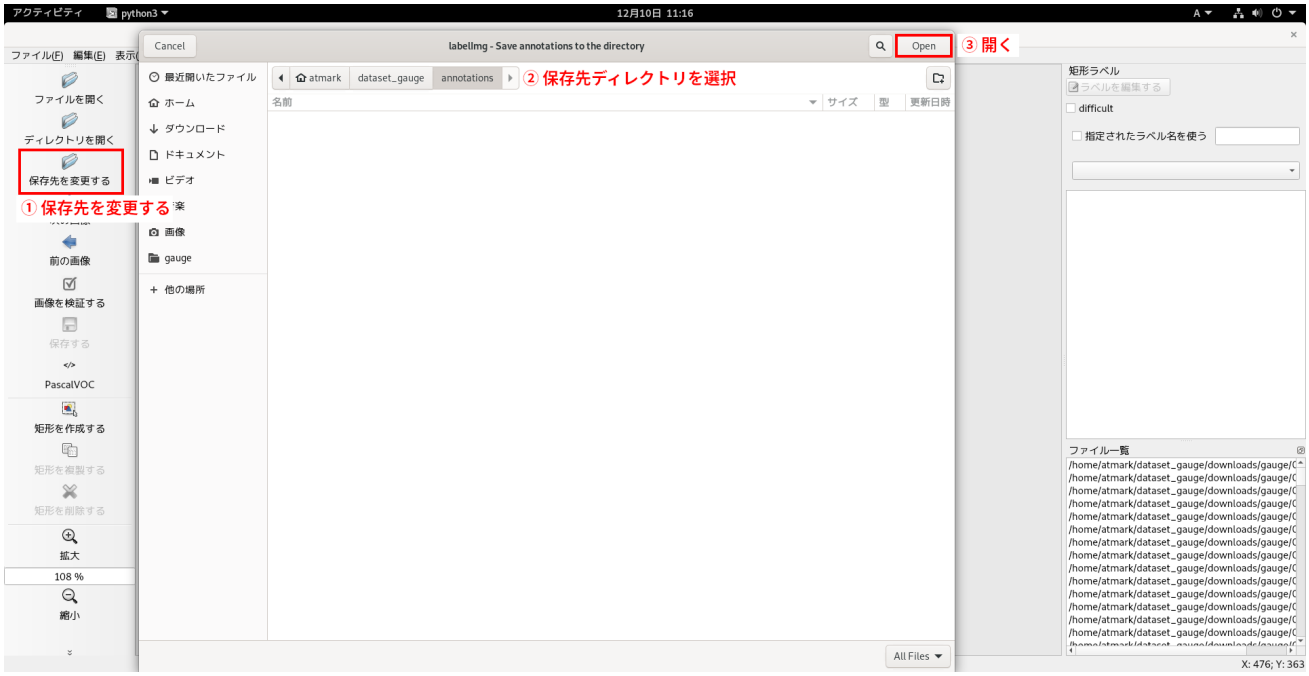


図 6.16 アノテーション情報の保存先の選択

その後作業効率化のために、画面上部のツールバーの[表示]から、以下を有効化をしておくことをお勧めします。

- 自動で保存する
- 単一のラベルだけをアノテーションする

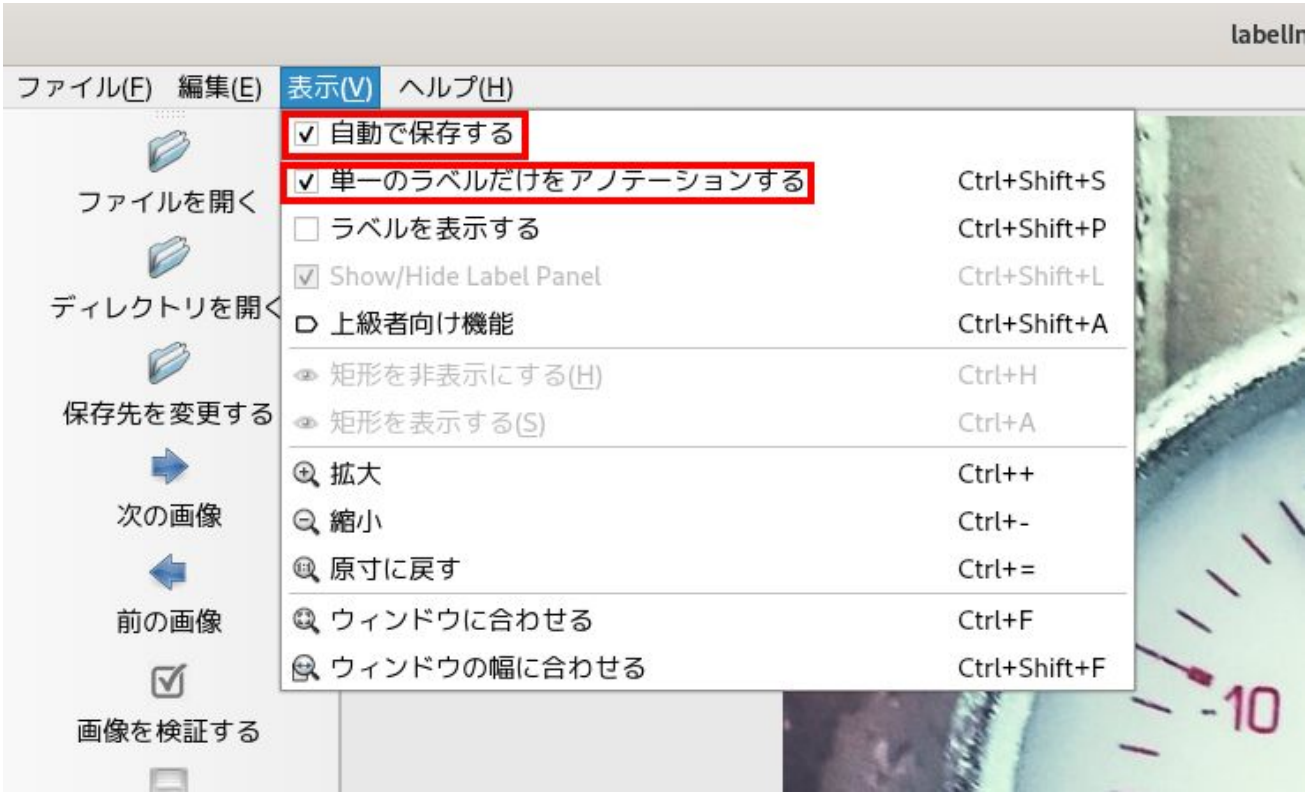


図 6.17 作業効率化のための設定 1

さらに、画面右側にて、「指定されたラベル名を使う」にチェックを入れ、ラベル名を「gauge」とします。

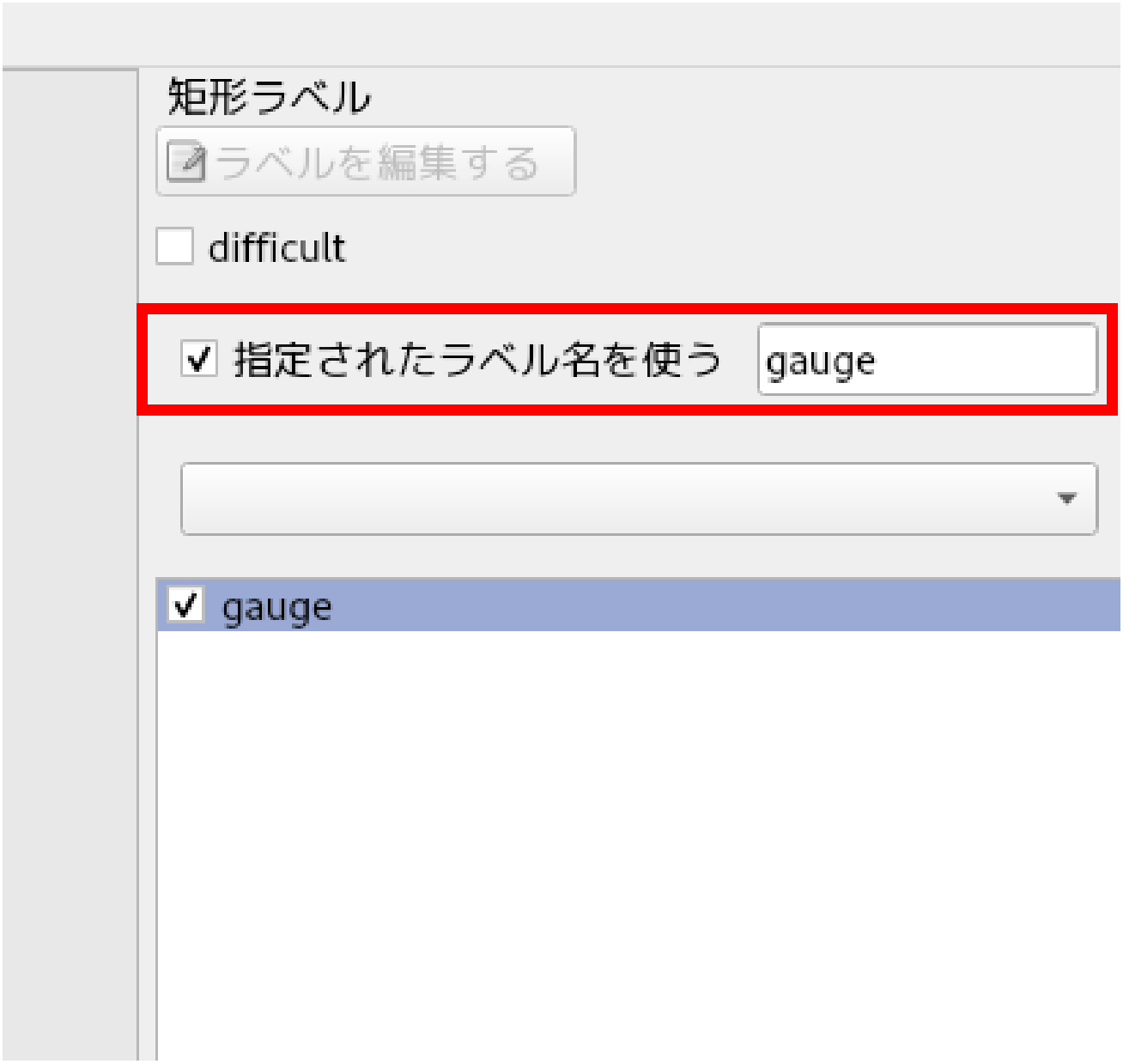


図 6.18 作業効率化のための設定 2

「表 6.3. labellmg でよく使用するショートカットキー一覧」に、labellmg を使用したアノテーション作業中によく用いるショートカットキーを示します。作業効率化のために使用することをお勧めします。詳しくは、labellmg の画面上部のツールバーの[ヘルプ]→[ショートカット一覧を見る(英語)]から確認できます。

表 6.3 labellmg でよく使用するショートカットキー一覧

ショートカットキー	役割
W	矩形選択モードに切り替え
D	次の画像へ
A	前の画像へ

上記設定後、W キーを押下して矩形選択モードに切り替え、マウス左ドラッグで画像内の対象物を選択します。選択すると自動的に選択範囲が gauge でラベル付けされます。

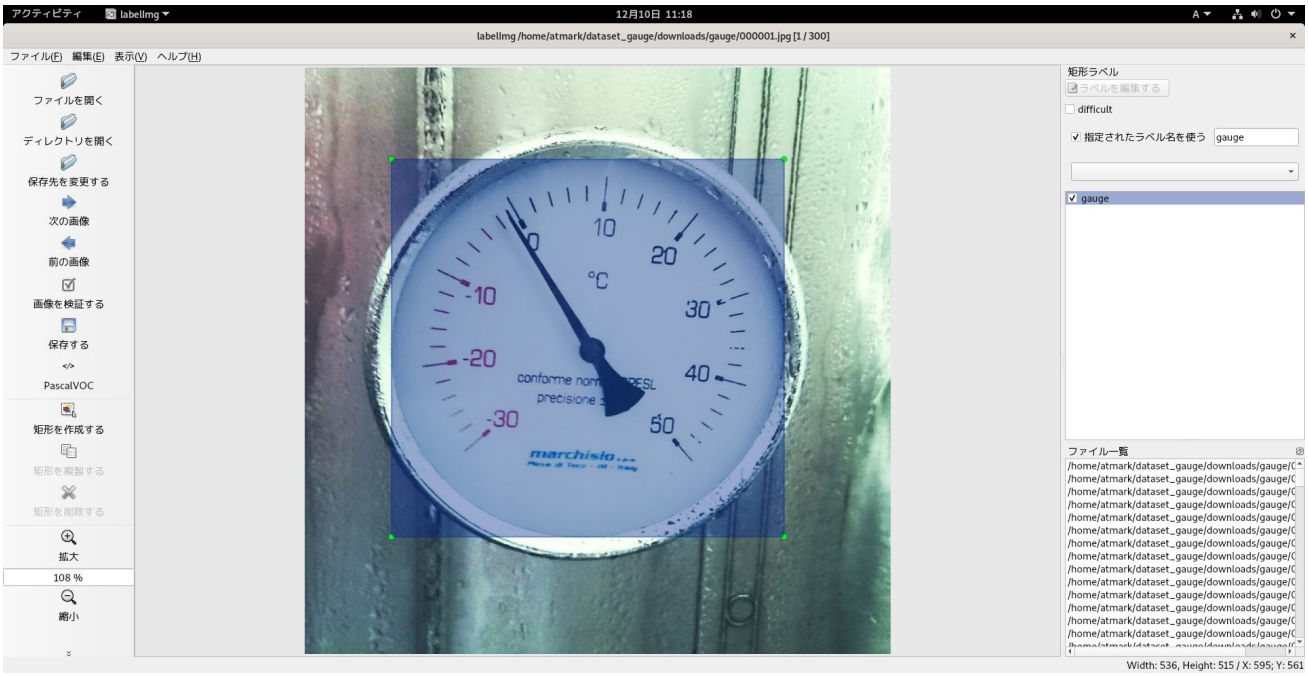


図 6.19 ラベル付け例

D キー押下で次の画像へ切り替えられます。

以上の手順を用意した全ての画像に対して行っていきます。

アノテーションする際の注意事項としまして以下が挙げられます。

- アノテーションのポリシーは作業中一貫させること
  - 見切れているものはアノテーションする/しない
  - 対象物の裏面や別角度の写真をアノテーションする/しない
  - 最終的に検出したい物を常に意識して作業する
- 用意した全ての画像に対して必ずしもアノテーションしなければならないわけではない
  - 画像を用意したが、対象物がよく写っている画像でないならアノテーションしないことも重要

アノテーションが完了したら、保存先のディレクトリにアノテーション情報(.xml ファイル)が保存されていることを確認してください。

```
[ATDE ~/dataset_gauge/annotations]$ ls *.xml
000001.xml
000002.xml
:（省略）
```

図 6.20 アノテーション情報ファイルを確認

次に、label_map.pbtxt というファイルを~/dataset_gauge/xmls/内に作成します。物体検出は、画像を推論モデルに入力し、結果として「どこに」、「何が」あるかが返ってくるのですが、「何が」の部分は 1 から始まる ID で返ってきます。label_map.pbtxt は、その ID とラベル名を紐付けるファイルです。

本サンプルアプリケーションでは"gauge"のみを認識し、画面上にはメーターが指し示す値を表示させる仕様なので、アプリケーション本体ではこのラベルファイルは使用しません。しかし、この後の推論モデルの学習時に使用するので、ここで作成しておきます。

今回作成する label_map.pbtxt の内容を「図 6.21. label_map.pbtxt 作成例」に示します。

```
item {
  id: 1
  name: 'gauge'
}
```

図 6.21 label_map.pbtxt 作成例

以上でアノテーション作業は完了です。

6.2.2. 推論モデルの生成

教師データが用意できたら、いよいよ学習して独自の物体検出モデルを作成します。

「3.2.2. 機械学習向け開発環境の準備 機械学習」でも述べましたが、今回推論モデルの学習は Google Colaboratory 上で行います。

ATDE の Web ブラウザで、Armadillo-IoT_G4_create_detection_model [https://colab.research.google.com/github/atmark-techno/Armadillo-Base-OS-dev-guide-colab/blob/master/Armadillo_IoT_G4_create_detection_model.ipynb] を Google Colaboratory で開いてください。そのノートブック内に転移学習と TFLite 形式への変換の具体的な手順をテキストとソースコード内のコメントにてまとめています。

各セルに対して Ctrl+Enter を押し、セルごとに実行していくことで転移学習済み TFLite モデルを取得することができます。実行には、全体で数十分程度時間がかかることにご注意ください。

結果として Armadillo 上で動作する TFLite 形式のモデル(model.tflite)が得られます。

以上で推論モデルの作成は完了です。

6.3. podman コンテナを作成する

推論モデルが作成できたらアプリケーションの作成を行いますが、先にそのアプリケーションが動作するコンテナイメージを作成します。

今回作成するサンプルアプリケーションでは、NPU や VPU をアプリケーション内で使用するため、アットマークテクノが提供しているサンプルコンテナイメージをベースとして使用します。

6.3.1. 開発前の準備

ここからは Armadillo-IoT ゲートウェイ G4 上で作業を行います。

Armadillo Base OS では全ての変更は RAM 上に保持され、電源を切った段階でそれらの変更は消えて変更前の状態に戻ります。運用時にはこの状態で問題ありませんが、開発時に変更した内容が再起動時に保持されないのは不都合です。

podman コンテナ上での作業も同様に全て消えてしまうので、開発中は基本的に「図 6.22. podman 関連ファイルを eMMC 上に保存する」に示すコマンドを実行して、コンテナイメージやコンテナの中身などの podman 関連のファイルは eMMC 上に保存するよう設定しておきます。

```
[armadillo /]# podman_switch_storage --disk
```

図 6.22 podman 関連ファイルを eMMC 上に保存する

また、予め Armadillo-IoT ゲートウェイ G4 の製品マニュアルの「VPU や NPU を使用する [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.11.0/ch08.html#sct.prepare-library]」の手順を行ってください。

6.3.2. サンプルコンテナをビルド

前述の通り、今回作成するサンプルアプリケーションは、アットマークテクノが提供しているサンプルコンテナイメージをベースに開発を行います。そのため、まずはそのサンプルコンテナイメージを生成します。

Armadillo-IoT ゲートウェイ G4 コンテナ [https://armadillo.atmark-techno.com/resources/software/armadillo-iot-g4/container]から「Debian 11 (bullseye) サンプル Dockerfile」をダウンロードしてください。[VERSION]は適宜読み替えてください。

```
[armadillo /]# wget https://armadillo.atmark-techno.com/files/downloads/armadillo-iot-g4/
container/at-debian-image-dockerfile-[VERSION].tar.gz
```

図 6.23 Dockerfile のダウンロード

「図 6.24. サンプルコンテナのイメージのビルド」に示すコマンドを実行することで、アットマークテクノが提供しているサンプルコンテナイメージを Armadillo-IoT ゲートウェイ G4 上でビルドできます。ここでは"abos-dev-guide"という名称で、"v0.0.0"というタグ名のコンテナを作成しています。

```
[armadillo /]# tar xzf at-debian-image-dockerfile-[VERSION].tar.gz
[armadillo /]# cd at-debian-image-dockerfile-[VERSION]
[armadillo /]# podman build -t abos-dev-guide:v0.0.0 .
:（省略）
[armadillo /]# podman images abos-dev-guide
REPOSITORY                TAG         IMAGE ID      CREATED             SIZE
localhost/abos-dev-guide  v0.0.0      c38bd7464cd8  About a minute ago  234 MB
```

図 6.24 サンプルコンテナのイメージのビルド

6.3.3. コンテナ内に入る

作成したコンテナの中に入り、開発作業を行います。

「図 6.25. コンテナ内に入る」に示すコマンドを実行することで、コンテナの中に入ることができます。この時、後にサンプルアプリケーションを動作させるために必要になるオプションも指定しておきます。

```
[armadillo /]# podman run -ti ¥
        --name=sample_container ¥
        --env=XDG_RUNTIME_DIR=/tmp ¥
        --env=LD_LIBRARY_PATH=/opt/firmware/usr/lib/aarch64-linux-gnu ¥
        --device=/dev/tty7 ¥
```

```
        --device=/dev/input ¥
        --device=/dev/dri ¥
        --device=/dev/galcore ¥
        --device=/dev/mxc_hantro ¥
        --device=/dev/mxc_hantro_vc8000e ¥
        --device=/dev/ion ¥
        --volume=/run/udev:/run/udev:ro ¥
        --volume=/opt/firmware:/opt/firmware:ro ¥
        --cap-add "SYS_TTY_CONFIG" ¥
        localhost/abos-dev-guide:v0.0.0 /bin/bash
[container /]#
```

図 6.25 コンテナ内に入る

6.4. アプリケーション実行の準備をする

アットマークテクノが提供しているサンプルコンテナには、必要最低限のパッケージしかインストールされていません。このままでは、サンプルアプリケーションのダウンロードや実行ができませんので、最初に必要なパッケージをインストールしておきます。

```
[container /]# apt update
[container /]# apt install -y ¥
   wget ¥
   procps ¥
   python3 ¥
   python3-pip ¥
   python3-opencv ¥
   python3-numpy ¥
   python3-tflite-runtime ¥
   nn-imx ¥
   tim-vx ¥
   tensorflow-lite-vx-delegate ¥
   gstreamer1.0-tools ¥
   gstreamer1.0-plugins-bad ¥
   gstreamer1.0-plugins-good ¥
   gstreamer1.0-imx
[container /]# pip3 install pillow
```

図 6.26 コンテナへパッケージをインストール

6.5. アプリケーションを作成する

podman コンテナ内で実行するアプリケーションを作成します。

以下では開発の際の手順の一例を紹介しますが、サンプルアプリケーションは こちら [https://download.atmark-techno.com/armadillo-iot-g4/example/armadillo-base-os-dev-guide/read_meter.tar.gz]から作成済みのものをダウンロードしてご使用ください。

コンテナ内で「図 6.27. サンプルアプリケーションのダウンロードと展開」に示すコマンドを実行することで、サンプルアプリケーションをダウンロード・展開できます。今回は、/opt 以下にアプリケーションファイルを展開します。

```
[container /]# cd /opt
[container /opt]# wget https://download.atmark-techno.com/armadillo-iot-g4/example/armadillo-base- ⏎
```

```
os-dev-guide/read_meter.tar.gz
[container /opt]# tar xzf read_meter.tar.gz
[container /opt]# rm read_meter.tar.gz
```

図 6.27 サンプルアプリケーションのダウンロードと展開

6.5.1. ファイル構成

ダウンロードしたファイルを展開すると、read_meter というディレクトリが現れます。

アプリケーションを構成するファイルは全て read_meter ディレクトリに配置されています。配置した各ファイルと、その役割を「図 6.28. サンプルアプリケーションを構成するファイル」に示します。

```
read_meter
|-- gauge_sample.mp4 ❶
|-- model.tflite ❷
|-- modules
|   |-- __init__.py
|   |-- inferencer.py ❸
|   `-- meter.py ❹
|-- parameter_sample.json ❺
`-- read_meter ❻
```

図 6.28 サンプルアプリケーションを構成するファイル

❶ アナログメーターが映り続けた動画です。USB カメラの代わりに入力映像として使用できます。

❷ TFLite 形式の推論モデルファイルです。

❸ 物体検出に関わる処理を記した Python モジュールファイルです。

❹ メーター読み取りに関わる処理を記した Python モジュールファイルです。

❺ 読み取るアナログメーターの中心座標や、各スケールの座標などを記録したファイルです。

❻ アプリケーション本体です。

6.5.2. コンテナ内のファイルの編集

実際に Armadillo で動かすソフトウェア開発を行う場合、ATDE 等の他の環境で作成したアプリケーションを Armadillo のコンテナ内に転送して動かすことが一般的です。

転送方法は様々ありますが、開発時に USB メモリを利用してコピーすること方法が手軽です。

```
[armadillo ~/]# mount /dev/sda1 /mnt ❶
[armadillo ~/]# podman cp /mnt/file sample_container:/ ❷
[armadillo ~/]# umount /mnt
[armadillo ~/]# podman exec sample_container ls /file ❸
file
```

図 6.29 ATDE から Armadillo へファイルを送信

❶ USB メモリをマウントして、中身を扱えるようにします。

❷ 「file」というファイルを sample_container の / ディレクトリにコピーします。

❸ コピーされたことを確認します。

もちろん Armadillo 内でファイルの編集をすることも可能です。アットマークテクノが提供しているサンプルコンテナイメージには、デフォルトではエディタが含まれていないので、「図 6.30. コンテナ内にエディタをインストールする」に示すコマンドを実行することでお好きなエディタ(この例では vim)をインストールできます。

```
[container /]# apt update && apt upgrade
[container /]# apt install -y vim
```

図 6.30 コンテナ内にエディタをインストールする

ただし、基本的にはエディタはアプリケーションを実行するだけの本番環境には必要ないため、最終的なコンテナイメージのサイズ削減のためにも、開発完了時にはアンインストールするか別途実行用のコンテナを作成することをお勧めします。

6.5.3. アプリケーションの動作確認

コンテナ内にアプリケーションを配置できたら動作確認をしてみましょう。予め Armadillo-IoT ゲートウェイ G4 を HDMI ケーブルでモニタと接続しておいてください。

