

JAPAN | 2024

aws SUMMIT



サーバーレス開発のベストプラクティス ～より効果的に、より賢く使いこなすために～

淡路 大輔

アマゾン ウェブ サービス ジャパン合同会社
ガバメントクラウド技術本部
ソリューションアーキテクト

Daisuke Awaji

Amazon Web Services Japan
Solutions Architect



@gee0awa



Aganda

- 01 サーバーレスとは？
- 02 サービスフルなサーバーレス
- 03 生成 AI とサーバーレス
- 04 まとめ

“サーバーレス”とは？

The History of Serverless

サーバーレスと出会ってから何年経ちましたか？

 Amazon SQS

 Amazon S3

 AWS Lambda

Available in Tokyo
5 Minute Functions

Access resources
within a VPC

 AWS Step Functions

 Support SQS as Event Source

Up to 15 minutes
Custom runtimes

Provisioned
concurrency

2006

2014

2015

2016

2017

2018

2019

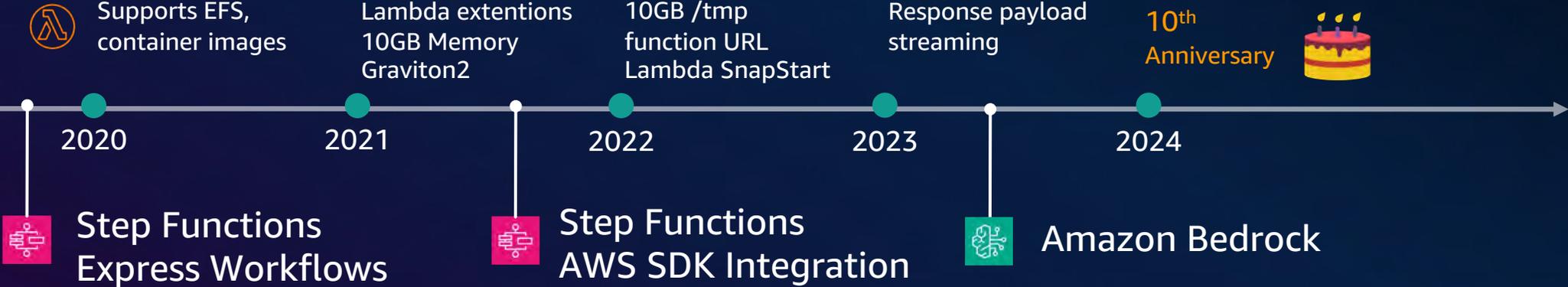
 Amazon API Gateway

 Amazon EventBridge



The History of Serverless

サーバーレスと出会ってから何年経ちましたか？



“サーバーレス” に求められること



サーバーやインフラの管理が
不要で実行できるコード

複雑なインフラを管理することなく
