クラウドネイティブ計画のステップと第一歩

東京エレクトロン デバイス株式会社
CNBU CN技術本部
システムエンジニアリング部
岡田大輝

※本資料に掲載されている会社名・製品・サービス名・ロゴは各社の商標または登録商標です。
また、写真・ロゴマーク・その他の著作物に関する著作権はそれぞれの権利を有する各社に帰属します。

Copyright © Tokyo Electron Device LTD. All Rights Reserved.

クラウドネイティブとは

connect Beyond

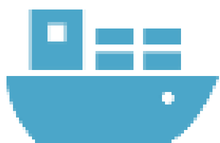
クラウドネイティブを推し進めている団体「Cloud Native Computing Foundation」（CNCF）の定義

「**CNCF Cloud Native Definition v1.0**」の抜粋
（https://github.com/cncf/toc/blob/master/DEFINITION.md）

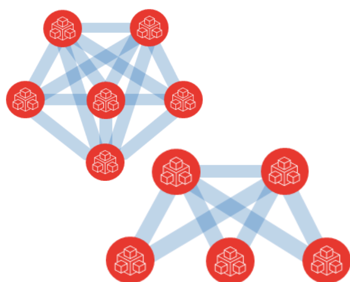
日本語版：

クラウドネイティブ技術は、パブリッククラウド・プライベートクラウド・ハイブリッドクラウドなどの近代的でダイナミックな環境において、スケーラブルなアプリケーションを構築および実行するための能力を組織にもたらします。このアプローチの代表例に、**コンテナ・サービスメッシュ・マイクロサービス・イミュータブルインフラストラクチャ**および**宣言型API**があります。
これらの手法により、**回復性・管理力**、および**可観測性**のある**疎結合システム**が実現します。
これらを堅牢な自動化と組み合わせることで、エンジニアはインパクトのある変更を最小限の労力で頻繁かつ予測どおりに行うことができます。

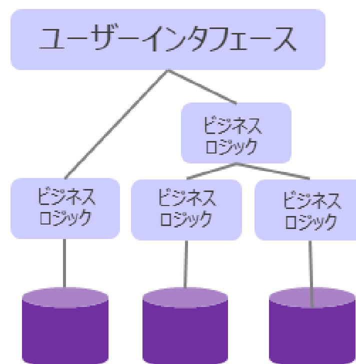
コンテナ



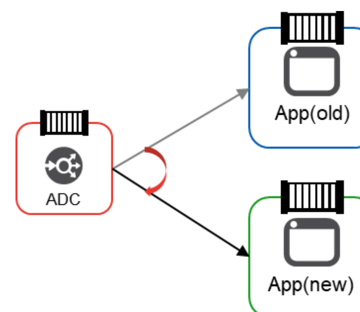
サービスメッシュ



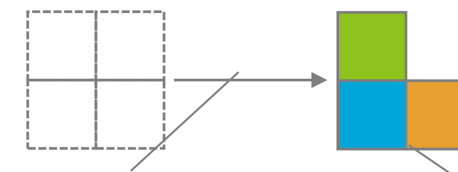
マイクロサービス



イミュータブル インフラストラクチャ



宣言型API



【手続き型】
1. ブロック枠左下を青
2. 青の上に緑を積む
3. 青の横に黄を配置

【宣言型】
Before
- 4つのからのブロック枠

After
- 左下が青
- 左上が緑
- 右下が黄

Copyright © Tokyo Electron Device LTD. All Rights Reserved.

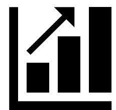
クラウドネイティブなアプリケーションの必要性

Connect Beyond

Webアプリケーションは、益々**データ活用を必要としており、**マーケティング・レコメンド・AI予測・IoTなど複数のアプリケーションで**ホストを跨った開発運用**が求められてきております。そして、**コスト・機能・実装のしやすさ・最適運用**など様々なことを考慮すると、**各ホストごとに機能分割化されたアプリケーションが連携**して動作する必要があります。

コスト重視

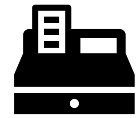
- アクセスログ管理システム
- 社内可視化システム



オンプレミス

コンピューティング

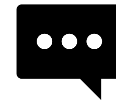
- POS管理システム
- ウェブ配信システム



A社 パブリッククラウド

AI

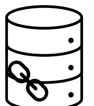
- レコメンドシステム
- 生成AI



B社 パブリッククラウド

データベース

- アンケートシステム
- 外部DB連携システム

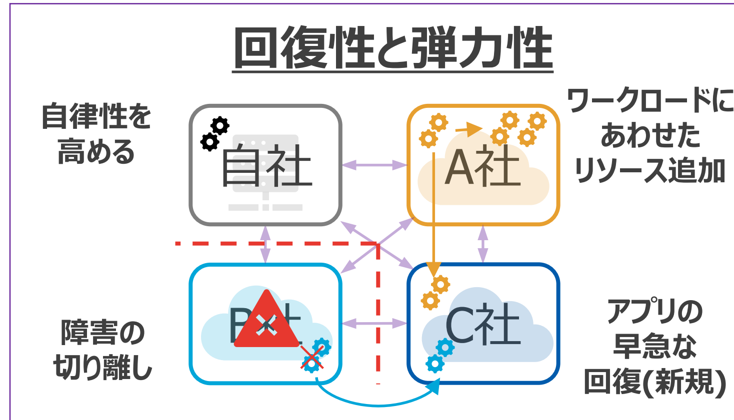


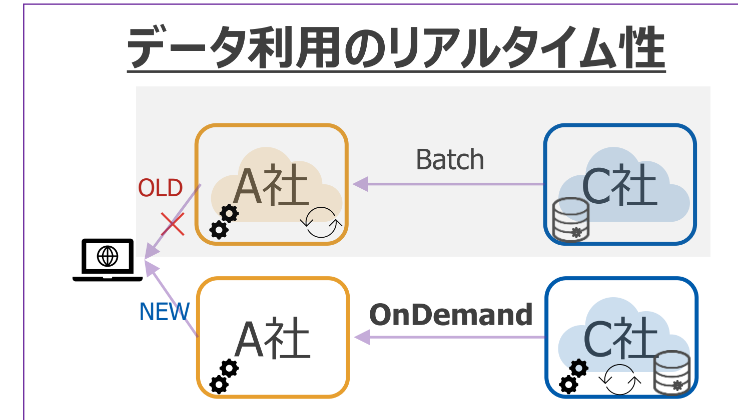
C社 パブリッククラウド

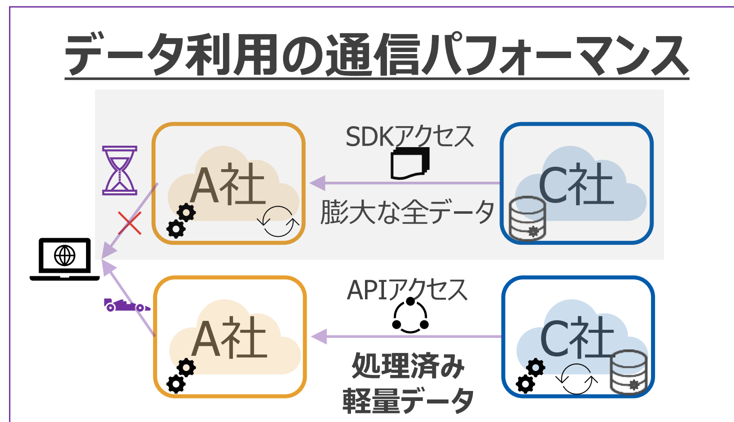
Copyright © Tokyo Electron Device LTD. All Rights Reserved.