今回はサンプルアプリケーションの動作確認として「図 6.31. サンプルアプリケーションの動作確認」に示すコマンドを実行します。ここでは USB カメラは使用せず、gauge_sample.mp4 を映像の入力として動作確認をします。

```
[container /]# cd /opt/read_meter
[container /opt/read_meter]# weston --tty=7 &
[container /opt/read_meter]# ./read_meter --model model.tflite --param_file parameter_sample.json ⏎
--video gauge_sample.mp4
```

図 6.31 サンプルアプリケーションの動作確認

「図 6.32. Armadillo-IoT ゲートウェイ G4 への周辺機器の接続」のような画面が表示されれば正しく動作しています。

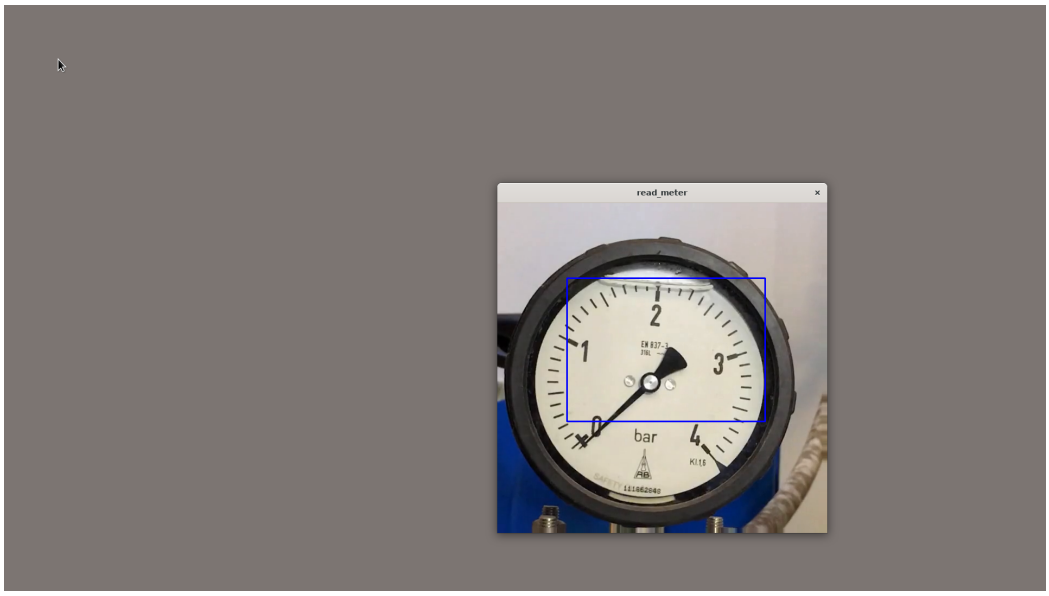


図 6.32 Armadillo-IoT ゲートウェイ G4 への周辺機器の接続

Ctrl+C でサンプルアプリケーションを終了させます。動作確認が完了したら、weston は終了させます。

```
[container /opt/read_meter]# pkill weston //バックグラウンドの weston を終了
```

図 6.33 weston を終了

以上でアプリケーションの作成は完了です。

6.5.4. podman コンテナの保存

コンテナ内の編集が完了しても、この後にコンテナを停止したり Armadillo の電源を OFF にしたりすると、ここまで編集してきたコンテナ内の変更の差分がなくなってしまいます。そのため、podman コンテナ内の整備が完了したらコンテナ内の変更を保存する必要があります。

まず、 `exit` コマンドを実行してコンテナから抜けます。

```
[container /]# exit
[armadillo ~]# //コンテナから抜けた
```

図 6.34 podman コンテナから抜ける

「図 6.34. podman コンテナから抜ける」の方法でコンテナから抜けると、コンテナは停止します。

起動中のコンテナは「図 6.35. 起動中の podman コンテナを確認する」に示すコマンドで確認できます。

```
[armadillo ~]# podman ps
CONTAINER ID  IMAGE                                    COMMAND     CREATED         STATUS
```

```
PORTS       NAMES
//sample_container が起動中でないことを確認
```

図 6.35 起動中の podman コンテナを確認する

停止中のコンテナは「図 6.36. 停止中の podman コンテナを確認する」に示すコマンドで確認できます。

```
[armadillo ~]# podman ps -a
CONTAINER ID  IMAGE                                COMMAND     CREATED        STATUS                    ⏎
PORTS       NAMES
9c908ab45ed8  localhost/abos-dev-guide:v0.0.0  /bin/bash   3 minutes ago  Exited (0) 12 seconds     ⏎
ago              sample_container
```

図 6.36 停止中の podman コンテナを確認する

次に、 podman commit コマンドで先程までの変更内容を保存します。「図 6.37. podman コンテナの変更を保存する」は、変更後の"sample_container"を"abos-dev-guide"というイメージ名、"v1.0.0"というタグ名で保存する例です。

```
[armadillo ~]# podman commit sample_container abos-dev-guide:v1.0.0
[armadillo ~]# podman images abos-dev-guide //保存されたことを確認
REPOSITORY                TAG         IMAGE ID      CREATED             SIZE
localhost/abos-dev-guide  v1.0.0      d0de8c2e70f0  About a minute ago  901 MB
localhost/abos-dev-guide  v0.0.0      c38bd7464cd8  2 hours ago         234 MB
```

図 6.37 podman コンテナの変更を保存する

タグ名の部分でコンテナイメージのバージョン管理を行うことが可能です。

podman commit することで、コンテナ内で行った変更が eMMC 内に保存され、Armadillo 本体の電源を切ったり、コンテナを削除しても commit した箇所までが保持されます。開発時に Armadiilo の電源を切る場合は、commit を行ったことを確認してから電源を切ってください。

6.5.5. podman コンテナのエクスポート

Armadillo-IoT ゲートウェイ G4 上で作成した podman コンテナイメージは、他の Armadillo-IoT ゲートウェイ G4 上でも動作します。作成した podman コンテナイメージを他の Armadillo に書き込む方法として以下があります。

1. podman コンテナイメージを外部ストレージなどに保存する
2. dockerhub などのリモートリポジトリに保存する

以下では、作成したコンテナイメージを USB メモリに保存する方法を紹介します。予め Armadillo に 4GB 以上の空き容量がある USB メモリを挿入してください。

「図 6.38. podman コンテナイメージを外部ストレージに出力する」は、USB メモリの第 1 パーティション(/dev/sda1)に abos-dev-guide.tar という名前で abos-dev-guide の v1.0.0 を保存する例です。

```
[armadillo ~]# mount /dev/sda1 /mnt ❶
[armadillo ~]# podman save -o /mnt/abos-dev-guide.tar abos-dev-guide:v1.0.0 ❷
```

図 6.38 podman コンテナイメージを外部ストレージに出力する

❶ USB メモリなどの外部記憶デバイスをマウントします。

❷ USB メモリにイメージ名:abos-dev-guide、タグ名:v1.0.0 を abos-dev-guide.tar で保存します。

保存したコンテナイメージは、SWUpdate の swu パッケージに含ませたり、 `podman load` コマンドで読み込んだりすることで、他の Armadillo 上で動かすことができます。

6.5.6. podman コンテナとアプリケーションの自動実行

最後に、Armadillo-IoT ゲートウェイ G4 の起動時に、作成した podman コンテナ及びコンテナ内の自作アプリケーションが自動実行するように設定します。

Armadillo Base OS では、起動時に `podman_start` というスクリプトを実行して、/etc/atmark/containers/以下の.conf ファイルを参照し、そこで指定されているコンテナを起動します。

各種設定方法について、詳しくは製品マニュアルをご確認ください。以下では、本サンプルアプリケーション向けの設定を示します。

6.5.6.1. conf ファイルの作成

コンテナの外で/etc/atmark/containers/sample_container.conf を作成します。

```
set_image "localhost/abos-dev-guide:v1.0.0"
add_devices /dev/tty7 /dev/input /dev/dri /dev/galcore
add_devices /dev/mxc_hantro /dev/mxc_hantro_vc8000e /dev/ion
add_volumes /run/udev:/run/udev:ro /opt/firmware:/opt/firmware
append_args --env=XDG_RUNTIME_DIR=/tmp
append_args --env=LD_LIBRARY_PATH=/opt/firmware/usr/lib/aarch64-linux-gnu
append_args --cap-add=SYS_TTY_CONFIG
append_args -w /opt/read_meter
set_command /bin/bash -c "
weston --tty=7 & ¥
exec python3 read_meter ¥
        --model model.tflite ¥
        --param_file parameter_sample.json ¥
        --video gauge_sample.mp4"
```

図 6.39 sample_container.conf 作成例

なお、「図 6.39. sample_container.conf 作成例」に示した conf ファイルは こちら [https://download.atmark-techno.com/armadillo-iot-g4/example/armadillo-base-os-dev-guide/sample_container.conf]からダウンロード可能です。

```
[armadillo /]# wget https://download.atmark-techno.com/armadillo-iot-g4/example/armadillo-base-os-dev-guide/sample_container.conf -O /etc/atmark/containers/sample_container.conf
```

図 6.40 sample_container.conf のダウンロード

6.5.6.2. 設定の確認

「6.5.6.1. conf ファイルの作成」で作成した conf ファイルの設定が正しく出来ているかを確認します。

「図 6.41. podman_start の実行例」を実行すると、/etc/atmark/containers/sample_container.conf ファイルの内容に従ってコンテナが起動します。これにより正常にコンテナとサンプルアプリケーションが起動することを確認します。

```
[armadillo ~]# podman_start sample_container
```

図 6.41 podman_start の実行例

正常にサンプルアプリケーションが実行されると、「6.5.3. アプリケーションの動作確認」で動作確認した時と同様の画面が表示されます。

ここまでの動作確認が完了したら、/etc/atmark/containers/sample_container.conf を persist_file コマンドで永続化します。

```
[armadillo ~]# persist_file /etc/atmark/containers/sample_container.conf
```

図 6.42 sample_container.conf の永続化

「図 6.42. sample_container.conf の永続化」を実行することにより、作成した sample_container.conf が再起動後も eMMC 内に残り、次回以降 Armadillo に電源を投入した際にコンテナが自動起動します。

podman_start によって起動したコンテナも「図 6.43. コンテナの停止・削除」に示すコマンドで停止、削除できます。

```
[armadillo ~]# podman stop sample_container ❶
[armadillo ~]# podman rm sample_container ❷
```

図 6.43 コンテナの停止・削除

❶ sample_container を停止します。

❷ sample_container を削除します。

6.6. 動作確認

最後に動作確認を行います。

周辺機器の接続後、Armadillo-IoT ゲートウェイ G4 に電源を投入してください。

しばらくすると自動的にアプリケーションが起動して、サンプルの動画ウィンドウを HDMI モニタに映し出します。青い四角は、物体検出にてアナログメーターだと検知した箇所です。

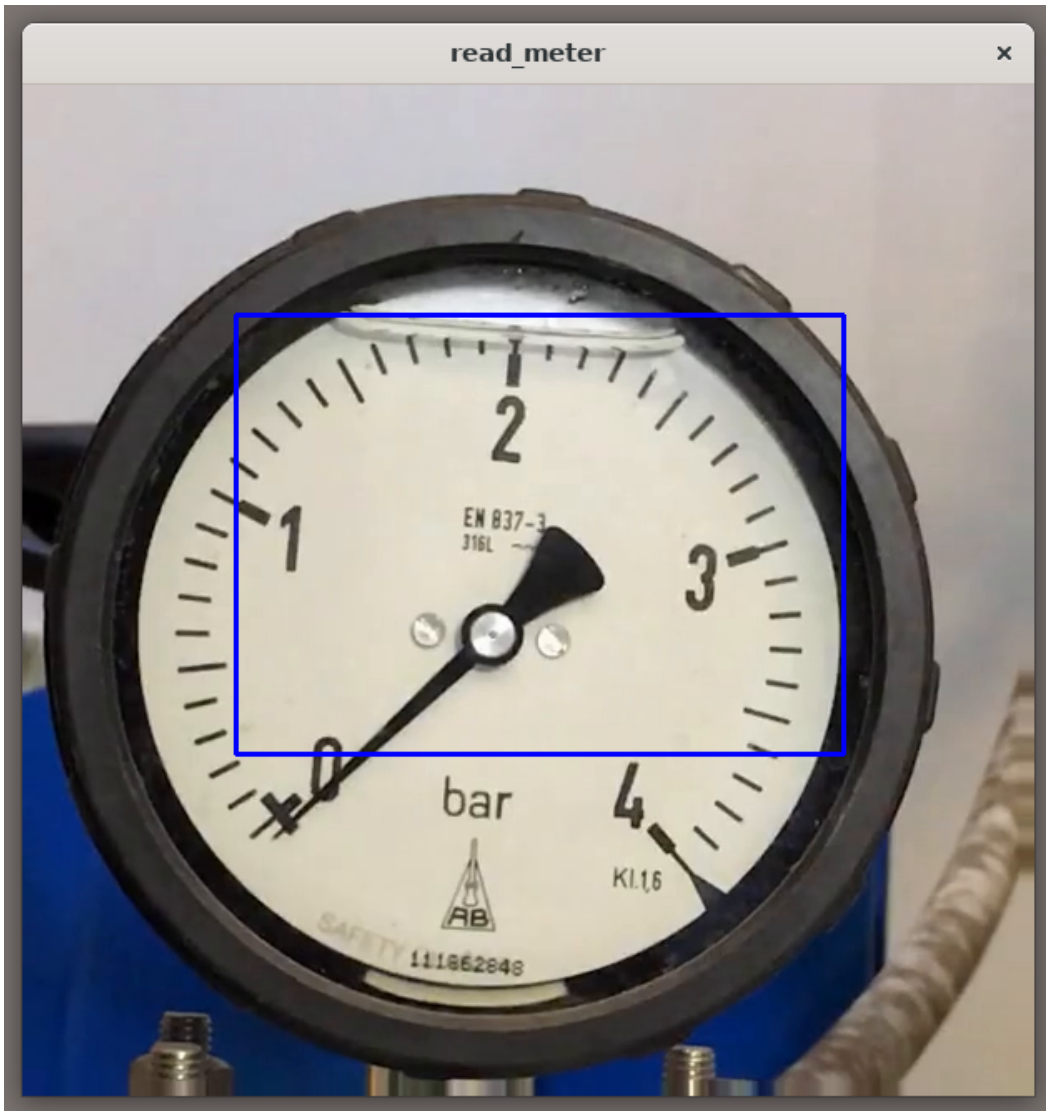


図 6.44 サンプル動画に対して物体検出

この状態でしばらく待つと、アナログメーターが指す値が変わっていき、スケールの最小値である 0 を超えた辺りから青い四角の上に読み取れた数値が表示されます。

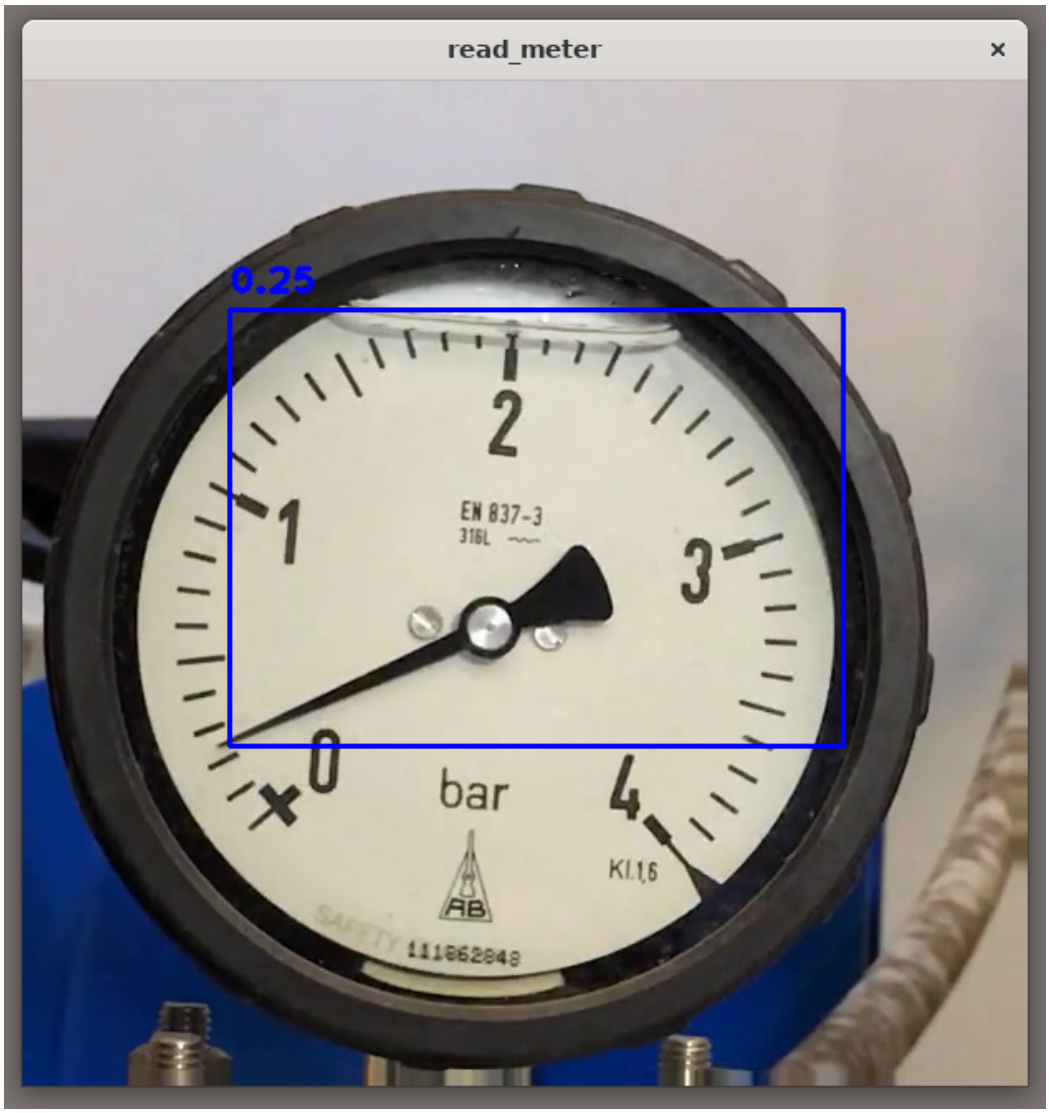


図 6.45 サンプル動画内のアナログメーターを読む

動画が終了すると、アプリケーション及びコンテナが停止します。

ここまでを確認できたならば、本ドキュメントにおけるサンプルアプリケーションは完成です。

6.7. USB カメラから映像を取得するように変更

以降の手順は USB カメラを使用するオプションです。

ここまでで作成したアプリケーションは、サンプル動画を映像入力としてメーターの読み取りを行っています。USB カメラと撮影対象のアナログメーターをお持ちの場合は、USB カメラから映像を取得し、任意のアナログメーターの値を読むことも可能です。

sample_container が起動している場合は、「図 6.43. コンテナの停止・削除」のコマンドを実行して停止・削除しておいてください。

6.7.1. 撮影対象の parameter.json を作成

本サンプルアプリケーションでは、アナログメーターの値を読む為に、各アナログメーターの中心とスケールがなす角度などのパラメータを保存した json ファイルが必要になります。サンプル動画以外のアナログメーターに対応させるためには、撮影対象となるアナログメーターの中心と各スケールの座標等を示す parameter.json を作成し、コンテナ内のアプリケーションから読み出す必要があります。

ここでは、サンプルアプリケーション向けのパラメータファイルを作成するための Python スクリプトについて紹介します。

アナログメーターを正面から捉えた画像から各点の座標を取得できる Python スクリプト(config_param.py)を こちら [https://download.atmark-techno.com/armadillo-iot-g4/example/armadillo-base-os-dev-guide/config_param.py]からダウンロードできます。

「図 6.46. config_param.py のダウンロードと起動」に、image.png をアナログメーターを正面から捉えた画像とした時の、config_param.py の実行例を示します。

```
[ATDE ~/]$ wget https://download.atmark-techno.com/armadillo-iot-g4/example/armadillo-base-os-dev-guide/config_param.py
[ATDE ~/]$ python3 config_param.py --image image.png
```

図 6.46 config_param.py のダウンロードと起動

図 6.47 アナログメーターを正面から捉えたサンプル画像(image.png)

実行すると以下のようなウィンドウが出現するので、アナログメーターの中心をクリックしてください。

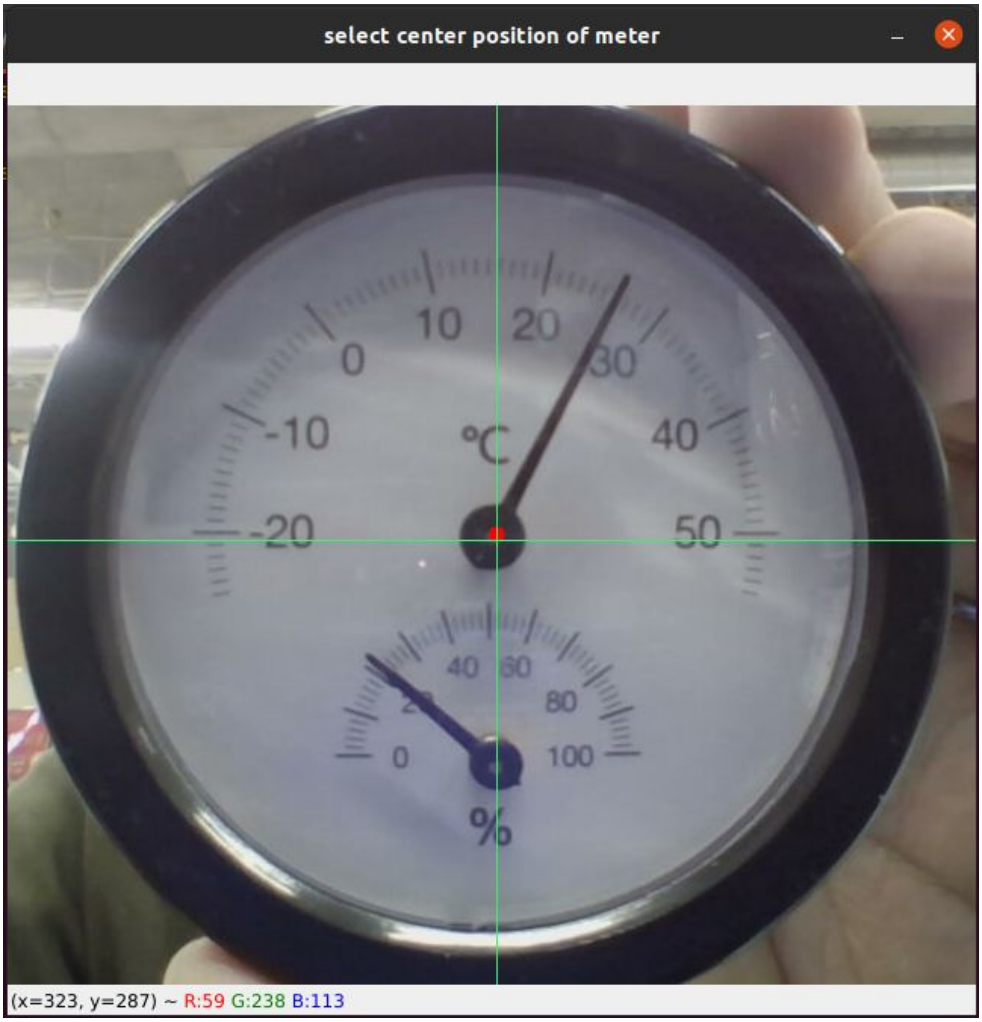


図 6.48 アナログメーターの中心をクリック

選択された中心点は赤い点で描画され、中心の選択は何度もやり直すことができます。選択が完了したら、何かキーを押下してください。

次に、以下のようなウィンドウが出現するので、アナログメーターが示すことができる最低値から順に、スケールの位置をクリックしてください。

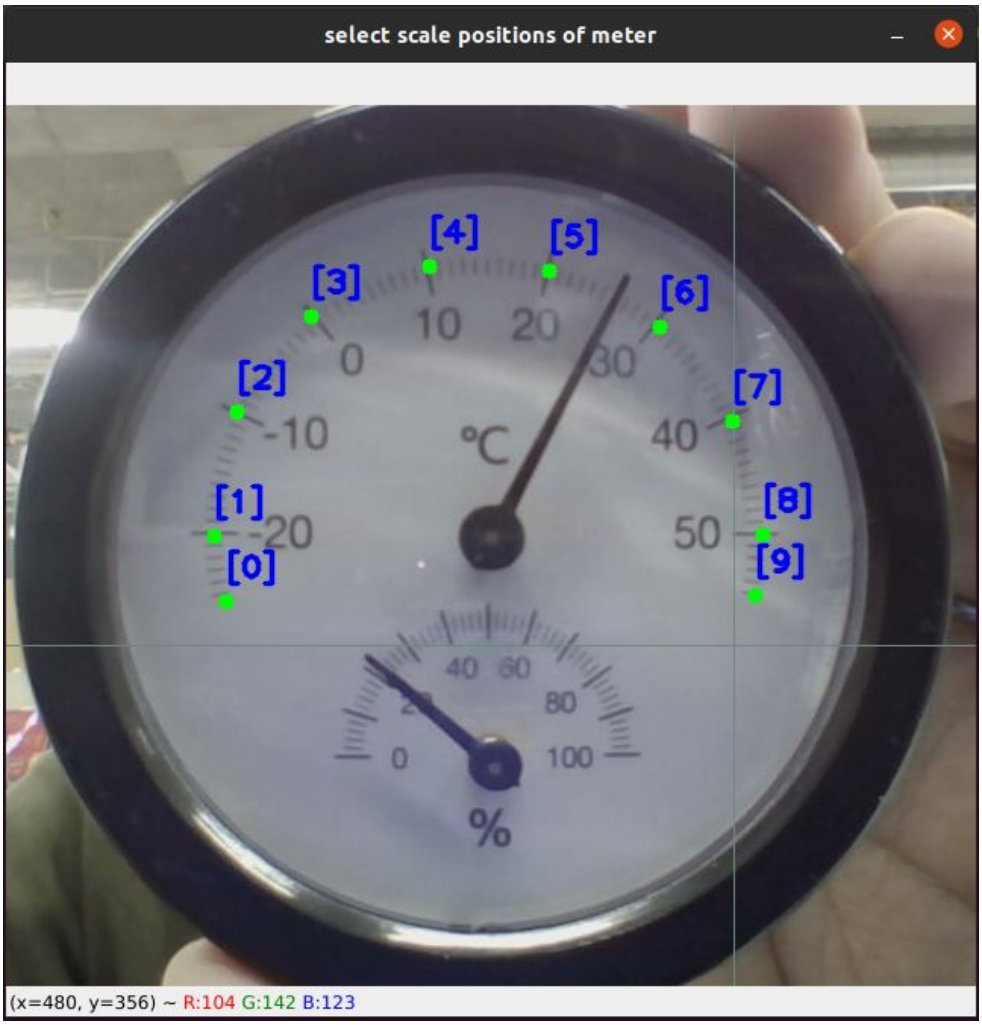


図 6.49 各スケールの位置をクリック

クリックした位置と順番は画面に描画されます。右クリックで一個前の選択を取り消すことができます。選択が完了したら、何かキーを押下してください。

次に、コンソールでクリックした位置の値を入力してください。コンソールで表示されている x 個目と画面内の選択点の数値は対応しています。入力したら Enter キーを押下し、選択した点の数だけこれを繰り返してください。

```
0 個目の数値> -25
1 個目の数値> -20
2 個目の数値> -10
3 個目の数値> 0
4 個目の数値> 10
5 個目の数値> 20
6 個目の数値> 30
7 個目の数値> 40
8 個目の数値> 50
9 個目の数値> 55
Done!
```

