Microsoft

生成 AI が業務システムのモダナイゼーションに与えるインパクト ～ Microsoft AI が変える基幹システムの再構築

日本マイクロソフト株式会社
クラウド＆AIソリューション事業本部
クラウド＆AIプラットフォーム統括本部
Azure 金融・パブリックセクター技術本部
シニア ソリューション エンジニア

佐藤 直樹

自己紹介

佐藤 直樹 (Naoki Sato)

X:@nksato, fb:nksato, https://www.linkedin.com/in/nksato/

(株)日立製作所でのソフトウエア開発、ロータス(株) (現 HCL Technologies)でのテクニカルサポートアナリストを経て、2000年にマイクロソフトに入社。

アプリケーションデベロップメントコンサルタント、プラットフォーム戦略アドバイザー、テクニカルエバンジェリスト、クラウドソリューションアーキテクト等を歴任し、パートナー企業やエンタープライズ企業の IT プロジェクトへ多数参画。

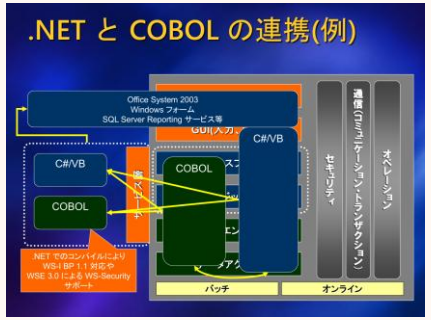
現在はクラウドサービスの Microsoft Azure および Azure AI を、パブリックセクターおよびパートナー企業に向けて推進、業務システムのクラウド化におけるアーキテクチャ策定など技術支援を行う

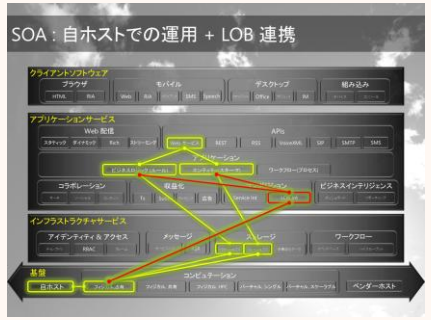
Agenda

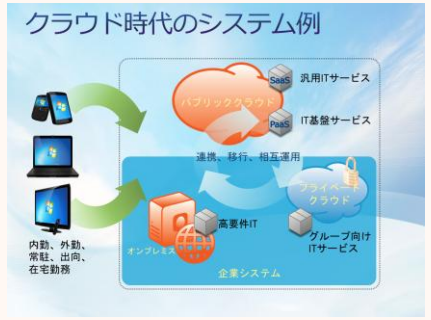
Agenda

COBOL フォーラムでの登壇内容

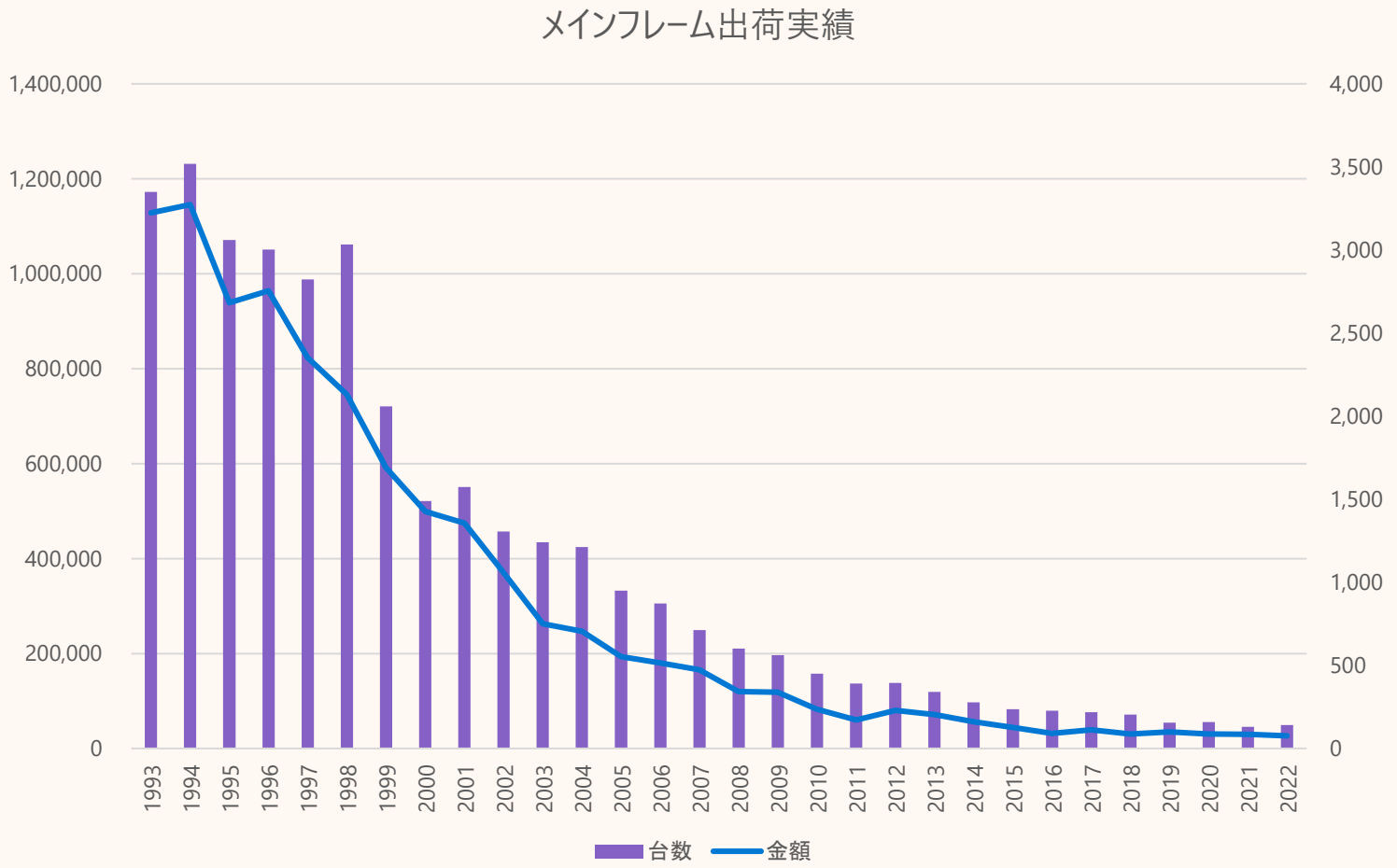
日付	セッションタイトル
2005/3/4	Windows Server 2003 の実力と Visual Studio .NET による開発
2006/2/16	企業の個性を引き出す .NET/COBOL 連携
2006/6/7	マイクロソフトの最新テクノロジー戦略と開発技術のロードマップ ～ COBOL は SOA と Web 2.0 に適応できるのか？
2008/6/17	サービス指向の視点で見たマイクロソフトプラットフォーム技術 ～ COBOL 資産を Software + Services で活かす～
2010/11/9	マイクロソフトのクラウド戦略と Windows Azure Platform 概要
2011/11/17	マイクロソフトプラットフォームの新潮流 ～モバイルからクラウドまで～
2012/11/8	マイクロソフトの最新プラットフォーム技術ご紹介 ～基幹系システム開発・実行環境とは～





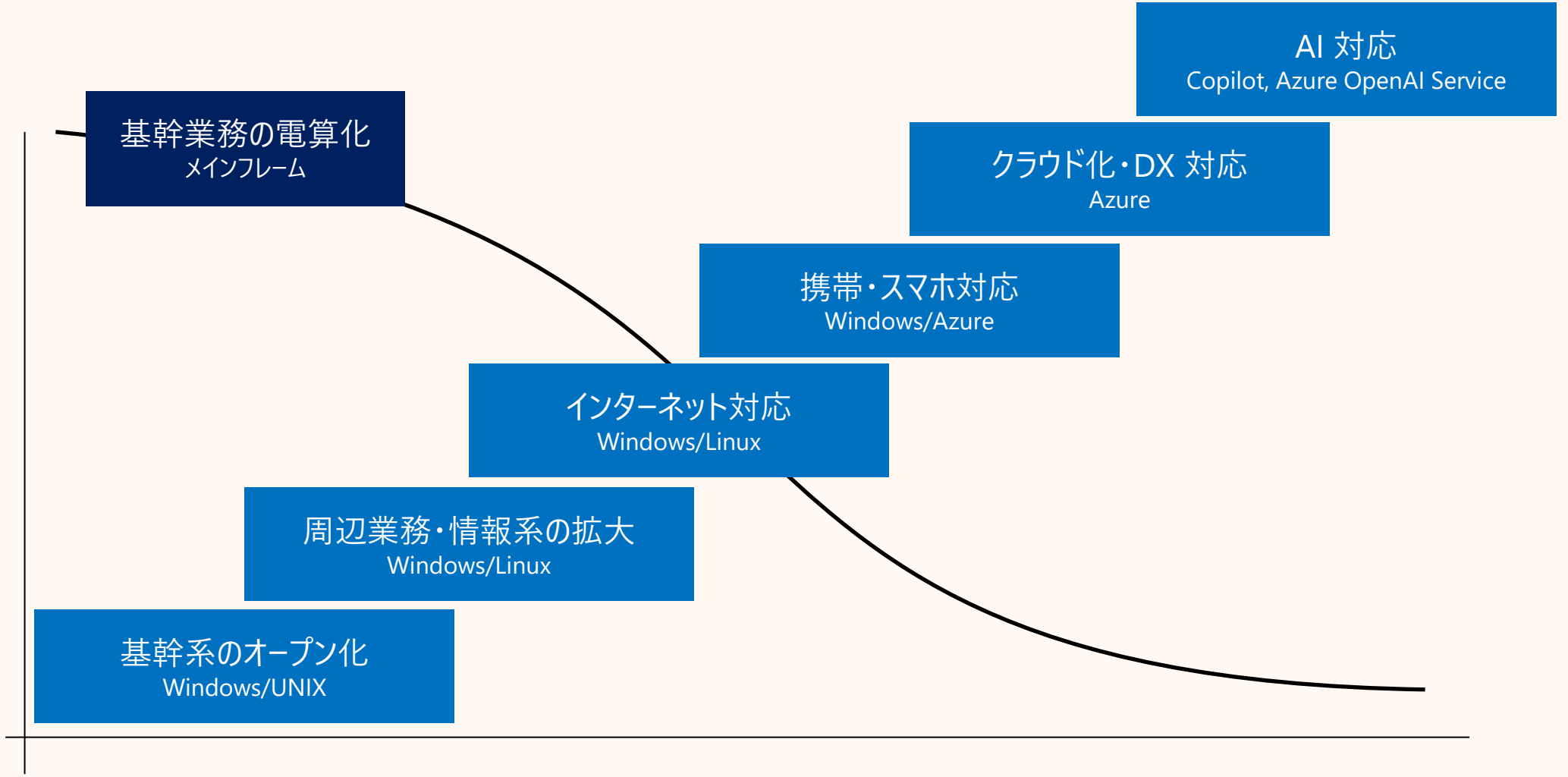


メインフレームの出荷台数



ソース：JEITA サーバー出荷実績 https://home.jeita.or.jp/cgi-bin/page/detail.cgi?n=1449&ca=1
2022年度でメインフレーム出荷統計は終了

