AWS SUMMIT JAPAN 2024

DEVELOPER ON LIVES - DAY2 [DOL-15]

チキン必見、開発者の心理的安全性を守る！ AWSアカウントの設計から見直すCI/CDパイプライン

白石　一乃

アマゾンウェブサービスジャパン合同会社

自己紹介

イベント情報など呟いています

X：@piko_san_0000

白石 一乃 (しらいし いちの)

ソリューションアーキテクト

- 西日本のお客様をメインで担当
- 国内 Sier 出身 Web アプリ開発、プロトタイピング、PM

チキン＝臆病者です

One-way / Two-way Door Decision

Amazonにおける、迅速な意思決定を行うための思考方法

意思決定を2種類に区別し、異なるアプローチを取り入れる

タイプ１：**一方通行のドア**
（元に戻すことがほぼ不可能な意思決定）

タイプ２：**両側通行のドア**
（簡単に元に戻せる決断）

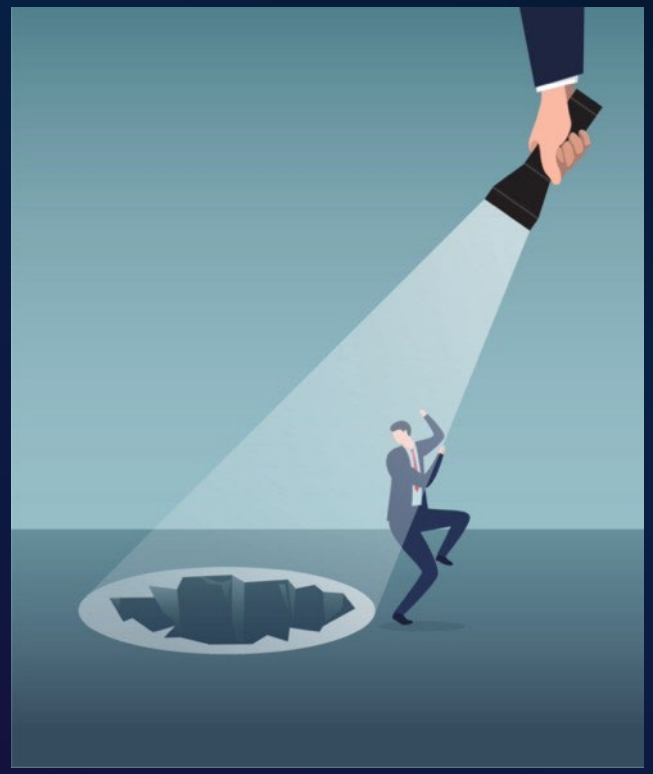
“組織は規模が大きくなるほど、タイプ2を含む多くの意思決定に、タイプ1用の意思決定プロセスを使う傾向があるようです。結果として判断が遅れ、リスク回避が増え、十分な実験が行われなくなり、発明が減ります。**この傾向に抗う術が必要です**”

- Jeff Bezos, 2015 shareholder letter

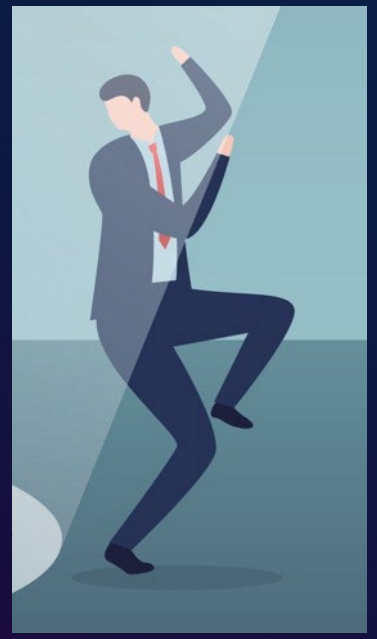
イノベーションを起こし続けるには
“メカニズム”が必要

セッションの目的

エンジニアが、より安心して 開発するための“メカニズム”



気にしていることは何か



みんな失敗が怖い

元に戻せなくなるのが怖い

ビジネスに影響を与えるのが怖い

失敗して上司・お客さんに怒られるのが怖い

自分の意図していないことが起こることを防ぎたい

- 意図せぬ構成 + それに気づかないこと
- 失敗をしたときに戻せるようにしたい

—> **スコープの話：開発環境、本番環境でも、与えるインパクトが違う**

ガードレールの仕組み　必要なのは「メカニズム」

スピーディーにビジネスを生み出す 仕掛け

（１）**「環境分離」**により影響範囲を小さくする

（２）望ましくない設定を**「検知・予防」**する

（３）インパクトの大きい環境に対しては**「自動化」**

ガードレールの仕組み　必要なのは「メカニズム」

スピーディーにビジネスを生み出す 仕掛け

（１）**「環境分離」**により影響範囲を小さくする

（２）望ましくない設定を**「検知・予防」**する

（３）インパクトの大きい環境に対しては**「自動化」**

本セッションで話す・見せること

- AWS マルチアカウント構成
- CI/CD パイプラインによる　本番環境メンテナンスの自動化（Demo）

（話さないこと）

- CI/CD の仕組み詳細、IaC、CDK、パイプライン設計、各AWSサービス詳細

（１）環境分離により影響範囲を小さくする

（１）環境分離により影響範囲を小さくする

AWSアカウント

- 様々なサービスを利用するための**区画**。
- **複数のユーザで利用**することができる。
- 単一の人・組織が**AWSアカウントを複数所有して**
  環境を分離することができる。