図 6.50 各スケールの数値を入力

最後まで入力すると、「Done!」と表示され、parameter.json が出力されています。

```
[ATDE ~/]$ ls parameter.json
parameter.json
```

図 6.51 各スケールの数値を入力

出力された parameter.json は、任意の方法で Armadillo に転送してください。

6.7.2. パラメータファイルの配置

作成したパラメータファイルを、コンテナ内のアプリケーションに読み込ませます。

読み取りたいアナログメーター 1 つにつき、パラメータファイルを 1 つ用意する都合上、何度も書き換えることになりそうなファイルです。このパラメータファイルをコンテナイメージの中に含めるとすると、ファイルの差し替え時には毎回コンテナイメージを作り直す必要があり手間です。

このようなファイルはコンテナイメージ内に含めず、Armadillo Base OS が提供するデータ保存用ディレクトリに保存し、コンテナに volume として渡す方法がお勧めです。

データ保存用ディレクトリは、/var/app/volumes と/var/app/rollback/volumes の 2 種類が用意されています。それぞれの詳細については、Armadillo-IoT ゲートウェイ G4 の製品マニュアルの「データを保存する [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.11.0/ch08.html#sct.data-hozon-container]」を参照してください。

今回は/var/app/rollback/volumes を使用します。

まず、コンテナの外で「図 6.52. /var/app/rollback/volumes/assets の作成」に示すコマンドで、/var/app/rollback/volumes/の下に assets ディレクトリを作成します。

```
[armadillo ~]# mkdir -p /var/app/rollback/volumes/assets
```

図 6.52 /var/app/rollback/volumes/assets の作成

次に、作成したパラメータファイル(parameter.json)を/var/app/rollback/volumes/assets の中に配置します。

```
[armadillo ~]# cp /path/to/parameter.json /var/app/rollback/volumes/assets
```

図 6.53 parameter.json を配置

以上でパラメータファイルの配置は完了です。

なお、データ保存用ディレクトリである/var/app/volumes 及び、/var/app/rollback/volumes に配置したファイルは、eMMC 上に保存されるため再起動後も消えずに残り続けます。

6.7.3. .conf ファイルの設定

以下の 2 点に対応させるために、/etc/atmark/containers/sample_container.conf を「図 6.54. USB カメラを使用する sample_container.conf 作成例」のように編集します。ほとんど USB カメラ対応前と同じですが、異なる部分のみ注釈で解説します。

1. サンプルアプリケーションの映像入力を動画ではなく USB カメラにする
2. 「6.7.2. パラメータファイルの配置」で配置した parameter.json をコンテナ内で使用する

```
set_image "localhost/abos-dev-guide:v1.0.0"
add_devices /dev/tty7 /dev/input /dev/dri /dev/galcore
add_devices /dev/mxc_hantro /dev/mxc_hantro_vc8000e /dev/ion
add_devices /dev/video* ❶
add_volumes /run/udev:/run/udev:ro /opt/firmware:/opt/firmware
add_volumes assets:/assets:ro ❷
append_args --env=XDG_RUNTIME_DIR=/tmp
append_args --env=LD_LIBRARY_PATH=/opt/firmware/usr/lib/aarch64-linux-gnu
append_args --cap-add=SYS_TTY_CONFIG
append_args -w /opt/read_meter
set_command /bin/bash -c "
weston --tty=7 & ¥
exec python3 read_meter ¥
        --model model.tflite ¥
        --param_file /assets/parameter.json ¥ ❸
        --camera 3" ❹
```

図 6.54 USB カメラを使用する sample_container.conf 作成例

❶ USB カメラを使用するために/dev/video*を追加します。

❷ 作成した assets ディレクトリをコンテナに渡します。

❸ パラメータファイルはコンテナ内では/assets/parameter.json に配置されていますので、そこを指定します。

❹ "--video…" の行を削除し、"--camera 3" に修正します。

volumes に相対パスを指定すると、/var/app/rollback/volumes 以下のディレクトリを指定したことになります。

サンプルアプリケーションは、起動時のオプションに"--camera <デバイス番号>"を指定することで、/dev/video<デバイス番号>のカメラデバイスから映像を取得します。<デバイス番号>は、USB カメラ接続時のログなどから判断してください。

6.7.4. 自動実行の確認

podman_start コマンドでコンテナが起動することを確認します。

```
[armadillo /]# podman_start sample_container
```

USB カメラから映像とアナログメーターの値を取得できることを確認します。

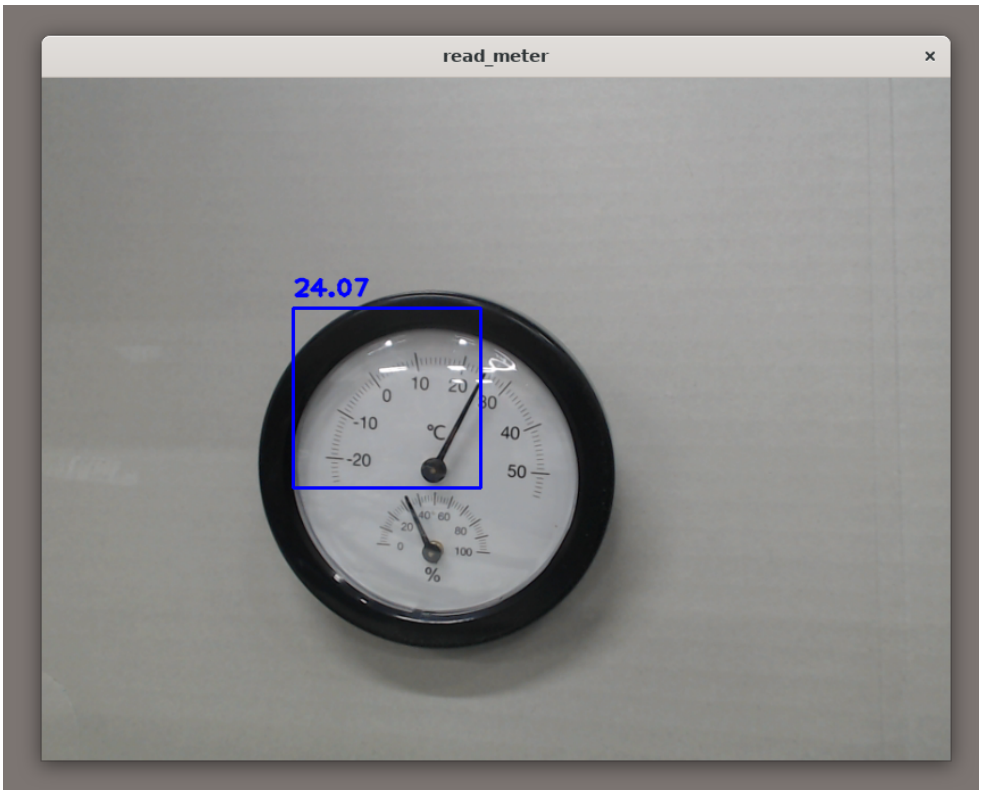


図 6.55 USB カメラの映像からアナログメーターの値を取得

次回 Armadillo 起動時に自動実行させるには、/etc/atmark/containers/sample_container.conf を persist_file コマンドで永続化させます。

```
[armadillo ~]# persist_file /etc/atmark/containers/sample_container.conf
```

図 6.56 sample_container.conf の永続化

7. 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。以下では、開発が完了し完成したソフトウェアを Armadillo Base OS 搭載製品に書き込む方法について紹介します。

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。

それぞれの手法の特徴を「表 7.1. インストールディスクと SWUpdate によるソフトウェア書き込みの比較」にまとめます。ソフトウェア書き込み工程を決定する際の参考にしてください。

表 7.1 インストールディスクと SWUpdate によるソフトウェア書き込みの比較

	Pros	Cons
インストールディスク	・インストールの前後処理を行なうシェルスクリプトのテンプレートが用意されている ・インストールの前後処理は、SD カード内にシェルスクリプトを配置するだけなので製造担当者にも編集しやすい	・インストール実行にはジャンパピンをショートにするために Armadillo のケースを開ける必要がある ・動いているシステムをそのままインストールディスクにするため、出荷時の標準イメージから手動で同じ環境を構築する手順が残らない
SWUpdate	・ジャンパピンをショートにせずに実行できるため、Armadillo のケースを開ける必要がない ・必ず必要となる初回アップデートを別途実行する必要がない	・swu イメージの作成には、mkswu を使用できる環境と desc ファイルの記述方法を知る必要があるため、開発担当者以外に swu イメージを更新させるハードルが少し高い ・ログの取得など、インストール前後の処理が必要な場合は自分で記述する必要がある

量産時のイメージ書き込みにインストールディスクを使用する場合は、「8. 量産用インストールディスクの作成」に進んでください。

量産時のイメージ書き込みに SWUpdate を使用する場合は、「9. 量産用 swu イメージの作成」に進んでください。

8. 量産用インストールディスクの作成

本章では、開発を経て完成したソフトウェア含む Armadillo を量産するためにインストールディスクを作成する手順について説明します。作成したインストールディスクを製造担当者に渡すことで、簡単かつミスも少なく Armadillo を量産することが可能です。

8.1. インストールディスクとは

インストールディスクイメージを microSD に書き込んだものがインストールディスクです。インストールディスクイメージには、インストール先の Armadillo に書き込む任意のルートファイルシステム及びブートローダーと、イメージ書き換え用のスクリプトが組み込まれます。SD ブートでインストールディスクから起動することで、自動的にイメージ書き換えスクリプトが動作し、eMMC 内のイメージを予めインストールディスクイメージに組み込んだ任意の OS やブートローダに書き換えることができます。

インストールディスクには 2 つの種類があります。

1. 初期化インストールディスク

   これを用いてインストールを実行することで、対象の Armadillo の eMMC 内のイメージを標準イメージに書き換えることができます。主に Armadillo を初期化する際に使用します。アットマークテクノが提供しているイメージですが、ユーザが同等のイメージを作成することも可能です。

2. クローンインストールディスク

   開発完了後の Armadillo を複製する為にユーザが作成するインストールディスクです。インストールディスクイメージには、開発完了後のルートファイルシステム、コンテナイメージやブートローダなどを組み込みます。

以下の手順ではクローンインストールディスクを作成し、製造の準備を行ないます。

本ドキュメントでは紹介しませんが、初期化インストールディスクは、Armadillo が起動しなくなった場合などに eMMC 内のイメージを標準イメージに書き換えるために使用されます。詳細は Armadillo-IoT ゲートウェイ G4 製品マニュアルの「 Armadillo のソフトウェアの初期化 [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.11.0/ch09.html#ch.yakushima-fwinitialize]」を参照してください。

8.2. インストールディスクを作成する準備を行なう

インストールディスク作成前にいくつかの前準備が必要なことがあります。以下に必要な前準備とその手順について紹介しますので、クローン元の Armadillo 上で作業を実施してください。

8.2.1. Armadillo Base OS の更新

基本的に Armadillo Base OS は常に最新版を使用することを推奨します。開発中に Armadillo Base OS の新しいバージョンがリリースされている場合もありますので、クローンインストールディスクを作成する前に Armadillo Base OS のバージョンを最新版にしてください。

8.2.1.1. /etc/swupdate_preserve_file への追記

ここでは SWUpdate というアップデート機能を使用して Armadillo Base OS のアップデートを行ないますが、このアップデートを行なうと、/etc/swupdate_preserve_files に記載の無いファイルは消えてしまいます。Armadillo Base OS のルートファイルシステム上に消えてほしくないファイルを開発中に配置していた場合は、「図 8.1. 任意のファイルパスを/etc/swupdate_preserve_files に追記する」に示すコマンドを実行することで/etc/swupdate_preserve_files にそのファイルが追記され、アップデート後も保持し続けるようになります。

一部のファイルやディレクトリは初めから/etc/swupdate_preserve_files に記載されている他、`podman commit` したコンテナイメージについてもアップデート後に引き継がれるので、本ドキュメントのサンプルアプリケーションの場合は実行する必要はありません。

```
[armadillo /]# persist_file -p <ファイルのパス>
```

図 8.1 任意のファイルパスを/etc/swupdate_preserve_files に追記する

8.2.1.2. アップデートの実行

Armadillo-IoT ゲートウェイ G4 Armadillo Base OS [https://armadillo.atmark-techno.com/resources/software/armadillo-iot-g4/baseos]から「Armadillo-IoT ゲートウェイ G4 用 SWU イメージファイル」の URL をコピーして、「図 8.2. Armadillo Base OS をアップデートする」に示すコマンドを実行することで Armadillo Base OS を最新版にアップデートできます。

```
[armadillo /]# swupdate -d '-u https://armadillo.atmark-techno.com/files/downloads/armadillo-iot-g4/image/baseos-x2-[VERSION].swu'
```

図 8.2 Armadillo Base OS をアップデートする

正常に実行された場合は自動的に再起動します。

8.2.2. SWUpdate の初期設定

これまでに一度も SWUpdate によるアップデートを行なっていない場合、「11.1.1. SWUpdate による初回アップデート」の手順にしたがって初回アップデートを実行してください。

Armadillo Base OS 搭載製品は、USB メモリにアップデートイメージを配置して Armadillo に接続することで、SWUpdate によるアップデートが行われます。一度も SWUpdate を実行していない個体に関しては、第三者が作ったアップデートイメージであっても受け付けてしまいます。そのため、初回アップデートを行なっていない Armadillo を量産して出荷することはとても危険です。

SWUpdate による初回アップデートでは、自分以外が作成したアップデートイメージを受け付けなくするように設定します。

初回 SWUpdate 時に作成した認証鍵は、出荷後のリモートアップデートなどに使用できるため削除しないよう注意してください。

8.2.3. パスワードの確認と変更

Armadillo の各ユーザの初回ログイン時に、ログインパスワードを設定しました。開発中はログインしやすいような単純なパスワードにしていることがよくあるので、製品に適用しないようにこのタイミングで強固なパスワードに変更しておきましょう。

```
[armadillo /]# passwd ❶
Changing password for root
New password: ❷
Retype password: ❸
passwd: password for root changed by root

[armadillo /]# passwd atmark ❹
Changing password for atmark
New password: ❺
Retype password: ❻
passwd: password for atmark changed by root

[armadillo /]# persist_file /etc/shadow ❼
```

図 8.3 パスワードを変更する

❶ root ユーザのパスワードを変更します。

❷ 新しい root ユーザ用パスワードを入力します。

❸ 再度新しい root ユーザ用パスワードを入力します。

❹ atmark ユーザのパスワードを変更します。

❺ 新しい atmark ユーザ用パスワードを入力します。

❻ 再度新しい atmark ユーザ用パスワードを入力します。

❼ パスワードの変更を永続化させます。

8.2.4. 開発中のみ使用していたコンテナイメージの削除

開発中に使用していて運用時には不要なコンテナ及びコンテナイメージは、インストールディスク作成前に削除しておくべきです。

「図 8.4. 作成済みコンテナ一覧を表示する」に示すコマンドを実行することで作成したコンテナの一覧を取得できます。

```
[armadillo /]# podman ps -a
CONTAINER ID  IMAGE                          COMMAND     CREATED        STATUS                    ⏎
PORTS  NAMES
3ca118e9473b  docker.io/library/alpine:latest  /bin/sh     3 seconds ago  Exited (0) 3 seconds      ⏎
ago         vibrant_easley
9c908ab45ed8  localhost/abos-dev-guide:v1.0.0  /bin/bash   3 minutes ago  Exited (0) 5 months       ⏎
ago          sample_container
```

図 8.4 作成済みコンテナ一覧を表示する

基本的に運用時におけるコンテナは/etc/atmark/containers/*.conf ファイルによって自動的に作成されますので、「図 8.4. 作成済みコンテナ一覧を表示する」のコマンドによって取得したコンテナは全て削除して問題無いはずです。

```
[armadillo /]# podman rm sample_container ❶
[armadillo /]# podman rm 3ca118e9473b ❷
```

```
[armadillo /]# podman ps -a
CONTAINER ID  IMAGE  COMMAND  CREATED  STATUS  PORTS  NAMES
❸
```

図 8.5 コンテナを削除する

❶ コンテナの名前で削除するコンテナを指定しています。

❷ コンテナの ID で削除するコンテナを指定しています。

❸ 何も表示されず、全てのコンテナが削除されたことを確認します。

「図 8.6. コンテナイメージ一覧を表示する」に示すコマンドを実行することでコンテナイメージの一覧を取得できます。readonly 領域に保存されているコンテナイメージが 1 つでもある場合は、 R/O 列が表示されます。R/O 列が表示されない場合は、全てのコンテナイメージの R/O が false であることを意味しています。

```
[armadillo /]# podman images
REPOSITORY                 TAG         IMAGE ID      CREATED       SIZE        R/O
docker.io/library/alpine   latest      6e30ab57aeee  2 weeks ago   5.56 MB     false
docker.io/library/busybox  latest      3c19bafed223  5 days ago    1.64 MB     true
localhost/abos-dev-guide   v1.0.0      2394ea5f177f  5 hours ago   932 MB      false
```

図 8.6 コンテナイメージ一覧を表示する

ここでは運用時に必要な localhost/abos-dev-guide:v1.0.0 を除いた、その他のコンテナイメージを削除します。

R/O が false のイメージは podman rmi コマンドで削除できます。

```
[armadillo /]# podman rmi docker.io/library/alpine
Untagged: docker.io/library/alpine:latest
Deleted: 6e30ab57aeeef1ebca8ac5a6ea05b5dd39d54990be94e7be18bb969a02d10a3f
```

図 8.7 R/O が false のイメージを削除する

R/O が true のイメージは abos-ctrl podman-rw rmi コマンドで削除できます。

```
[armadillo /]# abos-ctrl podman-rw rmi docker.io/library/busybox:latest
Untagged: docker.io/library/busybox:latest
Deleted: 3c19bafed22355e11a608c4b613d87d06b9cdd37d378e6e0176cbc8e7144d5c6
```

図 8.8 R/O が true のイメージを削除する

8.2.5. 開発したコンテナイメージを tmpfs に移行する

開発中は「6.3.1. 開発前の準備」で行なったように、podman のデータは電源を切っても保持されるように eMMC に保存していました。開発中はこのままで問題ありませんが、運用する場合には eMMC への書き込みを最小限にする観点から、podman のデータの保存先を tmpfs に変更しておくことを推奨します。

「図 8.9. podman のデータの保存先を tmpfs に変更する」に示すコマンドを実行することで、eMMC に保存されている開発完了後のコンテナイメージを tmpfs モードでも読み取り専用で使用できるように変更できます。

```
[armadillo /]# abos-ctrl podman-storage --tmpfs
List of images configured on development storage:
REPOSITORY                TAG         IMAGE ID      CREATED       SIZE
localhost/abos-dev-guide  v1.0.0      2394ea5f177f  5 hours ago   932 MB

What should we do? ([C]opy (default), [N]othing, [D]elete)
C ❶
Delete subvolume (no-commit): '/mnt/containers_storage'
Replacing development images to readonly storage succeeded
Switching back to tmpfs container storage.
Successfully reverted podman storage to tmpfs

[armadillo /]# abos-ctrl podman-rw image list ❷
REPOSITORY                 TAG         IMAGE ID      CREATED       SIZE
localhost/abos-dev-guide   v1.0.0      2394ea5f177f  5 hours ago   932 MB
```

図 8.9 podman のデータの保存先を tmpfs に変更する

❶ C を入力し Enter を押下します。

❷ tmpfs モードでコンテナイメージが読み込めていることを確認します。

8.3. 開発したシステムをインストールディスクにする

Armadillo Base OS では、 abos-ctrl make-installer コマンドを実行することで、現在起動しているルートファイルシステム及びブートローダーをそのままインストールディスクイメージとして出力し、microSD カードに書き込むことができます。

abos-ctrl make-installer コマンドを実行する前に、Armadillo がインターネットに接続されており、かつ microSD カードが挿入されていることを確認してください。microSD カード内のデータはインストールディスク作成時に上書きされて消えてしまうので、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。

```
[armadillo /]# abos-ctrl make-installer
An installer system is already available on SD card. Use it? [Y/n] ❶

Would you like to create a windows partition?
That partition would only be used for customization script at the end of
install, leave at 0 to skip creating it.
Custom partition size (MB, [0] - 30026): 500 ❷
Checking and growing installer main partition
GPT data structures destroyed! You may now partition the disk using fdisk or
other utilities.
The operation has completed successfully.
Trying to install mkfs.exfat (exfatprogs) in memory from internet
fetch https://download.atmark-techno.com/alpine/v3.16/atmark/aarch64/APKINDEX.tar.gz
fetch https://dl-cdn.alpinelinux.org/alpine/v3.16/main/aarch64/APKINDEX.tar.gz
fetch https://dl-cdn.alpinelinux.org/alpine/v3.16/community/aarch64/APKINDEX.tar.gz
```

```
(1/1) Installing exfatprogs (1.1.3-r0)
Executing busybox-1.35.0-r14.trigger
OK: 159 MiB in 215 packages
exfatprogs version : 1.1.3
Creating exFAT filesystem(/dev/mmcblk1p2, cluster size=32768)

Writing volume boot record: done
Writing backup volume boot record: done
Fat table creation: done
Allocation bitmap creation: done
Upcase table creation: done
Writing root directory entry: done
Synchronizing...

exFAT format complete!
Resize device id 1 (/dev/mmcblk1p1) from 400.00MiB to max
Currently booted on /dev/mmcblk2p1
Copying boot image
Copying rootfs
314572800 bytes (315 MB, 300 MiB) copied, 11 s, 28.5 MB/s
300+0 records in
300+0 records out
314572800 bytes (315 MB, 300 MiB) copied, 11.0192 s, 28.5 MB/s
Copying /opt/firmware filesystem
Copying appfs
At subvol app/snapshots/volumes
At subvol app/snapshots/boot_volumes
At subvol app/snapshots/boot_containers_storage
Cleaning up and syncing changes to disk...
Installer updated successfully!
```

図 8.10 開発完了後のシステムをインストールディスクイメージにする

❶ Enter キーを押下します。

❷ インストールディスク内にインストールログを保存したい場合など、自由に使用できる第 2 パーティションを指定したサイズ作成します。詳細は「8.3.1. インストール時に任意のシェルスクリプトを実行する」を参照してください。

「Installer updated successfully!」と表示されれば、正常に microSD カードにインストールディスクイメージを書き込むことができています。Armadillo から microSD カードを抜去してください。

これまでに一度も SWUpdate によるアップデートを行なっていない場合、abos-ctrl make-installer 実行時に以下のようなエラーメッセージが出力され実行に失敗します。

```
ERROR: Refusing to build installer image with public onetime swupdate
certificate installed
ERROR: Please install initial_setup.swu (from mkswu --init) first
```

このメッセージが出力された場合は、「11.1.1. SWUpdate による初回アップデート」の手順にしたがって初回アップデートを実行してください。SWUpdate による初回アップデートでは、自分以外が作成したアップデートイメージを受け付けなくするように設定します。