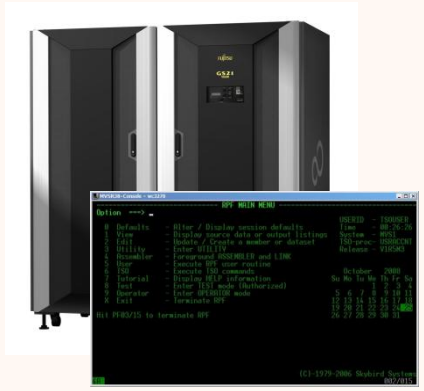
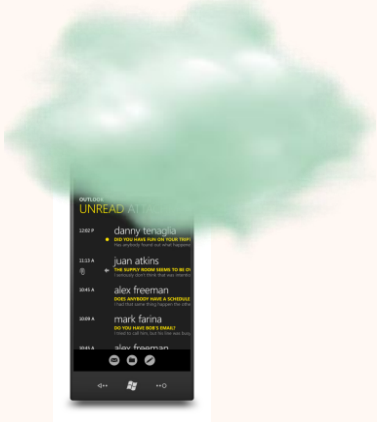
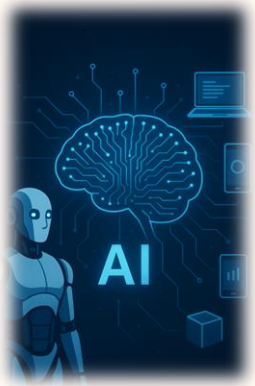
IT 投資領域の変遷



重要な企業データ・基幹系を安定運用しつつ、
最新技術を利用しデータの利活用

企業データを活用し続けるため、
技術投資が活発なプラットフォームを活用

IT 業界の変遷/テクノロジー視点

メインフレーム	クライアントサーバ	インターネット	クラウド/モバイル	生成 AI
				
集中処理 トランザクション バッチ処理	分散処理 エンドユーザコンピューティング ダウンサイジング	インターネット Web化 アプリケーションサーバ	仮想化、SOA SaaS、クラウド モバイル、アプリ配信	コンテナ、マイクロサービス、ゼロトラスト、GPU、AI データセンター
階層構造	クライアントサーバ	3 階層	サービス指向アーキテクチャ モバイル構成、クラウド基盤	マイクロサービス、サーバーレス、 API 連携、MCP/A2A
COBOL FORTRAN PL/I	Visual Basic, C/C++、 Delphi, SQL、Windows API	HTML/CSS, JavaScript, JavaApplet, ActiveX, Java, .NET, PHP	HTML5, JavaScript, Java, .NET, Objective-C、 Swift	Node.js, Python, TypeScript, Swift, Go, Rust, Kotlin, Java, .NET
ダム端末 専用回線	パソコン、電話回線、 イーサネット回線	Webブラウザ、携帯電話、 インターネット	スマートフォン、タブレット、 クラウド、IoT デバイス, 4G	ChatGPT, Copilot, AI デバイス, スマートグラス, 5G

COBOL はどれだけ利用されているのか？

COBOLの市場規模が従来の3倍に拡大、最新の独自調査で判明

グローバル調査により、COBOLアプリケーション活用のモダナイゼーションに巨大な市場機会があることが明らかに

※本資料は、Micro Focus社が、2月４日（現地時間）に発表したプレスリリースの抄訳です。

報道関係各位

2022年2月16日

Micro Focus（LSE： MCRO; NYSE: MFGP、以下マイクロフォーカス）は本日、独立系グローバル市場調査会社に委託した調査結果を発表しました。この調査により、かつてないほど大量のCOBOLコードが使用されており、アプリケーションモダナイゼーションに向け顕著な市場機会があることが明らかになりました。昨年の調査報告をさらに追跡した今年の調査では、世界のCOBOLアプリケーションの普及率は引き続き上昇しており、回答者の大多数が年内にCOBOLアプリケーションをモダナイズして継続利用し、クラウドをサポートする意向であることが示されました。

このグローバル調査によると、回答者の92%が引き続きCOBOLを戦略的な言語と見なしており、日常的に使用されているCOBOLコードの量は7,750～8,500億行と大幅に増加しています。マイクロフォーカスは、基幹業務アプリケーションのモダナイゼーションにおいて45年以上にわたる実績を有し、最近ではAWS Mainframe Modernizationサービスの主要パートナーになったことを発表しています。今回のCOBOLの利用状況に関する最新報告も、マイクロフォーカスが定期的に実施している市場分析プログラムの一環として委託したものです。

マイクロフォーカスのCOBOL製品マーケティング担当ディレクターであるエド・エアリー（Ed Airey）は次のように述べています。「企業は、モダナイゼーションやデジタルトランスフォーメーション（DX）の取り組みを通じてIT戦略を実現しようとしています。最新の調査結果で、アプリケーションモダナイゼーションや業務改革において、COBOLが引き続き重要な役割を果たしていることが明らかになりました。8,000億行ものコードが存在していることは、この最も信頼されている基幹業務システム技術の重要性をさらに高め、継続的な投資の必要性を裏付けるものです。市場に存在するこの膨大なCOBOLアプリケーションコードは、企業や組織にとって注目すべき価値であり、大規模なモダナイゼーション戦略の一環として継続的な投資を必要としています。ITリーダーにとって基幹業務システムを支えるCOBOLアプリケーションのモダナイゼーション（最新環境での活用）は、デジタル変革の中核をなすものです」

世界的な調査分析会社であるVanson Bourne社が実施したこのグローバル調査は、49か国のCOBOLに関わるアーキテクト、ソフトウェアエンジニア、開発者、開発マネージャー、およびIT管理者を対象として、本番システムで稼働するCOBOLアプリケーションコードの量、ビジネスにおけるCOBOLアプリケーションの戦略的重要性、将来のアプリケーションロードマップ、計画、およびリソースについて調査・算定を依頼しました。

世界のCOBOLコード量が過去最高を記録：8,000億行以上のコードが日常的に本番システムで実行されており、これまでの推定値を大きく上回っています。

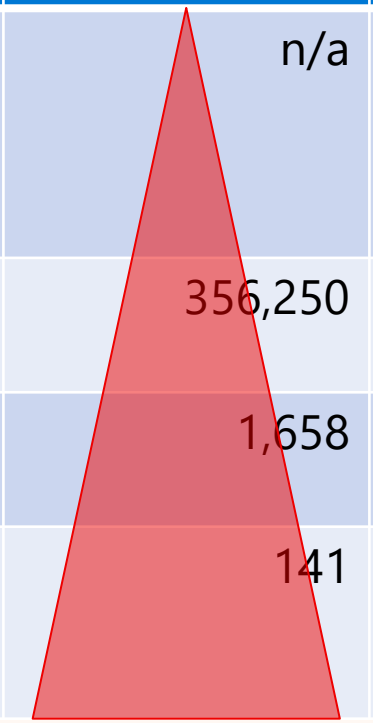
方向性は継続的な成長：回答者の約半数は、今後12か月間に自身の所属する企業や組織で使用されるCOBOLの量が増加すると予想しています。さらに、昨年の調査報告書では、回答者の半数以上（52%）がCOBOLアプリケーションは少なくとも今後10年間は存続すると予想しており、5人に4人以上が、自身が退職した後もCOBOLが引き続き使用されると考えています。したがって、次世代開発者のためにもCOBOLへの投資とモダナイゼーションを継続する必要性を感じています。

COBOLは企業や組織にとって依然として戦略的：回答者の92%が自身の所属する企業や組織のCOBOLアプリケーションは戦略的であり、将来のIT戦略とアプリケーションポートフォリオと最新技術の整合性が、モダナイゼーションの主な推進要因として挙げられています。

ソース：COBOLの市場規模が従来の3倍に拡大、最新の独自調査で判明
https://www.amc.rocketsoftware.co.jp/about/news/press/2022/0216/

実行環境の規模感

カテゴリ	年間出荷台数	用途の例
クラウド	n/a	メール/情報共有、ファイルサーバ、ID サーバー、LOB アプリ、Web アプリ、DB サーバ、開発環境、CAE/HPC、ML ※ SaaS, PaaS 等から、VM 数/インスタンス数など台数換算での統計情報なし
PC サーバ (IA サーバ)	356,250	LOB アプリ、Web アプリ、DB サーバ、メール/情報共有、ファイルサーバ、プリントサーバ、Active Directory、開発環境、CAE/HPC、ML、組み込み等
UNIX サーバ ミッドレンジ	1,658	「止められない基幹業務」「長期運用システム」(Linux に移行しにくいシステム) DB サーバ、LOB アプリ、Web アプリ等
メインフレーム	141	「止められない基幹業務」「長期運用システム」 基幹のマスターデータ管理、計算処理、ビジネストランザクション管理、資産管理等



ソース(メインフレーム/UNIX サーバ 2022年)： JEITA サーバー出荷実績 https://home.jeita.or.jp/cgi-bin/page/detail.cgi?n=1449&ca=1
統計参加会社：沖電気工業、セイコーエプソン、日本アイ・ビー・エム、日本電気、日立製作所、富士通、レノボ・エンタープライズ・ソリューションズ
ソース(PCサーバ)： MM総研「2022年度国内 PC サーバー出荷台数調査」 https://www.m2ri.jp/release/detail.html?id=584
※ガクラウド事業者などが ODM メーカーなどから調達する、自社専用設計のPCサーバーを含まず

Agenda

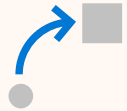
改めて「モダナイズ」「モダン化」とは

レガシー資産を
最新の技術基盤・開発運用プロセスへ移行することで、
拡張性・保守性・俊敏性を高め、
ビジネス価値を持続し向上させる取り組み

クラウドによるモダナイゼーションのきっかけ

データセンターの契約期限

S/W のサポート終了



S/W と H/W の更改

災害や障害、キャパシティ不足への対応

アプリと機能をより早く提供する

新しいビジネスの機会

セキュリティの脅威を特定

コンプライアンスに準拠

経験者や有識者の不足

「モダン化」による変化

レガシー資産を最新の技術基盤・開発運用プロセスへ移行し、拡張性・保守性・俊敏性を高め、ビジネス価値を持続させる取り組み

	現状	モダン化
アーキテクチャ	レガシーOSやフレームワーク、オンプレ専用の仕組み	クラウドネイティブ、Webベース、モジュール化された構成
	モノリシックアプリ	マイクロサービス
開発プロセス	ウォーターフォール中心	アジャイル/DevOps
	人手での手作業リリース	CI/CD パイプラインによる自動化
運用の高度化	インフラ管理	Infrastructure as Code
	障害対応	監視 + 自動リカバリ + 可観測性 (Observability)
	セキュリティ/コンプライアンス対応を導入時の基準で維持	セキュリティ/コンプライアンス対応を最新の基準から継続進化