AWSにおける環境分離　マルチアカウント構成

AWSアカウントの分離＝ セキュリティ／リソース境界

AWS アカウントから 別のAWSアカウントの中身をみること、アクセスすることは、
明示的に設定しない限りできない

全ての環境をひとまとめにすると・・・

影響範囲が大きい

＝ どの環境にも**同じレベルで気をつかう必要がある**

環境を分離すると・・・

開発環境

テスト環境

本番環境

影響範囲を小さく留めることができる

開発環境

テスト環境

本番環境

AWSアカウントの分離によって実現すること

セキュリティ境界

環境の完全な分離
１つのAWSアカウントから別の
AWSアカウントの中身を見ること、
アクセスすることは、
明示的に設定しない限りできない

リソースの分離

サービスクォータ（上限）の分離
AWSアカウントごとに利用可能
なサービスの上限（クォータ）が
ある

課金の分離

請求書の分離
コスト按分の明確化
AWSの請求書（Billing）は、
AWSアカウント単位で提供される

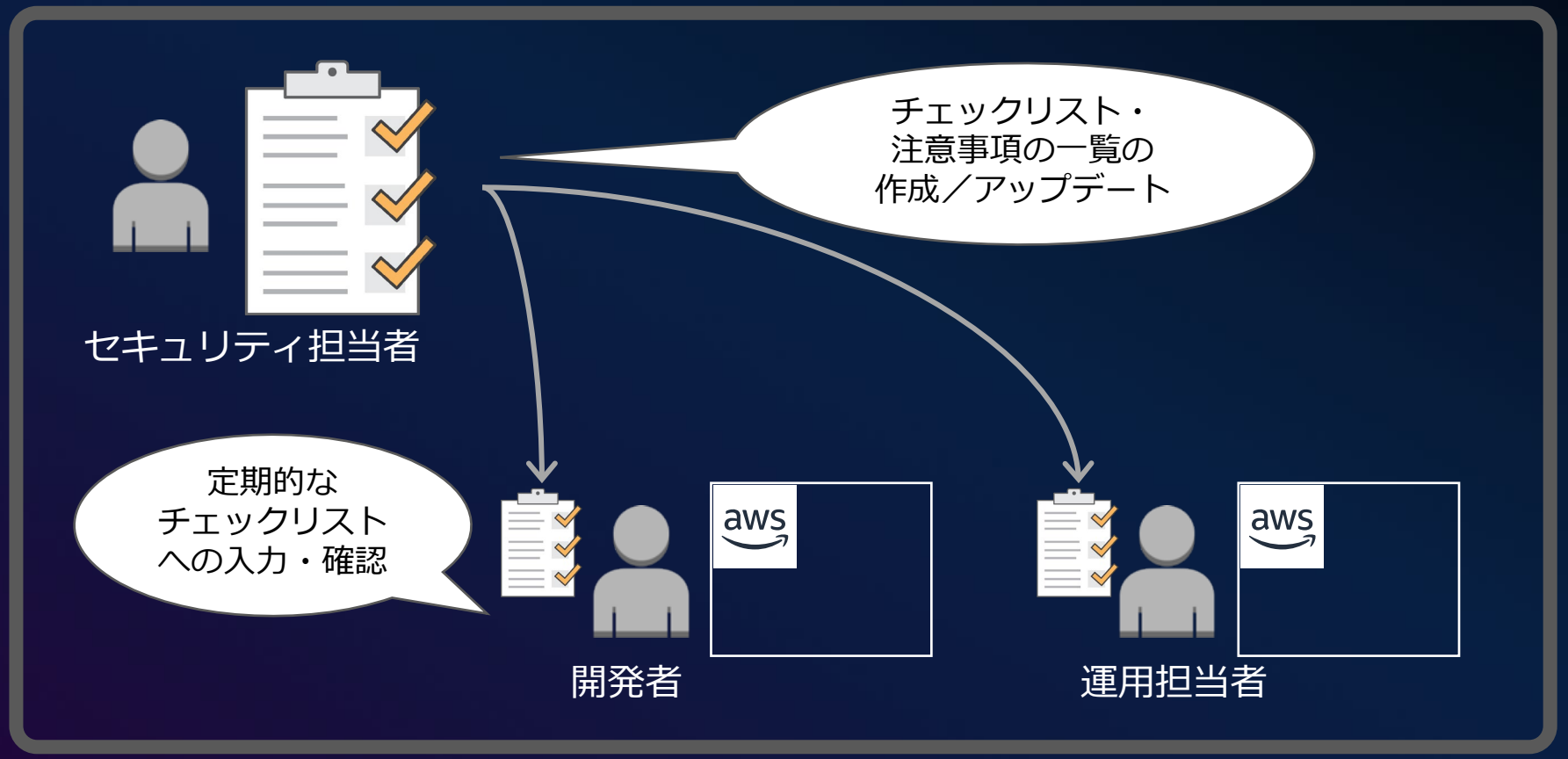
マルチアカウント構成：ベストプラクティスの変遷

現在のベストプラクティスでは、AWSアカウント単位で 環境を区切ることを推奨

（2）望ましくない設定を「検知・予防」する

（2）望ましくない設定を「検知・予防」する

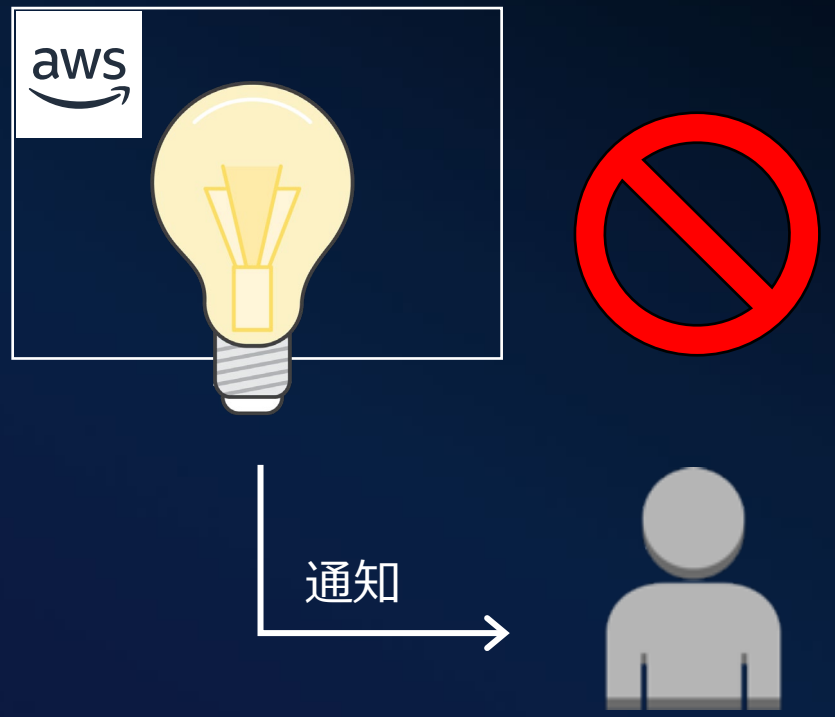
チェックリストの運用はつらい・・



（2）望ましくない設定を「検知・予防」する

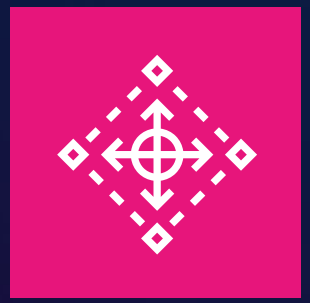
イノベーションを阻害しない仕組み

・望ましくない設定を「検知」
・リスクのある操作を「予防」



（2）望ましくない設定を「検知・予防」する

イノベーションを阻害しない仕組み



AWS Control Tower

（例：AWS Control Tower 上の「強く推奨」コントロール）

ルートユーザーの MFA が有効になっているかどうかを検出する

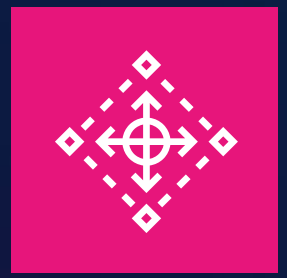
ルートユーザーのアクセスキーの作成を許可しない

ルートユーザーとしてのアクションを許可しない

Amazon S3 バケットへのパブリック読み取りアクセスが許可されているかどうかを検出する