Armadillo Base OS 搭載製品は、USB メモリにアップデートイメージを配置して Armadillo に接続することで、SWUpdate によるアップデートが行われます。一度も SWUpdate を実行していない個体に関しては、第三者が作ったアップデートイメージであっても受け付けてしまいます。そのため、初回アップデートを行なっていない Armadillo を量産して出荷することはとても危険です。

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

8.3.1. インストール時に任意のシェルスクリプトを実行する

作成したインストールディスクの所定の場所に、`installer_overrides.sh` というファイル名でシェルスクリプトを配置することで、インストール処理の前後で任意の処理を行なうことができます。

`installer_overrides.sh` に記載された「表 8.1. インストール中に実行される関数」に示す 3 つの名前の関数のみが、それぞれ特定のタイミングで実行されます。

表 8.1 インストール中に実行される関数

関数名	備考
preinstall	インストール中、eMMC のパーティションが分割される前に実行されます。
postinstall	send_log 関数を除く全てのインストール処理の後に実行されます。
send_log	全てのインストール処理が完了した後に実行されます。指定した場所にインストールログを保存できます。

`installer_overrides.sh` を書くためのサンプルとして、インストールディスクイメージの第 1 パーティション及び、「8.3. 開発したシステムをインストールディスクにする」で作成したのであれば第 2 パーティション直下に `installer_overrides.sh.sample` を用意してあります。このサンプルをコピーして編集するなどして、行ないたい処理を記述してください。

作成した `installer_overrides.sh` は、インストールディスクの第 1 パーティション(ラベル名は "rootfs_0")か、「8.3. 開発したシステムをインストールディスクにする」で作成したのであれば第 2 パーティション(ラベル名は"INST_DATA")の直下に配置することで実行されます。両方に配置した場合は、第 2 パーティションに配置した記述が適用されます。

インストールディスクの第 1 パーティションは btrfs、第 2 パーティションは exfat でフォーマットされているため、第 2 パーティションのみ Windows PC でもマウントして読み書きすることができます。

製造担当者が `installer_overrides.sh` を記述する場合に、仮に Windows PC しか作業環境がない場合でも、第 2 パーティションを作成しておくことで作業を行なうことができるというメリットもあります。

これを利用することで、複数台の Armadillo に対してそれぞれに異なる固定 IP アドレスを設定したり、各種クラウドへの接続鍵などを個体ごとに配置したりしたいなど、個体ごとに異なる設定を行なうなど柔軟な製造を行なうことも可能です。詳しくは「10.2.1. インストール時に任意の処理を行なう」で紹介します。

8.4. インストールディスクの動作確認

作成したインストールディスクの動作確認は必ず行なってください。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。

まず、インストール先の Armadillo にインストールディスクイメージを書き込んだ microSD カードを挿入してください。その後、JP1 をショートさせてから電源を入れます。Armadillo がインストールディスクから起動し、自動的にインストールスクリプトが動作します。

```
U-Boot SPL 2020.04-at8 (May 26 2022 - 08:44:15 +0000)
DDRINFO: start DRAM init
DDRINFO: DRAM rate 4000MTS
DDRINFO:ddrphy calibration done
DDRINFO: ddrmix config done
Normal Boot
Trying to boot from BOOTROM
image offset 0x8000, pagesize 0x200, ivt offset 0x0
NOTICE:  BL31: v2.4(release):lf-5.10.y-1.0.0-0-gba76d337e
NOTICE:  BL31: Built : 03:47:11, Dec 21 2021
:(省略)
Requesting system poweroff
[   23.441711] imx2-wdt 30280000.watchdog: Device shutdown: Expect reboot!
[   23.449006] reboot: Power down
```

図 8.11 インストールディスク使用時のログ

しばらくすると「reboot: Power down」と表示されるので、Armadillo の電源を切ります。その後 Armadillo から microSD カードを抜き、JP1 をオープンにします。再度電源を投入することで、インストールは完了です。

実際にクローンした Armadillo が想定した通りの動作をすることを確認してください。

動作確認の際には、クローンした Armadillo の起動時に「図 8.12. インストールに失敗した Armadillo に出力されるメッセージ」に示す末尾 3 行の WARNING メッセージが出力されないことを必ず確認してください。

```
armadillo login: root
Password:
Welcome to Alpine!

The Alpine Wiki contains a large amount of how-to guides and general
information about administrating Alpine systems.
See <http://wiki.alpinelinux.org/>.

:(省略)

You can change this message by editing /etc/motd.
WARNING: swupdate onetime public certificate is present, anyone can access this device
```

```
WARNING: Please install initial_setup.swu (from mkswu --init),
WARNING: or remove the first certificate from /etc/swupdate.pem
```

図 8.12 インストールに失敗した Armadillo に出力されるメッセージ

上記のメッセージが出力された場合は、インストールディスクによるイメージ書き込みが正しくできていません。「8.2. インストールディスクを作成する準備を行なう」からインストールディスクの作成をやり直してください。

9. 量産用 swu イメージの作成

本章では、開発を経て完成したソフトウェアを含む Armadillo を量産するために swu イメージを作成する手順について説明します。作成した swu イメージを製造担当者に渡すことで、簡単かつミスも少なくイメージ書き込み済み Armadillo を量産することが可能です。

9.1. swu イメージとは

Armadillo Base OS は、SWUpdate というアップデート機能を持っており、Armadillo 内の Linux カーネルイメージやユーザーランドなどを容易に書き換えることが可能です。swu イメージは、SWUpdate を行なうために必要なアップデート先のイメージなどが組み込まれたパッケージファイルを指します。

swu イメージを Armadillo に適用する方法として、swu イメージが書き込まれた USB メモリや SD カードを Armadillo に挿入する方法や、ネットワーク経由で swu イメージを取得する方法などがあります。詳しくは Armadillo-IoT ゲートウェイ G4 製品マニュアルの「Armadillo のソフトウェアをアップデートする [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.11.0/ch09.html#ch.yakushima-softwareupdate]」を参照してください。今回の例では USB メモリを用いた方法で進めていきます。

9.2. SWUpdate で量産を行なう際の流れ

アットマークテクノから出荷された量産向けの Armadillo は通常、標準イメージが初めから書き込まれています。これを SWUpdate によってアップデートする形で、ユーザー製品の量産用のイメージを書き込みます。

SWUpdate を用いて量産時にイメージ書き込みを行なう際の、一般的な作業の流れは以下の通りです。

1. 初回アップデート用 swu イメージの作成
2. 書き込むイメージの準備
3. desc ファイルの記述
4. swu イメージの作成
5. SWUpdate のテスト

各手順については、以下で順番に説明します。

9.3. 初回アップデート用 swu イメージの作成

まず初めに、SWUpdate の初回アップデートの準備を行ないます。

一度も SWUpdate を実行していない Armadillo は、第三者が作った swu イメージであっても受け付けてアップデートを実行してしまいます。そのため、初回アップデートを行なっていない Armadillo を使用した製品を量産して出荷することはとても危険です。SWUpdate による初回アップデートでは、Armadillo 内に専用の認証鍵を配置し、自分が作成した swu イメージ以外を受け付けなくするように設定しますので、必ず実行してください。

初回アップデート用 swu イメージの作成手順については、「11.1.1. SWUpdate による初回アップデート」を参照して「2. mkswu の初期設定」までを実行してください。今回は量産時に初回アップデートとソフトウェア書き込みを 1 度の SWUpdate で行なうため、今の段階では initial_setup.swu を生成した後に SWUpdate を実行する必要はありません。

9.4. 書き込むイメージの準備

次に、開発したソフトウェアを swu イメージにしていきます。

swu イメージは ATDE 上で作成するので、swu イメージに組み込むファイルを ATDE 内に配置しておいてください。

以下では「6. サンプルアプリケーションの作成」で作成したソフトウェアを Armadillo に組み込むことを例として考えます。最低限以下の 2 つを Armadillo に書き込むことで、起動時にアプリケーションを含むコンテナが自動的に実行されます。

1. abos-dev-guide-v1.0.0.tar(動かしたいアプリケーションを含むコンテナイメージ)
2. sample_container.conf(コンテナ自動実行用 conf ファイル)

ここでは、これらのファイルを ATDE に/home/atmark/mkswu/abos-dev-guide ディレクトリを作成して配置しておきます。

```
[ATDE ~/mkswu/abos-dev-guide]$ ls
abos-dev-guide-v1.0.0.tar  sample_container.conf
```

図 9.1 Armadillo に書き込みたいソフトウェアを ATDE に配置

9.5. desc ファイルの記述

desc ファイルの記述方法などの詳細については、Armadillo-IoT ゲートウェイ G4 製品マニュアルの「mkswu の desc ファイル [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.11.0/ch09.html#sct.mkswu-desc]」を参照してください。

SWUpdate 実行時に、「9.4. 書き込むイメージの準備」で挙げたファイルを Armadillo に書き込むような swu イメージを生成します。

swu イメージを作成するためには、SWUpdate 時に実行する処理等を示した desc ファイルを記述し、その desc ファイルを入力として mkswu コマンドを実行することで、swu イメージが出来上がります。

本書のサンプルアプリケーションを例とすると、Armadillo-IoT ゲートウェイ G4 においては以下のような abos-dev-guide.desc ファイルを作成します。

```
[ATDE ~/mkswu/abos-dev-guide]$ cat abos-dev-guide.desc
swdesc_option component=abos-dev-container
swdesc_option version=1
swdesc_option POST_ACTION=poweroff ❶

starting_cmd="cd /sys/class/leds/led1 && echo timer > trigger && echo 111 | tee delay_on > delay_off"
success_cmd="cd /sys/class/leds/led1 && echo none > trigger && cat max_brightness > brightness"
fail_cmd="cd /sys/class/leds/led1 && echo heartbeat > trigger"
```

```
swdesc_option NOTIFY_STARTING_CMD="$starting_cmd" ❷
swdesc_option NOTIFY_SUCCESS_CMD="$success_cmd" ❸
swdesc_option NOTIFY_FAIL_CMD="$fail_cmd" ❹

swdesc_embed_container "abos-dev-guide-v1.0.0.tar" ❺
swdesc_files --extra-os --dest /etc/atmark/containers/ "sample_container.conf" ❻
```

図 9.2 desc ファイルの記述例(Armadillo-IoT ゲートウェイ G4 の場合)

❶ SWUpdate 完了後に電源を切るように設定します。

❷ SWUpdate 中に LED3 を点滅させるように設定します。

❸ SWUpdate 成功時に LED3 を点灯させるように設定します。

❹ SWUpdate 失敗時に LED3 を周期的な 2 回の短点灯させるように設定します。

❺ コンテナイメージファイルを swu イメージに組み込み、Armadillo に転送します。

❻ コンテナ自動起動用 conf ファイルを Armadillo に転送します。

Armadillo-IoT ゲートウェイ A6E の場合は、初めから SWUpdate の実行時に LED が点滅する設定がされており、desc ファイルで LED の設定を行なう必要がないため、以下のような abos-dev-guide.desc ファイルとなります。

```
[ATDE ~/mkswu/abos-dev-guide]$ cat abos-dev-guide.desc
swdesc_option component=abos-dev-container
swdesc_option version=1
swdesc_option POST_ACTION=poweroff ❶

swdesc_embed_container "abos-dev-guide-v1.0.0.tar" ❷
swdesc_files --extra-os --dest /etc/atmark/containers/ "sample_container.conf" ❸
```

図 9.3 desc ファイルの記述例(Armadillo-IoT ゲートウェイ A6E の場合)

❶ SWUpdate 完了後に電源を切るように設定します。

❷ コンテナイメージファイルを swu イメージに組み込み、Armadillo に転送します。

❸ コンテナ自動起動用 conf ファイルを Armadillo に転送します。

9.5.1. イメージ書き込み時にログを残したい場合

SWUpdate によるイメージ書き込み時にログを残したい場合には、abos-dev-guide.desc を修正します。前述の通り、Armadillo-IoT ゲートウェイ A6E においては、初めから SWUpdate の実行時に LED が点滅する設定がされており、desc ファイルで LED の設定を行なう必要がないため、desc ファイルの記述も若干異なりますので、それぞれの記述例を以下に示します。

```
[ATDE ~/mkswu/abos-dev-guide]$ cat abos-dev-guide.desc
swdesc_option component=abos-dev-container
swdesc_option version=1
swdesc_option POST_ACTION=poweroff

starting_cmd="cd /sys/class/leds/led1 && echo timer > trigger && echo 111 | tee delay_on > delay_off"
```

```
success_cmd="mount -o rw,remount /dev/sda1 && ❶
             cp /var/log/messages /mnt/messages.¥$(date +%Y%m%d-%H%M%S).log && ❷
             cd /sys/class/leds/led1 && echo none > trigger && cat max_brightness > brightness"
fail_cmd="cd /sys/class/leds/led1 && echo heartbeat > trigger"

swdesc_option NOTIFY_STARTING_CMD="$starting_cmd"
swdesc_option NOTIFY_SUCCESS_CMD="$success_cmd"
swdesc_option NOTIFY_FAIL_CMD="$fail_cmd"

swdesc_embed_container "abos-dev-guide-v1.0.0.tar"
swdesc_files --extra-os --dest /etc/atmark/containers/ "sample_container.conf"
```

図 9.4 ログを残す際の desc ファイルの記述例(Armadillo-IoT ゲートウェイ G4 の場合)

❶ USB メモリは readonly でマウントされているため、書き込みを許可して再マウントします。

❷ /var/log/messages を USB メモリに保存します。

```
[ATDE ~/mkswu/abos-dev-guide]$ cat abos-dev-guide.desc
swdesc_option component=abos-dev-container
swdesc_option version=1
swdesc_option POST_ACTION=poweroff

success_cmd="mount -o rw,remount /dev/sda1 && ❶
             cp /var/log/messages /mnt/messages.¥$(date +%Y%m%d-%H%M%S).log && ❷
             cd /sys/class/leds/sys && echo none > trigger && cat max_brightness > brightness" ❸

swdesc_option NOTIFY_SUCCESS_CMD="$success_cmd"

swdesc_embed_container "abos-dev-guide-v1.0.0.tar"
swdesc_files --extra-os --dest /etc/atmark/containers/ "sample_container.conf"
```

図 9.5 ログを残す際の desc ファイルの記述例(Armadillo-IoT ゲートウェイ A6E の場合)

❶ USB メモリは readonly でマウントされているため、書き込みを許可して再マウントします。

❷ /var/log/messages を USB メモリに保存します。

❸ SWUpdate 成功時に LED3 を点灯させるように設定します。

上記の例では、SWUpdate 成功時にログを USB メモリ内に保存します。失敗時にはログは保存されませんが、自動で poweroff することもないので、その場でシリアルインターフェース経由で原因を調査することができます。

9.6. swu イメージの作成

「9.5. desc ファイルの記述」で作成した desc ファイルと、initial_setup.desc ファイルから swu イメージを生成し、USB メモリに書き込みます。

desc ファイルから swu イメージを生成するには、 mkswu コマンドを実行します。「図 9.6. mkswu コマンドの実行」は、initial_setup.desc と abos-dev-guide.desc から abos-dev-guide_setup.swu を生成をする例です。

```
[ATDE ~/mkswu/abos-dev-guide]$ mkswu -o abos-dev-guide_setup.swu ¥
../initial_setup.desc abos-dev-guide.desc

../initial_setup.desc を組み込みました。
abos-dev-guide.desc を組み込みました。
abos-dev-guide_setup.swu を作成しました。
```

図 9.6 mkswu コマンドの実行

このように、複数の desc ファイルをまとめてひとつの swu イメージを作成することが可能です。SWUpdate 時には mkswu コマンドで指定した順番に実行されるので、「図 9.6. mkswu コマンドの実行」の例では、initial_setup の処理の後に abos-dev-guide の処理が実行されます。

生成された abos-dev-guide_setup.swu を USB メモリに書き込みます。コマンド実行例を「図 9.7. 各イメージを USB メモリに配置」に示します。書き込み先である USB メモリは/media/atmark/USBDRIVE にマウントされているものとします。

```
[ATDE ~/mkswu/abos-dev-guide]$ cp abos-dev-guide_setup.swu /media/atmark/USBDRIVE/
```

図 9.7 各イメージを USB メモリに配置

USB メモリをアンマウントした後に、ATDE から USB メモリを取り外します。

```
[ATDE ~/]$ sudo umount /media/atmark/USBDRIVE
// 完了後に ATDE から USB メモリを取り外す
```

図 9.8 USB メモリをアンマウント

9.7. SWUpdate の実行とテスト

ここまでの手順で作成した swu イメージを実際に量産工程で使用する前に、誤ったイメージを書きこんでいないか、書き込んだイメージが正しく動作するかなどを確認するための十分なテストを必ず実施してください。そのために一度初期状態の Armadillo を 1 台用意し、それに作成した swu イメージを用いて SWUpdate を実際に実行してみることで、動作確認をしてみます。

9.7.1. SWUpdate の実行

標準イメージが書き込まれた初期状態の Armadillo-IoT ゲートウェイ G4 を 1 台用意し、実際に SWUpdate を実行してみます。

Armadillo に電源ケーブルと swu イメージが書き込まれた USB メモリを接続します。

量産時に行なう処理によっては他にも接続すべきものがある場合があります。例えば、SWUpdate 中にネットワーク経由でファイルを取得する処理がある場合にネットワークに接続するために Ethernet ケーブルを接続しておく必要がある、シリアルインターフェースからのログを残す為に microUSB ケーブルを接続する必要があるなど、実際の量産時の動きに合わせて必要なものを接続しておいてください。

USB メモリと電源ケーブルを接続し、電源を投入すると自動で起動して SWUpdate が開始されます。

「9.5. desc ファイルの記述」で紹介した desc ファイルをもとに作成された swu イメージでは、SWUpdate の進捗によって LED3 の状態が変化します。

SWUpdate の進捗と LED3 の状態の対応表を「表 9.1. SWUpdate の進捗と LED3 の状態の対応表」に示します。

表 9.1 SWUpdate の進捗と LED3 の状態の対応表

SWUpdate の進捗	LED3 の状態
電源投入～SWUpdate 開始前	点灯
SWUpdate 中	等間隔の点滅
SWUpdate 成功	消灯(Armadillo の電源 OFF)
SWUpdate 失敗	周期的な 2 回の短点灯

SWUpdate が成功した場合、Armadillo の電源が OFF になり LED3 が消灯します。Armadillo から USB メモリを抜去してください。

SWUpdate が失敗した場合、LED3 は周期的な 2 回の短点灯状態になり、電源は自動的に OFF になりません。シリアルインターフェース経由で Armadillo を操作し、SWUpdate の実行ログから失敗原因を調査・修正して、再度お試しください。

9.7.2. SWUpdate 後のテスト

SWUpdate 後の Armadillo が正しく動作するか、実際に動かして確認してみます。

本ドキュメントの例では、HDMI モニタと USB カメラを接続する必要があるので、電源投入前に接続しておきます。ここはシステムによって異なりますが、可能であれば本番環境になるべく近い形でのテストをお勧めします。

再度電源を投入して、期待したアプリケーションが動作することを確認してください。

ここまで完了したならば、量産用の swu イメージは完成です。

10. 製造・量産する

本章では、開発が完了した Armadillo Base OS 搭載製品を量産する際に役立つ情報を、量産製造の流れに沿って紹介します。

2022 年 6 月現在、アットマークテクノから出荷される Armadillo-IoT ゲートウェイ G4 に書き込まれるソフトウェアイメージはユーザが指定できず、基本的にその時の最新版の標準イメージが書き込まれます。

Armadillo-IoT ゲートウェイ G4 は、今後 BTO によるユーザーイメージの書き込みなどのカスタマイズに対応する予定です。

アットマークテクノが Armadillo-IoT ゲートウェイ G4 を出荷するまでの流れは以下の通りです。

1. 基板製造
2. 各種インターフェースの検査
3. 標準イメージの書き込み
4. 標準イメージでの起動検査
5. ケースへの組み込みなどの組立(ケースは製品モデルによっては付属しません)
6. 同梱品と共に梱包
7. 出荷

複数台の Armadillo-IoT ゲートウェイ G4 の製造を行なうのであれば、基本的にユーザの自社で製造工程を作り量産製造を行なう必要があります。

10.1. ユーザ製品の製造・量産の流れ

アットマークテクノが想定しているユーザの自社での製造工程は大まかに以下の通りです。

1. ケース付属の場合はケースを開ける(インストールディスクを使用する場合)
2. インストールディスクや SWUpdate を使用して eMMC 内のイメージを任意のものに書き換える
3. 外部センサなどの付属品を Armadillo に組み付ける
4. ケースを開けた場合はケースを閉じる
5. 動作確認等の検査する
6. 梱包・出荷する

前述の通り 2022 年 6 月現在、アットマークテクノから出荷される Armadillo-IoT ゲートウェイ G4 には標準イメージが書き込まれており、開発後のソフトウェアを書き込むサービスは展開しておりませ

ん。そのため、Armadillo-IoT ゲートウェイ G4 を用いた製品を開発して製造・量産するためには、ユーザ製品の製造工程でイメージの書き込みを行なう必要があります。

以下では Armadillo Base OS 搭載製品の製造時にイメージの書き込みを行なう手順について説明します。

インストールディスクを用いてイメージ書き込みを行なう場合には「10.2. クローンインストールディスクを用いてイメージ書き込みを行なう」を、SWUpdate を用いてイメージ書き込みを行なう場合には「10.3. SWUpdate を用いてイメージ書き込みを行なう」を参照してください。

10.2. クローンインストールディスクを用いてイメージ書き込みを行なう

本節では、インストールディスクを用いてユーザ製品の製造工程内で Armadillo にイメージ書き込みを行なう手順について説明します。インストールディスクについては、「8.1. インストールディスクとは」を参照してください。

開発の最終段階である「8. 量産用インストールディスクの作成」の手順で、開発完了後の Armadillo を複製するためのクローンインストールディスクを作成しました。このインストールディスクを使用することで、開発完了後の Armadillo と同等のイメージを他の Armadillo に書き込むことができます。

10.2.1. インストール時に任意の処理を行なう

量産する Armadillo ひとつひとつに重複しない固定 IP アドレスを設定したいなど、製造時に個体ごとに異なる設定を行ないたい場合があります。そのような場合には、インストールディスクにはインストール時に任意のシェルスクリプトを実行する機能を利用することで実現できます。

インストールディスクには任意のサイズの exfat でフォーマットされた、ユーザが自由に利用できる第 2 パーティションを作ることができます。作成方法は「8.3. 開発したシステムをインストールディスクにする」を参照してください。通常、インストールディスクの中身は Windows PC では見ることはできませんが、exfat でフォーマットされたパーティションは Windows PC であっても読み書きすることができます。

第 2 パーティションを持つインストールディスクを PC に接続すると、「 `INST_DATA` 」というボリューム名でマウントされます。

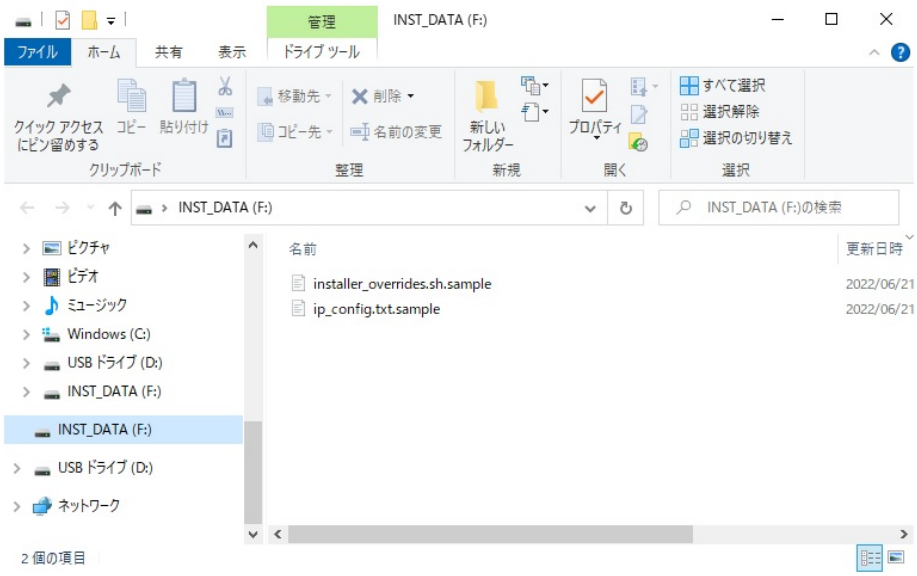
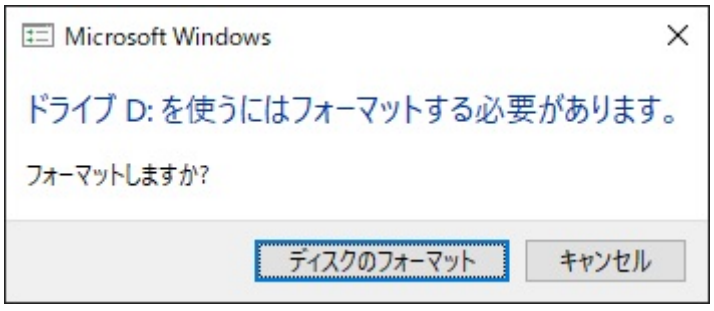


図 10.1 Windows PC にインストールディスクを接続する

PC にインストールディスクを接続しても INST_DATA がマウントされない場合は、インストールディスクを作る際に第 2 パーティションを作成していない可能性があります。インストールディスク作成者に第 2 パーティションを作成するように問い合わせてください。