クラウド移行戦略とモダナイゼーション戦略

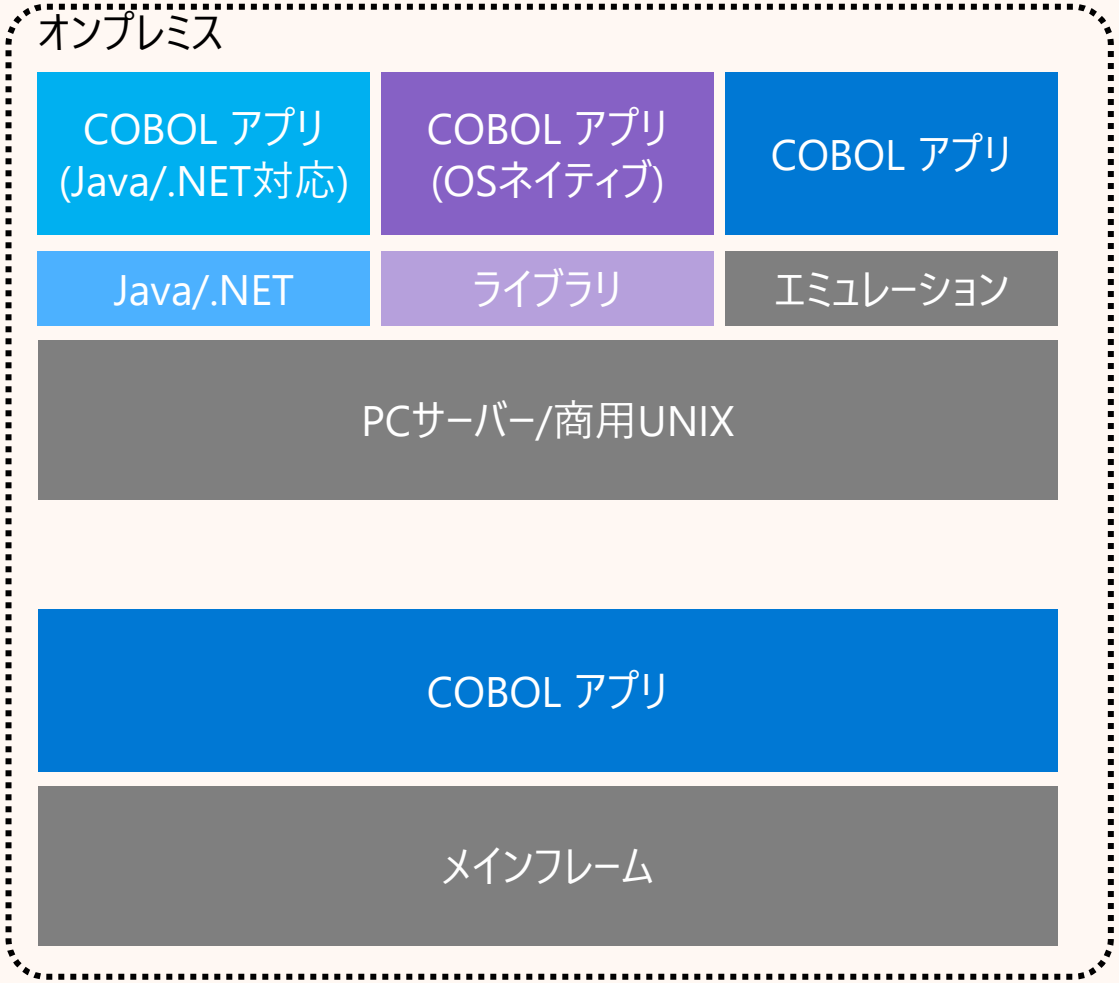
	Rehost	Refactor	Rearchitect	Rebuild
	As-is のままクラウドへ移行	クラウドを最大限に活用するように最小限の変更を加える	アプリケーションをサービスに合わせ大幅に変更/分解する	クラウド・ネイティブアプローチで書かれた新しいコード
ニーズ	• 初期投資を抑える • データセンターの空きを作る • 素早く ROI を出す	• 素早くアップデート • コードのポータビリティ • よりクラウドの効率性を活用 (リソース, スピード, コスト)	• 拡張性・柔軟性 • 新しいクラウドの能力を容易に適用 • 混合した技術スタック	• イノベーションを加速 • アプリを素早くビルド • 運用コストを削減
テクノロジー	**IaaS**	**Containers** **PaaS**	**PaaS** **Serverless** **Microservices**	

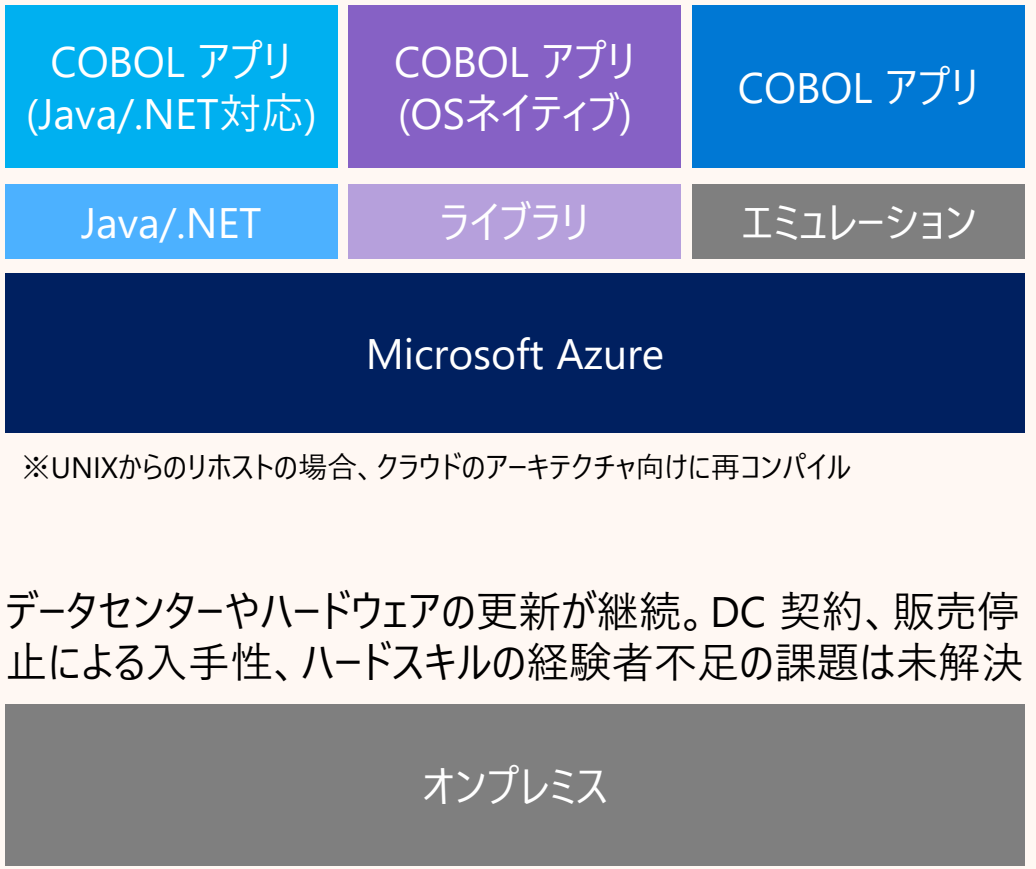
マイグレーション

アプリケーション モダナイゼーション

リホストによるマイグレーション

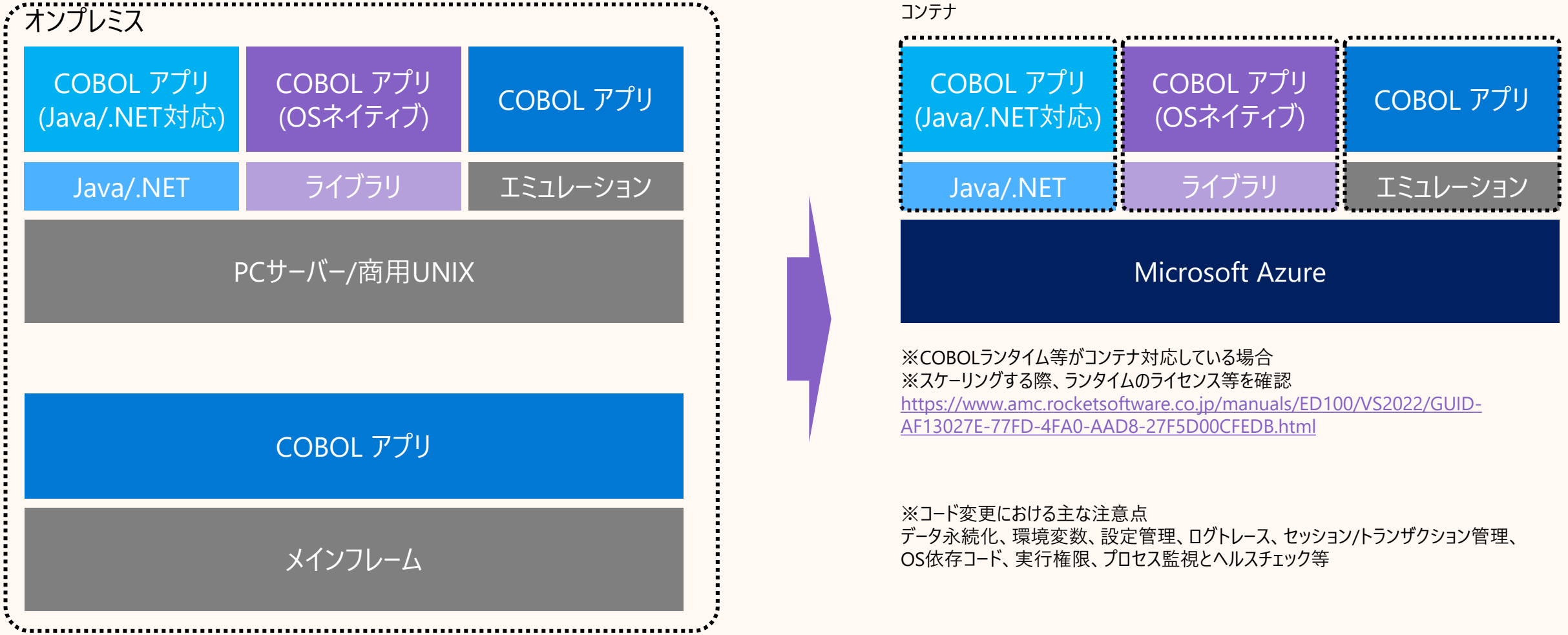
クラウドへの「リホスト」による、データセンター契約やハードウェア調達、経験者不足の課題の解決
互換性を重視、コードの大幅な見直しが難解・時間がかかる場合の打ち手