顧客に**価値を提供**できること

サービスフルなサーバーレス

Compose, Configure, then code

サーバーレスアプリケーション



AWS Lambda

サーバーレスアプリケーション

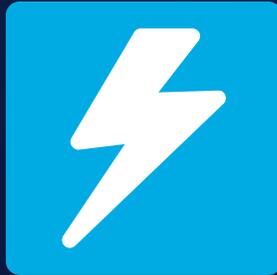
Function



Node.js
Python
Java
.NET
Go
Ruby
PowerShell
Runtime API

サーバーレスアプリケーション

Event source



Function



データの変更



APIリクエスト



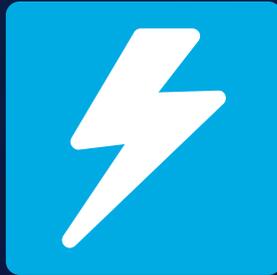
リソースの
状態変更



Node.js
Python
Java
.NET
Go
Ruby
PowerShell
Runtime API

サーバーレスアプリケーション

Event source



Function



Services



データの変更



APIリクエスト



リソースの
状態変更

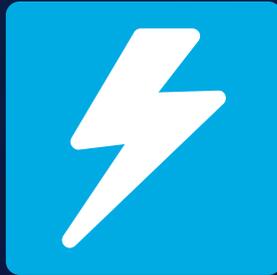


Node.js
Python
Java
.NET
Go
Ruby
PowerShell
Runtime API



サーバーレスアプリケーション

Event source



Services



データの変更



APIリクエスト



リソースの
状態変更





Transport (転送) ではなく Transform (変換) に使用する

イベント連携やフロー管理のサービスを活用する

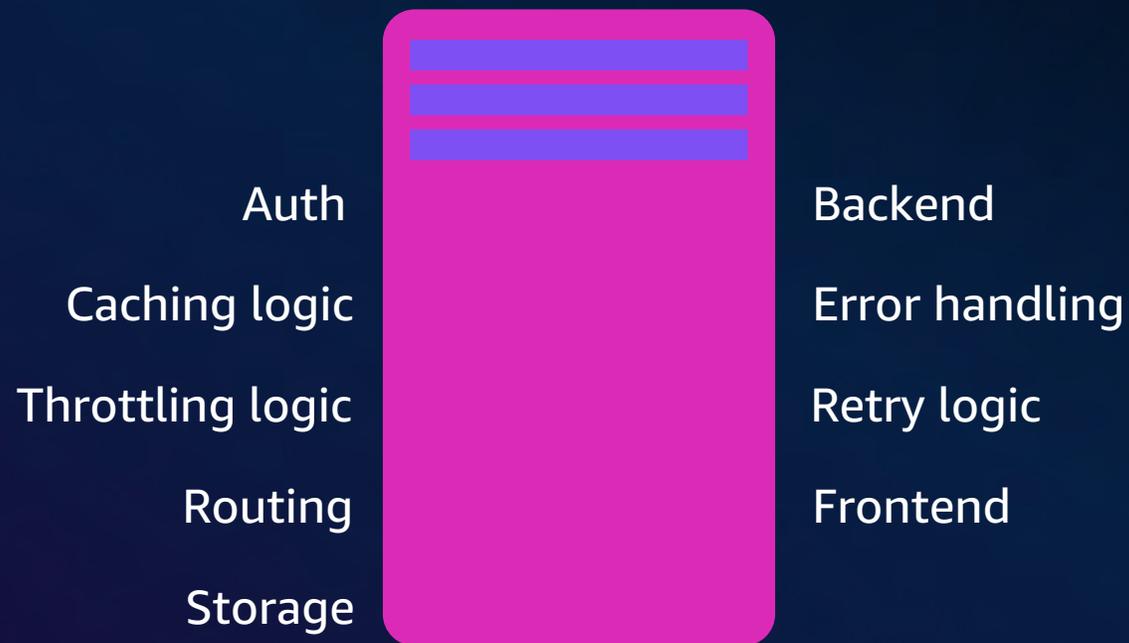
つなげる Lambda を減らし、“独自にやりたい処理”に Lambda を使う