Windows PC にインストールディスクを接続すると以下のようなメッセージが表示されますが、Windows で扱えないファイルシステムであるだけでデータは書き込まれているので、フォーマットは行わないでください。



この第 2 パーティション直下に installer_overrides.sh という名前でシェルスクリプトを配置することで、インストール処理中に任意の処理を行なうことが可能です。installer_overrides.sh については「8.3.1. インストール時に任意のシェルスクリプトを実行する」を参照してください。

INST_DATA の中にサンプルファイルとして installer_overrides.sh.sample と ip_config.txt.sample が配置されています。特に installer_overrides.sh.sample の内容は、 installer_overrides.sh を作成する際に参考にできますので確認してみてください。

10.2.1.1. 個体ごとに異なる固定 IP アドレスを設定する

インストール時に任意のシェルスクリプトを実行できる機能を利用して、複数の Armadillo に対して異なる固定 IP アドレスを割り当てる例を紹介します。

INST_DATA 内の installer_overrides.sh.sample と ip_config.txt.sample は個体ごとに異なる IP アドレスを割り振る処理を行なうサンプルファイルです。それぞれ installer_overrides.sh と ip_config.txt にリネームすることで、 ip_config.txt に記載されている条件の通りに個体ごとに異なる固定 IP アドレスを設定することができます。全てをここでは説明しませんので、詳細はそれぞれのファイル内の記述も参照してください。

今回はそれぞれのファイルの内容は変更せず使用します。サンプルそのままですが、 ip_config.txt の内容を「図 10.2. ip_config.txt の内容」に示します。

```
# mandatory first IP to allocate, inclusive
START_IP=10.3.4.2 ❶

# mandatory last IP to allocate, inclusive
END_IP=10.3.4.249 ❷

# netmask to use for the IP, default to 24
#NETMASK=24 ❸

# Gateway to configure
# not set if absent
GATEWAY=10.3.4.1 ❹

# DNS servers to configure if present, semi-colon separated list
# not set if absent
DNS="1.1.1.1;8.8.8.8" ❺

# interface to configure, default to eth0
#IFACE=eth0 ❻
```

図 10.2 ip_config.txt の内容

❶ このインストールディスクで割り振る IP アドレスの範囲の始まりを指定します。

❷ このインストールディスクで割り振る IP アドレスの範囲の終わりを指定します。

❸ ネットマスクを指定します。指定しない場合は 24 になります。デフォルトでコメントアウトされています。

❹ ゲートウェイアドレスを指定します。

❺ DNS アドレスを指定します。セミコロンで区切ることでセカンダリアドレスも指定できます。

❻ IP アドレスの設定を行なうインターフェースを指定します。指定しない場合は eth0 になります。デフォルトでコメントアウトされています。

インストール作業の並列化の為に、複数枚のインストールディスクで固定 IP アドレスを割り振る場合は、それぞれのインストールディスクが割り振る IP アドレスの範囲が被らないように ip_config.txt を設定してください。

これらのファイルを配置したインストールディスクでインストールを実行した Armadillo が、正しく設定できていることを確認します。

```
[armadillo /]# ip addr show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP qlen 1000
    link/ether 00:11:22:33:44:55 brd ff:ff:ff:ff:ff:ff
    inet 10.3.4.2/24 brd 10.3.4.255 scope global noprefixroute eth0
       valid_lft forever preferred_lft forever
    inet6 ffff::ffff:ffff:ffff:ffff/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

図 10.3 IP アドレスの確認

また、サンプルスクリプトをそのまま使用すると、インストールディスクの第 2 パーティションに allocated_ips.csv というファイルが生成されます。このファイルには、このインストールディスクを使用して IP アドレスの設定を行なった個体のシリアル番号、MAC アドレス、設定した IP アドレスが追記されていきます。

```
SN,MAC,IP
00C700010009,00:11:22:33:44:55,10.3.4.2
```

図 10.4 allocated_ips.csv の内容

2 台目以降の Armadillo にこのインストールディスクで IP アドレスの設定を行なう際に、 allocated_ips.csv を参照して次に割り振る IP アドレスを決めますので、誤って削除しないように注意してください。

10.2.1.2. インストール実行時のログを保存する

installer_overrides.sh 内の send_log 関数は、インストール処理の最後に実行されます。インストールしたルートファイルシステムやファームウェアのチェックサムなどの情報が記録されたログファイルのパスが LOG_FILE に入るため、この関数内でインストールディスクの第 2 パーティションに保存したり、外部のログサーバにアップロードしたりすることが可能です。

「図 10.5. インストールログを保存する」は、インストールディスクの第 2 パーティションにインストールログを保存する場合の send_log 実装例です。

```
send_log() {
        : "This function is called after aggregating logs for archival"
        local LOG_FILE="$1"

        if [ -n "$USER_MOUNT" ]; then
                        mount /dev/mmcblk1p2 "$USER_MOUNT" ❶
                        cp $LOG_FILE $USER_MOUNT/${SN}_install.log ❷
                        umount "$USER_MOUNT" ❸
```

```
        fi
}
```

図 10.5 インストールログを保存する

❶ send_log 関数中では、SD カードの第 2 パーティション(/dev/mmcblk1p2)はマウントされていないのでマウントします。

❷ ログファイルを <シリアル番号>_install.log というファイル名で第 2 パーティションにコピーします。

❸ 第 2 パーティションをアンマウントします。

これらの変更を行なったインストールディスクでインストールを実行した後に、インストールディスクを PC などに接続して正しくログを保存できていることを確認してください。保存したログファイルの中身の例を「図 10.6. インストールログの中身」に示します。

```
RESULT:OK
abos-ctrl make-rootfs on Tue Jun 21 17:57:07 JST 2022 4194304 6b8250df711de66b
abos-ctrl make-rootfs on Tue Jun 21 17:57:24 JST 2022 314572800 58a9b6664158943e
firm 8e9d83d1ba4db65d
appfs 5108 1fa2cbaff09c2dbf
```

図 10.6 インストールログの中身

10.2.2. インストールを実行する

インストールディスクを使用してインストールを実行する手順は以下の通りです。

1. Armadillo-IoT ゲートウェイ G4 がケースに組み込まれている場合はケース(上)を取り外します。
2. Armadillo-IoT ゲートウェイ G4 の CON1 にインストールディスクイメージが書き込まれた microSD カードを挿入します
3. JP1 をショート(SD ブートに設定)してから電源を投入します。
   - 電源が入ると、LED3 及び LED4 が点灯します。
4. イメージサイズなどによって前後しますが、「8. 量産用インストールディスクの作成」で作成したインストールディスクでは、1 分程度でインストール処理が完了しました。
   - インストールが完了すると LED3 及び LED4 が消灯します。
   - コンソールを接続している場合は、 `reboot: Power down` と表示されます。
5. ケース付属の場合は、ケース(上)を取り付けます。

Armadillo-IoT ゲートウェイ G4 のケースの取り外し方及び、取り付け方は Armadillo-IoT ゲートウェイ G4 製品マニュアルの「オプションケース(金属製) [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.11.0/ch13.html#sct.metal-case]」を参照してください。

以上でインストールディスクを使用した Armadillo の複製が完了しました。上記手順を繰り返して、同じイメージが書き込まれた Armadillo を量産することができます。

10.3. SWUpdate を用いてイメージ書き込みを行なう

本節では、SWUpdate を用いてユーザ製品の製造工程内で Armadillo にイメージ書き込みを行なう手順について説明します。SWUpdate については、「9.1. swu イメージとは」を参照してください。

開発の最終段階である「9. 量産用 swu イメージの作成」の手順で、開発完了後の Armadillo を複製するための swu イメージを作成しました。この swu イメージを使用することで、開発完了後の Armadillo と同等のイメージを他の Armadillo に書き込むことができます。

10.3.1. SWUpdate の処理内容を変更したい場合

SWUpdate は基本的に swu イメージ内に全てのファイルが組み込まれており、後から処理内容やイメージを変更したい場合には swu イメージを作り直す必要があります。

swu イメージの作成方法については「9. 量産用 swu イメージの作成」を参照してください。

10.3.2. インストールを実行する

SWUpdate を実行する手順は以下の通りです。

1. Armadillo-IoT ゲートウェイ G4 の CON4 に swu イメージが書き込まれた USB メモリを挿入します
2. 電源を投入します。
   - 電源が入ると、LED3 が点灯します。
3. SWUpdate が開始されると、進捗によって LED3 の状態が変化します。「表 9.1. SWUpdate の進捗と LED3 の状態の対応表」を参照してください。
4. SWUpdate で行なう処理や、各種ファイルサイズなどによって前後しますが、「9. 量産用 swu イメージの作成」で作成した swu イメージでは、2 分程度で SWUpdate 処理が完了しました。
   - SWUpdate が完了すると LED3 が消灯します。
   - コンソールを接続している場合は、 `reboot: Power down` と表示されます。

以上で SWUpdate を使用した Armadillo の複製が完了しました。上記手順を繰り返して、同じイメージが書き込まれた Armadillo を量産することができます。

11. Appendix

この章では、サンプルアプリケーション開発内で扱いきれなかった、Armadillo Base OS での開発に関わる情報を紹介します。

11.1. SWUpdate を用いてソフトウェアをアップデートする

Armadillo Base OS ではソフトウェアのアップデート方法として SWUpdate を利用できます。SWUpdate について詳細は Armadillo-IoT ゲートウェイ G4 の製品マニュアルの「Armadillo のソフトウェアをアップデートする [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.11.0/ch09.html#ch.yakushima-softwareupdate]」を参照してしてください。

今回の例では USB メモリを使用してアップデートしますので、4GB 以上の USB メモリを用意してください。

11.1.1. SWUpdate による初回アップデート

SWUpdate では、ソフトウェアアップデートを行うために swu イメージというファイルを使用しますので、まずは開発用 PC(本ドキュメントでは ATDE)で swu イメージを作成します。

出荷時の Armadillo は署名されていない swu イメージでも SWUpdate を行うようになっています。そのため、まずはじめに自分専用の署名鍵を生成して公開鍵を Armadillo に配置する作業を SWUpdate を用いて行うことで、自分が作成した swu イメージでしかアップデートを行わない Armadillo を作成します。

使用している Armadillo に、過去に一度でもこの初回アップデート作業を行っている場合は二度同じ作業を行うことはできず、する必要もありません。

1. mkswu の取得

   初めに、その swu イメージを作成するソフトウェアである mkswu をインストールします。

   ```
   [ATDE ~/]$ sudo apt update && sudo apt install mkswu
   ```

   図 11.1 mkswu の取得

2. mkswu の初期設定

   「図 11.2. mkswu の初期設定を行う」に示すコマンドを実行することで、swu パッケージの署名に用いる鍵や、公開鍵を Armadillo に書き込むための swu イメージを作成できます。

   ```
   [ATDE ~/]$ mkswu --init
   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: abos-guide-sample 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) y ❸
/home/atmark/mkswu/swupdate.aes-key を作成しました。
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。
[ATDE ~/]$ ls ~/mkswu ❾
initial_setup.desc  initial_setup.swu  mkswu.conf  swupdate.aes-key  swupdate.key ⏎
swupdate.pem
```

図 11.2 mkswu の初期設定を行う

❶ 会社や製品が分かるような任意の名称を指定します(今回は abos-guide-sample)。

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」を選択します。

❹ アットマークテクノが提供するアップデートイメージをインストールできるようにするかを選びます。例では有効にしています。

❺ ここで Armadillo の root ユーザーのパスワードを設定します。2 回入力してください。

❻ atmark ユーザーのパスワードも同様に設定します。2 回入力してください。

❼ Armadillo Base OS の定期アップデートの有効/無効を設定します。例では無効にしています。

❽ ABOS Web で使用するパスワードを設定します。

❾ 初期化処理の結果生成されたファイルを確認しています。swupdate.aes-key は swu イメージの暗号化を行うと作成されます。

3. SWUpdate の実行

上記の手順で生成された初回セットアップ用の swu イメージ(initial_setup.swu)を Armadillo に書き込みます。

ATDE に USB メモリを接続し、作成した swu イメージを配置します。

```
[ATDE ~/]$ df -h
Filesystem                  Size  Used Avail Use% Mounted on
:（省略）
/dev/sdb1                   15G   24K   14G    1% /media/atmark/USBDRIVE ❶

[ATDE ~/]$ cp ~/mkswu/initial_setup.swu /media/atmark/USBDRIVE/ ❷
[ATDE ~/]$ umount /media/atmark/USBDRIVE/ ❸
```

図 11.3 swu イメージの配置

❶ USB メモリがマウントされている場所を確認します。

❷ ファイルをコピーします。

❸ /media/atmark/USBDRIVE をアンマウントします。コマンド終了後に USB メモリを取り外してください。

swu イメージを配置した USB メモリを動作中の Armadillo に挿入すると、自動的にアップデートが開始され、アップデートが完了すると再起動します。

再起動後には root ユーザ、atmark ユーザともに、「図 11.2. mkswu の初期設定を行う」で設定したパスワードでログインできます。

これでこの Armadillo に公開鍵が配置され、手元の秘密鍵で作成された swu イメージでアップデート可能になりました。

Armadillo に公開鍵を配置した後に対応する鍵や、 `mkswu.conf` を紛失すると、当該の Armadillo に SWUpdate を行うことができなくなりますのでご注意ください。

11.2. 作成したコンテナを他の Armadillo に組み込む

「6.5. アプリケーションを作成する」では、Armadillo 上でコンテナイメージを作成する方法を紹介してきましたが、他の Armadillo など外部で作成されたコンテナイメージを別な Armadillo に組み込む方法を紹介します。

11.2.1. podman load でコンテナイメージを組み込む

「6.5.5. podman コンテナのエクスポート」で紹介した通り、 `podman images` コマンドで表示されるコンテナイメージは `podman save` コマンドを使用することでファイルとして保存することができました。

このファイルは、 `podman load` コマンドを使用することで、別な Armadillo 上であってもコンテナイメージとしてインポートすることができます。サンプルアプリケーションのコンテナイメージを使用した実行例を「図 11.4. コンテナイメージファイルをインポートする」に示します。

コンテナイメージのサイズはそのイメージの内容によって異なりますが、1GB を超えることもよくあります。OverlayFS 上で保存できるファイルサイズでは保存しきれないことがありますので、この方法でコンテナイメージをインポートする際には、イメージファイルは外部の USB メモリや SD カードに配置しておくことをお勧めします。

```
[armadillo /]# podman_switch_storage --disk ❶
[armadillo /]# mount /dev/sda1 /mnt && cd /mnt ❷
[armadillo /mnt]# wget https://download.atmark-techno.com/armadillo-iot-g4/example/armadillo-base-
os-dev-guide/abos-dev-guide-v1.0.0.tar ❸
[armadillo /mnt]# podman load -i abos-dev-guide-v1.0.0.tar ❹
[armadillo /mnt]# podman images abos-dev-guide ❺
REPOSITORY                TAG         IMAGE ID      CREATED       SIZE
localhost/abos-dev-guide  v1.0.0      05304c68979f  15 hours ago  917 MB
```

図 11.4 コンテナイメージファイルをインポートする

❶ podman コンテナイメージの保存先を eMMC に変更します。

❷ /dev/sda1 として認識されている USB メモリ(もしくは SD カード)を/mnt にマウントしディレクトリ移動します。

❸ USB メモリ上にサンプルアプリケーション用コンテナイメージをダウンロードします。

❹ `podman load` コマンドでコンテナイメージをインポートします。

❺ `podman images` コマンドでインポートできていることを確認します。

11.2.2. SWUpdate でコンテナイメージを組み込む

「6.5.5. podman コンテナのエクスポート」で紹介した、 `podman save` コマンドを用いて生成したコンテナイメージファイルは、SWUpdate によって他の Armadillo に適用することも可能です。以下では SWUpdate を用いてサンプルアプリケーションを含むコンテナイメージと、コンテナを自動実行するための sample_container.conf を Armadillo に書き込む手順を紹介します。

SWUpdate については「11.1. SWUpdate を用いてソフトウェアをアップデートする」を参照してください。以下の手順を実行するためには、予め「11.1.1. SWUpdate による初回アップデート」の手順を実行して SWUpdate の初回アップデートを実施しておく必要があります。

1. コンテナインストール用の swu イメージを作成する

   導入したいコンテナイメージを含んだ swu イメージを作成します。

   まず、~/mkswu/abos-dev-guide-descs ディレクトリを作成し、その中に Armadillo に組み込みたいコンテナイメージと、自動実行用の.conf ファイルを配置します。コンテナの自動実行については「6.5.6. podman コンテナとアプリケーションの自動実行」を参照してください。

   ```
   [ATDE ~/mkswu]$ mkdir -p abos-dev-guide-descs && cd abos-dev-guide-descs
   [ATDE ~/mkswu/abos-dev-guide-descs]$ wget https://download.atmark-techno.com/armadillo-iot-
   g4/example/armadillo-base-os-dev-guide/abos-dev-guide-v1.0.0.tar
   [ATDE ~/mkswu/abos-dev-guide-descs]$ wget https://download.atmark-techno.com/armadillo-iot-
   g4/example/armadillo-base-os-dev-guide/sample_container.conf
   ```

   図 11.5 作業用ディレクトリの作成と書き込むファイルのダウンロード

   次に、swu イメージ作成に用いる usb_container_sample.desc を以下のように作成します。

   ```
   [ATDE ~/mkswu/abos-dev-guide-descs]$ cat usb_container_sample.desc
   version=1
   ```

```
swdesc_usb_container "abos-dev-guide-v1.0.0.tar" ❶
swdesc_files --extra-os --dest /etc/atmark/containers sample_container.conf ❷
```

図 11.6 usb_container_sample.desc 作成例

❶ abos-dev-guide-v1.0.0.tar をコンテナイメージとして組み込みます。

❷ sample_container.conf を Armadillo 内の/etc/atmark/containers に配置します。

desc ファイルの詳細については Armadillo-IoT ゲートウェイ G4 の製品マニュアルの「 mkswu の desc ファイル [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.11.0/ch09.html#sct.mkswu-desc]」を参照してください。/usr/share/mkswu/examples 以下の desc ファイルはサンプルですので、そちらも参考にしてください。

swu イメージを作成します。

```
[ATDE ~/mkswu/abos-dev-guide-descs]$ mkswu usb_container_sample.desc
Enter pass phrase for /home/atmark/mkswu/swupdate.key: ❶
以下のファイルを USB メモリにコピーしてください：
'/home/atmark/mkswu/abos-dev-guide-descs/usb_container_sample.swu' '/home/atmark/mkswu/ ⏎
abos-dev-guide-descs/abos-dev-guide-v1.0.0.tar' '/home/atmark/mkswu/abos-dev-guide- ⏎
descs/.usb_container_sample/abos-dev-guide-v1.0.0.tar.sig'

usb_container_sample.swu を作成しました。
```

図 11.7 swu イメージの作成

❶ 「図 11.2. mkswu の初期設定を行う」で設定した証明書の鍵のパスワードを入力してください。

これでコンテナイメージをインストールし、Armadillo 起動時に自動実行するための swu イメージが完成しました。

2. USB メモリに swu イメージを書き込む

mkswu 実行時に表示された指示に従って、作成した swu イメージと付属のファイル群を USB メモリに配置します。ATDE に USB メモリを接続して以下を実行してください。

```
[ATDE ~/mkswu/abos-dev-guide-descs]$ df -h
Filesystem                Size  Used Avail Use% Mounted on
:（省略）
/dev/sdb1                  15G   24K   14G    1% /media/atmark/USBDRIVE ❶