無制限の着信 TCP トラフィックが許可されているかどうかを検出する

（補足）マルチアカウントの運用を楽にするための仕組みを利用する



AWS Control Tower

マルチアカウントのための環境を数クリックでセットアップ

① シングルサインオン

複数のAWSアカウントへの
ログインの切り替え

② ログ集約

AWSの操作ログの
自動収集

③ ガードレール

リスクのある操作の
予防・発見

④ アカウント作成

新規AWSアカウントの
自動セットアップ

詳細は Black Beltで検索！

（３）インパクトの大きい環境に対しては「自動化」

（3）インパクトの大きい環境に対しては「自動化」

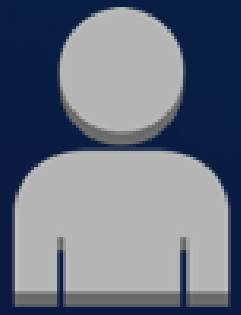
本番環境

人間の操作を極力排除
＝ **自動化**

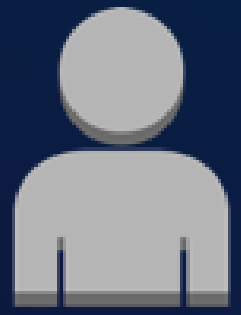
（ゼロタッチプロダクション）

インパクトの大きい環境に対しては「自動化」

Demo マルチアカウント構成でのCI/CDパイプライン



アプリケーション
開発者



アプリケーション
開発者

本番環境へのデプロイ
デプロイ手順書作って、レビューして
複数人で作業チェック・・・

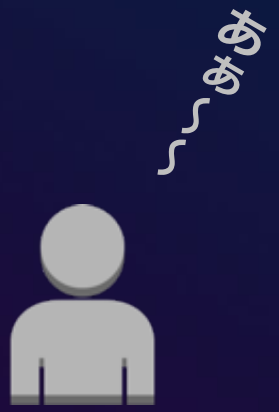
いつかミスしそう。。。

大変そうだな・・・
本番環境へのリリースは
半年に一回とかかな・・・

管理者

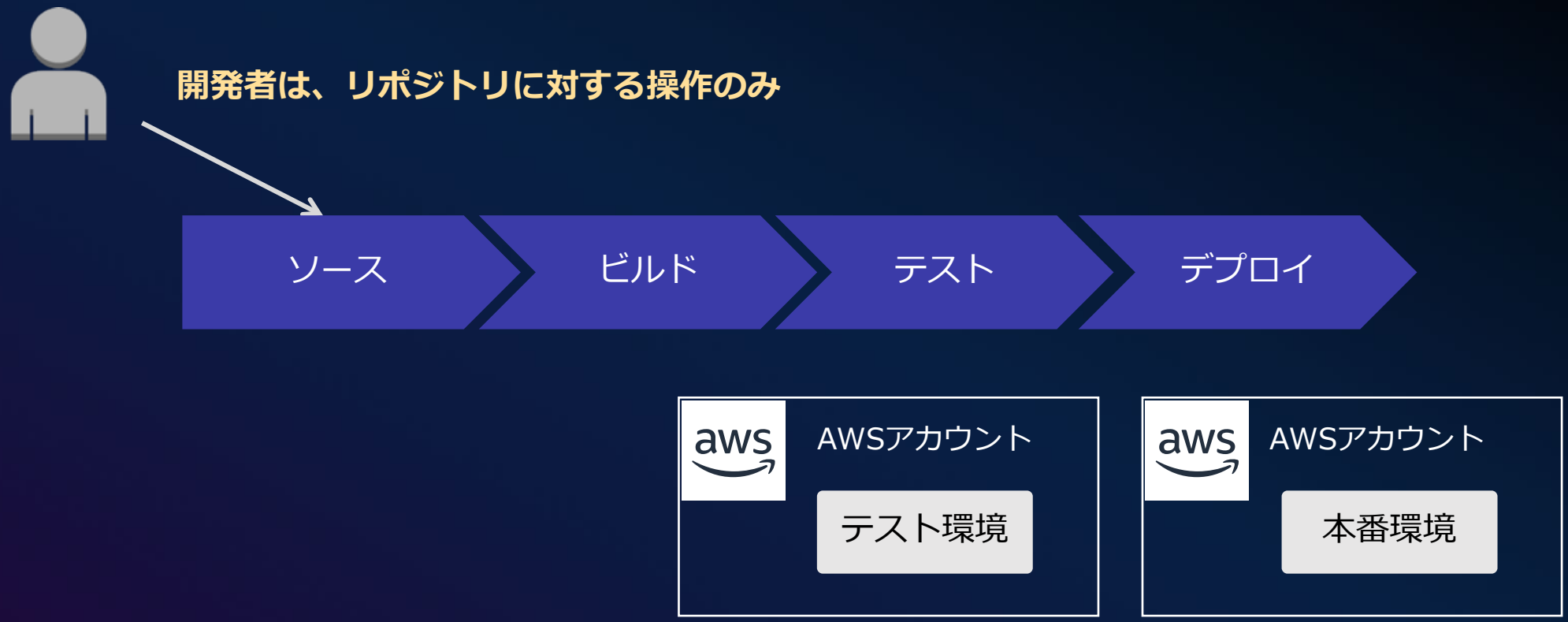
承認者も増やすか・・・

月日は流れ・・・
リリース作業を知る人間が減っていき・・・



リリース作業　＝　一方通行のドアの決断に・・・

マルチアカウント環境＋ CI / CDパイプライン



各環境に対して自動デプロイ＋本番環境は承認フローを挟む

プロジェクト初期：AWS アカウントの作成

プラットフォーム担当

プロジェクト開始時

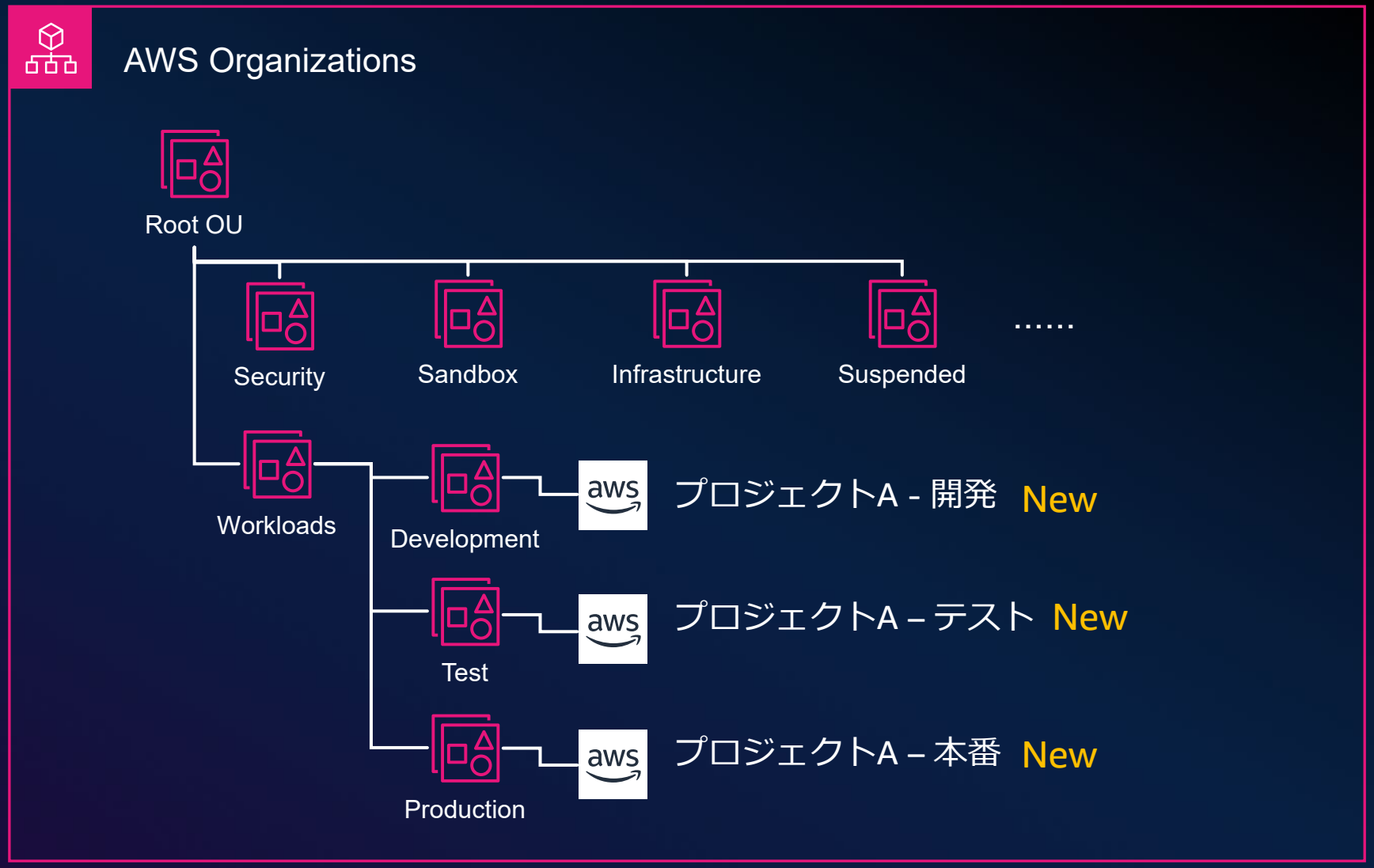
必要な環境の数だけ、
AWSアカウントを新規で
払い出す

AWS Control Tower

OU設計
AWSアカウントの新規作成

AWS IAM
Identity Center

AWSアカウントへの
アクセス権の付与
(IDP：全社共通)