リファクタによるモダナイゼーション

クラウドマイグレーションの恩恵 + 最小限のコード変更でのコンテナ化の恩恵の獲得
ロールバックが容易、柔軟なスケーリング、依存関係の分離による安定化など

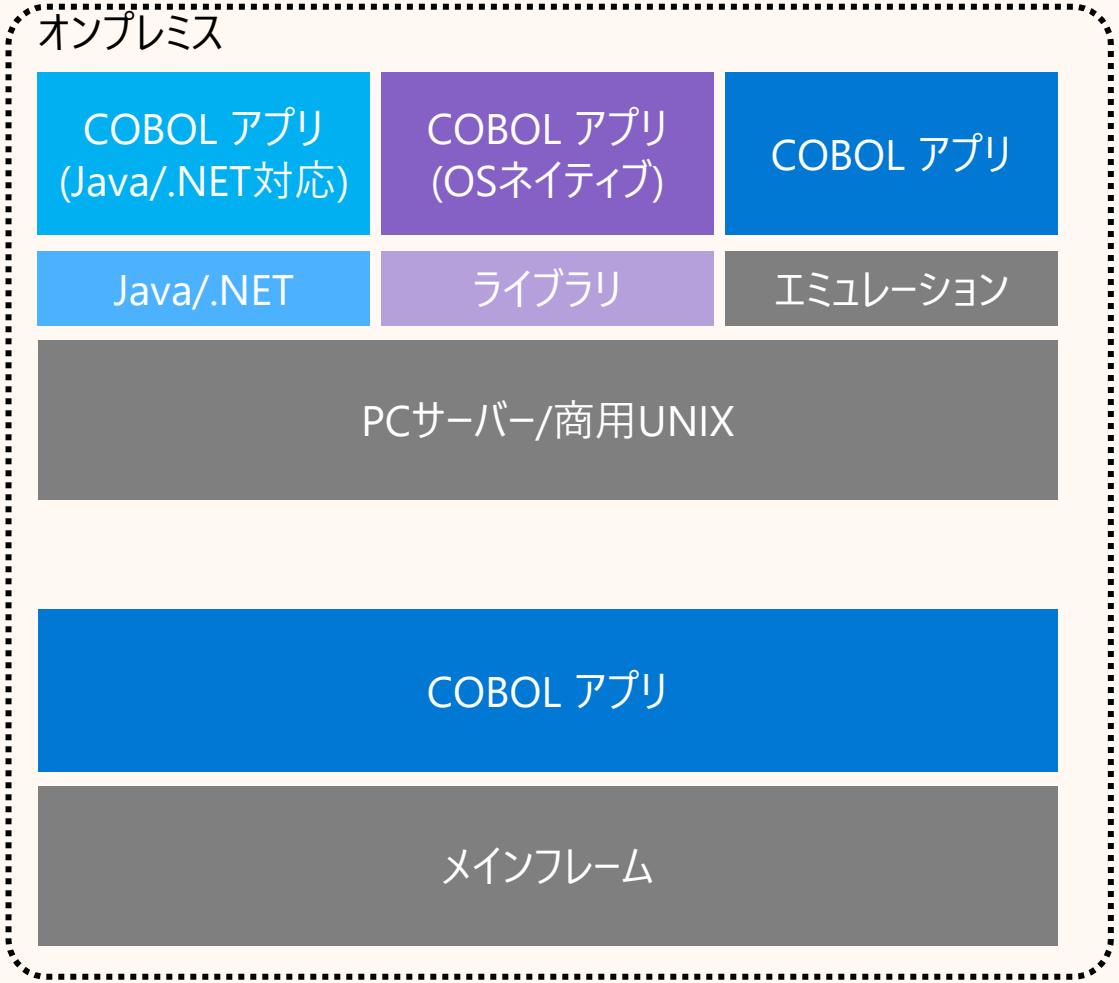


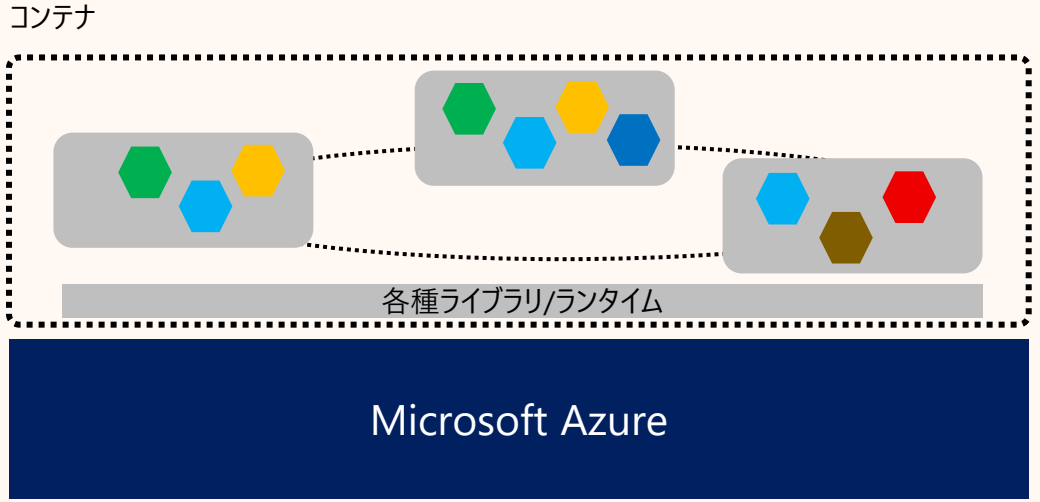
※COBOLランタイム等がコンテナ対応している場合
※スケーリングする際、ランタイムのライセンス等を確認
https://www.amc.rocketsoftware.co.jp/manuals/ED100/VS2022/GUID-AF13027E-77FD-4FA0-AAD8-27F5D00CFEDB.html

※コード変更における主な注意点
データ永続化、環境変数、設定管理、ログトレース、セッション/トランザクション管理、
OS依存コード、実行権限、プロセス監視とヘルスチェック等

リアーキテクト・リビルドによるモダナイゼーション

クラウドマイグレーションの恩恵 + クラウドネイティブによる拡張性と柔軟性、イノベーションの獲得
クラウド・マネージドサービスの能力を最大限に活用しデプロイ速度向上と運用コストの削減





選択のポイント
共通：
　類似ニーズのアプリが分散
　モノリシックアプリをマイクロサービス化
　更新サイクル頻度、アプリ利用時間の変動、大量トラフィック
リアーキテクト：既存コード活用を重要視
リビルド：AI、IoT など新機能の柔軟な追加を期待

クラウドへの COBOL モダナイゼーションによる課題解決

データセンターの契約期限

S/W のサポート終了

S/W と H/W の更改

災害や障害、キャパシティ不足への対応

アプリと機能をより早く提供する

新しいビジネスの機会

セキュリティの脅威を特定

コンプライアンスに準拠

経験者や有識者の不足

リホスト

リファクタ

リアーキテクト・リビルド

Azure における代表的なホスティング環境

リホスト

Azure VM

・Windows/Linux の仮想マシンをオンデマンドでプロビジョニング可能な IaaS

・ハイブリッド接続、拡張セットによるオートスケール等

リファクタ

Azure App Service

・フルマネージド Web アプリケーション ホスティング PaaS

・シングル、および、マルチコンテナが実行可能

Azure Container Instances (ACI)

・高速/シンプルなコンテナー ホスティング サービス

・シングル コマンドで実行可能なクラスタ フリー コンテナ環境

リアーキテクト・リビルド

Azure Kubernetes Service (AKS)

・マネージド Kubernetes サービス

・Master クラスターを Azure が管理

・Azure セキュリティ、ID、コスト管理、移行サービスと相互運用

Azure Red Hat OpenShift (ARO)

・フルマネージド OpenShift サービス、Kubernetes を拡張

・Entra ID との統合など強化されたセキュリティ

・Red Hat & Microsoft が共同でサポート

Azure Container Apps (ACA)

・コンテナ化されたサーバーレスプラットフォーム

・マイクロサービスとコンテナー化されたアプリケーションを実行できる

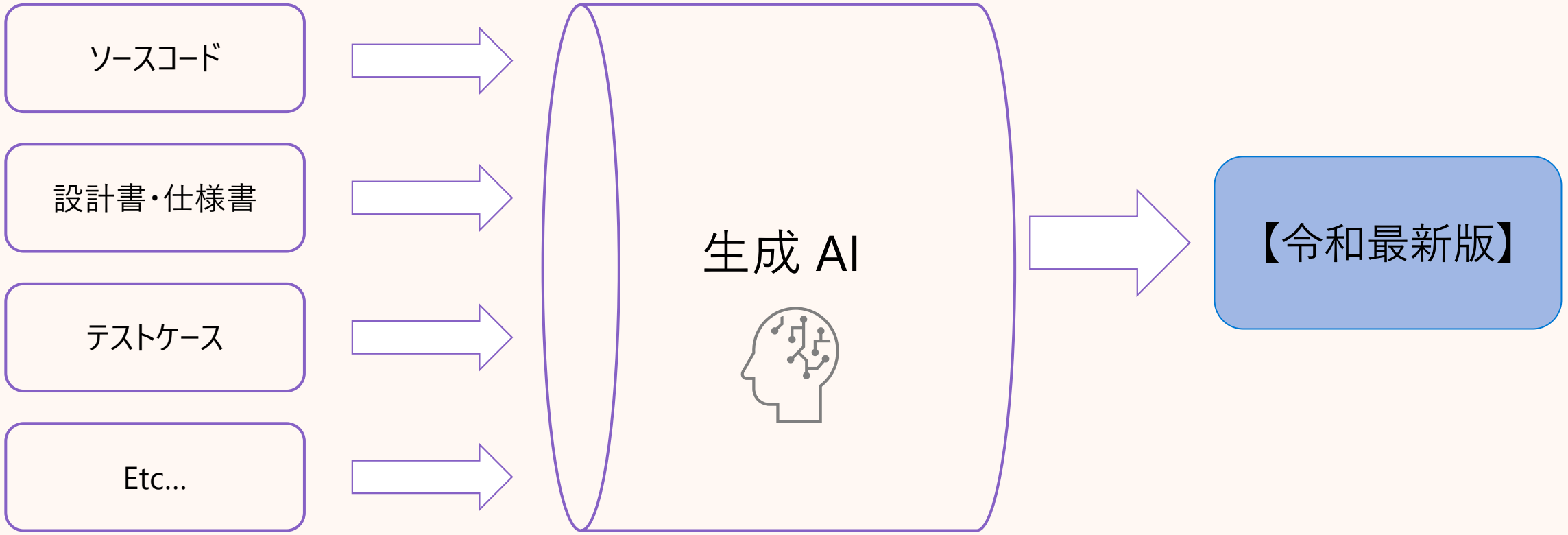
Agenda

開発における生成 AI 活用について、よくある問い合わせ

AI 対話によるコード生成	コメントがないコードの内容を分析し、開発を効率化したい 試行錯誤を増やし、品質と生産性を向上したい
他の開発言語への変換	C# で標準化を進めているので、VB6/VB.NET のコードを C# に変換したい C# エンジニアを多く抱えているのでスキルを有効活用したい
コードからドキュメント生成	ドキュメント作成や更新の手間をなくしたい コードとドキュメントの乖離をなくしたい
ドキュメントからコード生成	ドキュメントの変更からコード更新を加速させたい ドキュメントとコードを常に一致させることで説明責任を果たしたい

生成 AI 活用の願望

仕様書やコードなどなど、全部 AI に読み込ませると、いい感じにモダンに仕上げて欲しい



「生成 AI がモダナイズに使えるか？」生成 AI に聞いてみた

例：Windows クライアントから Web UI+Web API化における生成 AI 活用アイディア出し

フェーズ	発生作業	生成AI活用例	GitHub系活用可能性	関連ドキュメント
1. 現状分析・要件定義	・既存Windowsクライアントの機能棚卸し ・業務フロー分解（ユースケース抽出） ・依存関係の洗い出し・移行ロードマップ作成	・ソースコード要約、仕様抽出 ・ユースケース/シーケンス図自動生成 ・制約や考慮点のリスト化支援	Copilot Chatで既存コードの意味を質問し要件理解を効率化	要求仕様書 要件定義書
2. アーキテクチャ設計	・WebAPI境界定義 ・認証/認可設計 ・UIフレームワーク選定 ・非機能要件整理	・アーキテクチャパターン提示 ・雛形プロジェクト生成 ・セキュリティレビュー補助	Copilotでサンプル構成YAML/IaC雛形を生成、設計ドキュメント化支援	要件定義書 仕様書 設計書
3. UI/UX 設計	・画面リデザイン（Web化、レスポンシブ対応） ・UIコンポーネント設計・アクセシビリティ対応	・React/Blazor 等の画面モック生成 ・旧UIの改善提案 ・WinForms風UI → WebUI変換支援	CopilotでUIコードの雛形生成、デザインパターンの適用支援	設計書 ソースコード
4. API 開発	・API仕様策定（OpenAPI）・ビジネスロジックのAPI化 ・データアクセス抽象化	・DB/クラス定義からOpenAPI生成 ・レガシーコードのWebAPI化支援 ・テストコード生成	CopilotでWebAPI/DBアクセスコード生成、Coding Agentでリファクタ作業自動化	設計書 ソースコード
5. 移行・データ統合	・段階的API化（Strangler Fig）・データ移行/変換・並行稼働検証	・移行スクリプト生成 ・移行シナリオ提案 ・テストデータ自動生成	CopilotでSQL/ETLスクリプト補完、マイグレーションコード生成	設計書 ソースコード
6. テスト・検証	・単体/統合/E2Eテスト ・ブラウザ互換確認・性能テスト	・テストケース生成 ・E2Eテストコード生成（Playwright等） ・ログ解析と根本原因提示	Copilotでテストコード自動補完、Coding Agentでテストカバレッジ改善提案	仕様書 設計書 ソースコード
7. 運用・保守	・CI/CD整備 ・監視 ・ログ設計 ・サポート体制構築	・IaCコード生成（Terraform/Bicep） ・マニュアル要約/自動生成 ・障害解析支援（ログ→原因提案）	GitHub ActionsのYAML生成支援、Copilot Chatで障害解析やRunbook生成支援	仕様書 設計書

Microsoft 365 Copilot Chat による COBOL コード分析

この COBOL のコードを解析して　　CLASS-ID. BookRecordSet As "BookRecordSet".
　　REPOSITORY.
　　　CLASS cls-string As "System.String"
　　　CLASS cls-Decimal As "System.Decimal"
　　　CLASS cls-EndOfFileException As "EndOfFileException"
　　　CLASS cls-BookRecord As "BookRecord"
　　　.
　　STATIC.
　　END STATIC.

　　OBJECT.

　　INPUT-OUTPUT SECTION.
　　FILE-CONTROL.
　　　select bookfile assign to FilSpec
　　　　file status is ws-file-status
　　　　organization is indexed
　　　　access mode is sequential
　　　　record key is b-stockno
　　　　alternate record key is b-title with duplicates
　　　　alternate record key is b-author with duplicates
　　　　.