[ATDE ~/mkswu/abos-dev-guide-descs]$ sudo cp usb_container_sample.swu ¥
abos-dev-guide-v1.0.0.tar ¥
.usb_container_sample/abos-dev-guide-v1.0.0.tar.sig ¥
/media/atmark/USBDRIVE/ ❷
[ATDE ~/mkswu/abos-dev-guide-descs]$ umount /media/atmark/USBDRIVE/ ❸
```

図 11.8 アップデート用ファイル群の配置

❶ USB メモリがマウントされている場所を確認します。

❷ ファイル群をコピーします。

❸ /media/atmark/USBDRIVE をアンマウントします。コマンド終了後に USB メモリを取り外してください。

3. SWUpdate の実行

   initial_setup と同様に、Armadillo に USB メモリを挿入すると自動的にアップデートが実行され、コンテナイメージの組み込みとコンテナの自動実行設定が行われます。その後 Armadillo が自動的に再起動し、サンプルアプリケーションが自動実行します。

   以上で、SWUpdate を用いて Armadillo にコンテナイメージと自動実行用の conf ファイルの配置ができました。他にも SWUpdate を用いて出来ることは多々ありますので、詳細は Armadillo-IoT ゲートウェイ G4 の製品マニュアルの「 Armadillo のソフトウェアをアップデートする [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.11.0/ch09.html#ch.yakushima-softwareupdate]」を参照してください。

11.3. Device Tree を変更しハードウェアを拡張する

Armadillo の拡張インターフェースに外部デバイスを接続する際には、対象のピンの機能やパラメータを変更しなければならない場合があります。具体的には、Armadillo の各ピンの機能やパラメータを記述しているファイルである Device Tree Source を編集・ビルドし、Armadillo に書き込み、起動時にロードする必要があります。

こちらの詳細は Device Tree をカスタマイズする [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-2.10.0/ch03.html#sct.customize-dts] を参照してください。

11.4. コンテナデザインパターン

Armadillo Base OS では基本的にユーザーアプリケーションはコンテナ内に格納されます。場合によっては機能毎にコンテナを複数使用したり、コンテナ内からホストである Armadillo Base OS 側のリソースへのアクセスやコマンド実行をしたりしたい場合などがあります。この章では、上記のような場合に Armadillo Base OS 上でコンテナをどう構築するのが良いかを例を示しつつ紹介します。

11.4.1. 複数コンテナで処理を行いコンテナ間でデータを共有する

ここでは、以下のような処理を行うシステムについて考えます。

1. Armadillo に接続したセンサからデータを読み出す
2. 読み出したデータをローカルに保存
3. 保存したデータを Armadillo に接続した PC で表示できるようにする

このシステムは、1 つのコンテナに全ての処理を任せても実現可能ですが、役割毎にコンテナを分けることで以下のようなメリットを得ることができるためおすすめです。

- 不具合が発生した場合にコンテナ毎に切り分けや修正がしやすい
- アップデート時にアップデートするコンテナ以外への影響が少ない
- 再利用性が高い

・開発時に分業化しやすい

役割毎にコンテナを分けたシステムの構成図を「図 11.9. センサから読み取ったデータを PC に出力するシステム構成図」に示します。なお、図中でグレーアウトしている箇所は使用しない部分、枠が 2 つ重なっている箇所は二面化されている部分です。

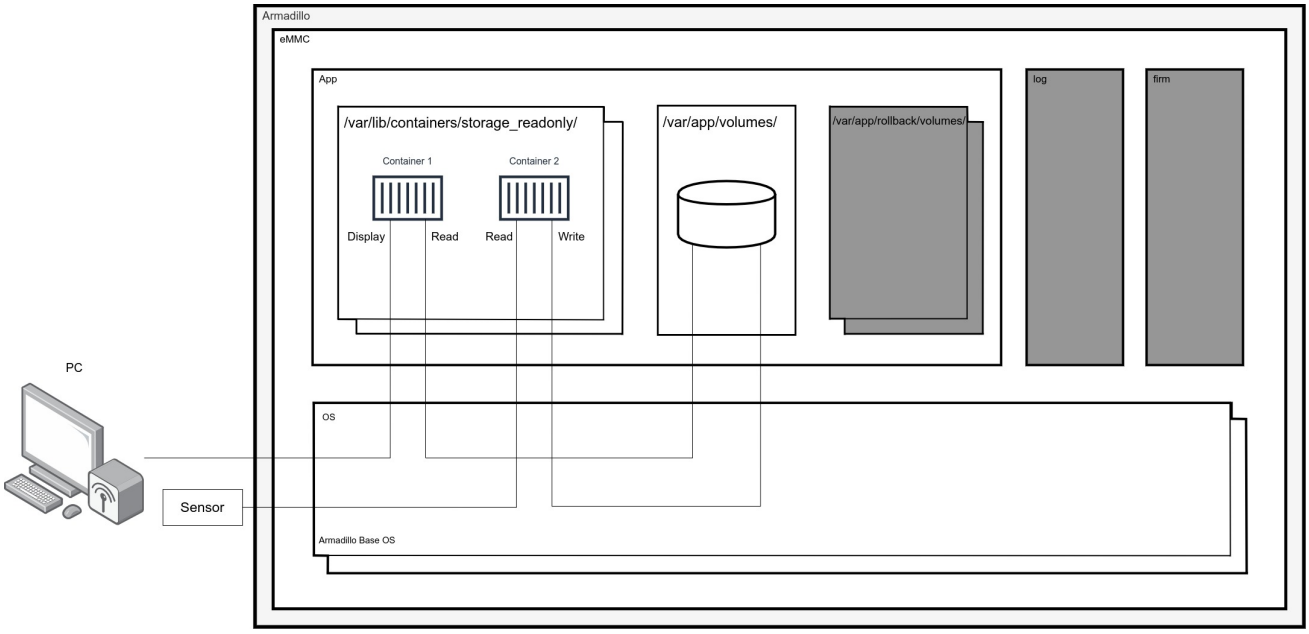


図 11.9 センサから読み取ったデータを PC に出力するシステム構成図

「図 11.9. センサから読み取ったデータを PC に出力するシステム構成図」に示したシステムにおける各コンテナは以下のような処理を行います。

・Container1

  ・定期的に/var/app/volumes 以下のデータベースからデータを取得する

  ・Armadillo に接続している PC でそのデータを見られるようにする

・Container2

  ・定期的に外部のセンサからデータを取得する

  ・センサから読み取った値を/var/app/volumes 以下にデータベースとして配置する

実装のポイントや注意点は以下の通りです。

・データは/var/app/volumes に保存します

  ・ここは二面化されておらず、rollback が発生しても変化はありません

  ・eMMC に書き込むため、電源を切るタイミングによってはデータが破損する可能性があります

  ・各コンテナは同じデータベースに読み書きするため、排他制御する必要があります

  ・USB メモリなどの外部ストレージに置き換えても問題ありません

11.4.2. 複数コンテナ間でデータを共有し、クラウドにアップロードする

ここでは、以下のような処理を行うシステムについて考えます。

1. Armadillo に接続したセンサからデータを読み出す
2. 読み出したデータをローカルに保存
3. 読み出したデータをクラウドにアップロードする

このシステムに関しても、役割毎にコンテナを分けることで「11.4.1. 複数コンテナで処理を行いコンテナ間でデータを共有する」と同様のメリットを得ることができるためおすすめです。

役割毎にコンテナを分けたシステムの構成図を「図 11.10. センサからデータを読み取りクラウドに出力するシステム構成図」に示します。なお、図中でグレーアウトしている箇所は使用しない部分、枠が 2 つ重なっている箇所は二面化されている部分です。

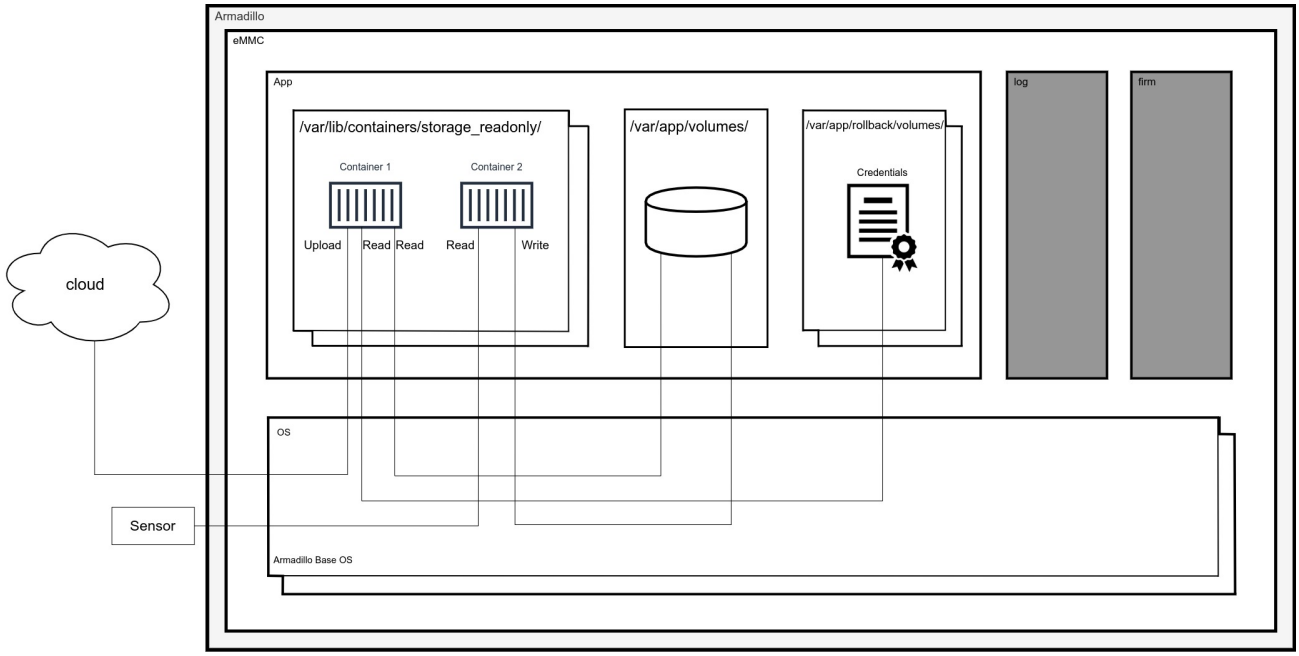


図 11.10 センサからデータを読み取りクラウドに出力するシステム構成図

「図 11.10. センサからデータを読み取りクラウドに出力するシステム構成図」に示したシステムにおける各コンテナは以下のような処理を行います。

- Container1
  - /var/app/rollback/volumes 以下に配置したクラウド認証情報を読み出してクラウドに接続する
  - 定期的に/var/app/volumes 以下のデータベースからデータを取得する
  - 外部のクラウドにデータをアップロードする
- Container2
  - 定期的に外部のセンサからデータを取得する

- センサから読み取った値を/var/app/volumes 以下にデータベースとして配置する

実装のポイントや注意点は以下の通りです。

- データは/var/app/volumes に保存します
  - ここは二面化されておらず、rollback が発生しても変化はありません
  - eMMC に書き込むため、電源を切るタイミングによってはデータが破損する可能性があります
  - 各コンテナは同じデータベースに読み書きするため、排他制御する必要があります
  - USB メモリなどの外部ストレージに置き換えても問題ありません
- クラウドの認証情報は/var/app/rollback/volumes に保存します
  - ここは二面化されており、rollback が発生するとこの中に配置されたファイルも前のバージョンに戻ります
  - 認証情報やコンフィグファイルなどの、アプリケーションのバージョンに依存するようなデータはここに保存することを推奨しています

11.4.3. コンテナを増やす

ここでは、以下のような処理を行うシステムについて考えます。

1. Armadillo に接続した 2 つのセンサからデータを読み出す
2. 読み出したデータを Armadillo 内部のデータベースに保存
3. データベース内のデータを Armadillo に接続した PC で表示できるようにする

基本的には、「11.4.1. 複数コンテナで処理を行いコンテナ間でデータを共有する」で紹介したシステムと同一ですが、入力となるセンサが 2 つに増えています。

この場合、各センサとのインターフェースの差異にもよりますが、先に挙げたメリットのひとつである再利用性を活かして、センサからデータを取得するコンテナを増やすだけで容易に対応することができます。

ただし、コンテナを増やす場合には考慮すべき問題があります。

1 つ目は、Armadillo の容量です。当然ではありますが、コンテナが増えれば増えるほど Armadillo の eMMC の容量を消費します。コンテナを必要以上に構築すると Armadillo の容量が足りなくなる恐れがあります。特に、コンテナのベースイメージが異なる場合は容量を大量に消費するので、複数のコンテナを使用する場合はベースイメージを統一するとコンテナのサイズを抑えることができます。

2 つ目は処理速度の低下です。コンテナを分ければ分けるほど、コンテナ 1 つで全ての処理を行なった際に比べてシステム全体としての処理速度は低下します。完成したシステムで動作確認を十分に行い、システム全体における動作速度に問題がないことを確認した上で運用してください。場合によってはセンサ 1 つに対してコンテナ 1 つでなく、全てのセンサに対してコンテナを 1 つにまとめるなど、コンテナを分ける粒度を荒くして動作速度の改善を考えるべきです。

処理毎にコンテナを分けたシステムの構成図を「図 11.11. 複数のセンサからデータを読み取るシステム構成図」に示します。なお、図中でグレーアウトしている箇所は使用しない部分、枠が 2 つ重なっている箇所は二面化されている部分です。

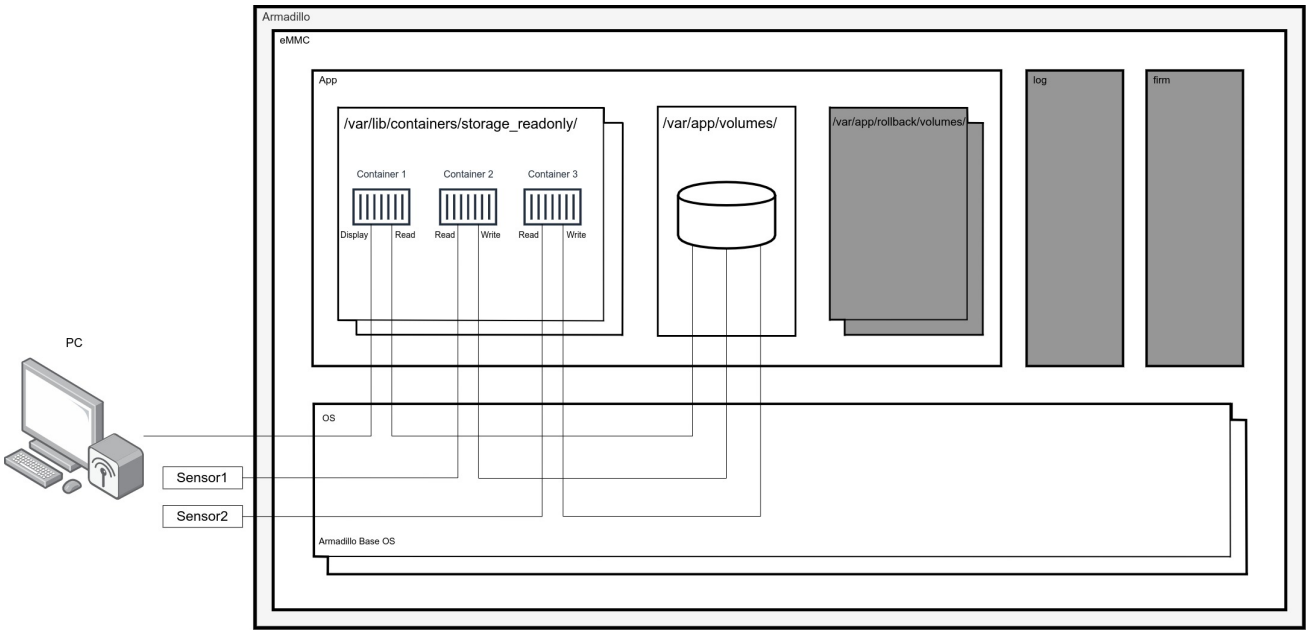


図 11.11 複数のセンサからデータを読み取るシステム構成図

「図 11.10. センサからデータを読み取りクラウドに出力するシステム構成図」に示したシステムにおける各コンテナは以下のような処理を行います。

- Container1
  - 定期的に/var/app/volumes 以下のデータベースからデータを取得する
  - Armadillo に接続している PC でそのデータを見られるようにする
- Container2
  - 定期的に外部のセンサ 1 からデータを取得する
  - センサから読み取った値を/var/app/volumes 以下にデータベースとして配置する
- Container3
  - 定期的に外部のセンサ 2 からデータを取得する
  - センサから読み取った値を/var/app/volumes 以下にデータベースとして配置する

実装のポイントや注意点は、「11.4.1. 複数コンテナで処理を行いコンテナ間でデータを共有する」に挙げたものに加えて以下が挙げられます。

- インターフェースの違いなどにもよりますが、Container2 と Container3 はほとんど同じものを使い回すことができます
- 各コンテナのベースイメージを統一することでコンテナのサイズを抑えることができます

11.4.4. ホストコマンドを実行する

ホスト(Arnadillo Base OS)のコマンドは、コンテナ内から直接実行することはできません。しかし、実現したいシステムによってはコンテナ内からホストコマンドを実行したいことがあります。

コンテナ内からホストコマンドを実行する方法を以下の 2 つのパターンに分けて紹介します。

- ボタン押下や、USB 挿抜などのイベントをトリガにホストコマンドを実行する場合
- コンテナ内から任意のタイミングでホストコマンドを実行する場合

11.4.4.1. イベントをトリガにホストコマンドを実行する

Armadillo Base OS が特定のイベントを検知した際にホストコマンドを実行するように設定できます。そのため、厳密に言うとコンテナ内からホストコマンドを実行することにはなりませんが、コンテナ内のアプリケーションが動作中でもホストコマンドが実行されます。

トリガとするイベントによって設定方法が異なります。以下では udev を用いてデバイスの挿抜を検知してホストコマンドを実行するパターンと、buttond を用いてユーザースイッチや外付けのキーボード等の入力を検知してホストコマンドを実行するパターンの 2 パターンについて説明します。

1. udev を用いて USB メモリの挿入を検知し mount コマンドを実行する

   udev は、Linux カーネル用のデバイス管理ツールです。デバイスが接続もしくは接続解除された時にカーネルは uevent を udev に通知します。この時 udev は予め設定しておいたルールに従って、受け取った uevent に対応した処理を行います。

   udev を活用することでデバイスの挿抜をトリガに任意のホストコマンドを実行できますので、コンテナ内で何かを設定する必要はありません。

   以下では USB メモリの挿抜時に mount コマンドを実行し、/mnt ディレクトリにマウントする例を紹介します。

   まず、/etc/udev/rules.d/に、99-usb-automount.rules というファイルを作成して、以下のように内容を編集します。

   ```
   [armadillo ~/]# cat /etc/udev/rules.d/99-usb-automount.rules
   ACTION==”add”, KERNEL==”sd*”, SUBSYSTEM==”block”, ENV{ID_FS_USAGE}==”filesystem”, RUN+=”/bin/mount /dev/%k /mnt”
   ```

   その後、以下のコマンドを実行して udev ルールの再読込みを行います。

   ```
   [armadillo ~/]# udevadm control -R
   ```

   以上で設定完了ですので、動作確認をしてみます。

   ```
   [armadillo ~/]# mount | grep /mnt ❶
   [armadillo ~/]# ❷
   [84250.573893] usb 1-1: new high-speed USB device number 8 using xhci-hcd
   [84250.732277] usb-storage 1-1:1.0: USB Mass Storage device detected
   : (省略)
   [84252.013348]  sda: sda1
   [84252.020039] sd 0:0:0:0: [sda] Attached SCSI removable disk
   [84252.214274] EXT4-fs (sda1): mounted filesystem with ordered data mode. Opts: (null)
   [armadillo ~/]# mount | grep /mnt ❸
   /dev/sda1 on /mnt type ext4 (rw,relatime)
   ```

❶ /mnt ディレクトリに何もマウントされていないことを確認します。

❷ ここで USB メモリを挿入します。

❸ USB メモリが/mnt ディレクトリに自動的にマウントされていることを確認します。

動作確認後、persist_file コマンドを実行して設定した udev ルールを永続化します。-p オプションでアップデートで保存するための /etc/swupdate_preserve_files にも記載します。

swupdate_preserve_files については Armadillo-IoT ゲートウェイ G4 の製品マニュアルの「swupdate_preserve_files について [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.11.0/ch09.html#sct.swupdate-preserve-files] 」を参照してください。

```
[armadillo ~/]# persist_file -p /etc/udev/rules.d/99-usb-automount.rules
```

上記手順では mount を例に紹介しましたが、他のコマンドやシェルスクリプトの実行なども可能です。

2. buttond を用いてユーザースイッチの押下を検知してホストコマンドを実行する

Armadillo Base OS には、ユーザースイッチや外付けのキーボードなどの入力をイベントとして検知し、予め定義したルールによってそのイベントに対応させた処理を実行できる buttond という仕組みがあります。

buttond についての詳細は、Armadillo-IoT ゲートウェイ G4 の製品マニュアルの「 Armadillo Base OS の操作 [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.11.0/ch09.html#ch.customize-baseos]」内の「ボタンやキーを扱う」を参照してしてください。

以下では、buttond を用いてユーザースイッチを 5 秒以上長押しすると reboot コマンドを実行する例を紹介します。

まず、/etc/atmark/buttond.conf を以下のように作成して、persist_file コマンドで永続化します。また、buttond サービスを再起動させます。

```
[armadillo ~/]# vi /etc/atmark/buttond.conf ❶
BUTTOND_ARGS="$BUTTOND_ARGS -l prog1 -t 5000 -a 'reboot'"
[armadillo ~/]# persist_file /etc/atmark/buttond.conf ❷
[armadillo ~/]# rc-service buttond restart ❸
```

❶ /etc/atmark/buttond.conf を作成・編集します。

❷ persist_file コマンドで永続化します。

❸ buttond サービスを再起動させます。

その後、Armadillo-IoT ゲートウェイ G4 のユーザースイッチ(SW1)を 5 秒長押しして再起動することを確認してください。

上記手順では reboot を例に紹介しましたが、他のコマンドやシェルスクリプトの実行なども可能です。

11.4.4.2. 任意のタイミングでホストコマンドを使用する

「11.4.4.1. イベントをトリガにホストコマンドを実行する」で紹介した方法は、デバイスの挿抜やスイッチの押下などのイベントをトリガとしてホストコマンドを実行するものでした。

イベントをトリガとせずに任意のタイミングでホストコマンドを実行したい場合は、コンテナからホストに対して ssh で接続して実行します。

1. Armadillo Base OS 側の準備 1

   まず、ssh サーバとなる Armadillo Base OS 側の設定を行います。以下のコマンドを実行して、ssh サーバを自動的に起動するよう設定し、sshd サービスを再起動します。

```
[armadillo ~/]# rc-update add sshd
 * service sshd added to runlevel default
[armadillo ~/]# persist_file /etc/runlevels/default/sshd
[armadillo ~/]# rc-service sshd restart
```

2. コンテナの準備

   以下では説明の為に at-debian-image をベースとしたコンテナを作成します。予め「6.3.2. サンプルコンテナをビルド」の手順に従って、at-debian-image の Dockerfile からイメージを生成しておく必要があります。

```
[armadillo ~/]# podman run -it --name=ssh_sample ¥
localhost/at-debian-image:latest /bin/bash
[container /]#
```

   コンテナ内で以下のコマンドを実行して ssh の準備をします。

```
[container /]# apt install -y openssh-client ❶
[container /]# ssh-keygen -t rsa -N "" -f /root/.ssh/id_rsa ❷
[container /]# ls /root/.ssh/id_rsa*    ❸
/root/.ssh/id_rsa     /root/.ssh/id_rsa.pub
[container /]# <Ctrl+P> <Ctrl+Q> ❹
[armadillo ~/]#
```

   ❶ openssh-client をコンテナ内にインストールします。

   ❷ 公開鍵認証のための認証鍵を生成します。

   ❸ 公開鍵・秘密鍵が生成されていることを確認します。

   ❹ コンテナから抜けます。

3. Armadillo Base OS 側の準備 2

   ssh サーバである Armadillo Base OS の方に、先程コンテナ内で生成した公開鍵を登録します。

```
[armadillo ~/]# mkdir /root/.ssh
[armadillo ~/]# chmod 600 /root/.ssh
```

```
[armadillo ~/]# podman exec -it ssh_sample /bin/cat /root/.ssh/id_rsa.pub >> /root/.ssh/
authorized_keys
```

4. ssh でコンテナ内からホストコマンドを実行

   再度コンテナ側に戻り、ssh 経由でホストコマンドである reboot を実行してみます。コンテナから Armadillo Base OS へは IP アドレス 10.88.0.1 でアクセスできます。

```
[armadillo ~/]# podman attach ssh_sample
[container /]# ssh root@10.88.0.1 reboot
:（省略）
```

   初回 ssh 時に、「Are you sure you want to continue connecting (yes/no/[fingerprint])?」と表示される場合があります。手動実行であれば y を選択すれば問題ありませんが、シェルスクリプトなど非手動実行の場合は ssh 実行時に `-o StrictHostKeyChecking=no` オプションを指定すると上記質問を回避できます。

   上記手順では reboot を例に紹介しましたが、他のコマンドやシェルスクリプトの実行なども可能です。

11.5. 機械学習と NPU の使いどころ

Armadillo-IoT ゲートウェイ G4 で採用している i.MX 8M Plus には、機械学習に特化した演算処理ユニットである NPU (Neural Processor Unit) が搭載されています。機械学習と聞くと、何でもできるといったイメージが先行してしまいがちですが、そうとも言い切れません。機械学習を使うためには解決しなければならない課題も多く、これから開発しようとしている製品やサービスに機械学習という手法がマッチするかどうかは、それらの課題をよく考慮した上で決定する必要があります。この章では、機械学習と NPU を採用する前に考慮するべき点、および機械学習以外の方法について説明します。

11.5.1. 機械学習を活用できる範囲

機械学習は主に以下に挙げる 4 つの範囲で活用できます。

表 11.1 機械学習の活用範囲

分野	具体例
画像・映像データの処理	不良品検知 顔認識 文字認識
テキストデータの処理	チャットボット 多言語への翻訳 文章要約
音声データの処理	音声認識
数値データの処理	EC サイトのレコメンド機能

Armadillo-IoT ゲートウェイ G4 でこれらのことを実現したい場合は、NPU を活用した機械学習は選択肢の一つとなります。これら以外のことを実現したいのであれば、機械学習以外の方法を検討することも必要かもしれません。

11.5.2. NPU でできること

NPU は学習済みデータを使って推論を行うことに特化した演算処理ユニットです。このため、学習自体には適していません。学習は別途 PC やクラウドサービスなどを使って行う必要があります。また、

NPU は INT8 で量子化された学習データで推論が高速になるように設計されており、INT8 ではない学習データは CPU で演算が行われるため処理速度は落ちてしまいます。

NPU は以下のツールキットから利用できます。

- TensorFlow Lite
- ONNX Runtime
- ArmNN

これらは、アットマークテクノが提供しているコンテナ at-debian-image であれば、apt コマンドでインストールすることができます。プログラミング言語としては Python が広く使われています。

```
[container /]# apt install tensorflow-lite tensorflow-lite-dev python3-tflite-runtime ¥
tim-vx tensorflow-lite-vx-delegate
```

図 11.12 TensorFlow Lite のインストール

```
[container /]# apt install onnxruntime onnxruntime-dev onnxruntime-tools python3-onnxruntime
```

図 11.13 ONNX Runtime のインストール

```
[container /]# apt install libarmnn22 libarmnn-dev python3-pyarmnn armnn-examples
```

図 11.14 ArmNN のインストール

11.5.3. 機械学習と NPU を使うことの課題

機械学習を用いると従来のアルゴリズムではできなかったことができるようになる反面、「5. 仕様を検討・決定する」 でも言及していますが、以下のような課題を考える必要があります。

- 学習用に大量のデータを用意する必要がある。
- 期待したような認識率がでない場合は、いろいろとパラメータを変えて試行錯誤する必要がある。
- 確率的な処理があるために自動テストが難しい。
- なぜそういう結果になったのかという理由がブラックボックス化されてしまっている。
- 長期運用しているとトレンドの変化などで入力の傾向が変化する。
- 精度が 100%となることはない。

11.5.4. 機械学習以外の方法

機械学習を活用しなくても、例えば画像処理や音声処理などであれば以下のようなライブラリがあります。これらのライブラリでも十分にサービスを提供できるのであれば、採用を検討してもよいかもしれません。

- OpenCV

・ PIL (Python Image Library)

・ Julius (音声認識エンジン)

この内、OpenCV と Julius はアットマークテクノが提供しているコンテナ at-debian-image であれば、apt コマンドでインストールできます。PIL は pip コマンドでインストールできます。

```
[container /]# apt install libopencv-dev python3-opencv
```

図 11.15 OpenCV のインストール

```
[container /]# apt install julius
```

図 11.16 Julius のインストール

```
[container /]# pip3 install pillow
```

図 11.17 PIL のインストール

11.5.4.1. Julius を動かす

ここでは、前章で機械学習以外の方法として紹介した音声認識エンジンである Julius のデモ実行手順を説明します。細かい設定などの詳細な情報については Julius の 公式サイト [https://julius.osdn.jp/index.php]のマニュアル等を参照してください。

なお、音声認識のデモであるため Armadillo-IoT ゲートウェイ G4 本体に USB マイク等の音声入力デバイスを接続しておく必要があります。

準備として、「6.3.2. サンプルコンテナをビルド」 に示した手順通りにコンテナイメージを作成してください。

コンテナを起動してコンテナ内に入ります。

```
[armadillo /]# podman run -ti ¥
        --name=julius_demo_container ¥
        --device=/dev/snd ¥
        --volume=/run/udev:/run/udev:ro ¥
        localhost/abos-dev-guide:v0.0.0 /bin/bash
[container /]#
```

図 11.18 コンテナ内に入る

Julius をインストールします。

```
[container /]# apt install julius
```

図 11.19 コンテナへ Julius をインストール

Julius の 公式サイト [https://julius.osdn.jp/index.php?q=dictation-kit.html] から音声認識パッケージをダウンロードします。音声認識パッケージパッケージには、音声認識に必要な音響モデルが含まれており、「ディクテーションキット」、「話し言葉モデルキット」、「講演音声モデルキット」の 3 つがあります。ここではディクテーションキットを使用します。他のキットに関する詳細は公式サイトを参照してください。

```
[container /]# cd
[container ~]# apt install -y unzip wget
[container ~]# wget --trust-server-names https://osdn.net/projects/julius/downloads/71011/
dictation-kit-4.5.zip/
[container ~]# unzip dictation-kit-4.5.zip
```

図 11.20 ディクテーションキットのダウンロード

julius コマンドで音声認識を実行します。<<< please speak >>> と表示された後に、マイクに向かって発話すると、認識された文章が表示されます。