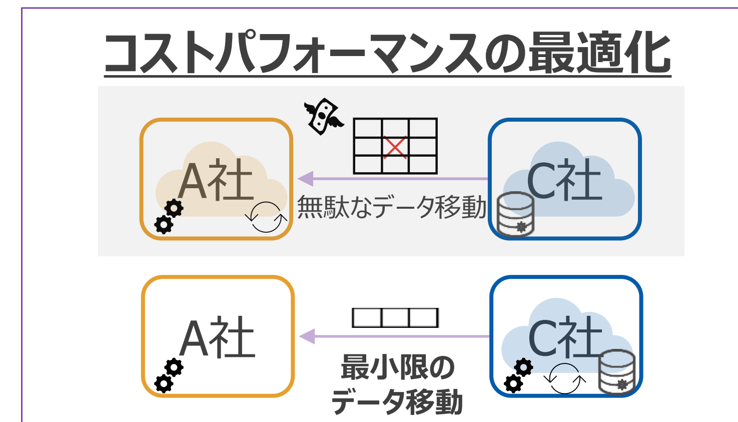
クラウドの効率的な利用とは？！

Connect Beyond









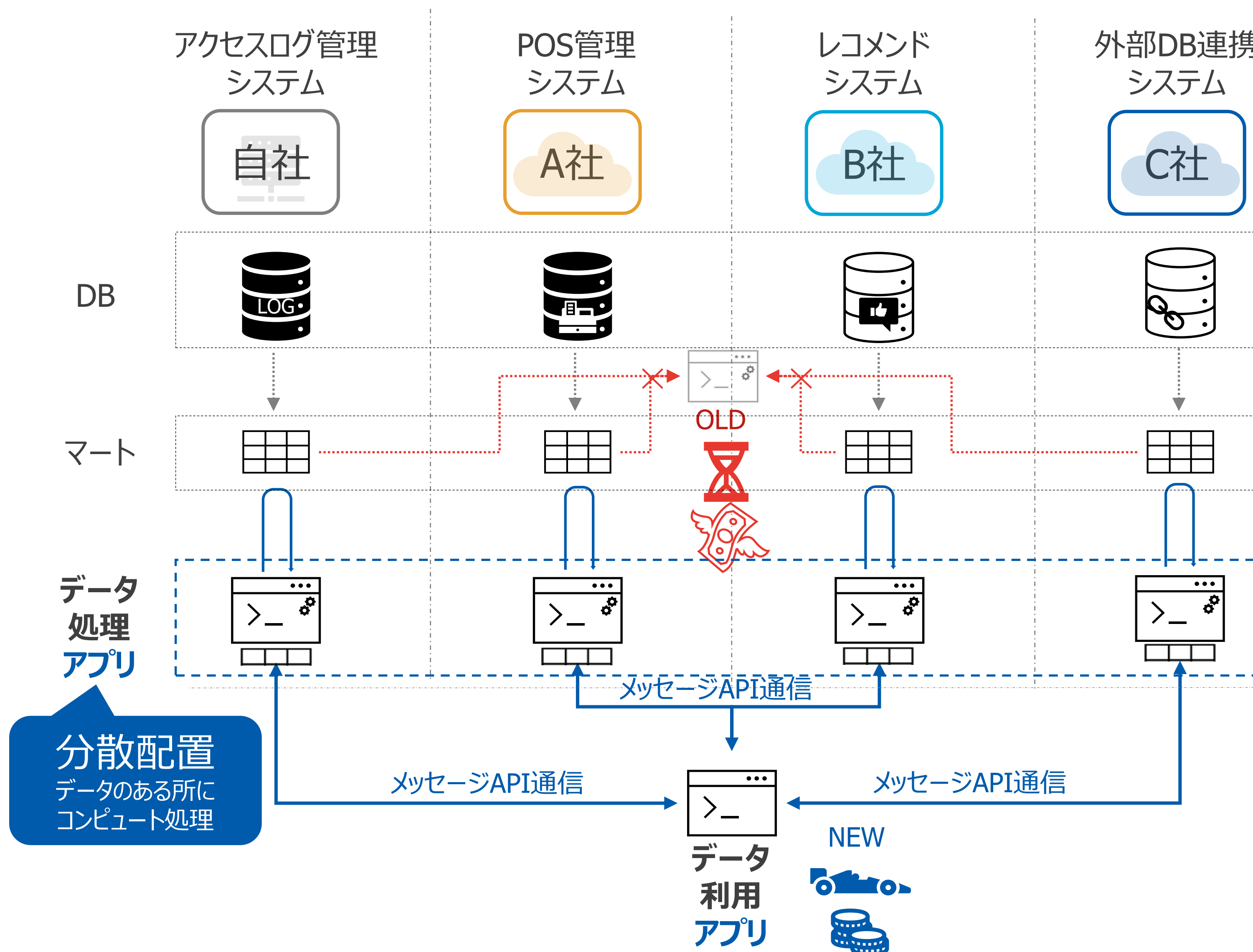
Copyright © Tokyo Electron Device LTD. All Rights Reserved.

東京エレクトロン デバイス

クラウドの効率的な利用の一例

Connect Beyond

データの地産地消



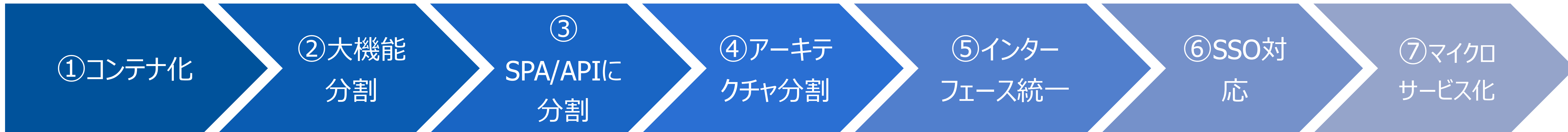
(データ利用のリアルタイム性)
(データ利用の通信パフォーマンス)
(コストパフォーマンスの最適化)

Copyright © Tokyo Electron Device LTD. All Rights Reserved.