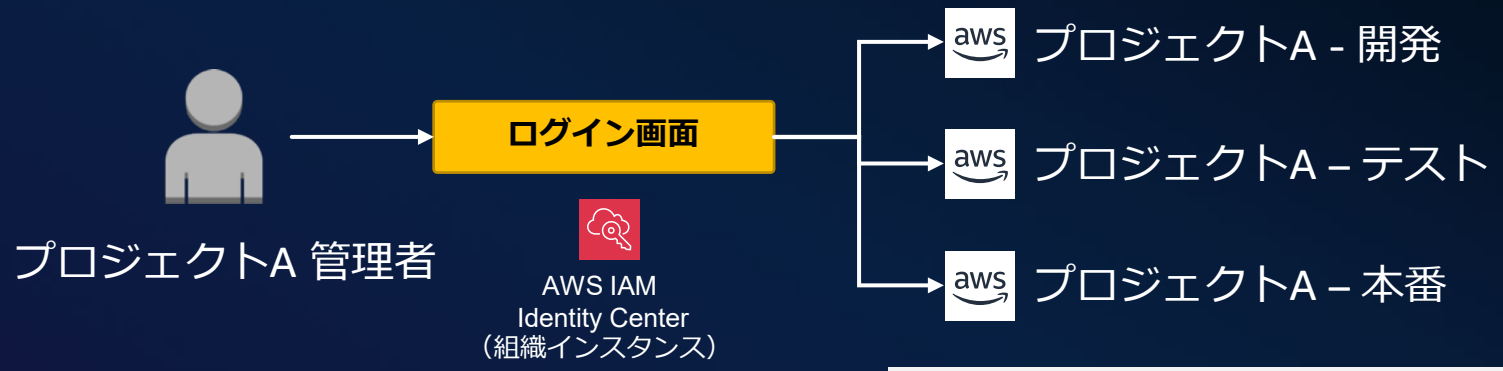


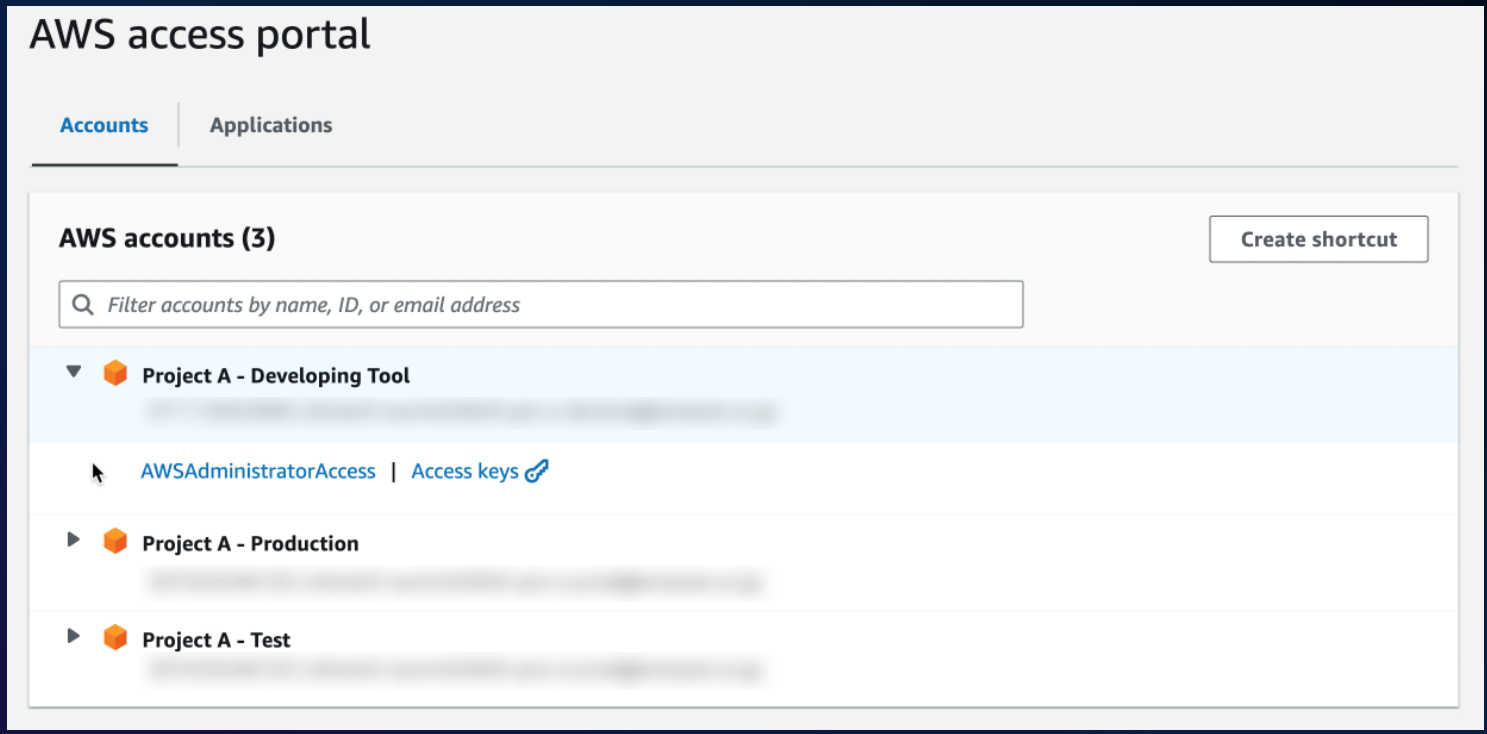
Demo – シナリオ設定

	役割	責任
 プラットフォーム担当	組織全体の AWSアカウント管理 社員IDPの管理  AWS Organizations  AWS IAM Identity Center	全社共通 セキュリティベースラインの提供 （予防・検知・証跡ログ）  AWS Control Tower
 プロジェクトA 管理者	プロジェクトレベルのAWS環境管理 チームコラボ環境・開発環境の提供 本番環境リリース承認  Amazon CodeCatalyst	プロジェクトレベルの アクセス管理 （プロジェクト用 IDP）※ 本番環境への責務  AWS IAM Identity Center アカウントインスタンス
 プロジェクトA 開発担当	リポジトリ管理（コード・リソース） 環境ごとプロファイル管理 デプロイ・監視  Amazon CodeCatalyst	ソースリポジトリ 自身の開発環境