```
[container ~]# cd dictation-kit-4.5
[container ~]# export ALSADEV="plughw:1,0"
[container ~]# julius -C main.jconf -C am-gmm.jconf -demo
STAT: include config: main.jconf
STAT: include config: am-gmm.jconf
STAT: jconf successfully finalized
: (省略)
<<< please speak >>>
sentence1:  テスト
```

図 11.21 音声認識を実行

11.6. ネットワーク

本章では、Armadillo-IoT ゲートウェイ G4 各製品のネットワーク構成や Podman のネットワーク概要等について説明したうえで、具体的なネットワーク構成を挙げ、その設定方法を記載しています。さらに、量産製造時に一括してネットワークの設定を行う方法や、運用を開始した後の設定変更の方法についても案内しています。

なお、ネットワーク設定方法を記載していますが、必ずしも記載の通り設定する必要はありません。ご利用の環境・システム構成にあわせて適宜読み替えてください。

11.6.1. Armadillo-IoT ゲートウェイ G4 のネットワーク概要

ここでは、Armadillo-IoT ゲートウェイ G4 のネットワークに関して記載します。

11.6.1.1. ネットワークデバイスの種類

Armadillo-IoT ゲートウェイ G4 は以下の通信規格に対応しています。利用可能な種別・通信規格と Linux から使用するネットワークデバイスの対応を以下に示します。

表 11.2 種別・通信規格とネットワークデバイス

種別・通信規格	ネットワークデバイス	備考
有線 LAN(Ethernet)	eth0	有線 LAN インターフェース 1

種別・通信規格	ネットワークデバイス	備考
有線 LAN(Ethernet)	eth1	有線 LAN インターフェース 2 [a]
モバイル通信(3G/LTE)	ttyCommModem	LTE モデルのみ搭載 [b]

[a]10BASE-T 非対応

[b]Quectel 製 EC25-J

11.6.1.2. ネットワーク設定方法

Armadillo-IoT ゲートウェイ G4 では、通常の Linux システムと同様、ネットワークインターフェースの設定は NetworkManager を使用します。NetworkManager はデフォルトで eth0 と eth1 が自動で up し、DHCP でネットワーク設定を取得するようになっています。

NetworkManager はすべてのネットワーク設定をコネクションとして管理します。コネクションには「どのようにネットワークへ接続するか」、「どのようにネットワークを作成するか」を記述し、 /etc/NetworkManager/system-connections/ に保存します。また、1 つのデバイスに対して複数のコネクション設定を保存することは可能ですが、1 つのデバイスに対して有効化できるコネクションは 1 つのみです。

NetworkManager は、従来の /etc/network/interfaces を使った設定方法もサポートしていますが、本書では nmcli と nmtui を用いた方法を中心に紹介します。

1. nmcli

   nmcli は NetworkManager を操作するためのコマンドラインツールです。

   「図 11.22. nmcli のコマンド書式」に nmcli の書式を示します。このことから、 nmcli は「オブジェクト (OBJECT) というものが存在し、それぞれのオブジェクトに対してコマンド (COMMAND) を実行する。」という書式でコマンドを入力することがわかります。また、オブジェクトそれぞれに help が用意されていることもここから読み取れます。

   ```
   nmcli [ OPTIONS ] OBJECT { COMMAND | help }
   ```

   図 11.22 nmcli のコマンド書式

   なお、基本的な使い方については、Armadillo-IoT ゲートウェイ G4 の製品マニュアルの「ネットワーク [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.11.0/ch07.html#sect.operation-check-network]」を参照してください。

2. nmtui

   nmtui は NetworkManager を操作するためのユーザーフレンドリーなツールです。

   NetworkManager を操作するための画面がテキストユーザーインターフェースで表示されるため、一つずつコマンドを入力する必要がなく、現在の設定項目を確認しながら設定することができます。ただし、nmtui で設定できる項目には限りがあるため、設定する内容によっては nmcli を使用する必要があります。

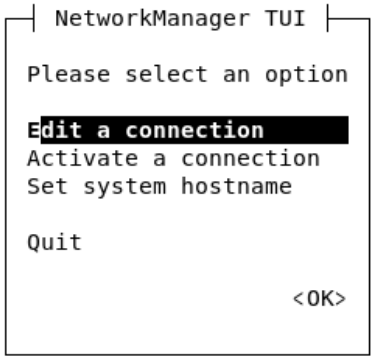


図 11.23 nmtui 起動後の画面

11.6.2. podman のネットワークの仕組み

Armadillo Base OS では ユーザーアプリケーションを「podman コンテナ」内で動作させますが、ネットワークの設定は基本的に Armadillo Base OS (ホスト OS) の設定に帰結します。ここでは、podman のネットワークの仕組みについて説明します。

なお、本項のコマンド実行結果や構成図については Armadillo-IoT ゲートウェイ G4 LTE モデルの内容を記載しており、記述中の `ppp0` は LTE モデルにのみ存在するネットワークインターフェースです。

11.6.2.1. podman のネットワークモード

podman のネットワークモードはブリッジモードがデフォルトです。その他のモードに指定する場合は、コンテナ作成時に `--net` オプションを付けることで指定することができます。例として、podman run 実行時に `--net=host` を指定すると、host モードになります。

下記でブリッジモードと host モードの 2 つについて説明します。その他のモードについては、Podman のドキュメントを参照してください。

> **Podman - podman run コマンド --net オプション詳細**
>
> https://docs.podman.io/en/latest/markdown/podman-run.1.html#network-mode-net

1. ブリッジモード

   前述の通り、ブリッジモードがデフォルトとなります。

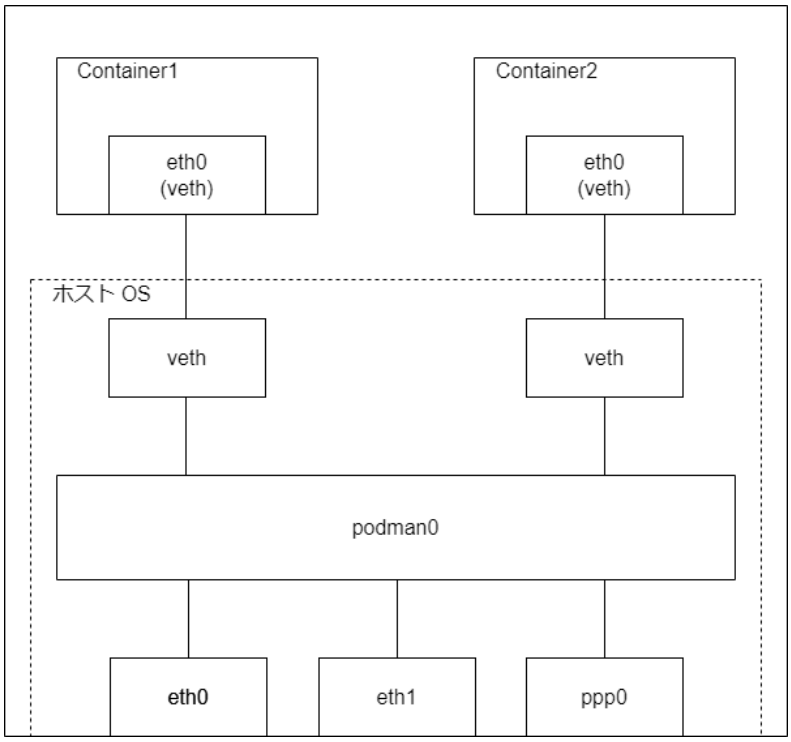


図 11.24 ブリッジモードの構成

仮想ブリッジ podman0 がコンテナとホスト OS のネットワークインターフェースを繋ぐ役割を担い、コンテナは veth を経由して仮想ブリッジに接続します。コンテナから外部に接続する場合は、仮想ブリッジで IP マスカレードが行われます。そのため、ホスト OS 側で複数のネットワークインターフェースを持っていたとしても、コンテナ内のネットワークインターフェースは eth0(veth) となります。

ネットワークの設定が基本的に Armadillo Base OS の設定に帰結するのはこの仕組みによるものです。

実際にコンテナを `podman run` コマンドで作成し、ホスト OS 側のネットワークインターフェースを確認すると、 podman0 と veth が作成されています。

```
[armadillo ~]# podman run -itd --name=my_container --cap-add=NET_ADMIN --cap-add=NET_RAW
docker.io/alpine:latest /bin/sh
eca23e90f6a59bbeee0a92edd88e3c3c24575f9ee061e43ae8a8238e27c2cd4e
[armadillo ~]# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq state DOWN qlen 1000
    link/ether 00:11:00:11:0c:00 brd ff:ff:ff:ff:ff:ff
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP qlen 1000
    link/ether 0c:00:0a:c4:0a:c3 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.18/24 brd 192.168.1.255 scope global dynamic noprefixroute eth1
       valid_lft 86328sec preferred_lft 86328sec
    inet6 240d:18:38:cd00:2e4e:94c4:ce84:bc8/64 scope global dynamic noprefixroute
```

```
       valid_lft 9031sec preferred_lft 9031sec
    inet6 fe80::a61c:e27c:705a:4d96/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
4: ppp0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN ⏎
qlen 3
    link/ppp
    inet 10.231.34.86/32 scope global noprefixroute ppp0
       valid_lft forever preferred_lft forever
5: podman0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP qlen 1000
    link/ether 72:70:b6:93:a5:34 brd ff:ff:ff:ff:ff:ff
    inet 10.88.0.1/16 brd 10.88.255.255 scope global podman0
       valid_lft forever preferred_lft forever
    inet6 fe80::4a4:7dff:fe59:ddfe/64 scope link
       valid_lft forever preferred_lft forever
6: veth7b836e00@eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue master ⏎
podman0 state UP qlen 1000
    link/ether 72:70:b6:93:a5:34 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::7070:b6ff:fe93:a534/64 scope link
       valid_lft forever preferred_lft forever
```

図 11.25 ブリッジモード時のネットワークインターフェース一覧 (ホスト OS)

コンテナ内のネットワークインターフェースはこのようになります。

```
[container /]# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0@if5: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP qlen ⏎
1000
    link/ether 3e:81:21:41:ed:f2 brd ff:ff:ff:ff:ff:ff
    inet 10.88.0.2/16 brd 10.88.255.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::3c81:21ff:fe41:edf2/64 scope link
       valid_lft forever preferred_lft forever
```

図 11.26 ブリッジモード時のネットワークインターフェース一覧 (コンテナ内)

2. host モード

   コンテナ作成時に --net=host オプションをつけると host モードになり、ホスト OS のネットワークインターフェースをそのままコンテナに渡すことができます。

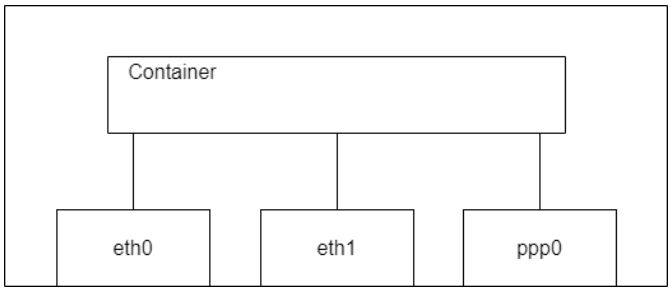


図 11.27 host モードの構成

コンテナを podman run コマンドで作成した時に表示されるログを見ると、ブリッジモードとは異なり、ホスト OS 側に podman0 と veth は存在しません。

```
[armadillo ~]# podman run -itd --name=my_container --net=host --cap-add=NET_ADMIN --cap-
add=NET_RAW docker.io/alpine:latest /bin/sh
3f9ce7adcf7c583b6cec0b82f7ec0f3ab134cef41550bd7f9514d48bd6a4986b
[armadillo ~]# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq state DOWN qlen 1000
    link/ether 00:11:00:11:0c:00 brd ff:ff:ff:ff:ff:ff
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP qlen 1000
    link/ether 0c:00:0a:c4:0a:c3 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.18/24 brd 192.168.1.255 scope global dynamic noprefixroute eth1
       valid_lft 85893sec preferred_lft 85893sec
    inet6 240d:18:38:cd00:a5a3:440a:4c47:a41c/64 scope global dynamic noprefixroute
       valid_lft 7850sec preferred_lft 7850sec
    inet6 fe80::648f:5ec2:82fb:f291/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
4: ppp0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN
qlen 3
    link/ppp
    inet 10.231.34.86/32 scope global noprefixroute ppp0
       valid_lft forever preferred_lft forever
```

図 11.28 host モード時のネットワークインターフェース一覧 (ホスト OS)

コンテナ内のネットワークインターフェースは以下の通り、ホスト OS と同じ内容になります。

```
[container /]#  ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc mq state DOWN qlen 1000
    link/ether 00:11:00:11:0c:00 brd ff:ff:ff:ff:ff:ff
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP qlen 1000
```

```
    link/ether 0c:00:0a:c4:0a:c3 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.18/24 brd 192.168.1.255 scope global dynamic noprefixroute eth1
       valid_lft 85621sec preferred_lft 85621sec
    inet6 240d:18:38:cd00:a5a3:440a:4c47:a41c/64 scope global dynamic noprefixroute
       valid_lft 7578sec preferred_lft 7578sec
    inet6 fe80::648f:5ec2:82fb:f291/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
4: ppp0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UNKNOWN ⏎
qlen 3
    link/ppp
    inet 10.231.34.86/32 scope global noprefixroute ppp0
       valid_lft forever preferred_lft forever
```

図 11.29 host モード時のネットワークインターフェース一覧 (コンテナ内)

host モードは、コンテナからホスト OS のネットワークインターフェースの設定を行いたい場合に適しています。

11.6.2.2. コンテナの IP アドレス

コンテナをブリッジモードにすると veth が作られますが、この veth のアドレスがコンテナ自身のアドレスとなります。同じネットワークのコンテナ (接続されている仮想ブリッジが同じことを指します) の間では、このアドレスを用いて通信を行うことができます。コンテナ間の通信については、コンテナ名でも行うことができます。

なお、ユーザー定義のネットワークの設定を行うことにより、固有の IP アドレスを設定することができます。設定方法の詳細は Armadillo-IoT ゲートウェイ G4 の製品マニュアルの「ネットワークを扱う [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.11.0/ch09.html]」を参照してください。

11.6.2.3. ネットワーク関連の Capability

コンテナを作成する際には、コンテナ内のアプリケーションの動作内容に応じて、ネットワーク関連の権限を渡すことが必要となります。開発時は `--privileged` を設定し全ての権限を渡すことが有用ですが、運用時はセキュリティの都合上、最低限の Capability とすることを推奨します。ネットワークに関連する代表的な Capability を以下に記載します。

表 11.3 ネットワークに関連する代表的な Capability

Capability	
NET_ADMIN	ネットワークスタックを操作する場合に指定
NET_RAW	RAW と PACKET sockets を利用する場合に指定
NET_BIND_SERVICE	特権ポート(1024 以下)でリッスンする場合に指定。ただし、コンテナに root ユーザーで login する場合は考慮不要。

11.6.3. ネットワーク構成例とその設定方法

ここでは、具体的なネットワーク構成を挙げ、その設定方法を紹介します。なお、必ずしも記載の通り設定する必要はありません。ご利用の環境・システム構成にあわせて適宜読み替えてください。また、Armadillo-IoT ゲートウェイ G4 のネットワーク設定が何も行われていない場合を想定して記載しています。

ハードウェア接続方法等については、Armadillo-IoT ゲートウェイ G4 の製品マニュアルの「接続方法 [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.11.0/ch04.html#sct.connect-method]」を参照してください。

11.6.3.1. 各 Ethernet ポートを ローカルネットワーク に接続し 3G/LTE は WAN に接続する

以下の通り、各 Ethernet ポートそれぞれを別セグメントのローカルネットワークに接続し、3G/LTE で WAN に接続します。このとき、 3G/LTE 接続に使用する SIM は固定 IP アドレスが割り当てられるなど WAN 側から接続できるものを想定し、外部からの攻撃を防ぐためファイアーウォールを設定します。

eth0 側のネットワークで シーケンサなどの機器を制御、eth1 側のネットワークは工場内 LAN に接続しそのデータを集約し、3G/LTE 経由で Armadillo-IoT ゲートウェイ G4 のコンテナに ssh 接続し、診断を行う場合などに利用できる構成です。

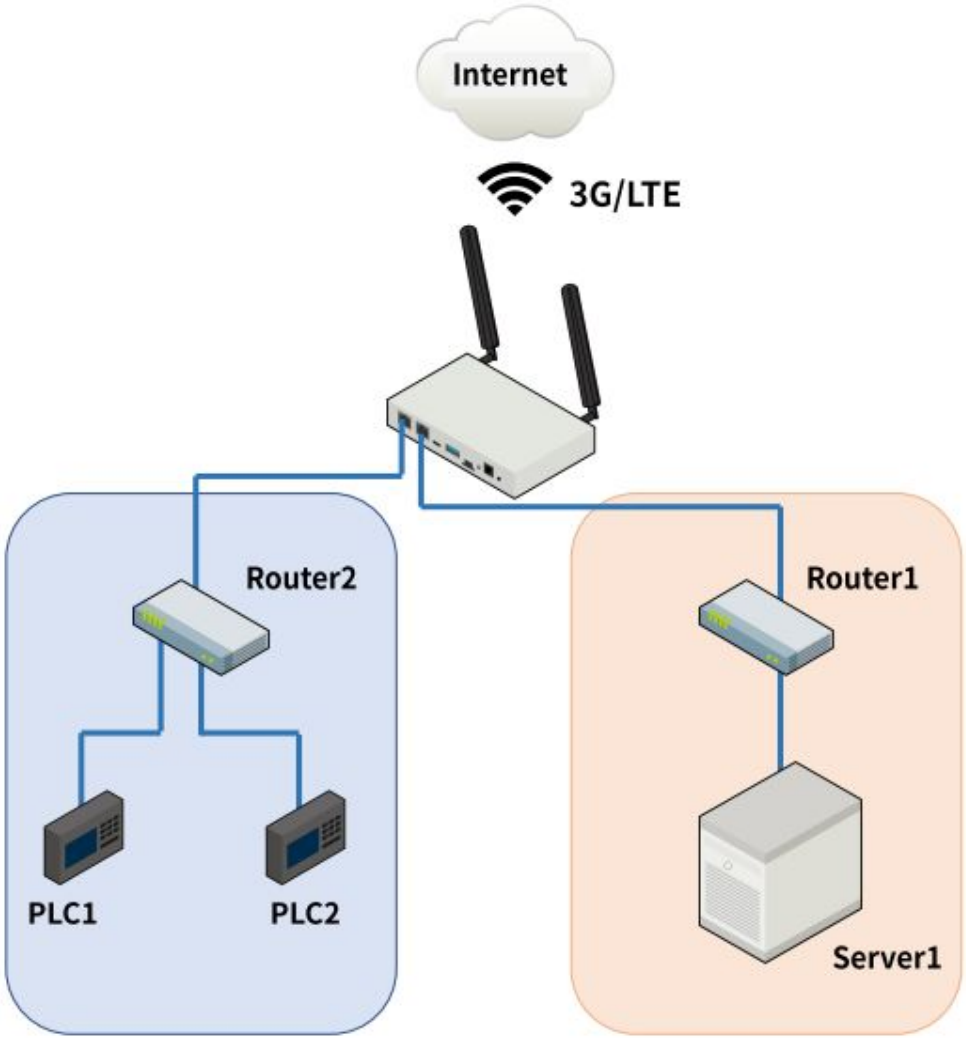


図 11.30 ネットワークの構成

表 11.4 ネットワークのアドレス情報

ノード名	ネットワークデバイス	IP アドレス	ネットワークアドレス
Armadillo	eth0	192.168.10.10	192.168.10.0/24
	eth1	192.168.20.20	192.168.20.0/24
	ttyCommModem	xxx.xxx.xxx.xxx	xxx.xxx.xxx.xxx

ノード名	ネットワークデバイス	IP アドレス	ネットワークアドレス
Router1	-	192.168.10.1	192.168.10.0/24
Server1	-	192.168.30.1	192.168.30.0/24
Router2	-	192.168.20.1	192.168.20.0/24
PLC1	-	192.168.40.1	192.168.40.0/24
PLC2	-	192.168.40.2	192.168.40.0/24

・ 設定手順

上記のネットワークを構築する際のネットワーク設定手順は以下のようになります。全てコンソール上で実行します。有線 LAN インターフェースの設定については nmcli と nmtui のそれぞれの手順を記載しています。使いやすい方を選択して設定を行ってください。

なお、Armadillo-IoT ゲートウェイ G4 は overlayfs を採用しているため、シャットダウンや再起動でシステムを OFF すると作成したファイルが消えてしまいます。それにより nmcli コマンド等で指定した設定も消えてしまうため、各設定の最後には設定の永続化を行っています。

また、全ての設定はホスト OS 側で実施します。

1. 有線 LAN インターフェースの設定

・ nmcli

下記にコマンドを記載します。「表 11.5. 有線 LAN インターフェース の設定情報(nmcli)」 に記載する内容に置き換えて 有線 LAN インターフェース 1(eth0)、2(eth1) それぞれを設定してください。

表 11.5 有線 LAN インターフェース の設定情報(nmcli)

	eth0	eth1
interface name	eth0	eth1
connection name	ethernet-eth0	ethernet-eth1
address	"192.168.10.10/24"	"192.168.20.20/24"
route	"192.168.30.0/24 192.168.10.1"	"192.168.40.0/24 192.168.20.1"

```
[armadillo ~]# nmcli connection add type ethernet ifname [interface name] ❶
[armadillo ~]# nmcli connection modify [connection name] ipv4.method manual
ipv4.addresses [address] ❷
[armadillo ~]# nmcli connection modify [connection name] ipv4.routes [route] ❸
[armadillo ~]# nmcli connection modify [connection name] ipv4.never-default yes ❹
[armadillo ~]# nmcli connection down [connection name] ❺
[armadillo ~]# nmcli connection up [connection name] ❻
```

図 11.31 有線 LAN インターフェースの設定を行う (nmcli)

❶ 有線 LAN インターフェースのコネクションを作成します。

❷ 有線 LAN インターフェースのコネクションに固定 IP アドレスを設定します。

❸ 有線 LAN インターフェースのコネクションに経路情報を追加します。

❹ 有線 LAN インターフェースのコネクションのデフォルトゲートウェイを無効化します。

❺ 修正を反映させるため、有線 LAN インターフェースのコネクションを無効化します。

❻ 有線 LAN インターフェースのコネクションを有効化します。

・nmtui

下記の通り、nmtui を実行して設定を行います。入力する内容は 「表 11.6. 有線 LAN インターフェース の設定情報(nmtui)」 に記載する内容に置き換えて 有線 LAN インターフェース 1(eth0)、2(eth1) それぞれを設定してください。

表 11.6 有線 LAN インターフェース の設定情報(nmtui)

	eth0	eth1
Profile name	ethernet-eth0	ethernet-eth1
Device	eth0	eth1
Addresses	192.168.10.10/24	192.168.20.20/24
Destination/Prefix	192.168.30.0/24	192.168.40.0/24
Next Hop	192.168.10.1	192.168.20.1

nmtui を起動します。