東京エレクトロン デバイス

クラウドネイティブなアーキテクチャへ移行するための7ステップ

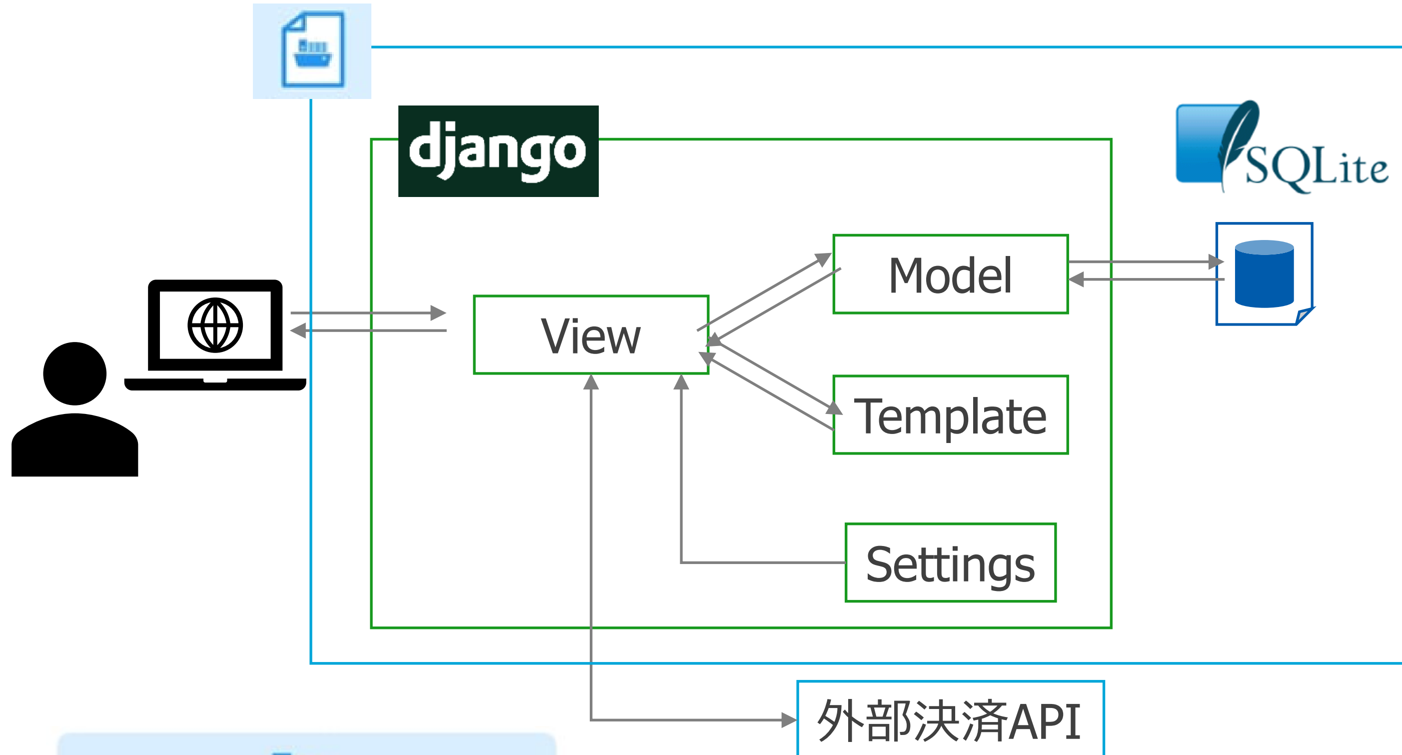
Connect Beyond

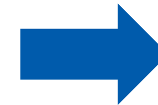


① **コンテナ化：**Webアプリをコンテナ化
② **大機能分割：**Webアプリを大きな機能ごとに分解
③ **SPA/API分割：**Webアプリをシングルページアプリケーション（SPA）/APIサーバーに分解
④ **アーキテクチャ分割：**WebアプリのAPIを、表示、バッチ、スケールアウトに分解
⑤ **インターフェース統一：**ビジネスロジックと必要なデータごとにインターフェース統一
⑥ **SSO対応：**Webアプリや各APIの認証・認可周りでシングルサインオン対応
⑦ **マイクロサービス化：**更に独立化できる機能ごとにコンテナに分けてマイクロサービス化

Copyright © Tokyo Electron Device LTD. All Rights Reserved.

STEP1. Webアプリをコンテナ化





コンテナ定義ファイル

Dockerfile 2.81 KiB

```
1  FROM alpine as builder
2  MAINTAINER  .........
3
4  # パッケージをインストール
5  RUN apk update && \
6      apk --no-cache add \
7      bash \
8      git \
9      openssh
10
11 # 非rootユーザーを使用
12 ENV USER dev
13 ENV HOME /home/$USER
14 RUN addgroup -S $USER && \
15     adduser -S -u 1000 -G $USER $USER && \
16     chown -R $USER:$USER $HOME
```

```
#------------------------------------------------------------
# イメージ登録
#------------------------------------------------------------
export DOCKER_BUILDKIT=1
docker build --progress=plain --no-cache --rm --pull --secret id=sshkeyfile,src=${SSHKEYFILE} --secret id=sshconfig,src=${SSHCONFIG} . -t ${TAG}
```

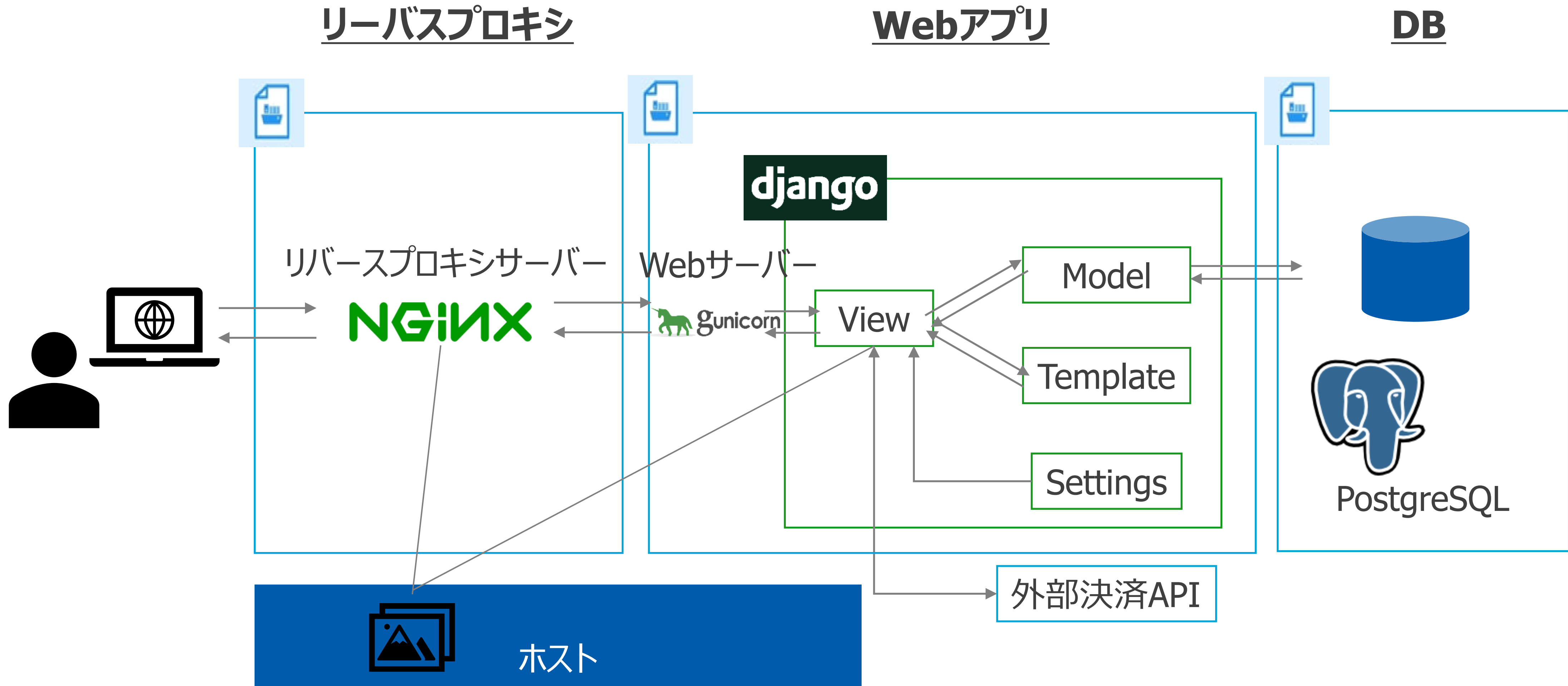
★ポイント
- マルチステージビルド
- 環境変数の活用
- docker-entrypoint.shの活用

➡ **デモンストレーションにて**

Copyright © Tokyo Electron Device LTD. All Rights Reserved.