LOGIC





“Traditional” なアプリを考える



The lift and shift - リフト & シフト



Auth
Caching logic
Throttling logic
Routing



Backend
Error handling
Retry logic
Frontend

Storage



The migration – サービスに責務を移譲する



Frontend

Auth
Caching logic



Throttling logic
Routing

Backend



Storage



単一責務の原則と Lambda 関数の単位

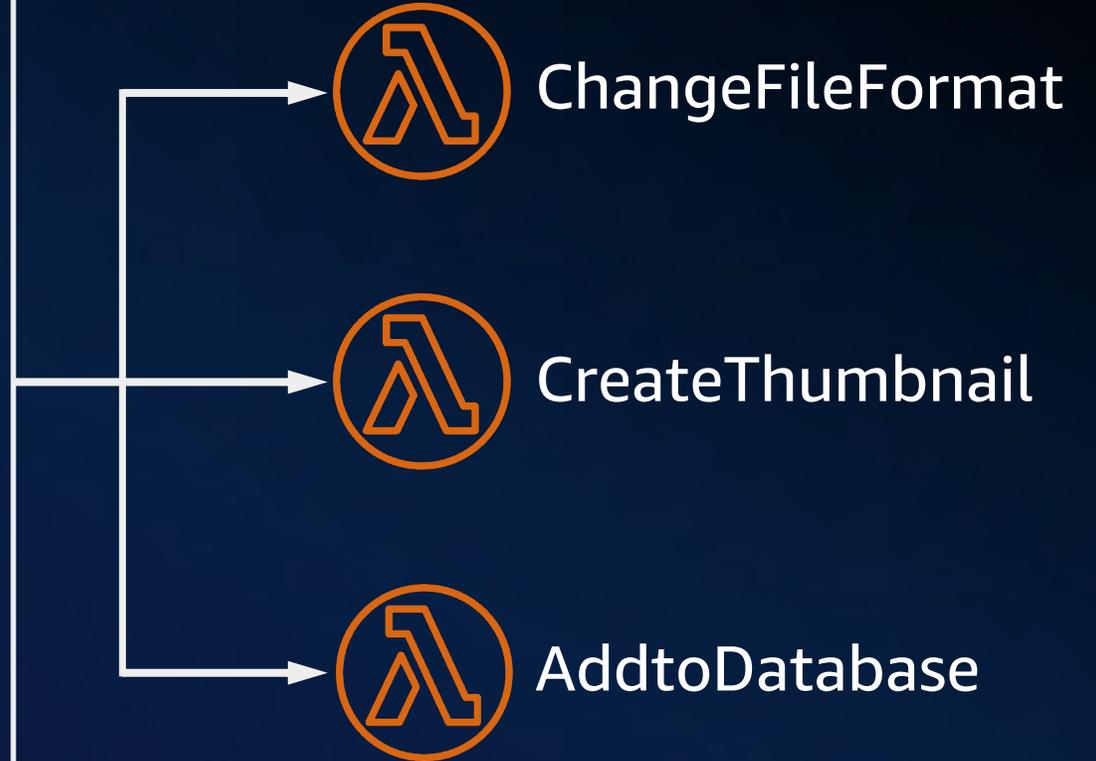
```
CODE EDITOR

 画像処理を実行する関数

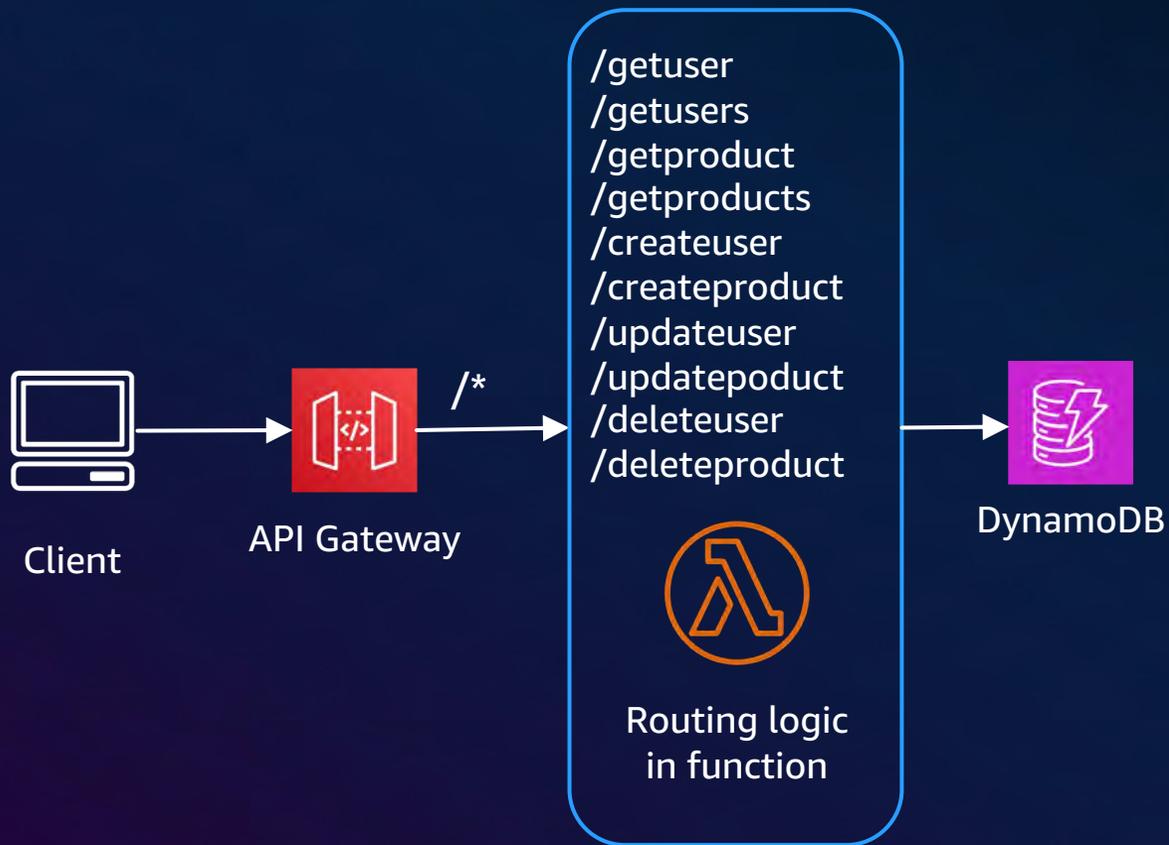
# ファイルのフォーマットを変更する
function ChangeFileFormat(event, context)
{...}

# サムネイル画像を作成する
function CreateThumbnail(event, context)
{...}

# データベースに保存する
function AddtoDatabase(event, context)
{...}
```