※ IDP は単一が理想のため推奨しているわけではないが
管理部門がまたがってアジリティが落ちる・柔軟性が必要な場合は
アカウントインスタンスという手段もある

プロジェクト初期：プロジェクト環境の事前準備





プロジェクト初期：プロジェクト環境の事前準備

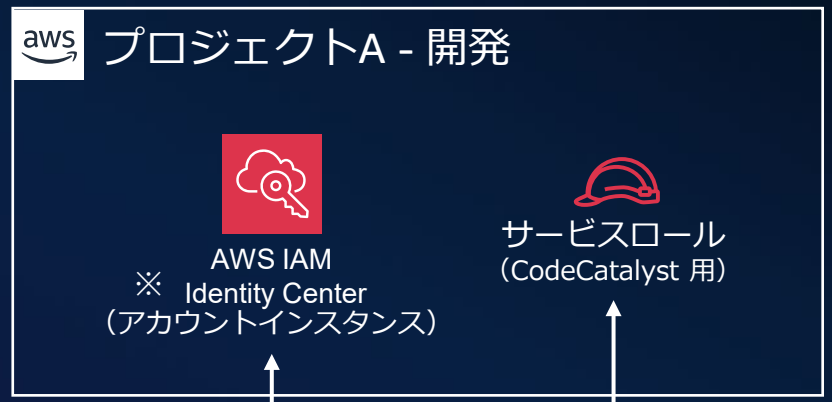
プロジェクトA 管理者

開発プロジェクトの環境整備

Amazon CodeCatalyst と
連携するための設定（SSO、サービス
ロール）をセットアップ

プロジェクト初期：プロジェクト環境の事前準備

プロジェクトA 管理者

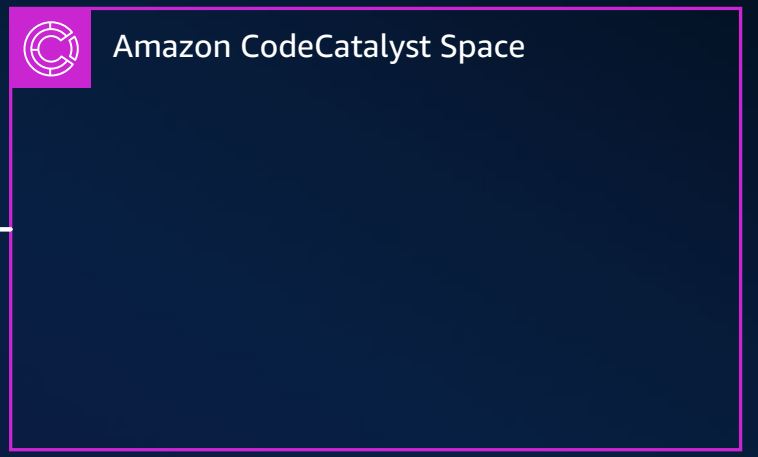


※ IDP は単一が理想のため推奨しているわけではない前提で、
アカウントレベルでもインスタンスが構築できる例
（あとから組織インスタンスに変更も可能）

利用ユーザはグループで指定する

開発プロジェクトの環境整備

Amazon CodeCatalyst と
Space を作成して連携



※CodeCatalyst の Space は、お客様の AWS アカウントの
外部に存在する

プロジェクト初期：プロジェクト環境の事前準備

プロジェクトA 管理者

aws プロジェクトA - 開発

AWS IAM
※ Identity Center
（アカウントインスタンス）

サービスロール
（CodeCatalyst 用）

aws プロジェクトA – テスト

サービスロール
（CodeCatalyst 用）

aws プロジェクトA – 本番

サービスロール
（CodeCatalyst 用）

同様にテスト・本番用
AWSアカウントそれぞれでも、
CodeCatalyst Space との連携を承認
サービスロールを作成

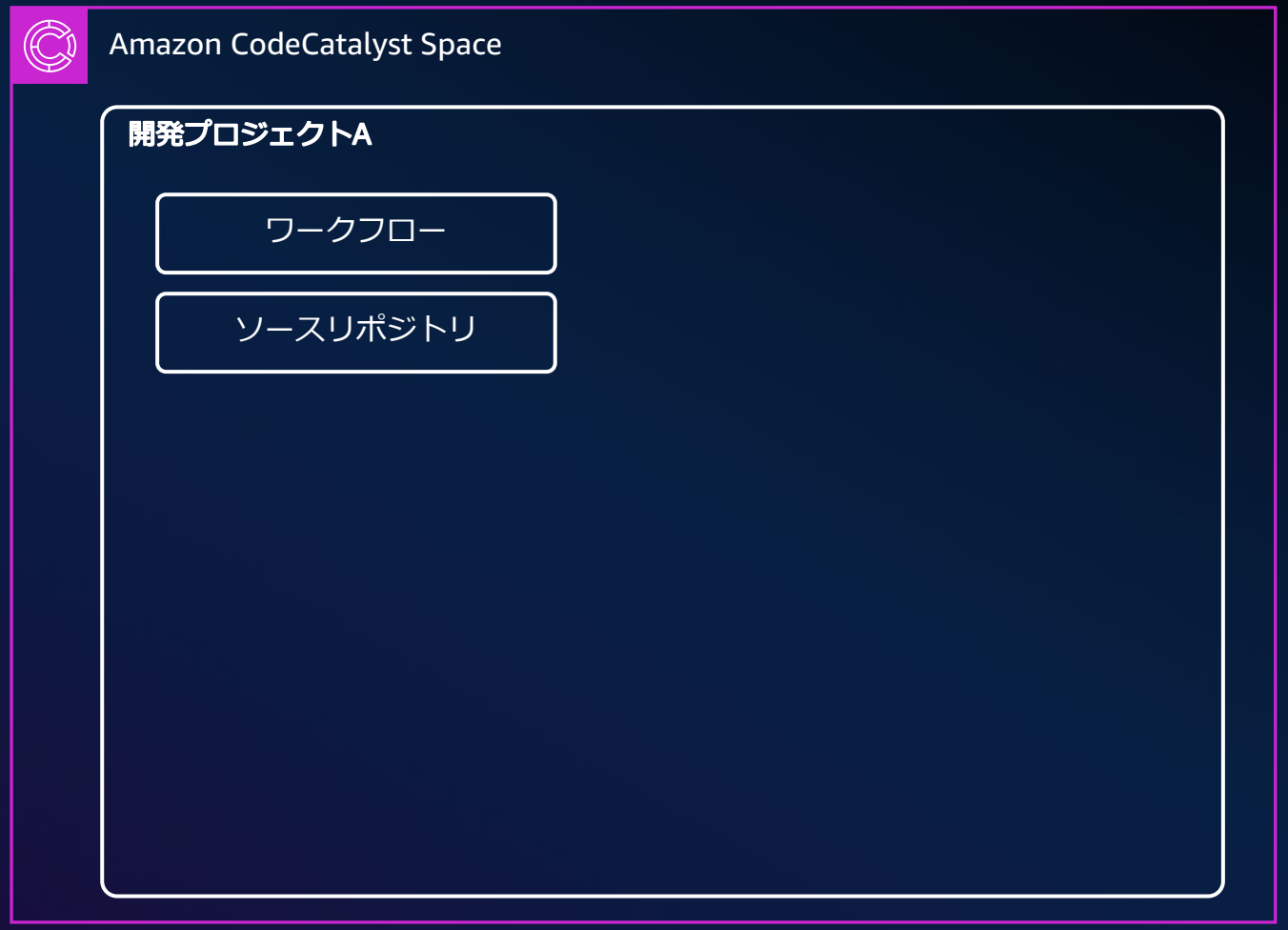
Space の設定で連携するAWSアカウントを登録

Amazon CodeCatalyst Space

プロジェクト初期：プロジェクト環境の事前準備

プロジェクトA 管理者

CodeCatalyst の Space に
プロジェクトを作成
利用ユーザの設定
環境の設定 etc....