東京エレクトロン デバイス

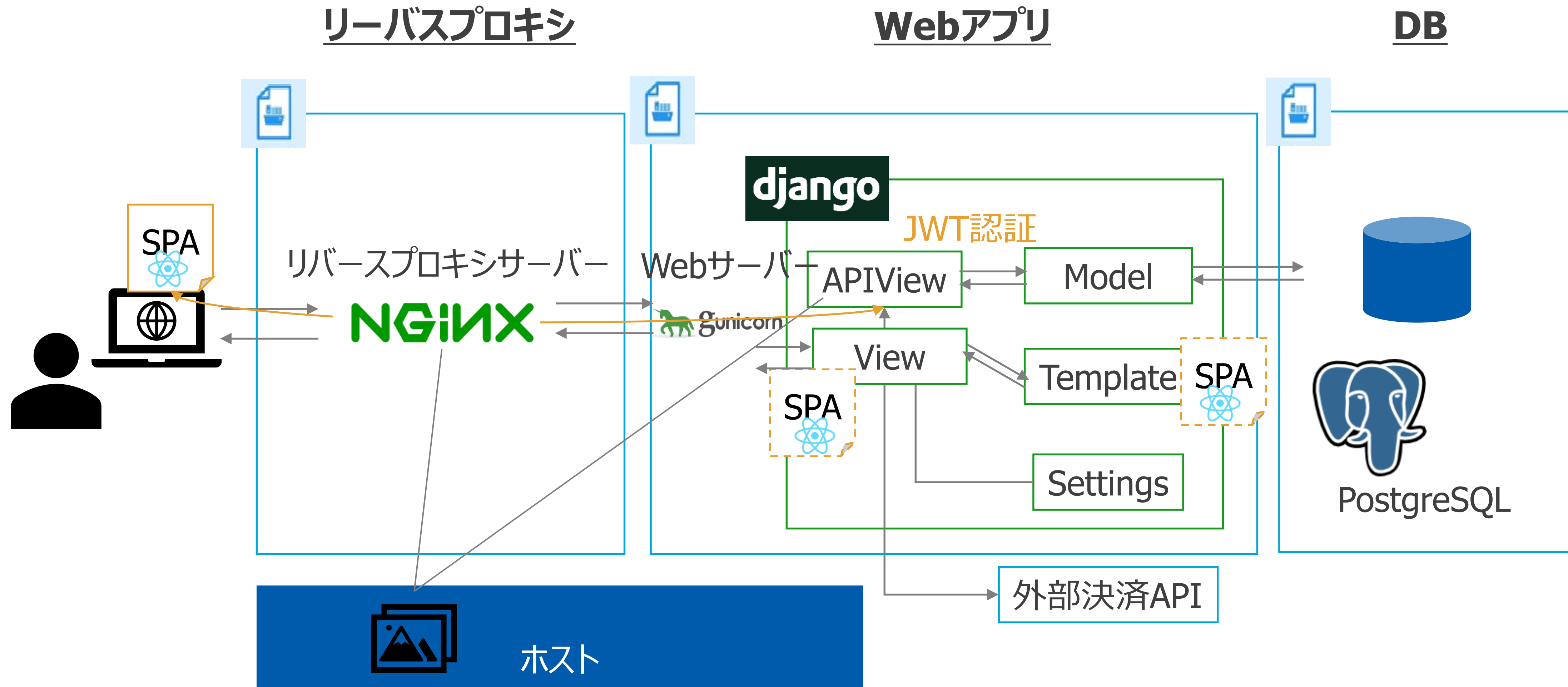
STEP2. Webアプリを大きな機能ごとに分解



Copyright © Tokyo Electron Device LTD. All Rights Reserved.