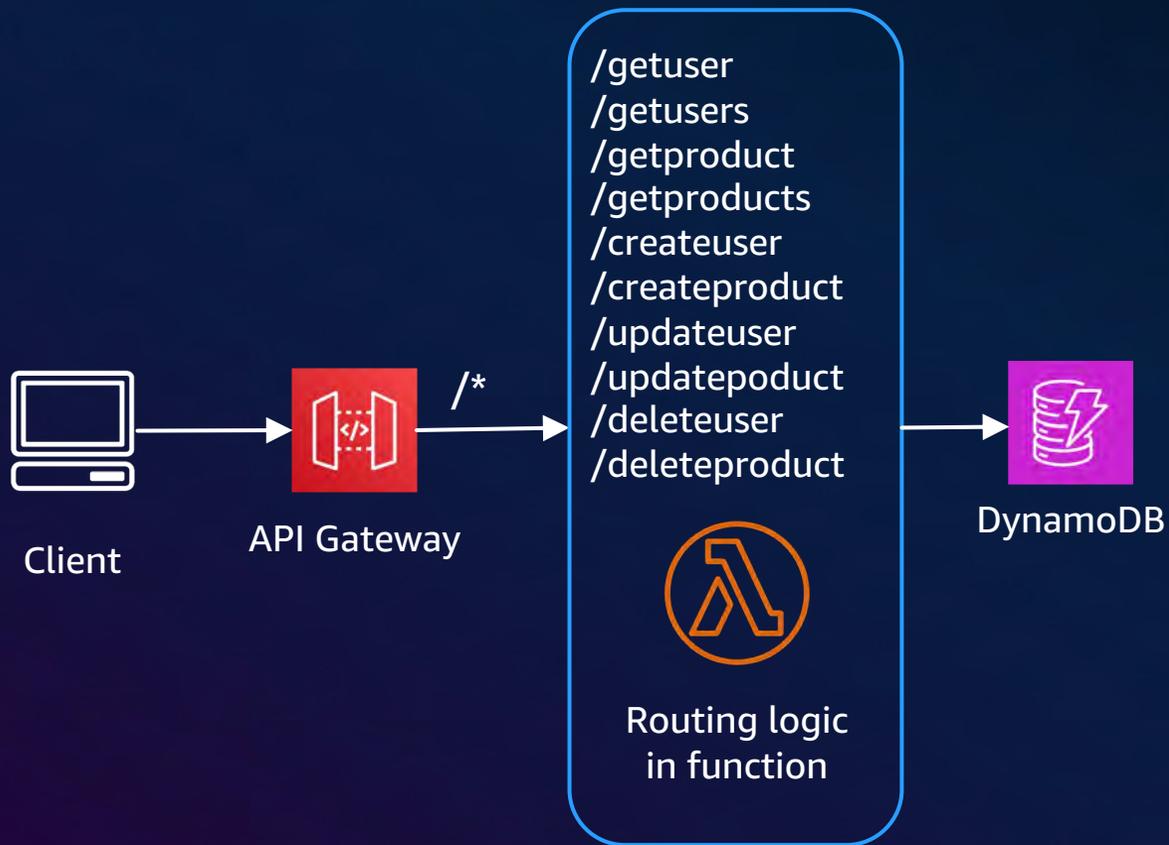


Lambda 関数に何を含めるべきか？



“Lambda-lith”

Lambda 関数に何を含めるべきか？



“Lambda-lith”