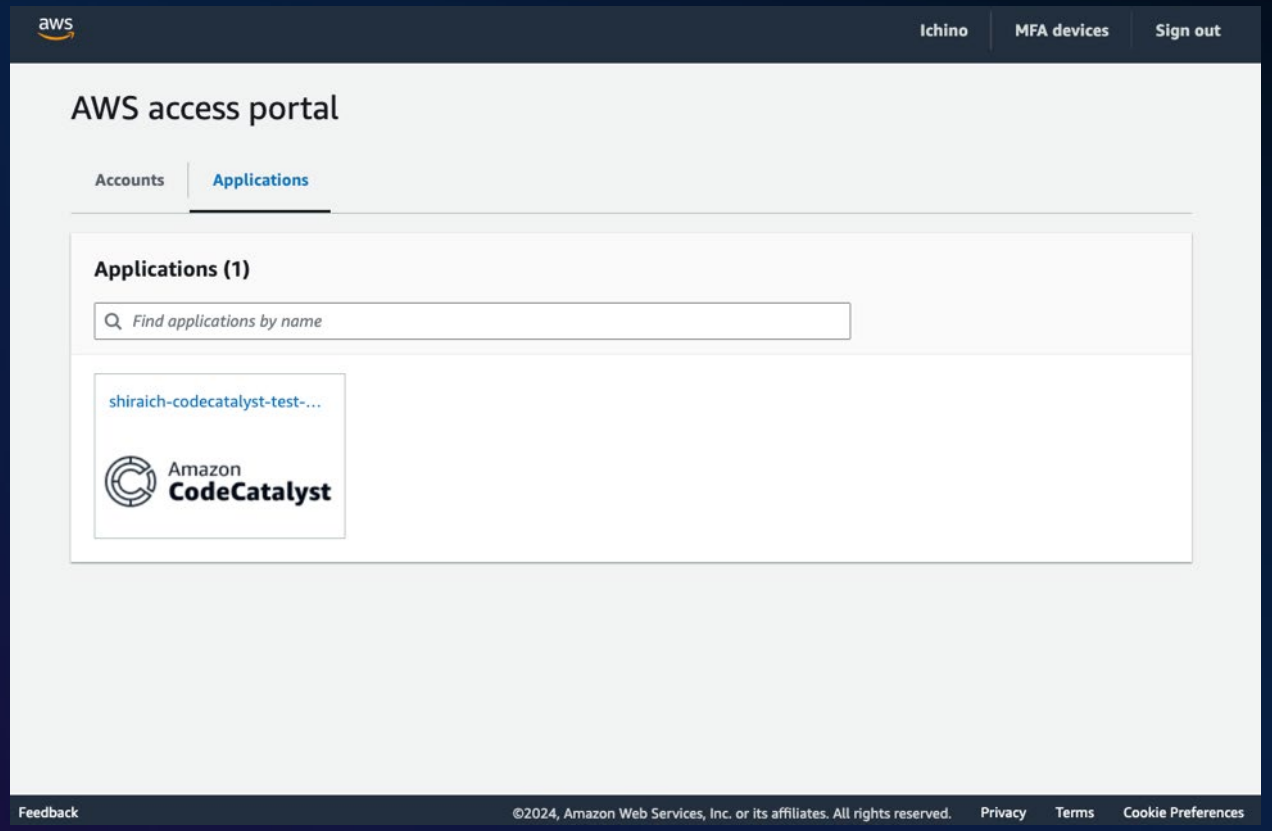
開発担当の作業

プロジェクトA 開発担当

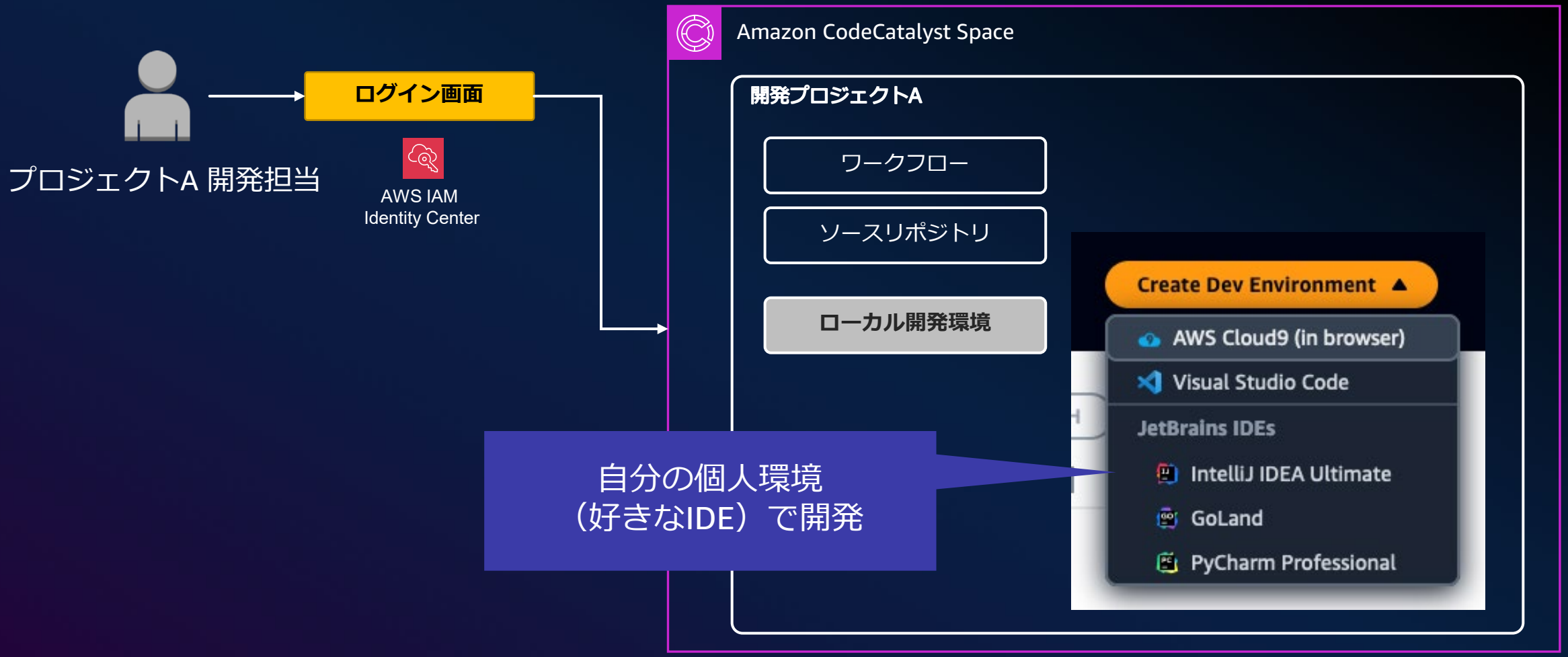
ログイン画面

AWS IAM
Identity Center

Amazon CodeCatalyst Space



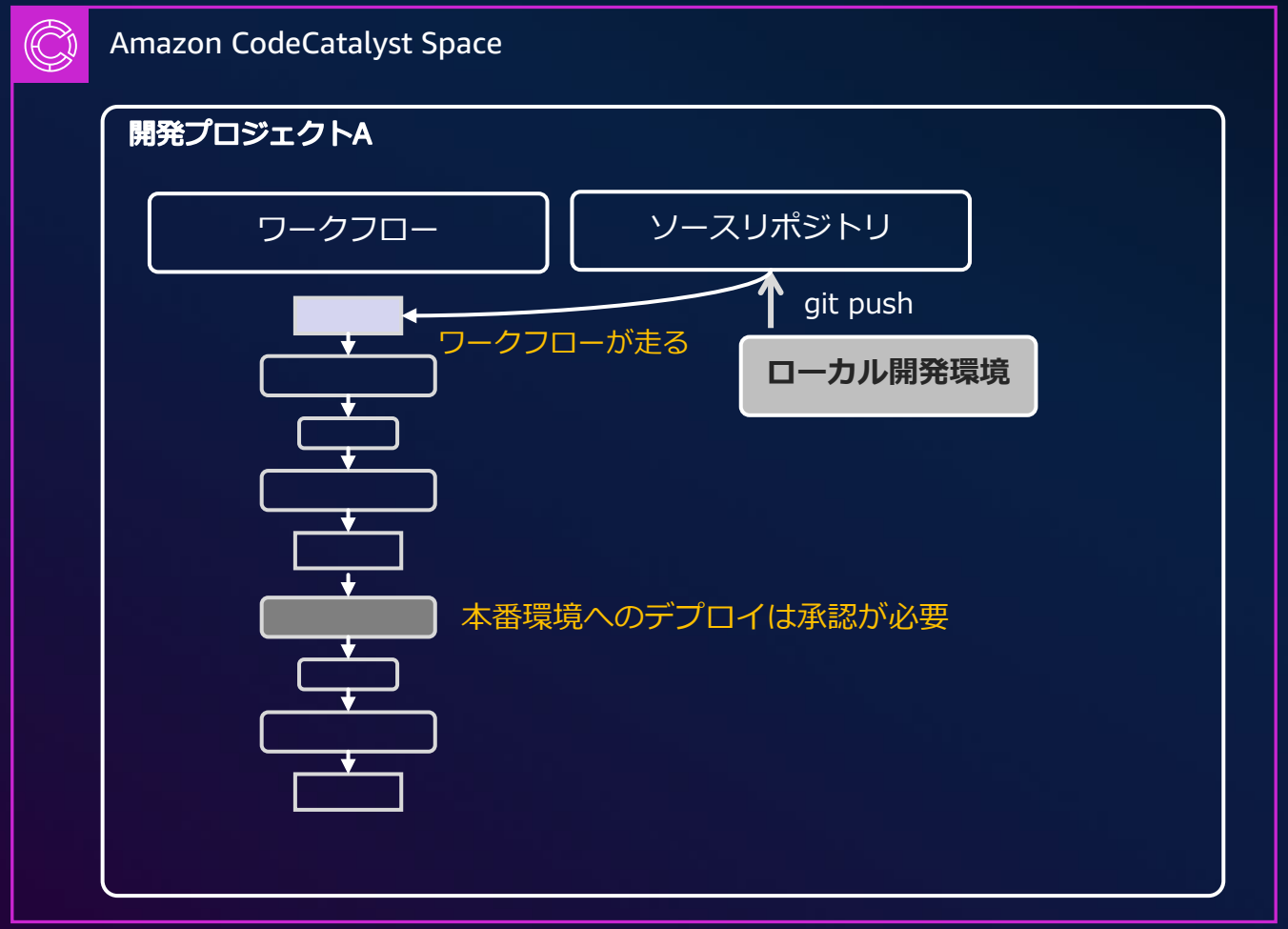
開発担当の作業



Demo：ワークフロー

開発者は、リポジトリに対する操作のみ

プロジェクトA 開発担当



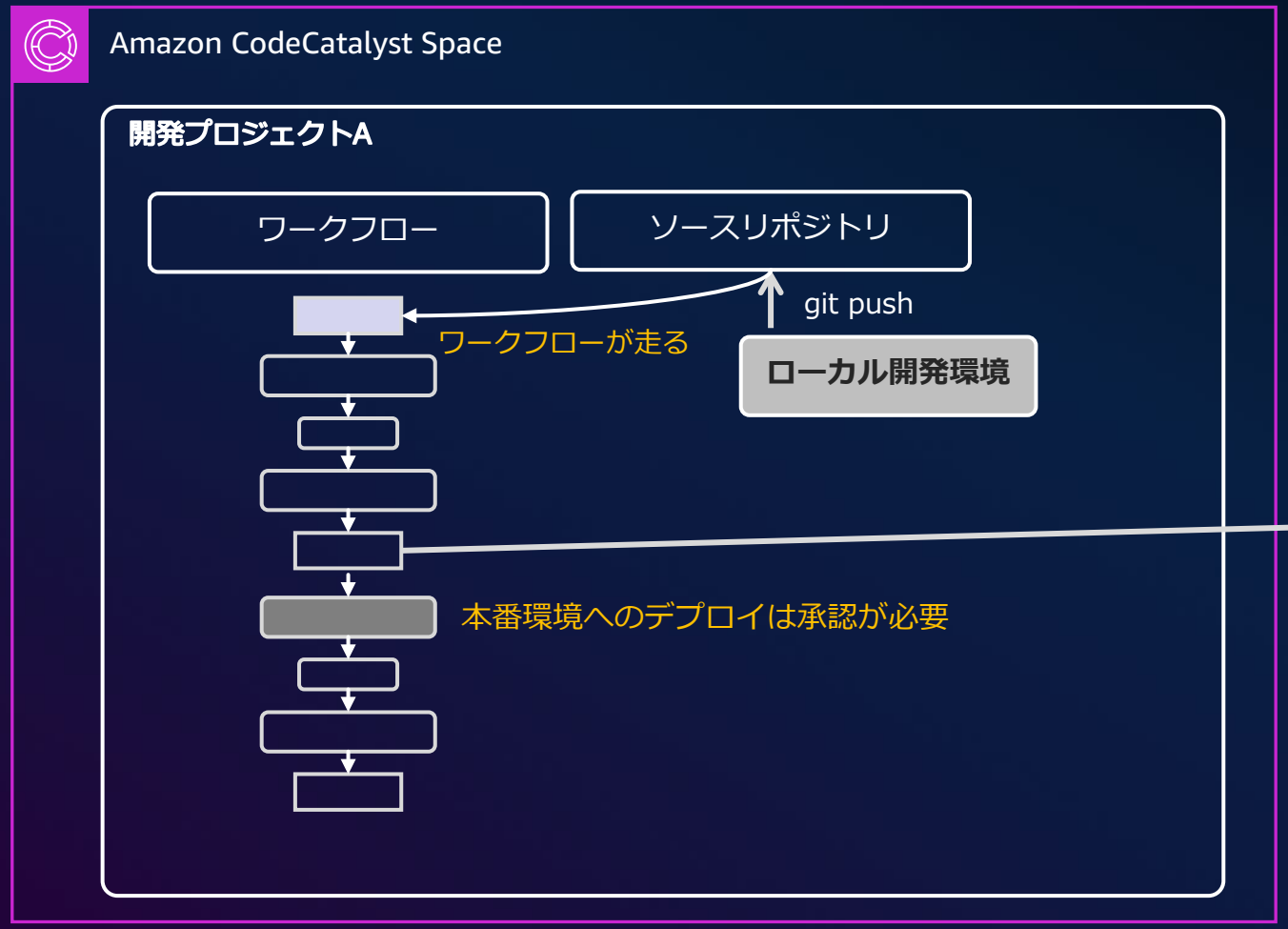
リポジトリの変更を受けて、
自動で各環境にデプロイする

（※図は概念図になります。Artifactの実態を表しているわけではありません）

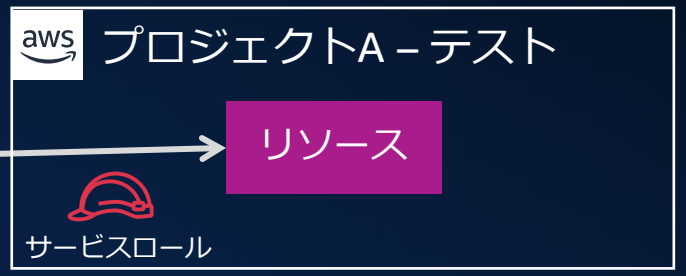
Demo：ワークフロー

開発者は、リポジトリに対する操作のみ

プロジェクトA 開発担当



リポジトリの変更を受けて、
自動で各環境にデプロイする

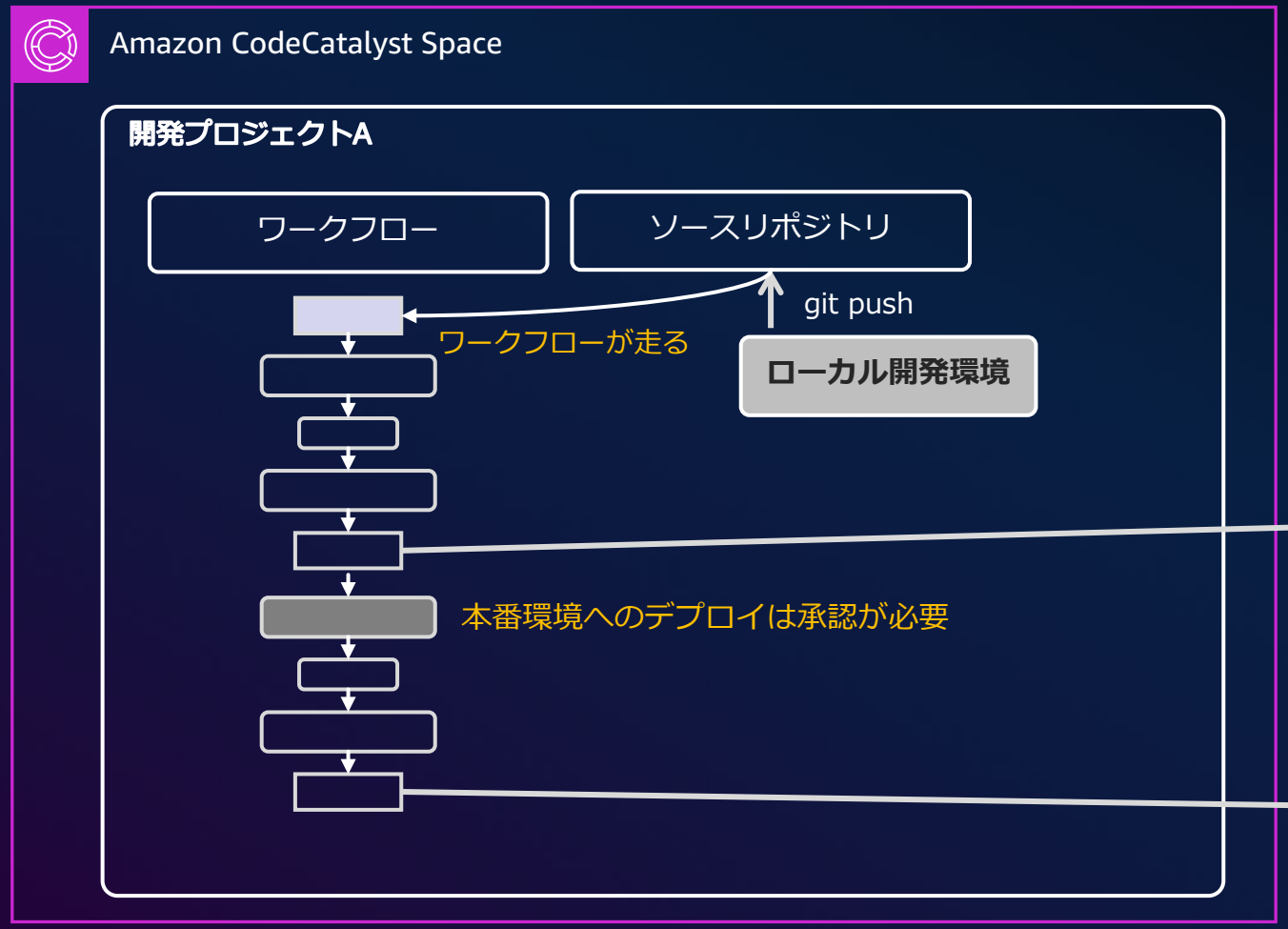


（※図は概念図になります。Artifactの実態を表しているわけではありません）

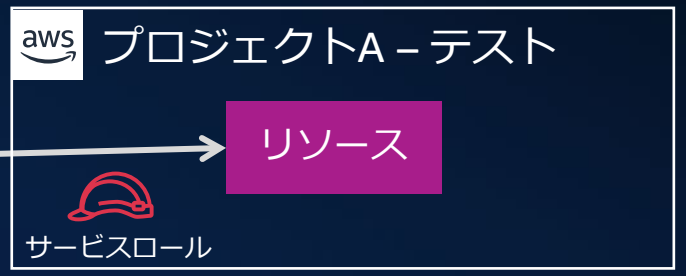