```
[armadillo ~]# nmtui
```

図 11.32 nmtui を起動する

Edit a connection を選択します。

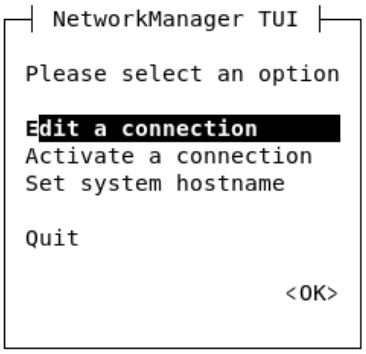


図 11.33 nmtui 起動後の画面

<Add> を選択します。

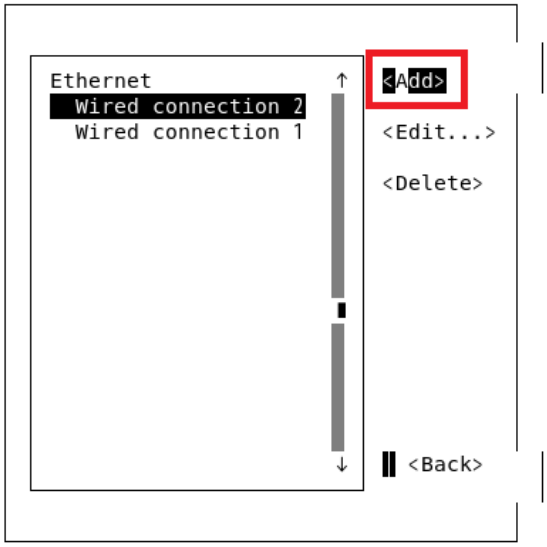


図 11.34 nmtui コネクション選択画面

コネクション種別の選択画面が表示されるので Ethernet を選択します。

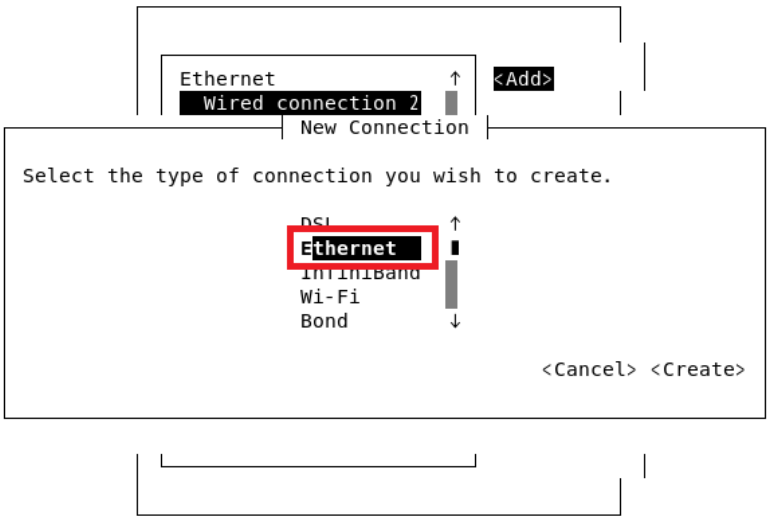


図 11.35 nmtui コネクション種別選択画面

Profile name Ethernet と Device を入力します。

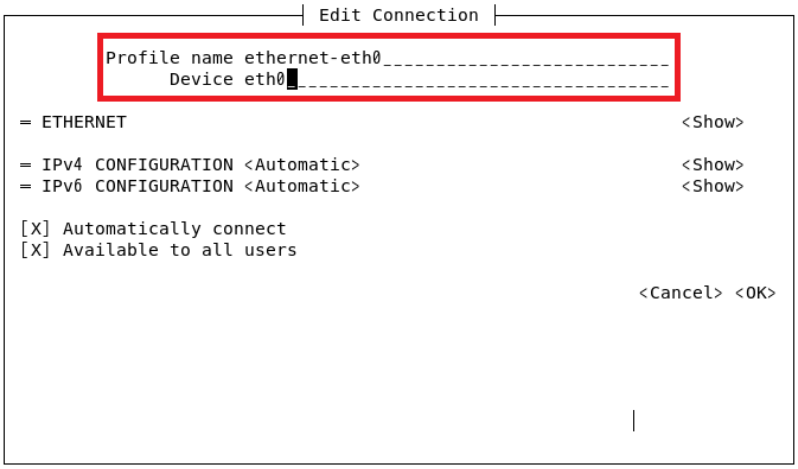


図 11.36 nmtui コネクション入力画面

IPv4 CONFIGURATION を <Manual> に変更し、Show を選択します。入力画面が表示されるので、 Addresses と Routing を入力します。

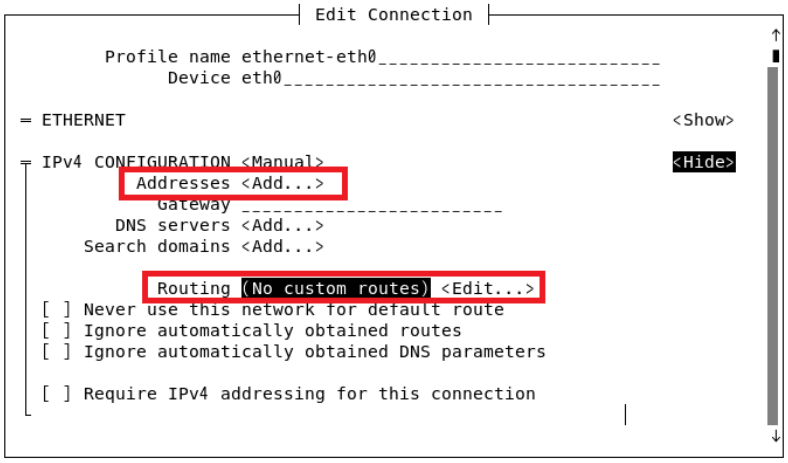


図 11.37 nmtui IPv4 設定画面

Routing については、 <Edit…> 選択後に表示される画面の <Add…> を選択すると入力画面が表示されるので、項目入力後 <OK> を選択してください。

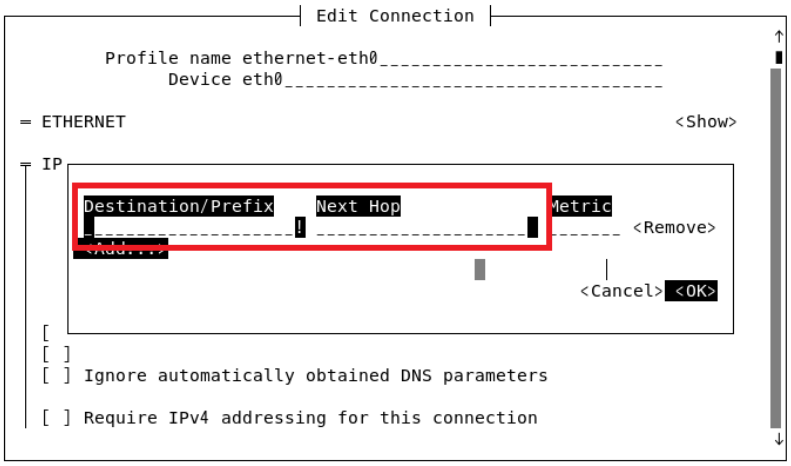


図 11.38 nmtui Route 設定画面

デフォルトゲートウェイを無効化するため、 <Never use this network for default route> にチェックを入れます。カーソルを合わせ、スペースキーを押すと、チェックが入ります。

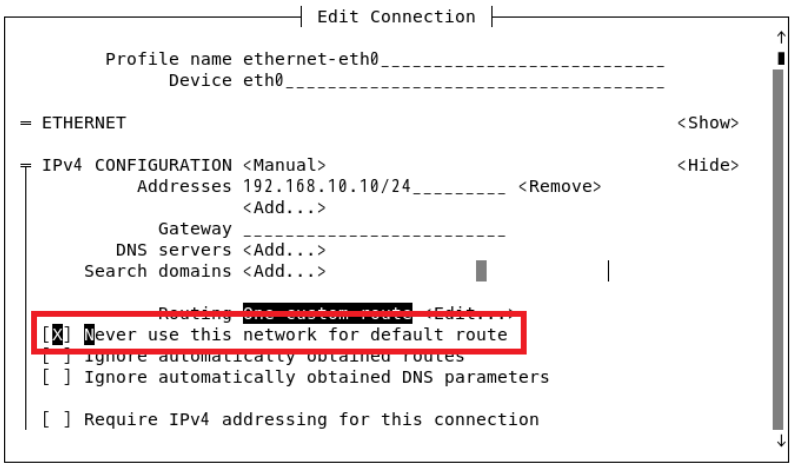


図 11.39 nmtui デフォルトゲートウェイ無効化

全ての設定入力後、 <OK> を選択し設定を保存してください。

次に、ネットワーク設定を反映させるために一度コネクションを Deactivate します。コネクション選択画面で <Back> を選択し、 Activate a connection を選択してください。

現在 Active になっているコネクションには、コネクション名の横に *(アスタリスク) がついています。もし *(アスタリスク) がついていない場合は、Deactivate 作業はスキップし、Activate を行ってください。

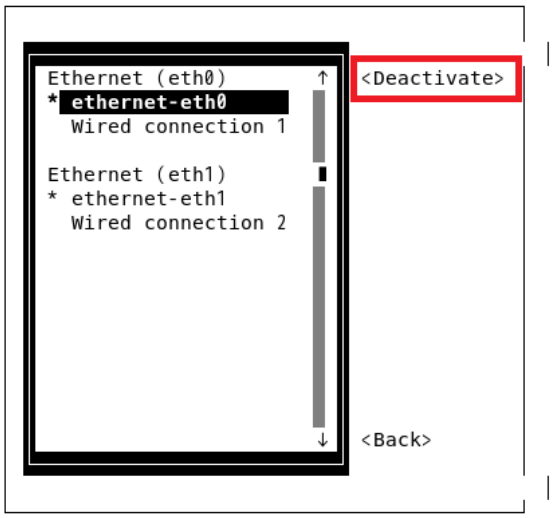


図 11.40 nmtui コネクション Activate 画面

対象のコネクションにカーソルを合わせ、 <Deactivate> を選択すると、 *(アスタリスク) が消えます。

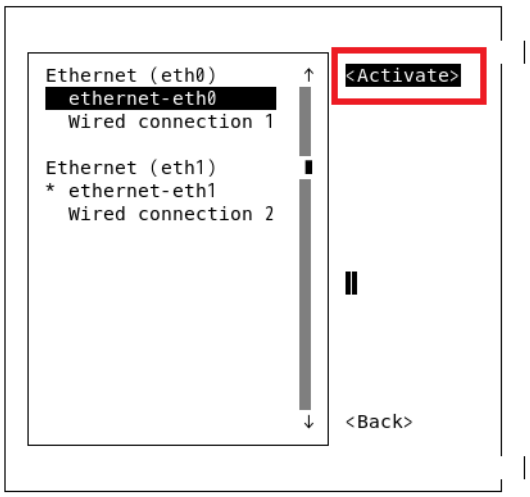


図 11.41 nmtui コネクション Deactivate 後の画面

Deactivate 後、再び対象のコネクションにカーソルを合わせ、 <Activate> を選択すると、ネットワーク設定が反映され、コネクション名の横に *(アスタリスク) が表示されます。

2. 有線 LAN インターフェース設定の永続化

```
[armadillo ~]# persist_file /etc/NetworkManager/system-connections/ethernet-
eth0.nmconnection ❶
[armadillo ~]# persist_file /etc/NetworkManager/system-connections/ethernet-
eth1.nmconnection ❷
```

図 11.42 有線 LAN インターフェース設定を永続化する

❶ 有線 LAN インターフェース 1(eth0) のコネクション設定ファイルを永続化します。

❷ 有線 LAN インターフェース 2(eth1) のコネクション設定ファイルを永続化します。

3. 3G/LTE(ttyCommModem) の接続設定

```
[armadillo ~]# nmcli connection add type gsm ifname ttyCommModem apn [apn] user [user] ⏎
password [password] ❶
[armadillo ~]# persist_file /etc/NetworkManager/system-connections/gsm- ⏎
ttyCommModem.nmconnection ❷
```

図 11.43 3G/LTE 回線の接続設定を行う

❶ 3G/LTE(ttyCommModem) のコネクションを作成します。

❷ 3G/LTE(ttyCommModem) のコネクション設定ファイルを永続化します。

4. ファイアーウォールの設定

3G/LTE(ttyCommModem) のファイアーウォールを設定します。今回は、外部から 50000 番ポートにアクセスされた場合のみ通信を許可するよう設定します。また、 ICMP プロトコルと DNS(53 番ポート) も許可しています。

ここでは、iptables を用いて設定を行っています。IPv6 の場合は、以下の手順の iptables の箇所を ip6tables に置き換えて実行してください。

```
[armadillo ~]# iptables -A INPUT -i ppp0 -p tcp --dport 50000 -j ACCEPT ❶
[armadillo ~]# iptables -A INPUT -i ppp0 -p tcp --dport 53 -j ACCEPT ❷
[armadillo ~]# iptables -A INPUT -i ppp0 -p udp --sport 53 -j ACCEPT
[armadillo ~]# iptables -A INPUT -i ppp0 -p icmp -j ACCEPT ❸
[armadillo ~]# iptables -A INPUT -i ppp0 -j DROP ❹
```

図 11.44 ファイアーウォールの設定を行う

❶ 3G/LTE のネットワークデバイス ppp0 について、50000 番ポートのパケットを許可するよう設定します。

❷ 3G/LTE のネットワークデバイス ppp0 について、 DNS (53 番ポート) を許可します。

❸ 3G/LTE のネットワークデバイス ppp0 について、 ICMP プロトコルを許可します。なお、 IPv6 の場合は icmp を icmpv6 に置き換えてください。

❹ 3G/LTE のネットワークデバイス ppp0 について、全てのパケットを破棄するよう設定します。

iptables の設定は overlayfs に関係なくシャットダウンや再起動でシステム OFF すると消えてしまうため、以下の手順で設定を永続化します。

```
[armadillo ~]# /etc/init.d/iptables save ❶
[armadillo ~]# persist_file -p /etc/iptables/rules-save ❷
```

```
[armadillo ~]# rc-update add iptables ❸
[armadillo ~]# persist_file -p /etc/runlevels/default/iptables ❹
```

図 11.45 ファイアーウォールの設定を永続化する

❶ iptables の設定をファイルに書き出します。コマンドを実行すると、 /etc/iptables/rules-save に保存されます。

❷ 書き出したファイルを永続化し、-p オプションで OS アップデートの際にもファイルが削除されないように設定します。

❸ iptables サービスを有効にします。これにより、Armadillo-IoT ゲートウェイ G4 が起動時に iptables が自動起動します。

❹ iptables サービスの自動起動設定を永続化し、-p オプションで OS アップデートの際にも設定が保持されるように設定します。

ipv6 を使う予定がない場合は完全に無効化できます。IPv4 にフィルターが必要と判断した場合には必ず IPv6 に同様なフィルターを設定するか、IPv6 を無効化してください。

```
[armadillo ~]# vi /etc/sysctl.d/20-disable-ipv6.conf
net.ipv6.conf.all.disable_ipv6=0
[armadillo ~]# persist_file /etc/sysctl.d/20-disable-ipv6.conf
```

図 11.46 IPv6 の無効化を行う

・ 設定確認

1. ネットワークデバイスの状態確認

   設定したネットワークデバイスの状態が connected になっていることを確認します。

```
[armadillo ~]# nmcli device
DEVICE        TYPE      STATE      CONNECTION
ttyCommModem  gsm       connected  gsm-ttyCommModem
eth0          ethernet  connected  ethernet-eth0
eth1          ethernet  connected  ethernet-eth1
lo            loopback  unmanaged  --
ppp0          ppp       unmanaged  --
```

図 11.47 ネットワークデバイスの状態を確認する

2. IP アドレス設定確認

```
[armadillo ~]# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
```

```
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP qlen 1000
    link/ether 00:11:00:11:0c:00 brd ff:ff:ff:ff:ff:ff
    inet 192.168.10.10/24 brd 192.168.10.255 scope global noprefixroute eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::aa84:668a:56ce:5d51/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP qlen 1000
    link/ether 0c:00:0a:c4:0a:c3 brd ff:ff:ff:ff:ff:ff
    inet 192.168.20.20/24 brd 192.168.20.255 scope global noprefixroute eth1
       valid_lft forever preferred_lft forever
    inet6 fe80::bdee:42d6:1307:325e/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
4: ppp0: <POINTOPOINT,MULTICAST,NOARP,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state ⏎
UNKNOWN qlen 3
    link/ppp
    inet 10.231.34.86 peer 10.64.64.64/32 scope global ppp0
       valid_lft forever preferred_lft forever
    inet 10.231.34.86/32 scope global noprefixroute ppp0
       valid_lft forever preferred_lft forever
```

図 11.48 IP アドレスの設定を確認する

3. ルーティングテーブルの確認

```
[armadillo ~]# route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
default         *               0.0.0.0         U     700    0        0 ppp0
10.64.64.64     *               255.255.255.255 UH    0      0        0 ppp0
192.168.10.0    *               255.255.255.0   U     103    0        0 eth0
192.168.20.0    *               255.255.255.0   U     104    0        0 eth1
192.168.30.0    192.168.10.1    255.255.255.0   UG    103    0        0 eth0
192.168.40.0    192.168.20.1    255.255.255.0   UG    104    0        0 eth1
```

図 11.49 ルーティングテーブルの内容を確認する

4. ファイアーウォールの確認

```
[armadillo ~]# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:50000
ACCEPT     tcp  --  anywhere             anywhere             tcp dpt:domain
ACCEPT     udp  --  anywhere             anywhere             udp spt:domain
ACCEPT     icmp --  anywhere             anywhere
DROP       all  --  anywhere             anywhere

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

```
[armadillo ~]# iptables -S
-P INPUT ACCEPT
-P FORWARD ACCEPT
-P OUTPUT ACCEPT
-A INPUT -i ppp0 -p tcp -m tcp --dport 50000 -j ACCEPT
-A INPUT -i ppp0 -p tcp -m tcp --dport 53 -j ACCEPT
-A INPUT -i ppp0 -p udp -m udp --sport 53 -j ACCEPT
-A INPUT -i ppp0 -p icmp -j ACCEPT
-A INPUT -i ppp0 -j DROP
```

図 11.50 ファイアーウォールの設定を確認する

・ コンテナ作成例

ここでは、ホスト OS の 50000 番ポートへのアクセスでコンテナの 22 番ポートに接続することができるコンテナの作成手順を紹介します。コンテナ内での IP アドレスの設定等は必要ありません。

なお、本項では podman_start を使用してコンテナを作成・起動する方法を記載しています。「6.5.6. podman コンテナとアプリケーションの自動実行」 で述べている通り、Armadillo Base OS では /etc/containers/ 以下の .conf ファイルを参照し、記述されている内容に従ってコンテナを作成します。

以下に、 IPv4 と IPv6 それぞれの .conf ファイル作成例を記載します。また、作成した .conf ファイルについても永続化の設定が必要なため、あわせて記載しています。

```
[armadillo ~]# cat /etc/atmark/containers/my_container.conf
set_image docker.io/alpine

add_args -ti
add_ports 50000:22 ❶
set_autostart no ❷

set_command sh

[armadillo ~]# persist_file /etc/atmark/containers/my_container.conf ❸
```

図 11.51 .conf ファイル設定例 (IPv4 の場合)

❶ コンテナの 22 番ポートをホストの 50000 番ポートに割り当てます。

❷ 自動起動を無効にしています。自動起動させたい場合は、この行を削除してください。

❸ 設定ファイル my_container.conf ファイルを永続化します。

```
[armadillo ~]# cat /etc/atmark/containers/my_net.conf
set_type network
add_args --ipv6 ❶

[armadillo ~]# cat /etc/atmark/containers/my_container.conf
set_image docker.io/alpine

add_args -ti
add_ports 50000:22 ❷
```

```
set_network my_net ❸
set_autostart no ❹

set_command sh

[armadillo ~]# persist_file /etc/atmark/containers/my_net.conf ❺
[armadillo ~]# persist_file /etc/atmark/containers/my_container.conf ❻
```

図 11.52 .conf ファイル設定例 (IPv6 の場合)

❶ --ipv6 オプションを指定してユーザー定義のネットワークを作成します。

❷ コンテナの 22 番ポートをホストの 50000 番ポートに割り当てます。

❸ 作成したユーザー定義のネットワーク名を指定します。

❹ 自動起動を無効にしています。自動起動させたい場合は、この行を削除してください。

❺ 設定ファイル my_net.conf を永続化します。

❻ 設定ファイル my_container.conf を永続化します。

コンテナの起動方法はそれぞれ下記の通りです。

```
[armadillo ~]# podman_start my_container ❶
```

図 11.53 コンテナ起動方法 (IPv4 の場合)

❶ コンテナを起動します。

```
[armadillo ~]# podman_start my_net ❶
[armadillo ~]# podman_start my_container ❷
```

図 11.54 コンテナ起動方法 (IPv6 の場合)

❶ コンテナ起動前にユーザー定義のネットワークを作成します。なお、my_net.conf 設定後は Armadillo-IoT ゲートウェイ G4 起動時に自動的にネットワーク `my_net` が作成されます。

❷ コンテナを起動します。

また、下記のコマンドでコンテナ内に入ることができます。

```
[armadillo ~]# podman attach my_container
```

図 11.55 コンテナ内に入る (IPv4, IPv6 共通)

11.6.4. 量産製造時のネットワーク設定

量産製造時に一括してネットワーク設定を行う場合の方法を紹介します。以下のいずれの場合であっても、 「10.2. クローンインストールディスクを用いてイメージ書き込みを行なう」 に示す通り、クローンインストールディスクを用いてネットワーク設定を行います。

11.6.4.1. 全ての機器が同一の設定内容の場合

設定する 3G/LTE の接続情報が同一の場合など、全ての機器で同一のネットワーク設定となる場合は、「8. 量産用インストールディスクの作成」 の手順に従って作成したクローンインストールディスクを用いることで一括して設定を行うことができます。

11.6.4.2. 機器ごとに設定内容が異なる場合

機器ごとに異なる固定 IP アドレスを設定する場合など、各機器でネットワーク設定が異なる場合は、「10.2.1. インストール時に任意の処理を行なう」 に示す通り、インストール時に任意の処理を行うようにすることで異なる設定を行うことができます。

異なる固定 IP アドレスを設定する場合の手順については、 「10.2.1.1. 個体ごとに異なる固定 IP アドレスを設定する」 に記載しています。また、インストール時に実行する処理の内容を変えることで、固定 IP アドレス以外にも様々なネットワーク設定を柔軟に行うことができます。

11.6.5. 運用開始後に設定を変更する

ここでは、運用開始後にネットワークの設定を変更する場合の方法について記述します。

運用中の機器に対してネットワークの設定の変更は SWUpdate で実施します。

SWUpdate の詳細は Armadillo-IoT ゲートウェイ G4 の製品マニュアルの「Armadillo のソフトウェアをアップデートする [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.11.0/ch09.html#ch.yakushima-softwareupdate]」を参照してください。

11.6.5.1. SWUpdate でネットワーク設定を変更する

SWUpdate は swu というイメージ形式のファイルを用いてアップデートを実行します。

以下は 有線 LAN インターフェース(eth0) の ネットワーク設定を変更する場合の手順を例に説明します。このパターンでは、 swu イメージにコネクションファイルを含める形となります。

swu ファイル作成に必要となるツール mkswu については、Armadillo-IoT ゲートウェイ G4 の製品マニュアルの「Armadillo のソフトウェアをアップデートする [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.11.0/ch09.html#ch.yakushima-softwareupdate]」を参照してください。なお、 「11.1.1. SWUpdate による初回アップデート」 に示す mkswu を用いた初回アップデート作業が完了していることを前提としています。

1. swu イメージの作成

   swu イメージの作成は、 ATDE 上で実施します。

   ATDE 上にコネクションファイルをコピーし、パーミッションを 600 に設定します。

```
[ATDE ~/mkswu]# mkdir network-setting && cd network-setting
[ATDE ~/mkswu/network-setting]# cp /path/to/ethernet-eth0.nmconnection . ❶
[ATDE ~/mkswu/network-setting]# ls
ethernet-eth0.nmconnection
[ATDE ~/mkswu/network-setting]# chmod 600 ethernet-eth0.nmconnection ❷
```

図 11.56 ATDE 上へのコネクションファイルのコピーとパーミッションの設定を行う

❶ コネクションファイルをコピーします。コピー元のパスは適宜読み替えてください。

❷ コネクションファイルのパーミッションを 600 に設定します。

swu イメージ作成に用いる network-setting.desc ファイルを以下の通り作成します。

```
[ATDE ~/mkswu/network-setting]$ cat network-setting.desc
component=extra_os.nw_eth0
version=1

swdesc_files --dest /etc/NetworkManager/system-connections "ethernet-eth0.nmconnection" ❶
```

図 11.57 network-setting.desc ファイルを作成する

❶ ethernet-eth0.nmconnection を Armadillo-IoT ゲートウェイ G4 の /etc/NetworkManager/system-connections/ に配置します。

mkswu を実行して swu イメージを作成します。

```
[ATDE ~/mkswu/network-setting]# mkswu network-setting.desc
Enter pass phrase for /home/atmark/mkswu/swupdate.key: ❶
network-setting.swu を作成しました。

[ATDE ~/mkswu/network-setting]# ls
ethernet-eth0.nmconnection  network-setting.desc  network-setting.swu
```

図 11.58 コネクションファイルを含む swu イメージを作成する

❶ 初回アップデート作業時に指定した鍵のパスワードを入力してください。

2. swu イメージのインストール

swu イメージの Armadillo-IoT ゲートウェイ G4 への転送方法ですが、既にネットワークに接続されている場合はリモートアップデートで実施、ネットワーク接続していない機器に対し設定を行う場合は USB メモリを用いて実施するなど、それぞれの環境に応じて選択してください。

インストール方法については、Armadillo-IoT ゲートウェイ G4 の製品マニュアルの「Armadillo のソフトウェアをアップデートする [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.11.0/ch09.html#ch.yakushima-softwareupdate]」を参照してください。

Armadillo Base OS 開発ガイド

改訂履歴

バージョン	年月日	改訂内容
1.0.0	2021/12/28	・初版発行
1.1.0	2022/04/27	・「11.1. SWUpdate を用いてソフトウェアをアップデートする」を追加 ・「11.3. Device Tree を変更しハードウェアを拡張する」 を追加 ・「11.4. コンテナデザインパターン」 を追加 ・その他軽微な修正
1.2.0	2022/05/27	・「11.5. 機械学習と NPU の使いどころ」 を追加
1.3.0	2022/06/28	・「8. 量産用インストールディスクの作成」 を追加 ・「10. 製造・量産する」 を追加 ・「11.4. コンテナデザインパターン」 に複数コンテナを使用する場合のデザインパターンを追加 ・「11.6. ネットワーク」 を追加
1.3.1	2022/07/27	・at-debian-image からコンテナを作成する例において、 device として渡す tty の番号を 1 から 7 に修正
1.4.0	2022/11/28	・「7. 量産時のイメージ書き込み手法」 を追加 ・「9. 量産用 swu イメージの作成」 を追加
1.4.1	2023/05/29	・「11.6. ネットワーク」 に IPv6 を無効にする方法を追加
1.4.2	2024/04/23	・「8.3. 開発したシステムをインストールディスクにする」にインストール実行時に再生成するファイルに関する説明を追加
1.4.3	2024/06/26	・「11.3. Device Tree を変更しハードウェアを拡張する」 のデバイスツリーカスタマイズ方法の説明を製品マニュアルへのリンクに変更 ・誤記修正

Armadillo Base OS 開発ガイド
Version 1.4.3
2024/06/26