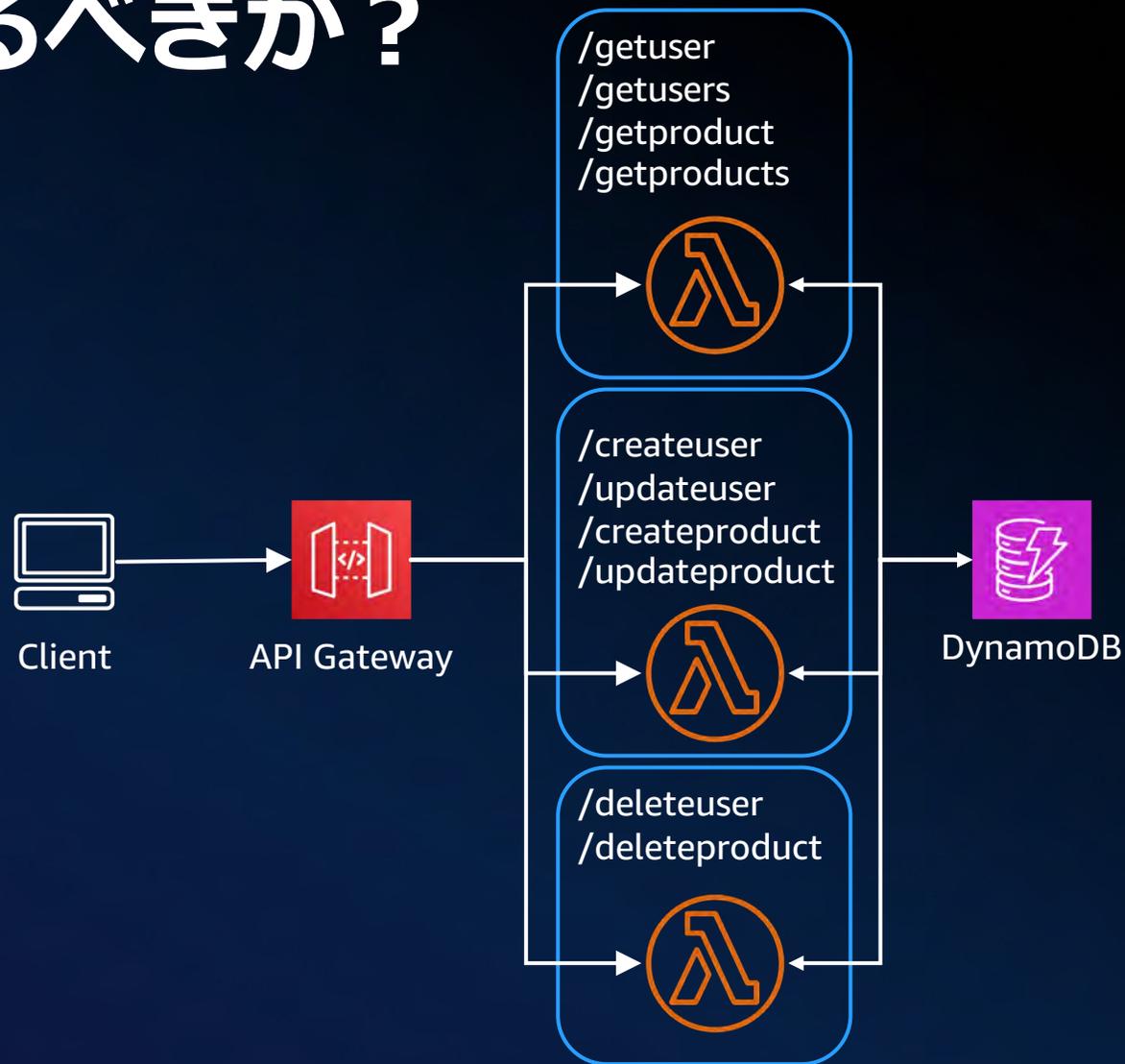


“Micro Lambda”

Lambda 関数に何を含めるべきか？

グループ化の例

- 境界付けられたコンテキスト
- 開発組織の構造
- IAMパーミッションのスコープ
- 共通的なコードの依存関係
- 配下のリソースとの依存関係
- 初期化時間とコールドスタート
- メモリ割り当て