　　DATA DIVISION.
　　File section.
　　FD　bookfile.
　　copy "book-rec.cpy" replacing ==(prefix)== by ==b==.
　　OBJECT-STORAGE SECTION.
　　01 bookTitle　　OBJECT REFERENCE cls-string
　　　property as "bookTitle".
　　01 bookType　　OBJECT REFERENCE cls-string
　　　property as "bookType".
　　01 bookAuthor OBJECT REFERENCE cls-string
　　　property as "bookAuthor".
　　01 stockNumber　OBJECT REFERENCE cls-string

Copilot

このCOBOLコードは、.NET環境で動作する**オブジェクト指向COBOL**（たとえばMicro Focus Visual COBOLなど）で書かれており、書籍情報を管理するクラス BookRecordSet と、ファイルの終端に達したときにスローされるカスタム例外クラス EndOfFileException を定義しています。以下に、コードの構造と動作をわかりやすく解説します。

📘 BookRecordSet クラスの概要

🔹 目的

- 書籍情報を格納したファイル（bookfile.dat）を読み込み、1件ずつレコードを取得する。
- ファイルの終端に達した場合は EndOfFileException をスローする。

🔹 主な構成

🗂️ ファイル定義

COBOL

```
1  select bookfile assign to FilSpec
```

- bookfile.dat というインデックス付きファイルを読み込み対象とする。
- 主キーは b-stockno、副キーとして b-title と b-author を使用。

🧱 オブジェクトストレージ

COBOL

```
1  01 bookTitle OBJECT REFERENCE cls-string property as "bookTitle".
```

- 各フィールドは .NET のクラス（System.String, System.Decimal など）を参照。
- property as により、.NET 側からプロパティとしてアクセス可能。

🛠️ メソッド

New メソッド

COBOL

```
1  METHOD-ID. New.
```

Copilot へメッセージを送る

＋　ツール

AI で生成されたコンテンツは誤りを含む可能性があります。

Microsoft 365 Copilot Chat によるコード変換例

以下の Pro＊Cのコードを COBOLに変換してください

```
include <sqlca.h>
EXEC SQL BEGIN DECLARE SECTION;
int emp_id;
char emp_name[100];
EXEC SQL END DECLARE SECTION;

EXEC SQL SELECT employee_name INTO :emp_name FROM
employees WHERE employee_id = :emp_id;
```

Copilot

以下は、提示された Pro*C コードを COBOL に変換した例です。COBOL では、埋め込み SQL を使用するために **EXEC SQL** 構文を利用します。

COBOL 変換例

COBOL

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. EMPLOYEE-SELECT.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       EXEC SQL INCLUDE SQLCA END-EXEC.

       EXEC SQL BEGIN DECLARE SECTION END-EXEC.
       01 EMP-ID        PIC S9(9) COMP.
       01 EMP-NAME      PIC X(100).
       EXEC SQL END DECLARE SECTION END-EXEC.

       PROCEDURE DIVISION.
           MOVE 12345 TO EMP-ID  *> 例として社員IDを設定

           EXEC SQL
               SELECT EMPLOYEE_NAME
               INTO :EMP-NAME
               FROM EMPLOYEES
               WHERE EMPLOYEE_ID = :EMP-ID
           END-EXEC.

           DISPLAY "社員名: " EMP-NAME.

           STOP RUN.
```

簡易表示

主なポイント

- **SQLCA** は EXEC SQL INCLUDE SQLCA END-EXEC でインクルード。
- Pro*C の int emp_id は COBOL では PIC S9(9) COMP。

接続処理とエラー処理を含む完全なCOBOLサンプルを見せて

Copilot

以下は、**接続処理**と**エラー処理**を含む完全な COBOL サンプルです。DB2 を想定した埋め込み SQL の例です。

✅ 完全 COBOL サンプル（DB接続＋SELECT＋エラー処理）

COBOL

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. EMPLOYEE-SELECT.

       ENVIRONMENT DIVISION.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       EXEC SQL INCLUDE SQLCA END-EXEC.

       EXEC SQL BEGIN DECLARE SECTION END-EXEC.
       01 USER-ID        PIC X(20) VALUE 'DBUSER'.
       01 USER-PASS      PIC X(20) VALUE 'DBPASS'.
       01 EMP-ID         PIC S9(9) COMP VALUE 12345.
       01 EMP-NAME       PIC X(100).
       EXEC SQL END DECLARE SECTION END-EXEC.

       PROCEDURE DIVISION.
       MAIN-PARA.
           *> --- DB接続 ---
           EXEC SQL
               CONNECT TO SAMPLE USER :USER-ID USING :USER-PASS
           END-EXEC.
           PERFORM CHECK-SQLCODE.

           *> --- データ取得 ---
           EXEC SQL
               SELECT EMPLOYEE_NAME
               INTO :EMP-NAME
               FROM EMPLOYEES
               WHERE EMPLOYEE_ID = :EMP-ID
           END-EXEC.
           PERFORM CHECK-SQLCODE.

           DISPLAY "社員名: " EMP-NAME.

           *> --- DB切断 ---
           EXEC SQL
               DISCONNECT CURRENT
           END-EXEC.
           PERFORM CHECK-SQLCODE.

           STOP RUN.

       *> --- エラー処理 ---
       CHECK-SQLCODE.
           IF SQLCODE NOT = 0
               DISPLAY "SQL ERROR: " SQLCODE
               DISPLAY "SQLERRMC: " SQLERRMC
               STOP RUN
           END-IF.
```

簡易表示

✅ ポイント解説

- **SQLCA** は必須 (EXEC SQL INCLUDE SQLCA END-EXEC)。
- **接続処理**：CONNECT TO SAMPLE USER :USER-ID USING :USER-PASS
- **エラー処理**：SQLCODE をチェックし、エラー時は SQLERRMC で詳細表示。
- **切断処理**：DISCONNECT CURRENT
- ホスト変数は：を付ける。
- 実際の環境では **DB名・ユーザー名・パスワード** を適切に設定。

💡 追加できます：

- トランザクション制御（COMMIT / ROLLBACK）
- 複数レコード取得（カーソル使用）
- 詳細なエラーコード別処理

質問：

この後、**カーソルを使って複数行を取得する COBOL サンプル**も見せましょうか？

それとも、**COMMIT/ROLLBACK を含むトランザクション処理付きサンプル**が必要ですか？

AI で生成されたコンテンツは誤りを含む可能性があります。

Microsoft 365 Copilot Chat によるコード変換例

Pro*Cのコードを COBOLに変換して

```
#include <stdio.h>
#include <sqlca.h>int main() {
    char emp_name[50];
    int emp_id;

    EXEC SQL SELECT ename INTO :emp_name
            FROM emp
            WHERE empno = :emp_id;
}
```

Copilot

以下は、提示されたPro*CコードをCOBOLに変換した例です。COBOLでは、埋め込みSQLを使う場合、EXEC SQL構文を利用します。

COBOL版コード例

COBOL

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. EMP-SELECT.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 EMP-NAME        PIC X(50).
       01 EMP-ID          PIC 9(9).

       EXEC SQL INCLUDE SQLCA END-EXEC.

       PROCEDURE DIVISION.
           DISPLAY "Enter Employee ID: ".
           ACCEPT EMP-ID.

           EXEC SQL
               SELECT ENAME
               INTO :EMP-NAME
               FROM EMP
               WHERE EMPNO = :EMP-ID
           END-EXEC.

           IF SQLCODE = 0
               DISPLAY "Employee Name: " EMP-NAME
           ELSE
               DISPLAY "Error: SQLCODE = " SQLCODE
           END-IF.

           STOP RUN.
```

簡易表示

ポイント

- EXEC SQL INCLUDE SQLCA END-EXEC. はSQL通信エリアを取り込むために必須。
- :EMP-NAMEや:EMP-IDのように、ホスト変数はコロン付きで指定。

接続処理とエラーハンドリングを含めた完全なサンプルを作成して

Copilot

以下は、**接続処理・エラーハンドリング・トランザクション制御**を含めた完全なCOBOLサンプルです。Pro*COBOLを前提としています。

COBOL完全サンプル（Oracle接続＋SELECT＋エラーハンドリング）

COBOL

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. EMP-SELECT.

       ENVIRONMENT DIVISION.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 EMP-NAME        PIC X(50).
       01 EMP-ID          PIC 9(9).
       01 USERNAME        PIC X(20) VALUE "scott".
       01 PASSWORD        PIC X(20) VALUE "tiger".
       01 CONNECT-STRING  PIC X(50) VALUE "ORCL".

       EXEC SQL INCLUDE SQLCA END-EXEC.

       PROCEDURE DIVISION.
       MAIN-PARA.
           DISPLAY "Connecting to Oracle...".

           EXEC SQL
               CONNECT :USERNAME IDENTIFIED BY :PASSWORD USING :CONNECT-STRING
           END-EXEC.

           PERFORM CHECK-SQLCODE.

           DISPLAY "Enter Employee ID: ".
           ACCEPT EMP-ID.

           EXEC SQL
               SELECT ENAME
               INTO :EMP-NAME
               FROM EMP
               WHERE EMPNO = :EMP-ID
           END-EXEC.
```

DBはSQL Serverです。Pro*COBOL ではない完全コードを作成して

Copilot

SQL Server用のCOBOLコードは、Pro*COBOLではなく**ODBCを利用した標準COBOL**で実装するのが一般的です。以下は、**接続処理・エラーハンドリング・SELECT実行・切断処理**を含む完全なサンプルです。

COBOL + ODBC 完全サンプル（SQL Server対応）

COBOL

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID. EMP-SELECT.

       ENVIRONMENT DIVISION.

       DATA DIVISION.
       WORKING-STORAGE SECTION.
       01 EMP-NAME        PIC X(50).
       01 EMP-ID          PIC 9(9).
       01 SQLSTATE        PIC X(5).
       01 SQLCODE         PIC S9(9) COMP-5.
       01 CONNECT-STRING  PIC X(100)
           VALUE "DSN=SQLServerDSN;UID=sa;PWD=yourpassword".