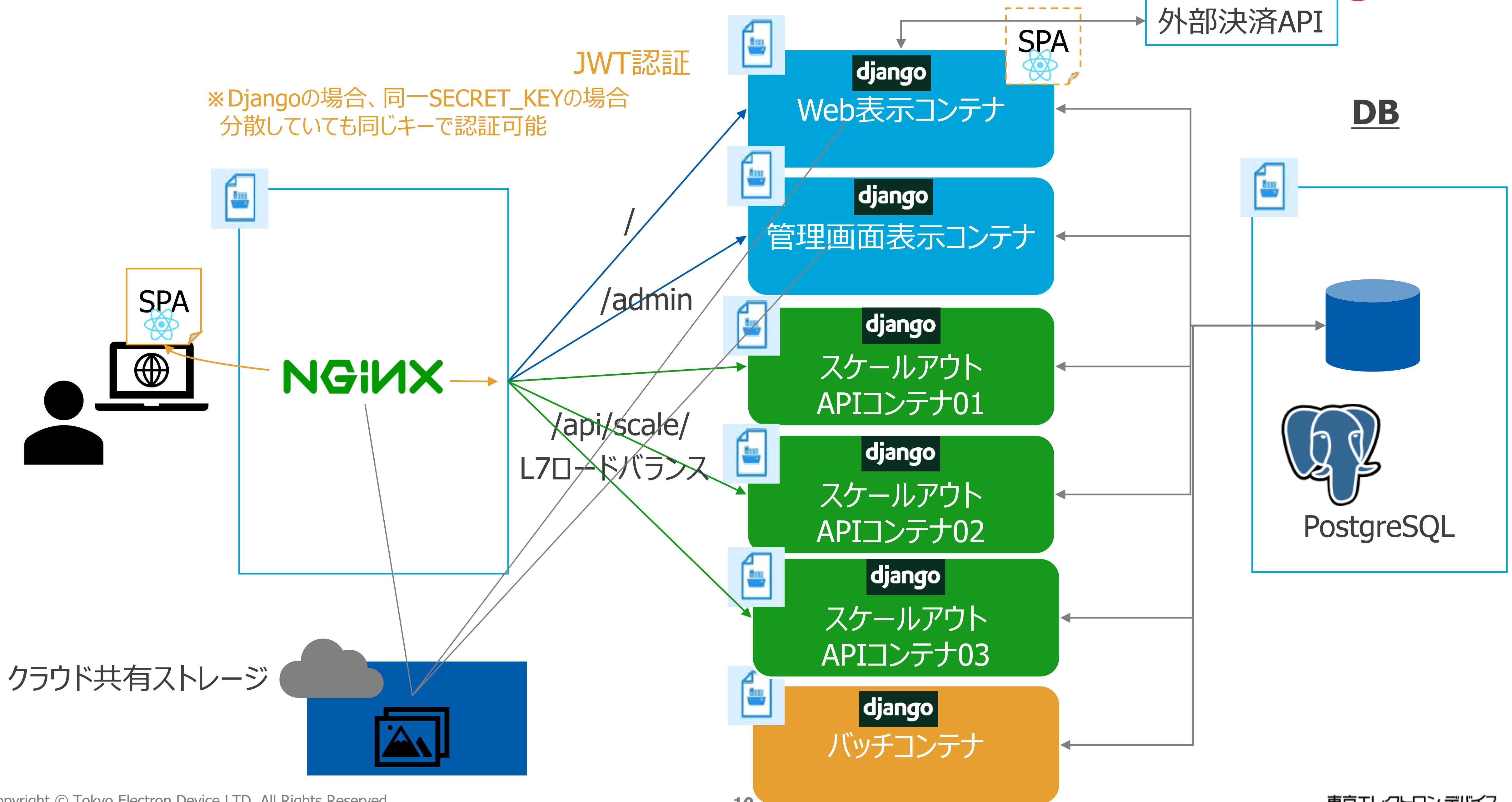
STEP3. WebアプリをSPA/APIサーバーに分解

onnect Beyond



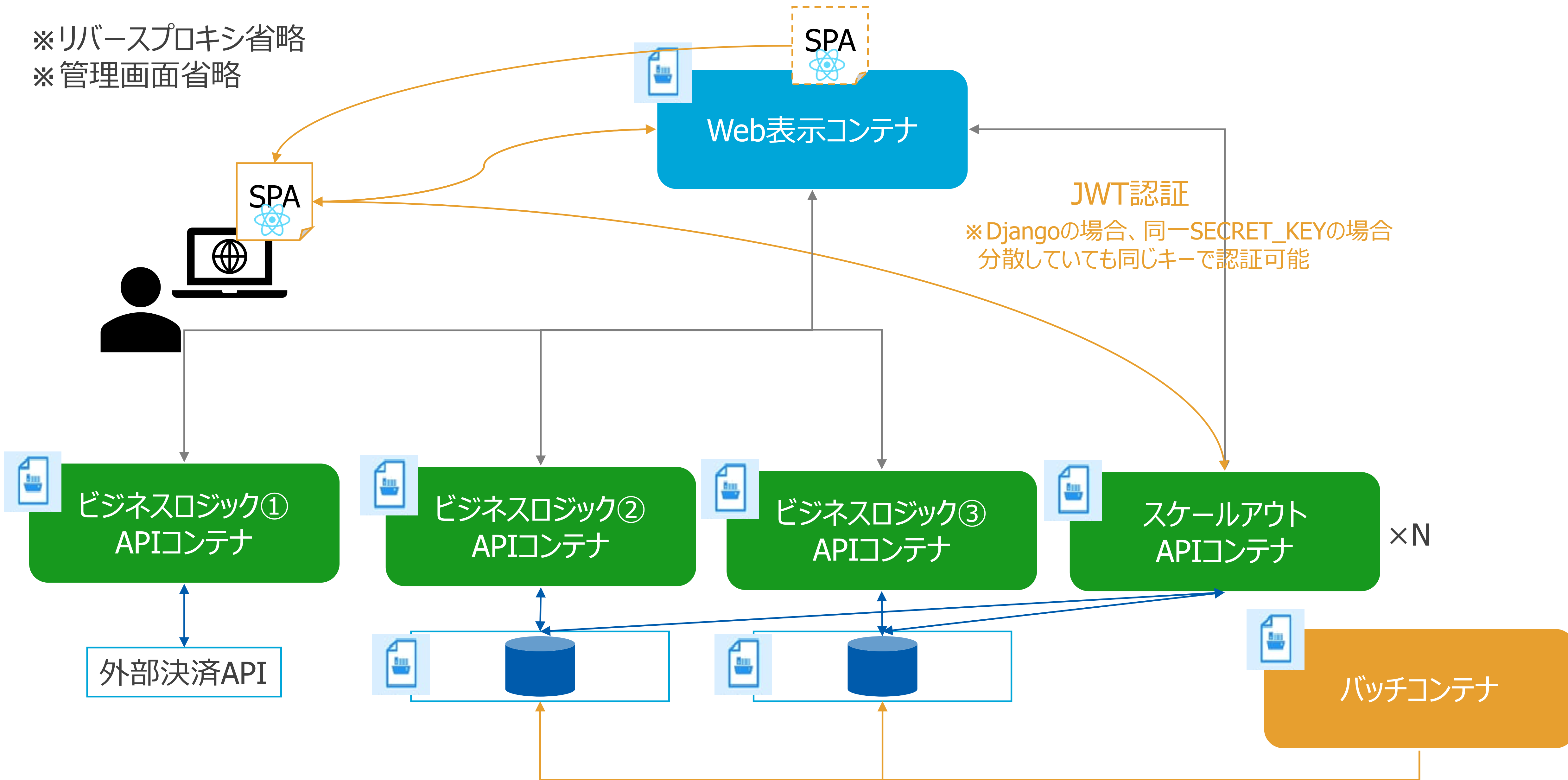
Copyright © Tokyo Electron Device LTD. All Rights Reserved.

STEP4. WebアプリのAPIを、表示、バッチ、スケールアウトに分解



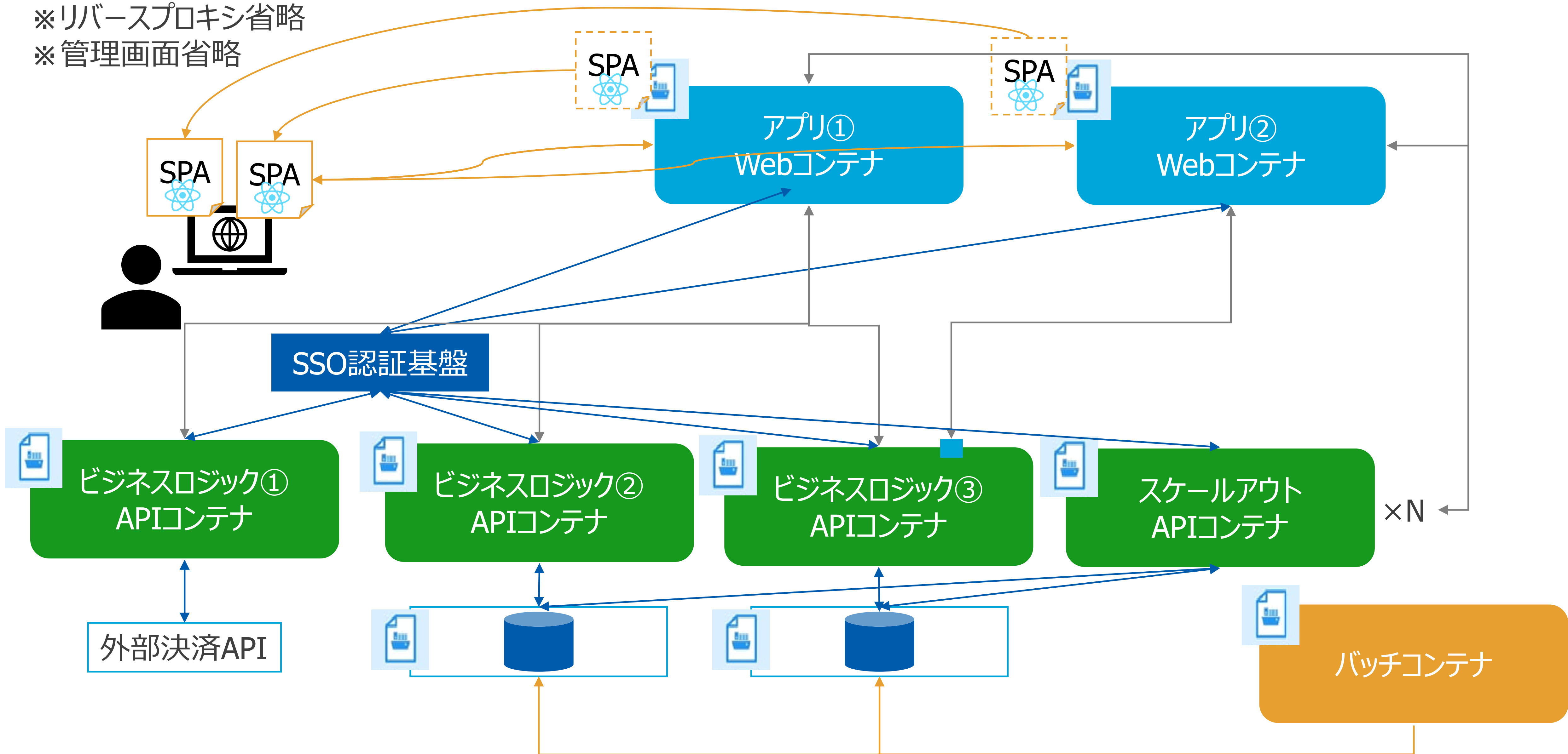
Copyright © Tokyo Electron Device LTD. All Rights Reserved.