Demo：ワークフロー

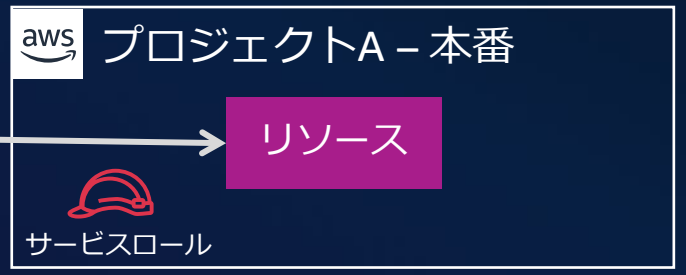
開発者は、リポジトリに対する操作のみ

プロジェクトA 開発担当



リポジトリの変更を受けて、
自動で各環境にデプロイする





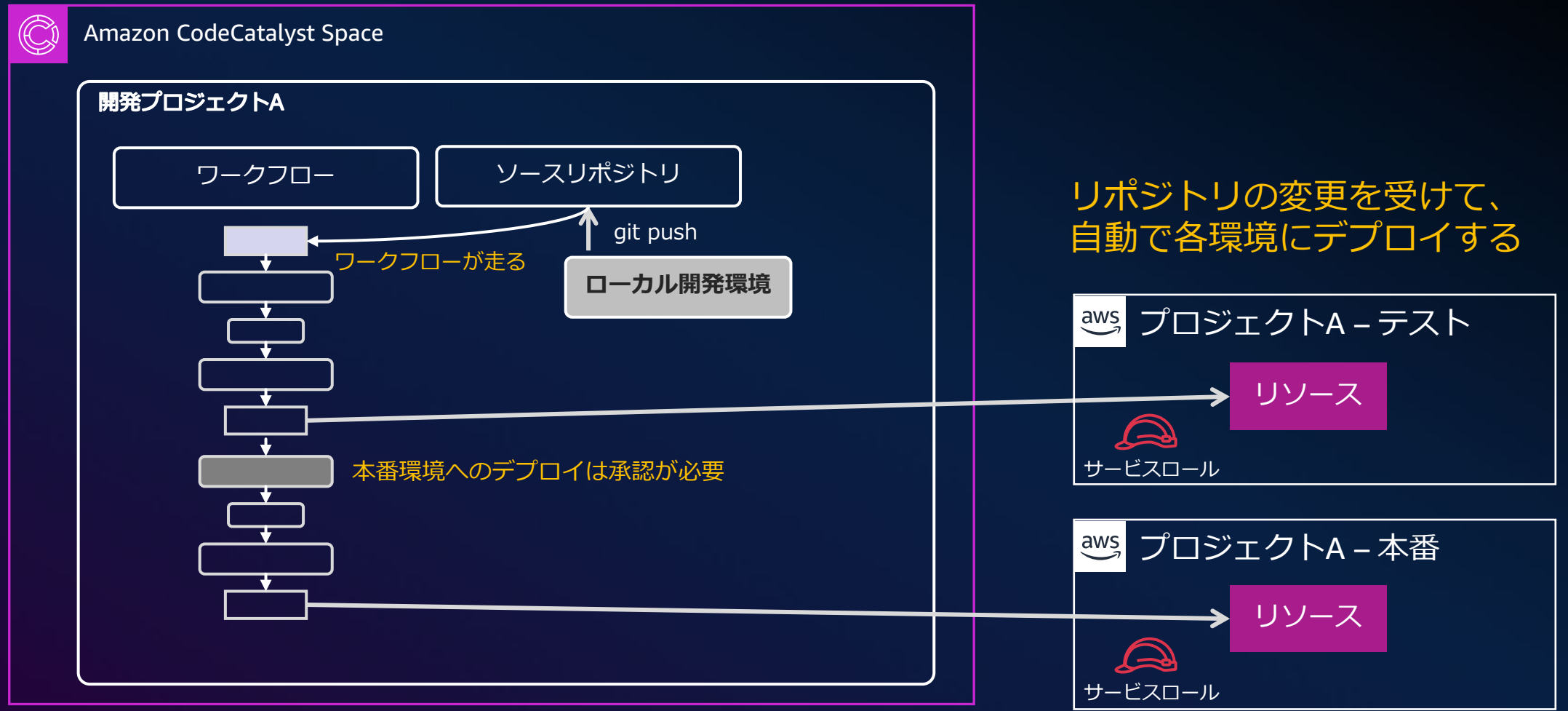
（※図は概念図になります。Artifactの実態を表しているわけではありません）

Demo：ワークフロー

再掲

開発者は、リポジトリに対する操作のみ

プロジェクトA 開発担当



（※図は概念図になります。Artifactの実態を表しているわけではありません）

（3）インパクトの大きい環境に対しては「自動化」

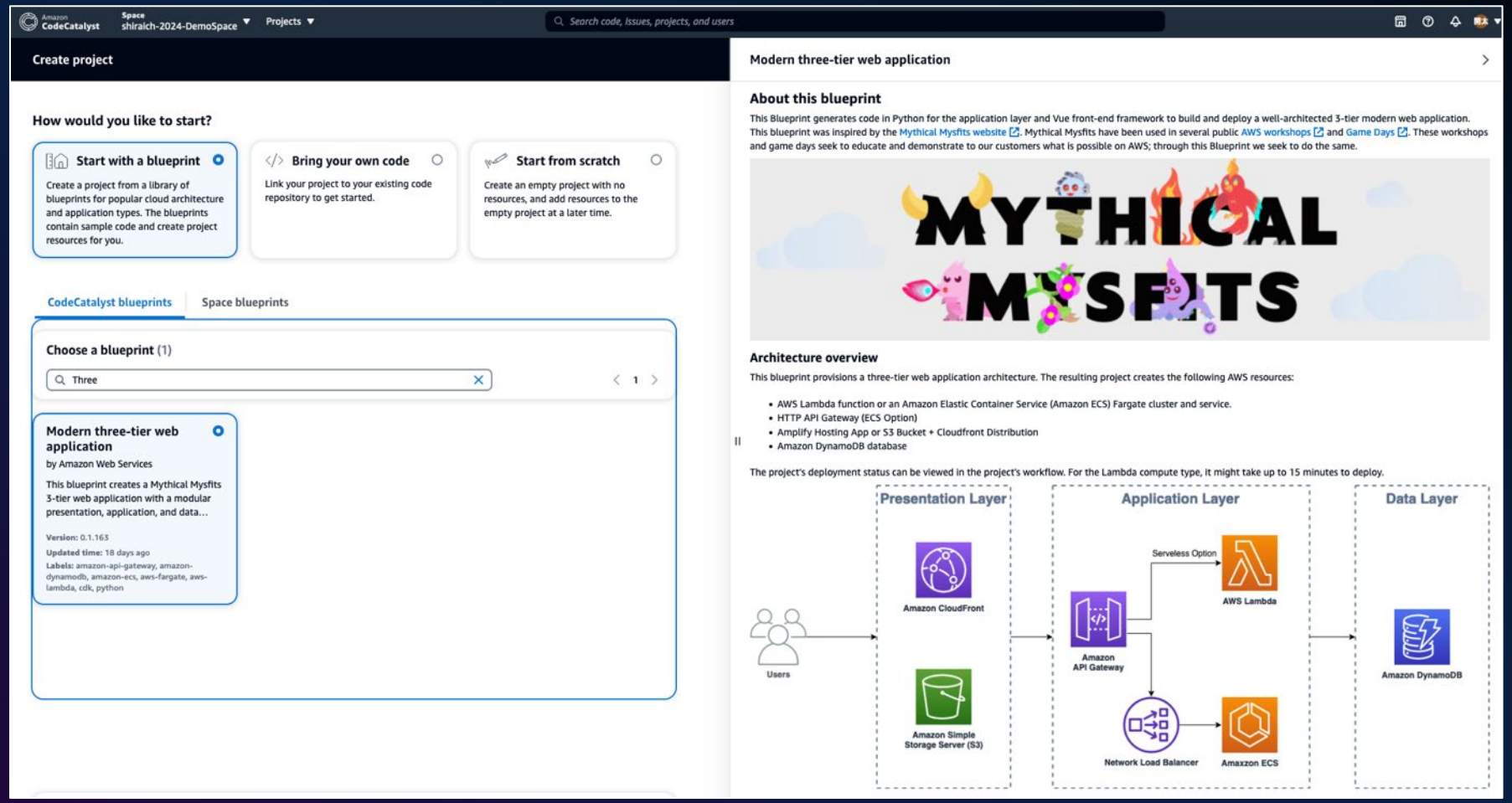
本番環境

人間の操作を極力排除
＝ **自動化**

（ゼロタッチプロダクション）

自動化作業も一度だけではなく
繰り返し実践する、「成功体験を積み上げること」

本日の お見せしたプロジェクトは Amazon CodeCatalyst の Blueprint から作成が可能



Amazon CodeCatalyst で Environment を利用したマルチアカウントのデプロイ
https://aws.amazon.com/jp/blogs/news/using-environments-multi-account-deployments-with-amazon-codecatalyst/

本日省略したCI/CDパイプラインの実装における詳細・考慮事項の参考

[buiders.flash] AWS でデプロイの自動化を実現するベストプラクティスをグラレコで解説
https://aws.amazon.com/jp/builders-flash/202005/awsgeek-ci-cd/

[AWSブログ] 新規 ― デプロイメントパイプラインのリファレンスアーキテクチャとリファレンス実装
https://aws.amazon.com/jp/blogs/news/new_deployment_pipelines_reference_architecture_and_-reference_implementations/

まとめ

再掲

ガードレールの仕組み　必要なのは「メカニズム」

スピーディーにビジネスを生み出す 仕掛け

（１）**「環境分離」**により影響範囲を小さくする

（２）望ましくない設定を**「検知・予防」**する

（３）インパクトの大きい環境に対しては**「自動化」**

イノベーションを起こし続けるには

“メカニズム”が必要

恐がらずに行動できる仕掛け

Thank you!

Ichino Shiraishi
X : @piko_san_0000