       PROCEDURE DIVISION.
```

その他の行を表示する

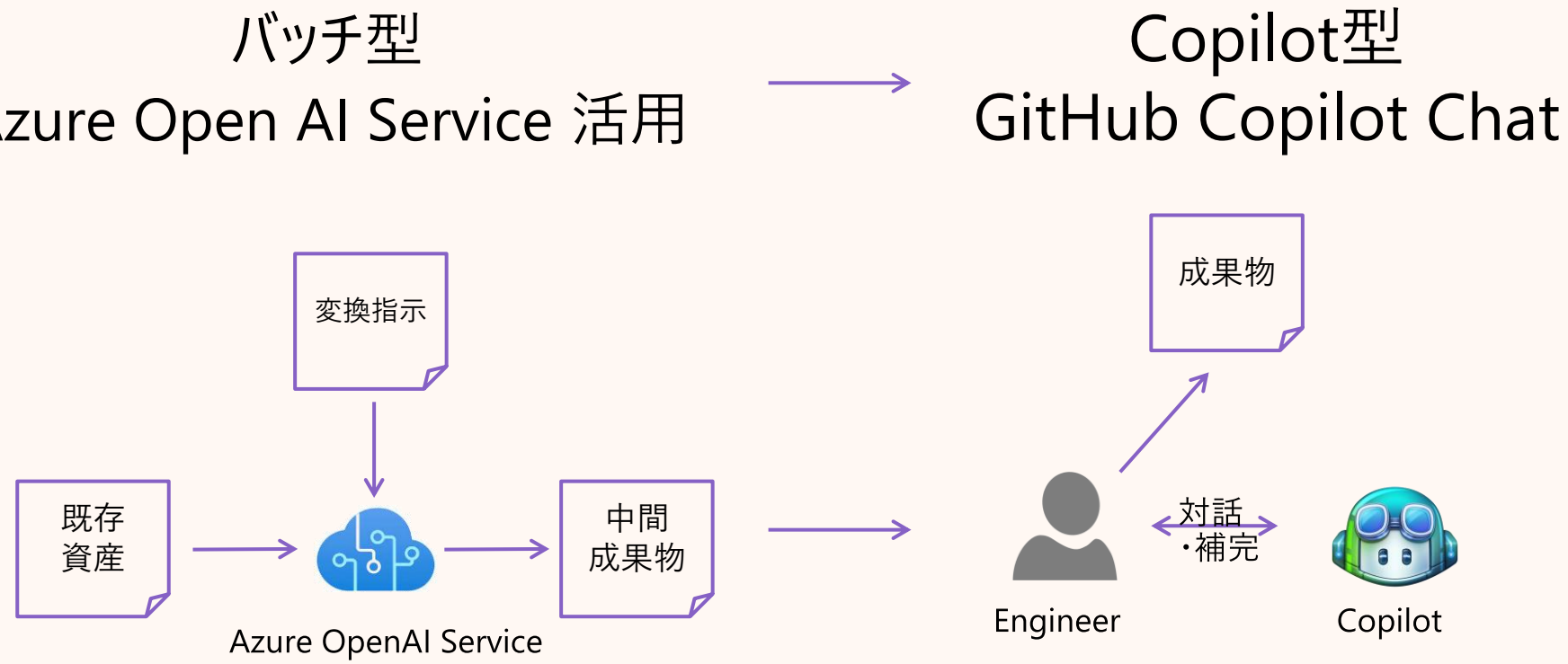
ポイント

- **ODBC API呼び出し**
  - SQLConnect：DSNで接続
  - SQLExecDirect：SQL文実行
  - SQLFetch：結果取得
  - SQLDisconnect：切断
- **DSN設定**
  WindowsのODBCデータソースにSQLServerDSNを設定しておく必要があります。

AI で生成されたコンテンツは誤りを含む可能性があります。

AI 活用の現実的なアプローチ

- 大量のファイルをバッチ型で処理し、人 + AI の協調作業で品質を上げる
  - プロジェクトにある大量のコードとドキュメントをまとめて処理：バッチ型
  - コードやドキュメントの処理結果の確認を AI が支援：Copilot 型



AI 活用のアプローチごとの違い

- バッチ型と Copilot 型でそれぞれ活用戦略や期待値を変える
  - どちらのパターンにおいてもコードの生成を AI に任せっきりにしない
  - 人間と AI が協調してコンテンツを生成していくという意識の必要性は共通

利用用途	特徴	AI が持つ責任	AI 活用の戦略	AI に求めるべき期待値
バッチ型	• 一度に大量のコンテンツを生成可能	• 最終的なアウトプットは人間の手が必要であるものの、中間生成物としての品質には AI の責任が大きい	• より高い品質のアウトプットを得られるよう、人間による品質の高いプロンプトの定義が必要	• いかに品質の高いプロンプトを定義したとしても、期待値通りの回答は得られないという前提で、AI で生成されたコンテンツは人間によるレビューが必要
Copilot 型	• 人間と協調しながら、少量で高速にコンテンツを生成	• パイロットはあくまで開発者であるため、最終的なアウトプットの品質は AI ではなく開発者が持つ	• 開発中に調べたいことが出てくる度に AI に質問を行い、AI からの回答の取捨選択が必要	• AI は開発者の生産性を高めるための手段であり、利用のハードルが低い一方で、AI で生成された内容の確認は必要

GitHub Copilot とは

- GitHub Copilot =「AIコーディングアシスタント」
- 共に作業することで、コード生成を速くして開発の効率化に寄与
- 最近はコーディング以外のGitHubの機能でもCopilotが活躍

Copilot導入のメリット

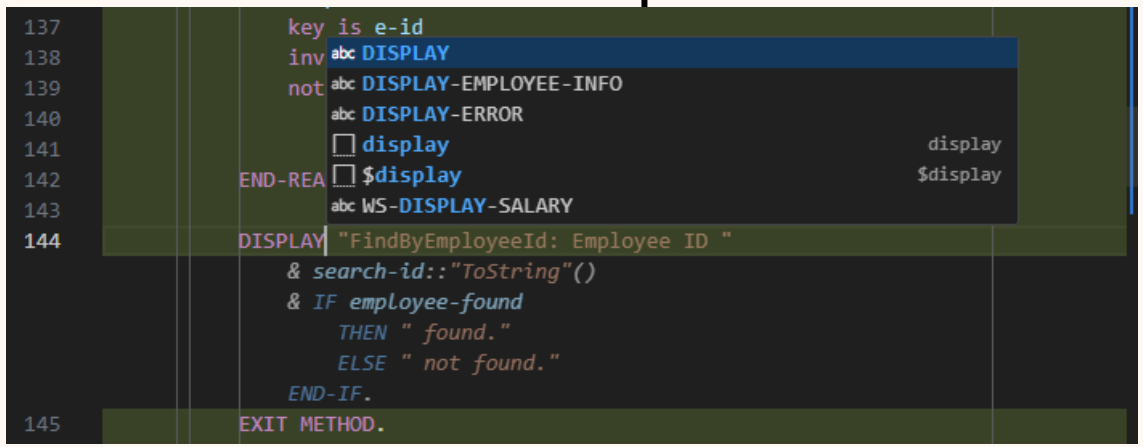
- コード理解・生成のスピード向上
- ベストプラクティスに従った一貫性のある編集で品質向上
- 反復作業やレビューを担うことでチーム全体の作業効率化
- 自力で思いつかない解決策の提案による学習支援

※GitHub Copilot は、パブリック リポジトリで公開されているすべての言語でトレーニングされています。各言語で受け取る提案の品質は、その言語のトレーニング データの量と多様性に応じて異なる場合があります。(https://github.com/features/copilot?locale=ja)

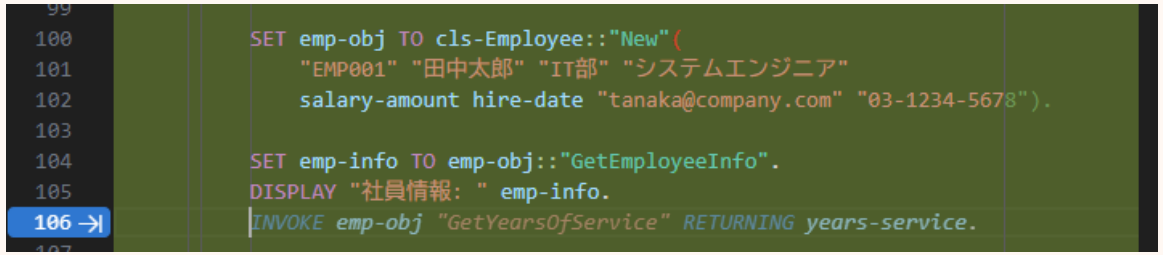
GitHub Copilot を使った COBOL 開発

コード補完

・ソースコードエディター内で Copilot がコード補完を提案する



・Next Edit Suggestions (NES) で関連箇所の改修を続けて提案する



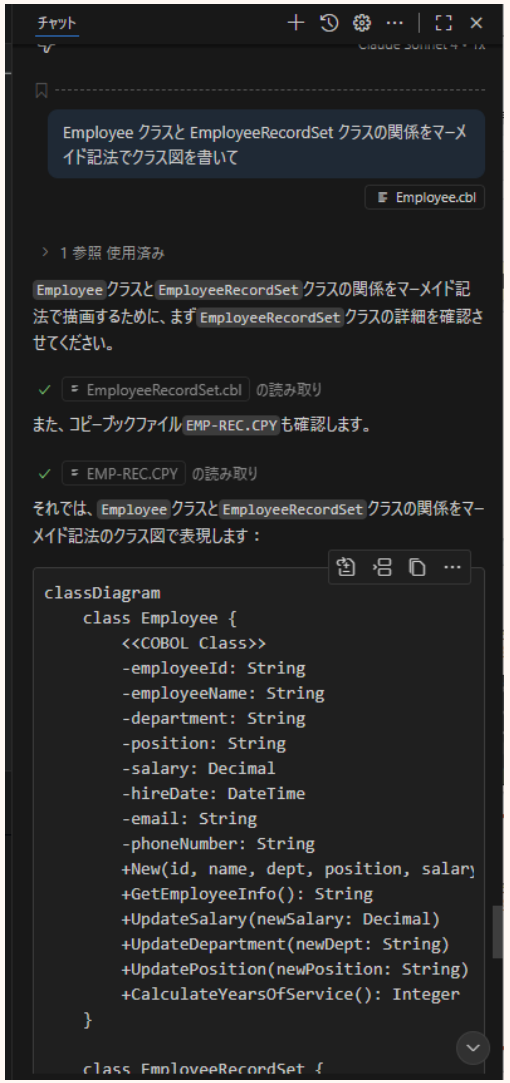
※GitHub Copilot は、パブリック リポジトリで公開されているすべての言語でトレーニングされています。各言語で受け取る提案の品質は、その言語のトレーニング データの量と多様性に応じて異なる場合があります。(https://github.com/features/copilot?locale=ja)

GitHub Copilot を使った COBOL 開発

Copilot Chat

・チャットで Copilot より改修内容の提案や技術的な回答を得る

・モデルによっては画像も読み込める



インラインチャット

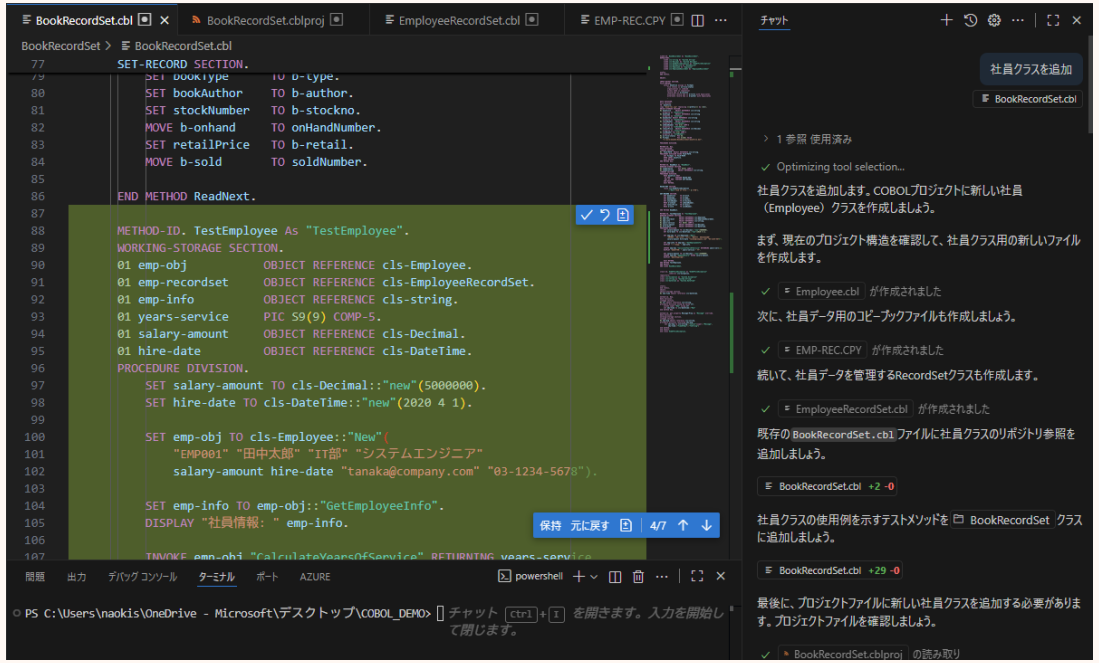
・エディター内で Copilot Chat を使う

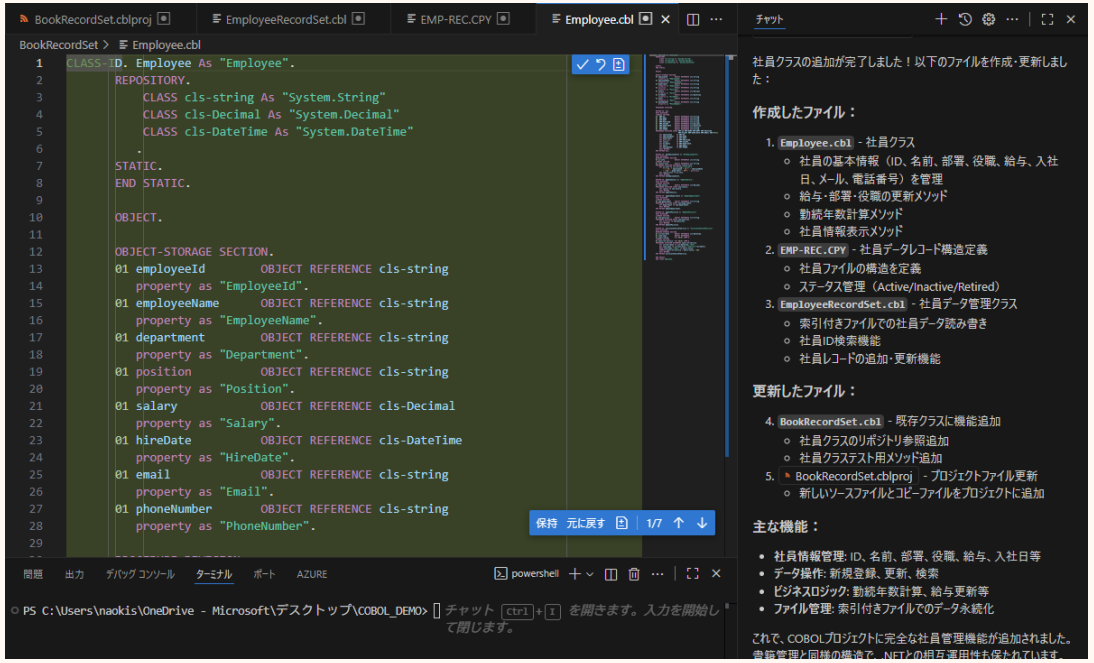
・開いている箇所の回答や修正提案を貰える

※GitHub Copilot は、パブリック リポジトリで公開されているすべての言語でトレーニングされています。各言語で受け取る提案の品質は、その言語のトレーニング データの量と多様性に応じて異なる場合があります。(https://github.com/features/copilot?locale=ja)

GitHub Copilot を使った COBOL 開発

- GitHub Copilot Agent Mode
  - 単純なコード補完や対話チャットを超えた、AI が開発者の指示に対して複数のステップを自律的に実行できるモード





画面ショットの動作環境：Visual Studio Code + GitHub Copilot + Rocket® COBOL extension
https://marketplace.visualstudio.com/items?itemName=RocketSoftware.rocket-cobol