STEP5. ビジネスロジックと必要なデータごとにインターフェース統一



Copyright © Tokyo Electron Device LTD. All Rights Reserved.

STEP6. WebアプリやAPIの認証・認可周りでシングルサインオン対応

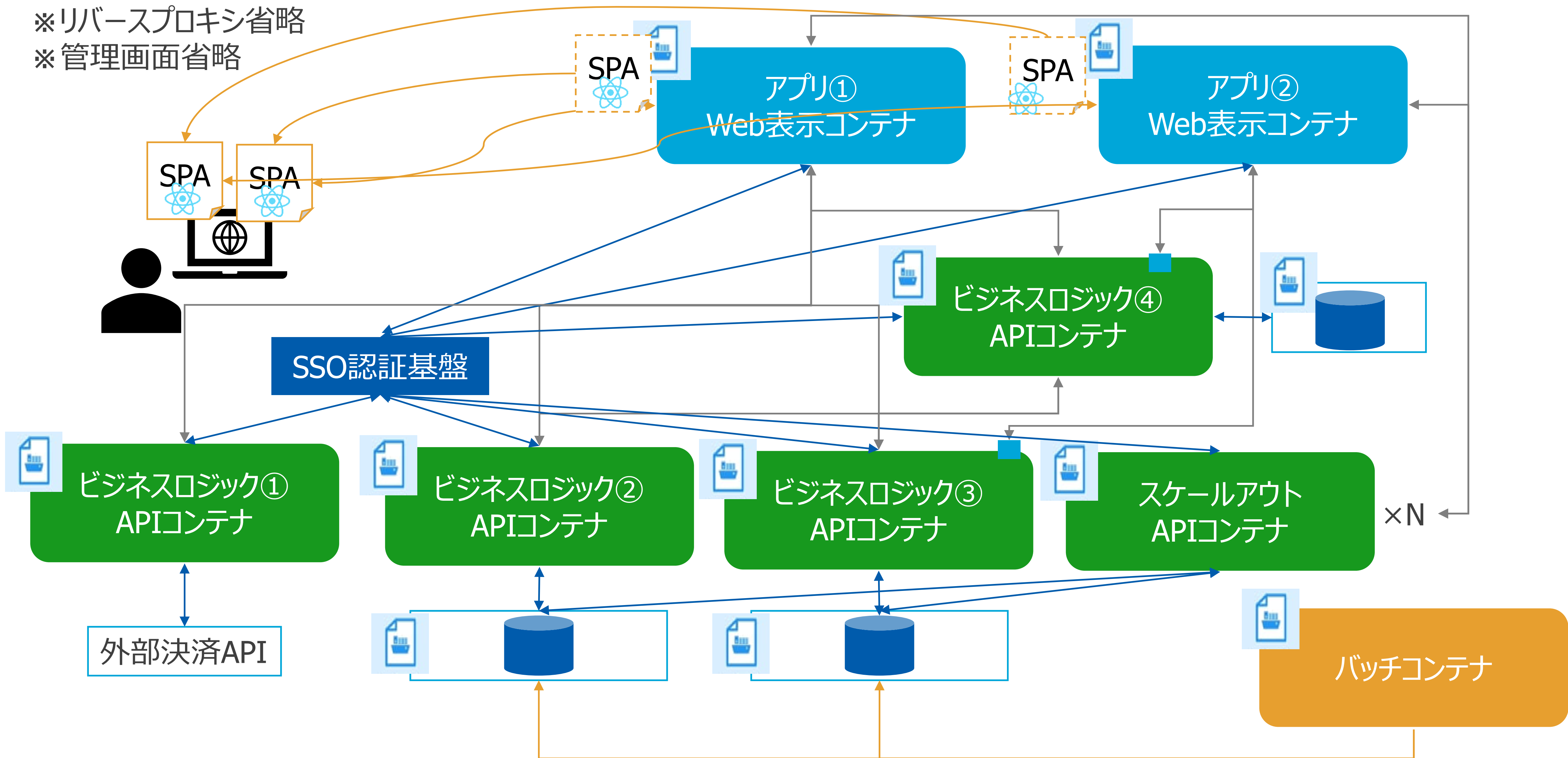


Copyright © Tokyo Electron Device LTD. All Rights Reserved.

東京エレクトロン デバイス

STEP7. 更に独立化できる機能ごとにコンテナに分けてマイクロサービス化

Connect Beyond



Copyright © Tokyo Electron Device LTD. All Rights Reserved.

東京エレクトロン デバイス

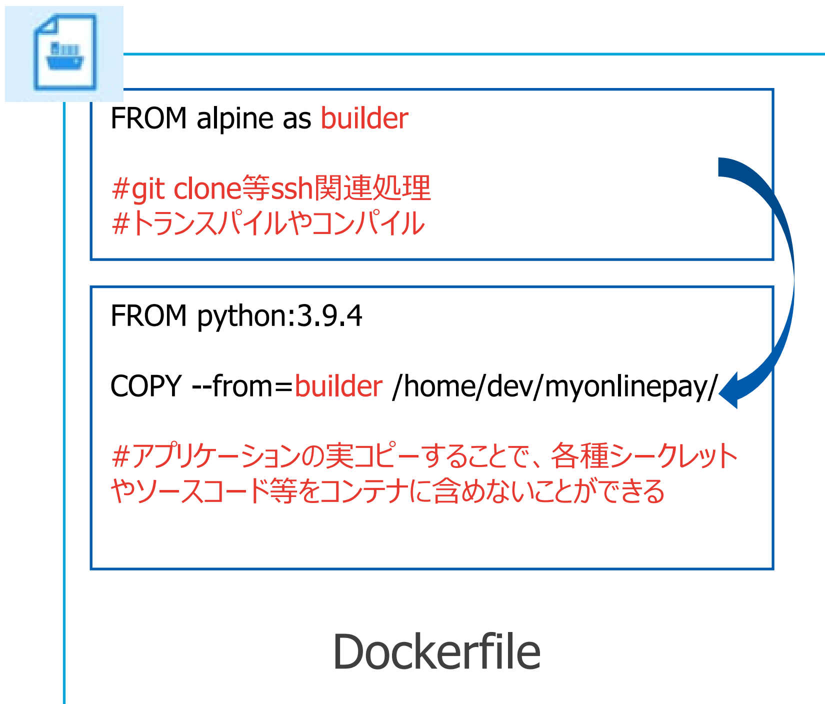
第一歩として/STEP1コンテナ化のデモンストレーション

onnect Beyond

例としては、外部のオンライン決済APIを含むpythonのdjangoアプリケーションをローカルで開発して、そのアプリケーションをコンテナ化し、F5 XC上にあるvk8sにデプロイする所をデモンストレーションします。

★コンテナ化のポイント

マルチステージビルド



環境変数の活用

local_settings.py

```
# 外部オンライン決済APIのパブリックキー
STRIPE_PUBLIC_KEY = ${DJANGO_STRIPE_PUBLIC_KEY}
# 外部オンライン決済APIのシークレットキー
STRIPE_SECRET_KEY = ${DJANGO_STRIPE_SECRET_KEY}
# 外部オンライン決済APIのWebhookのシークレットキー
STRIPE_WEBHOOK_SECRET = ${DJANGO_STRIPE_WEBHOOK_SECRET}
```

・本番環境/開発環境など
環境ごとに変わる変数を環境変数化

${DJANGO_STRIPE_PUBLIC_KEY}
${DJANGO_STRIPE_SECRET_KEY}
${DJANGO_STRIPE_WEBHOOK_SECRET}

docker-entrypoint.shの活用

docker-entrypoint.sh

```
envsubst '${DJANGO_STRIPE_PUBLIC_KEY}
        ${DJANGO_STRIPE_SECRET_KEY}
        ${DJANGO_STRIPE_WEBHOOK_SECRET}
        ' < myonlinepay/local_settings.py.template >
myonlinepay/local_settings.py
```

・コンテナ起動時に環境変数化された変数を設定ファイルなどに書き込み環境ごとに異なる設定で動作するようにすることができる

Copyright © Tokyo Electron Device LTD. All Rights Reserved.