※GitHub Copilot は、パブリック リポジトリで公開されているすべての言語でトレーニングされています。各言語で受け取る提案の品質は、その言語のトレーニング データの量と多様性に応じて異なる場合があります。(https://github.com/features/copilot?locale=ja)

コード生成やってます！仕様書の Excel パラメータ表で

強み	内容・具体例
安定性と成熟した運用/信頼性	多くの SI 案件で長年使われており、仕様書テンプレート、レビュー・承認プロセス、変更管理、ドキュメンテーション等の慣習が確立
大規模システムでの適用経験とノウハウが豊富	基幹系システムやレガシーシステムなど、複雑性・稼働規模が大きいプロジェクトへの対応実績があり、非機能要件への配慮（可用性・信頼性・保守性・運用性）などが仕様段階で比較的手厚く扱われる
明文化・検証可能な仕様書/契約ドキュメントとしての価値	発注者・顧客との約束・合意を仕様書で取ることが標準的。変更管理・トレーサビリティ・コンプライアンスの観点でも仕様書が法的/監査的役割を持つ
コード自動生成による定型処理/ボイラープレート作成の効率	データモデル -> CRUD API, DB ORM, 画面テンプレート等の反復的・定型的処理のコストを削減可能。Excel, DSL, テンプレートでパラメータを与える方式は熟成しており高い信頼性
設計詳細と非機能要件に対する明示的設計	画面仕様書、データ定義書、通信プロトコル、性能要件などを仕様段階で細かく詰めることが多く、設計の予見性・性能保証・運用性の担保に強み
リスク管理・ガバナンス体制が整っている	品質保証・レビュー、テスト設計、保守性・セキュリティの仕様があらかじめ定義され、ステークホルダー・契約者間での確認手続きが明確

生成 AI の強み

領域	生成 AI の強み	Excel パラメータテンプレートの特徴
初期要件の曖昧性の解消	初期段階で仕様を明確にし、人と AI の対話・反復で要求の抜け漏れを減らす	従来から仕様レビューや顧客承認プロセスで曖昧さを潰す手法・プロセスがあるが、人手依存・時間コストが高い
プロジェクト速度・PoC、MVPの効率	AI によるドラフト生成やタスク分割が早く、最初の機能実装までの速度が出しやすい	定型部分の自動生成で速度は出るが、非定型部分は手作業でコーディング。仕様確定・書類化に時間がかかり、初動が重い
柔軟性・技術スタックの変更または新技術導入	技術非依存な仕様設計 -> 技術スタックを後から変えても仕様が活かせる可能性あり。AI を使えば複数実装案を並行比較可能	技術スタックやテンプレートが固定されていることが多く、新技術導入や変更が難しく、変更コストが高い。手作業コーディング後の仕様変更で、手作業でのマージが煩雑化
チーム間・ドキュメント整合性、オンボーディング	最新の仕様書が常に参照でき、仕様・タスク・実装の関係が見えるため、チームでの認識合わせがしやすい。	詳細な仕様書があるが、ドキュメントが更新されなかったり、散逸することが多い。実装と仕様、ドキュメントとの齟齬が発生
品質保証・非機能要件	仕様からテストや検証を設計段階で組み込みやすい。仕様-実装の整合性チェックが可能	非機能要件を仕様レベルで厳密に記述し、それを担保するプロセス（レビュー・テスト設計・性能試験など）が確立されている一方、曖昧な仕様とドキュメント齟齬から品質が低下

ご参考：パートナーソシューション
生成 AI を活用したモダナイゼーション支援

アバナードの生成AIに関する取組み

マイクロソフトの緊密なイノベーションパートナーとして、グローバル・日本のAIエキスパート・エンジニアを活用しお客様のDX化へ貢献します。

4500+
エンジニアリング、データサイエンス、AIソリューションなどを網羅するデータ＆AIのスペシャリスト。日本は600名弱

6000+
350社以上の顧客の自動化の成功に貢献

1/10
生成AI Copilotのグローバルプレビューパートナー

90+
マイクロソフト・アライアンス・パートナー・オブ・ザ・イヤーを17回受賞

100+
世界中のクライアントに生成AIプロジェクトを提供

神戸
神戸の「Microsoft AI Co-Innovation Lab」に参画し、AI/IoT でビジネス変革を支援

End-to-end services

エマージングテクノロジー › エクスペリエンスサービス › アドバイザリサービス › ソリューションデリバリー › マネージドサービス

生成AIを活用した開発のモダナイゼーション支援

Avanadeでは、アーキテクチャのモダナイゼーションと合わせて開発モダナイゼーションを支援します。GitHubなどのツール・プロセスをメインフレーム領域の開発で活用することで、生成AIを活用したコード生成、開発生産性の向上をはかることが可能になります。開発モダナイゼーションの導入・トレーニング、定着化のための伴走型サポートを提供致します。

GitHub Enterprise / Copilot 導入支援サービス

- GitHub Enterpriseの構成
- GitHub Copilotの有効化
- GitHub Copilot利用状況レポート
- GitHub運用/ガバナンス設計

GitHub Enterprise / Copilot トレーニング

- GitHub Enterpriseを理解し、最適な設計ができるようにする
- 管理のベストプラクティスを習得する
- セキュアなソフトウェア開発体制とガバナンスを整備するための知識を習得し、運用を自走できる状態を目指す