まとめ

connect Beyond

- **クラウドネイティブアーキテクチャ移行への7つのステップ**
  - ① **コンテナ化：** Webアプリをコンテナ化　→　第一歩として本セッションで解説したこと
  - ② **大機能分割：** Webアプリを大きな機能ごとに分解
  - ③ **SPA/API分割：** Webアプリをシングルページアプリケーション（SPA）/APIサーバーに分解
  - ④ **アーキテクチャ分割：** WebアプリのAPIを、表示、バッチ、スケールアウトに分解
  - ⑤ **インターフェース統一：** ビジネスロジックと必要なデータごとにインターフェース統一
  - ⑥ **SSO対応：** Webアプリや各APIの認証・認可周りでシングルサインオン対応
  - ⑦ **マイクロサービス化：** 更に独立化できる機能ごとにコンテナに分けてマイクロサービス化
- **STEP1.　コンテナ化におけるDevOpsにおけるポイント**
  - コンテナにシークレットを残さない工夫　→　マルチステージビルド
  - 環境によって変わる設定は環境変数化　→　外部API連携部分
  - 各種コンテナの起動時処理　→　docker-entrypoint.shの活用

Copyright © Tokyo Electron Device LTD. All Rights Reserved.

東京エレクトロン デバイス

Thank you!

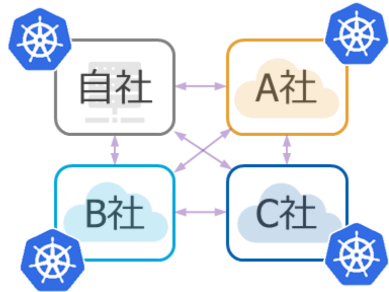
Appendix：
クラウドネイティブなWebアプリを容易にデプロイを実現するインフラ

Distributed
Cloud Services

Copyright © Tokyo Electron Device LTD. All Rights Reserved.

マネージドk8s

Connect Beyond



マルチサイト共通でアプリケーションをデプロイできる環境としてのk8sがデファクト

→ アプリケーションをコンテナ化することで**配置環境ごとのソフトウェアスタック準備が不要**

→ 同じk8sのため**デプロイジョブを共通化**可能

管理・デプロイの手軽さを実現

インフラ管理コスト減

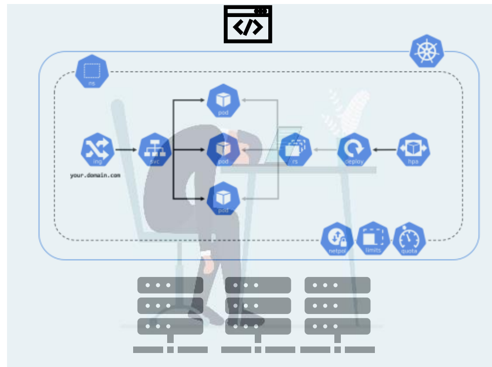
一般的にマネージドk8sと呼ばれる範囲

セルフマネージドk8s

自社でのk8sクラスタ管理

- ソフトウェアスタック管理
- （物理）ホスト管理

▶ 大変

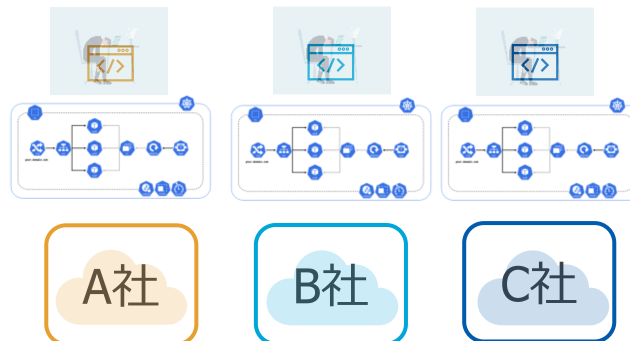


プロバイダマネージドk8s

各クラウドプロバイダでのk8sクラスタ管理

- 各クラウドごとの管理
- 各クラウドごとの構築

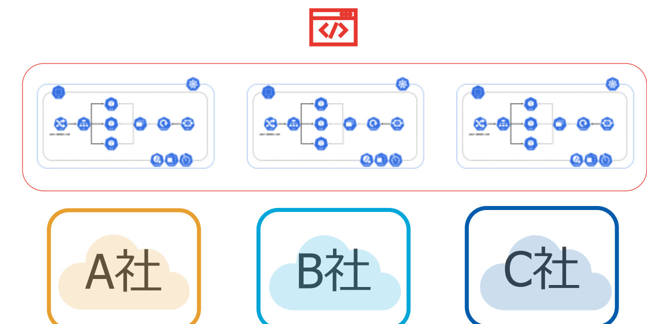
▶ 大変



プロバイダマネージド仮想k8s

1コンソールから各クラウドプロバイダのk8sクラスタを仮想的に統合管理

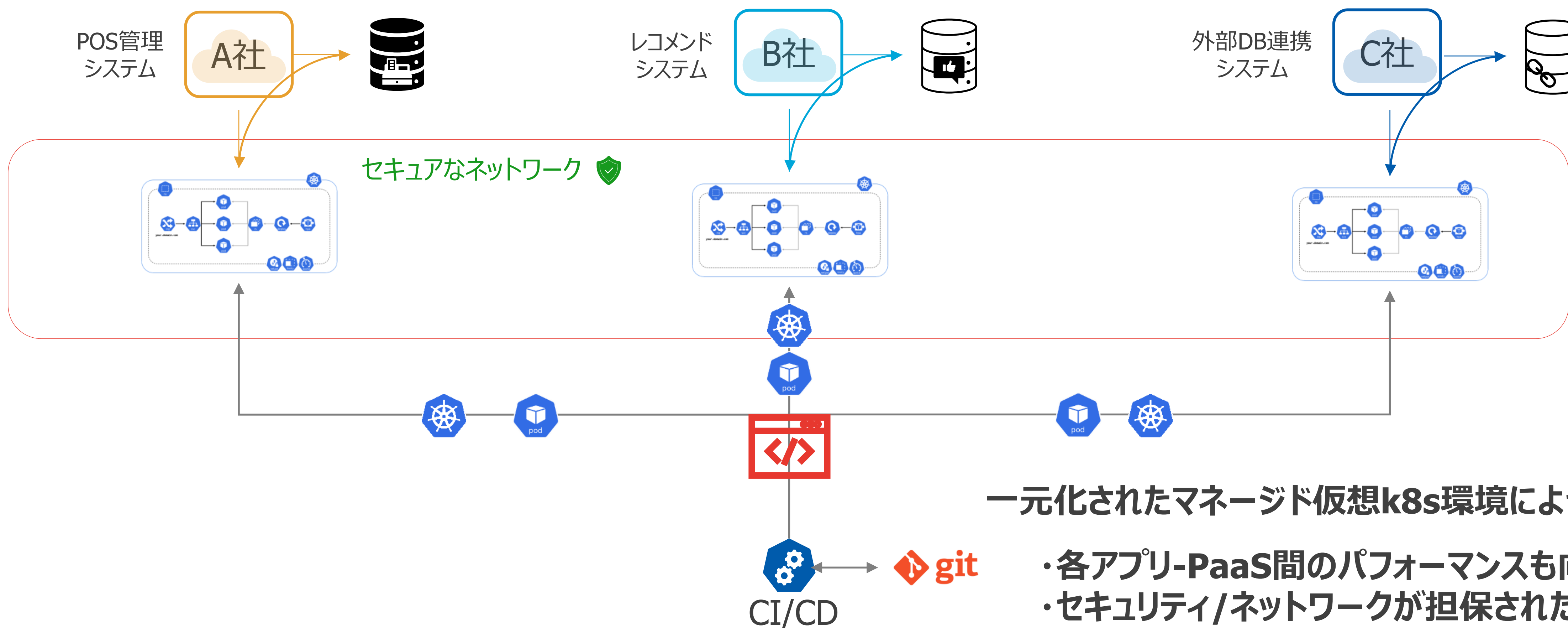
各クラウドプロバイダ環境にマネージドk8sを展開してグルーピングすることで仮想的に1つに見せる



Copyright © Tokyo Electron Device LTD. All Rights Reserved.

マネージドk8sとアプリケーションの分散管理

プロバイダマネージド仮想k8sを用いたk8sクラスター/アプリケーションの一元管理や一括デプロイ



一元化されたマネージド仮想k8s環境によって

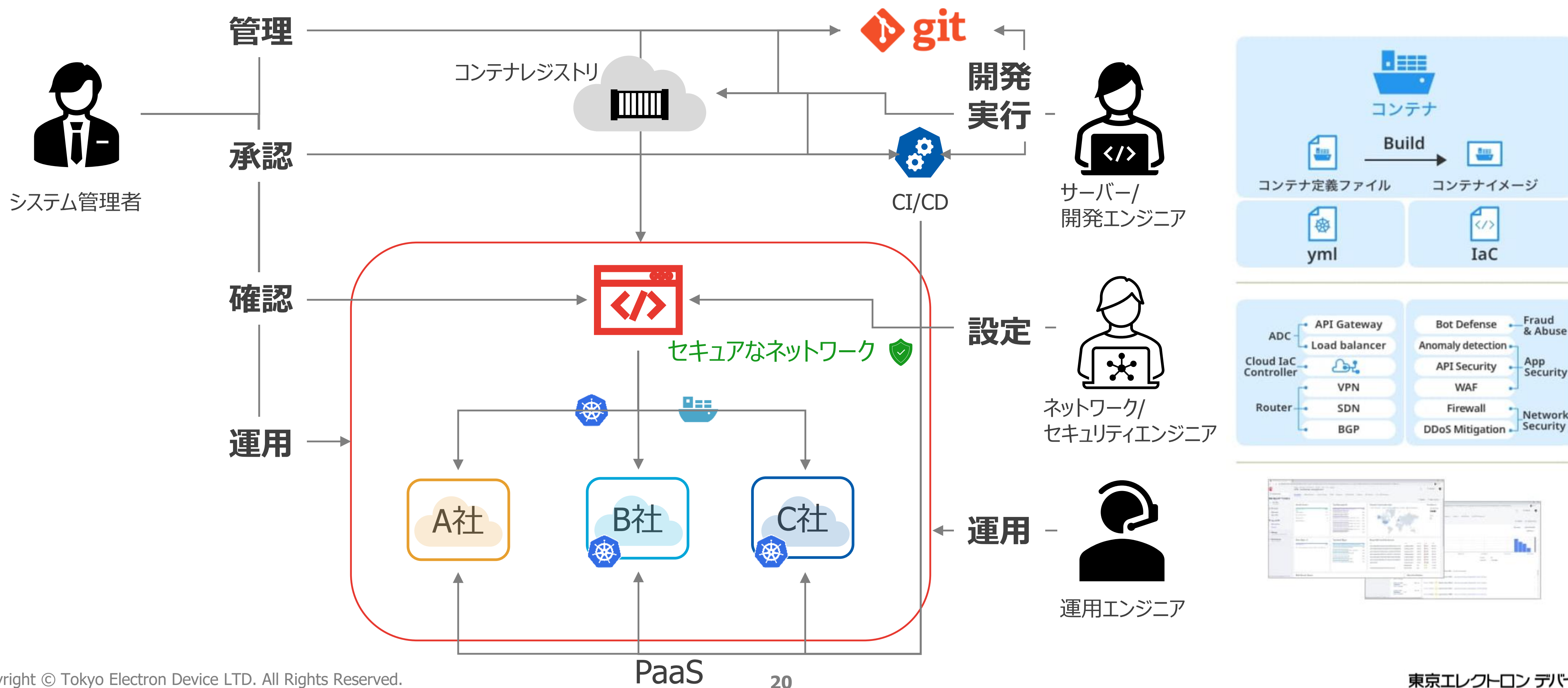
- **各アプリ-PaaS間のパフォーマンスも向上**
- **セキュリティ/ネットワークが担保された環境**
- **共通基盤として扱えるため開発工数も低減**

Copyright © Tokyo Electron Device LTD. All Rights Reserved.

安全かつシームレスなアプリケーション配信の実現

Connect Beyond

ネットワーク、サーバー、クラウド、セキュリティ、アプリケーションエンジニアが、**シームレスに連携**し、
今までの知識を生かしながら、モダンサービスの展開と運用（DevOps）が行えることが必要となってくる。

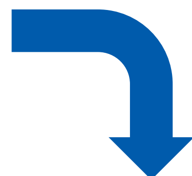


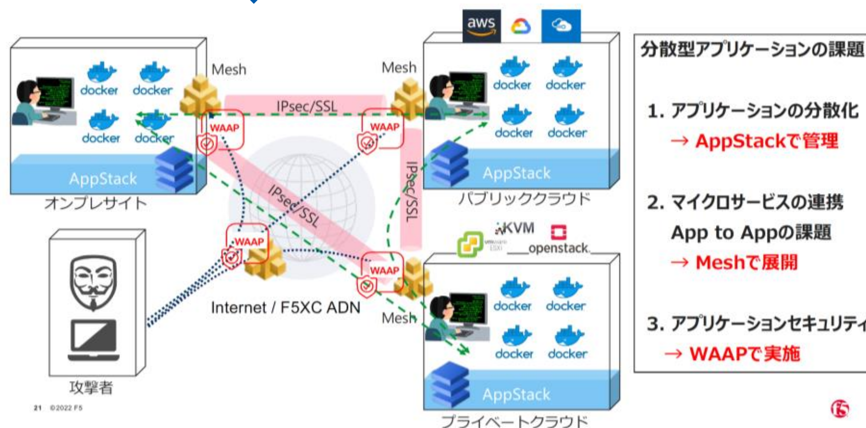
Copyright © Tokyo Electron Device LTD. All Rights Reserved.

東京エレクトロン デバイス

クラウドネイティブを容易に実現するF5 XC

F5のSaaSでマルチクラウド/ハイブリットクラウドのアプリケーション管理プラットフォームです。
F5 XCとパブリッククラウドやお客様拠点を接続し、F5 XC経由でそれぞれの環境にあるアプリケーションを配信する事ができます。
F5 XCはWAF等のセキュリティサービスも提供しておりセキュアにアプリケーションの配信を行う事ができます。





マルチサイト・クラウドネットワーク接続

マネージドk8s
アプリケーションサービス

SaaS型
セキュリティサービス

XC Consoleからお客様環境のXC Mesh、XC App Stackを一元管理

Copyright © Tokyo Electron Device LTD. All Rights Reserved.

東京エレクトロン デバイス