開発者成熟度評価診断

- チームのデリバリー性能と開発者体験を定量化し、業界ベンチマークとのギャップを把握
- DevEx変革の5つの軸で課題を分類し、改善インパクトと実現難易度で優先度を決定
- 継続的改善ループと自動化により、エンジニアリング投資のROIを向上させ、企業OKR/KPI への寄与を加速

DevOps・GitHub併走型サポートサービス / AaS

- 導入時や運用時に発生するさまざまな技術的課題をサポート
- 専門の技術者による伴走型サポートで企業ニーズに合わせた支援が可能

開発ライフサイクルのモダナイゼーション・ロードマップ

- SDLCトレーニングと診断
- DevSecOpsのBlueprint作成や標準化
- KPI/生産性指標の収集
- Azure AI ServicesやGitHub Copilotを活用したアプリケーション開発

お問い合わせ先：https://www.avanade.com/ja-jp/contact

Agenda

これからのマイグレーション & モダナイズの方向性

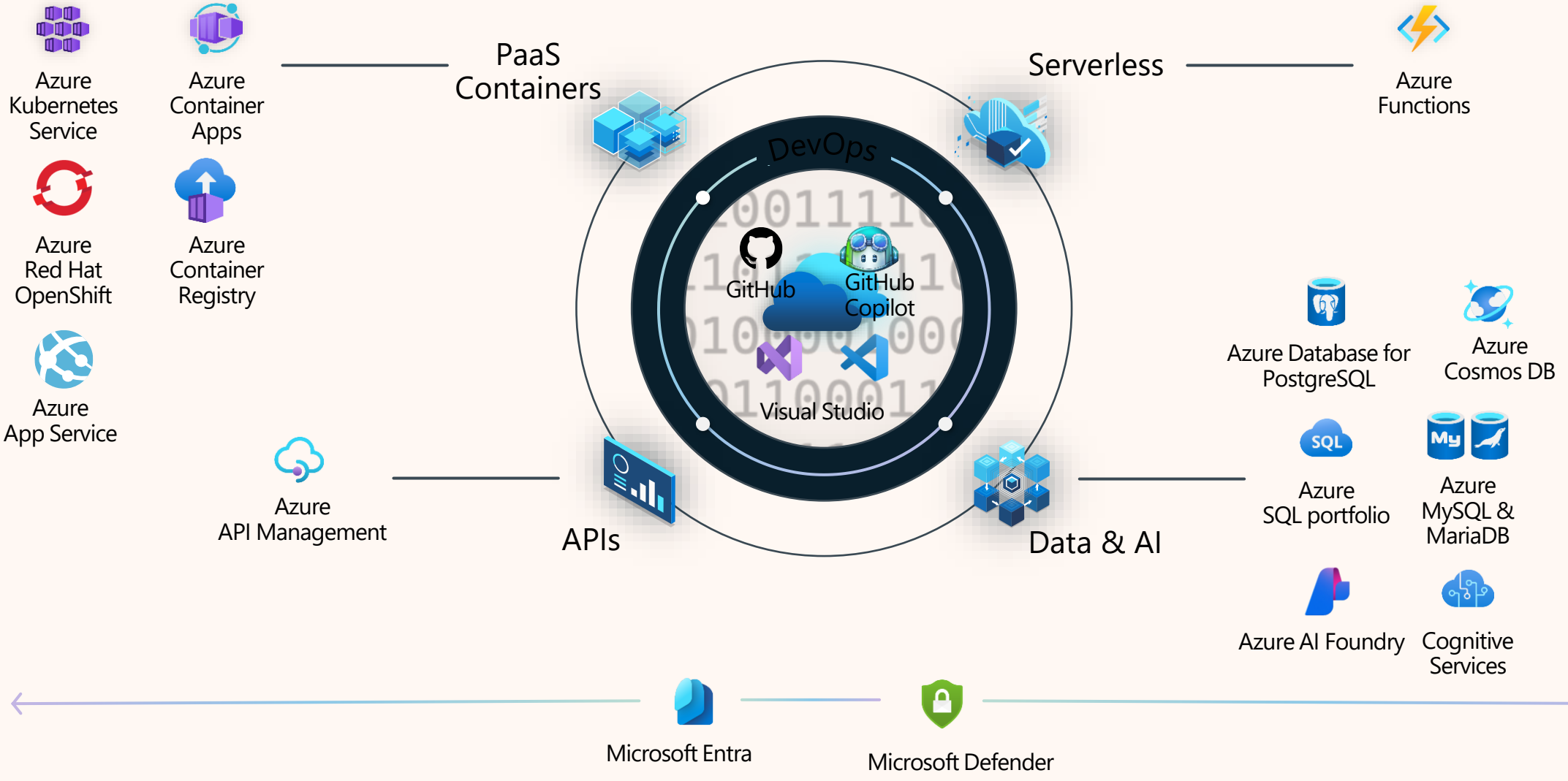
従来のマイグレーション＆モダナイゼーション戦略

SaaS ファースト戦略： 可能な限り開発せず既製サービスを活用
セキュリティ・ゼロトラスト対応：モダナイズ時に認証・監査の仕組みを最新化

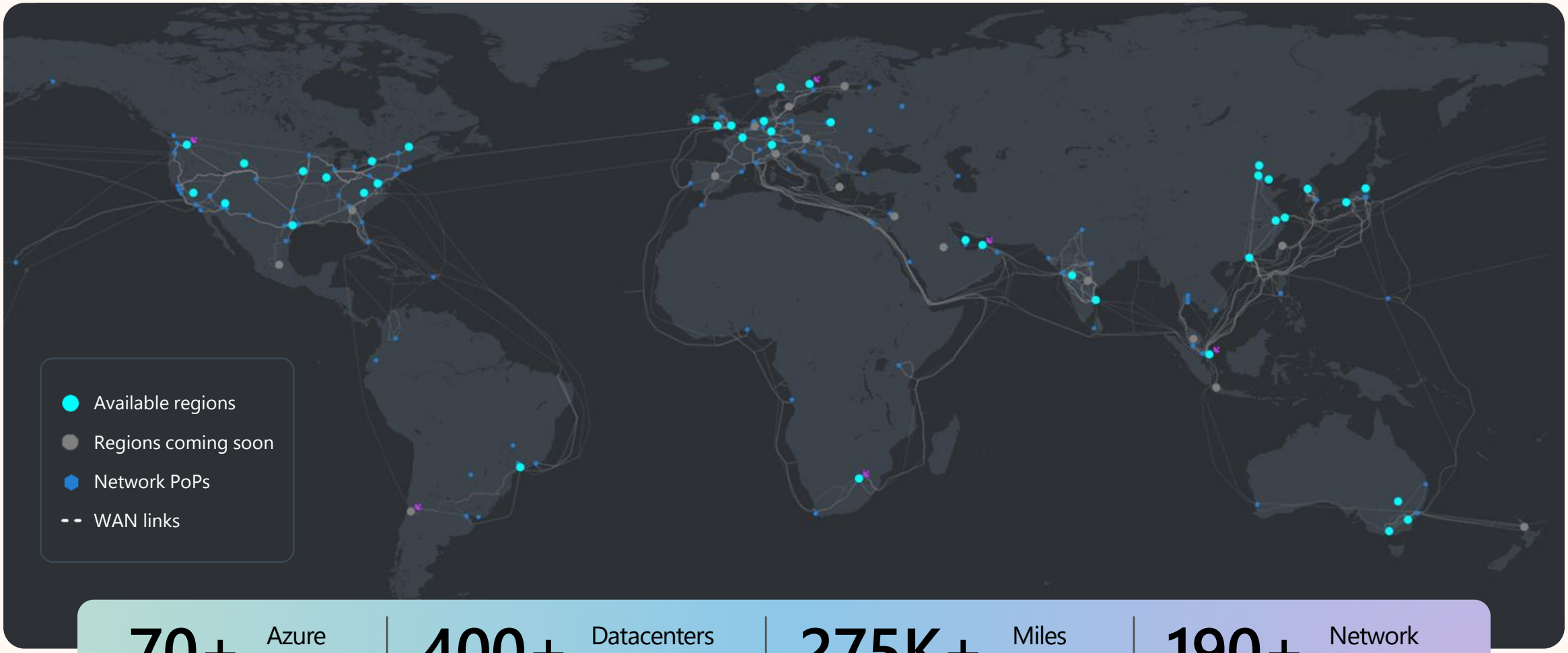
サステナビリティとイノベーションの両立

サステナビリティ対応：省電力・再生可能エネルギー利用
AI ネイティブ化：業務プロセスやシステムに AI 機能を組み込む

ビジネスアジリティを向上させる Azure 上でのクラウドネイティブ



Microsoft のグローバルクラウドインフラストラクチャ



70+ Azure regions | **400+** Datacenters worldwide | **275K+** Miles of fiber | **190+** Network PoPs

AI モデルを進化させるデータセンターの革新

エコノミーとエコロジーの両立に向けて

数十万基のNVIDIA GB200 GPUを相互接続

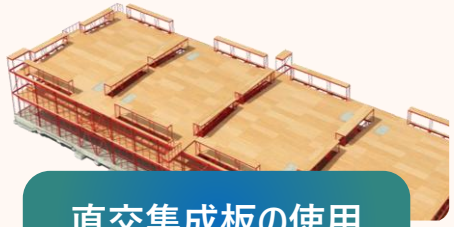
数百万基のコンピューティングコア、エクサバイト規模のストレージ

複数の DC を 400 Tb/s 帯域の AI WAN で接続

電力 100% 再生可能エネルギー

建設時の炭素排出量の削減

65% 削減



直交集成板の使用

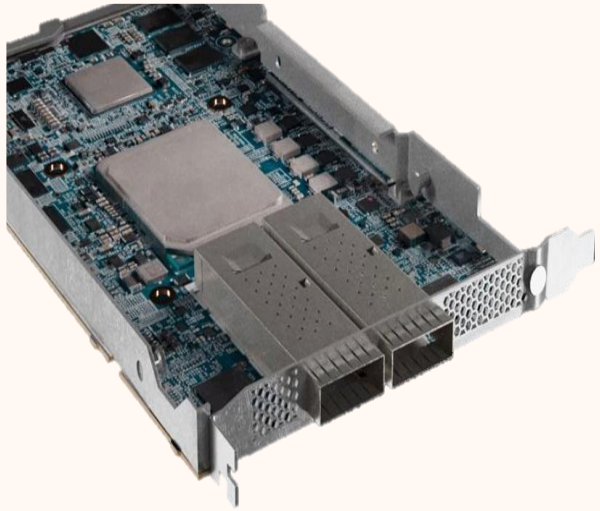
50% 削減

バイオコンクリート配合

AIモデルを進化させるシリコンへのアプローチ

Azure Boost

高効率と低電力を実現する
I/Oアクセラレータシリコン



各仮想マシン（VM）に対して1秒あたり40万の
ネットワーク接続、ローカルストレージで660万IOPS、
リモートストレージで80万IOPSを実現

Azure Cobalt 100 VMs

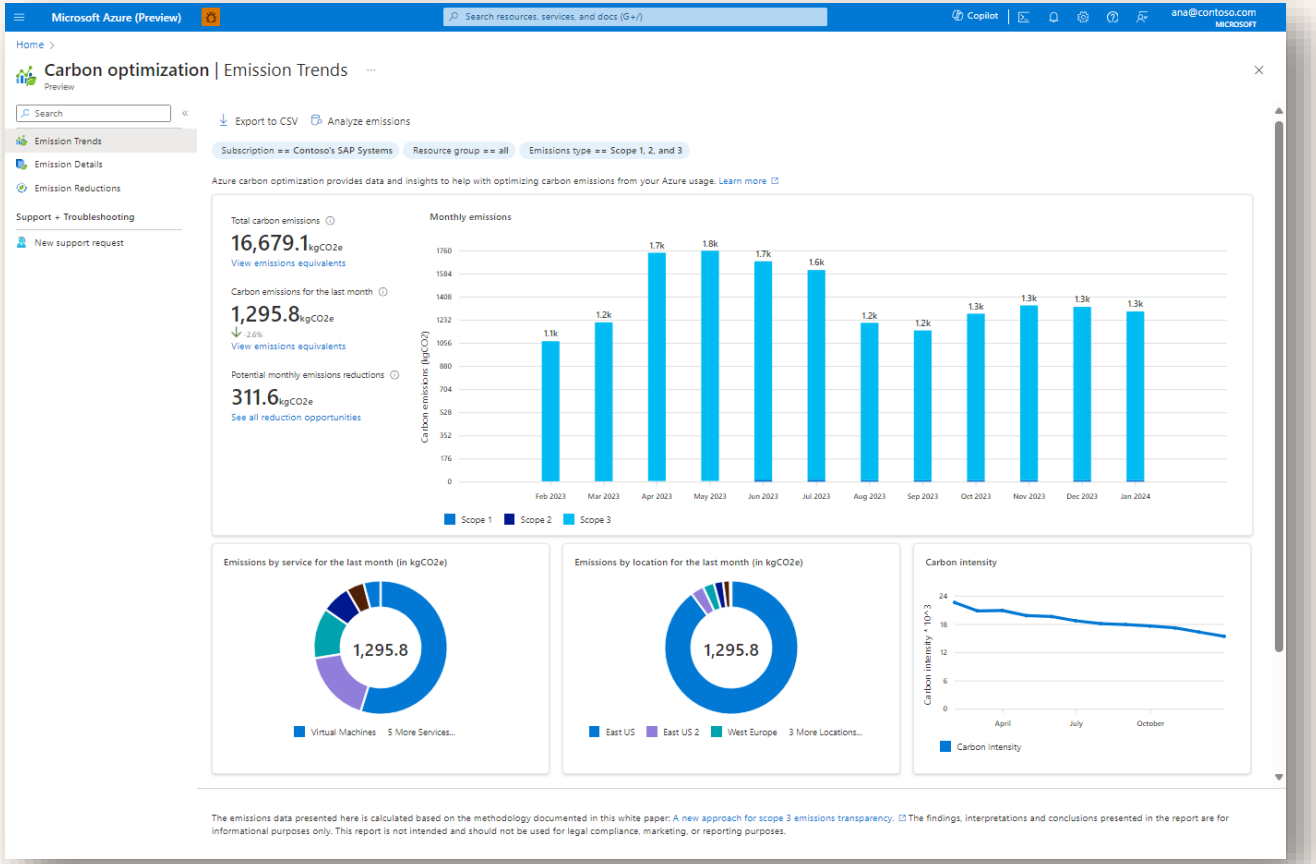
汎用コンピューティングに最適な
コストパフォーマンスを備えた ARM64 CPU

.NETアプリケーションで最大2倍のパフォーマンス向上を実現
AMDやIntelベースと比較し20％コストセーブ

排出量の影響を民主化

排出量の影響をみんなで共有し、改善に参加できるようにする

- Azure リソースの排出量を可視化
- すべてのスコープにわたって各リソースの環境への影響を理解
- クラウドの効率性と持続可能性を改善するための推奨事項を発見



Azure Carbon Optimization (GA)

Agenda

まとめ

COBOL 資産は、デジタルトランスフォーメーション、AIトランスフォーメーションの土台として重要な資産

クラウドや生成 AI を活用した COBOL 資産のモダナイゼーションは、レガシー資産を未来につなぐ技術戦略

Microsoft Azure、Microsoft 365 Copilot、GitHub Copilot、そしてパートナーソリューションが COBOL 資産のモダナイゼーションとイノベーション基盤の構築を強力に支援

- 本書に記載した情報は、本書各項目に関する発行日現在の Microsoft の見解を表明するものですMicrosoftは絶えず変化する市場に対応しなければならないため、ここに記載した情報に対していかなる責務を負うものではなく、提示された情報の信憑性については保証できません
- 本書は情報提供のみを目的としています Microsoft は、明示的または暗示的を問わず、本書にいかなる保証も与えるものではありません
- すべての当該著作権法を遵守することはお客様の責務ですMicrosoftの書面による明確な許可なく、本書の如何なる部分についても、転載や検索システムへの格納または挿入を行うことは、どのような形式または手段（電子的、機械的、複写、レコーディング、その他）、および目的であっても禁じられていますこれらは著作権保護された権利を制限するものではありません
- Microsoftは、本書の内容を保護する特許、特許出願書、商標、著作権、またはその他の知的財産権を保有する場合がありますMicrosoftから書面によるライセンス契約が明確に供給される場合を除いて、本書の提供はこれらの特許、商標、著作権、またはその他の知的財産へのライセンスを与えるものではありません