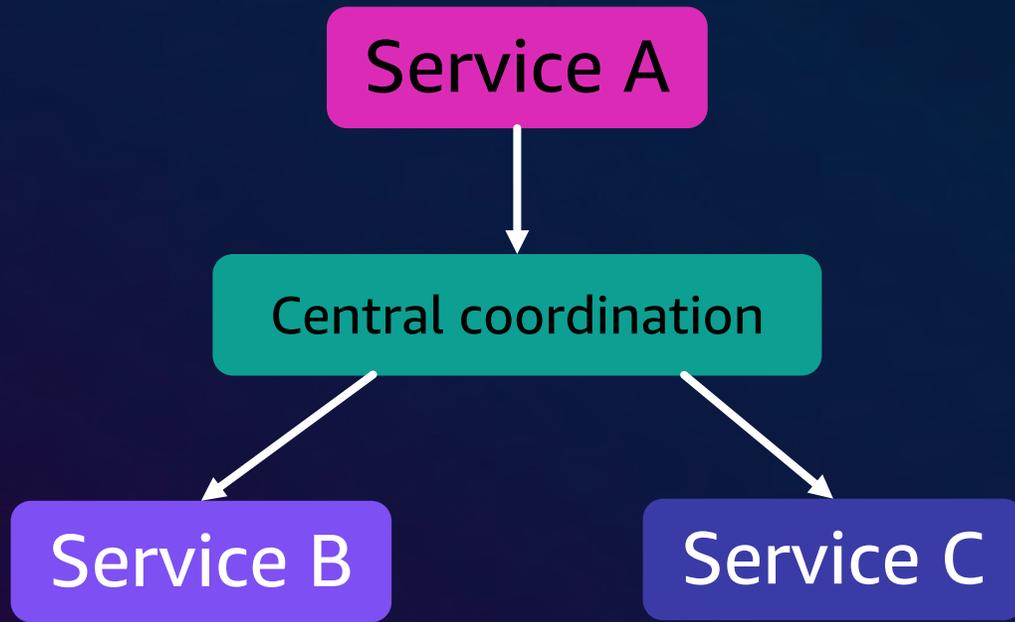


“Pragmatic Lambda”

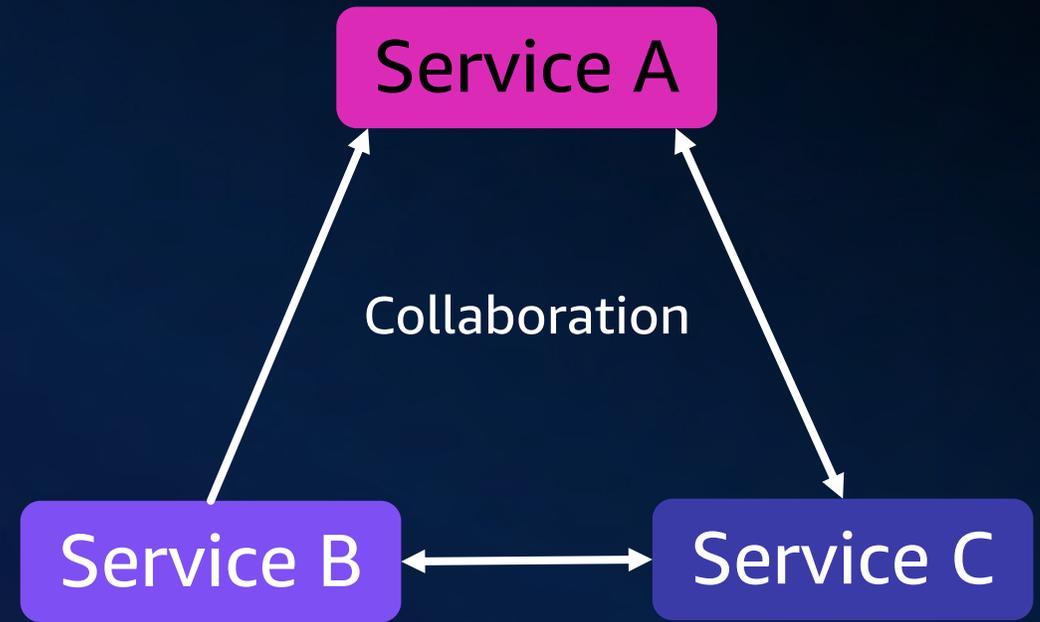
オーケストレーションとコレオグラフィ

独自のコードを記述するのではなく、CONFIGURATIONとして組み込むことを考える

オーケストレーション



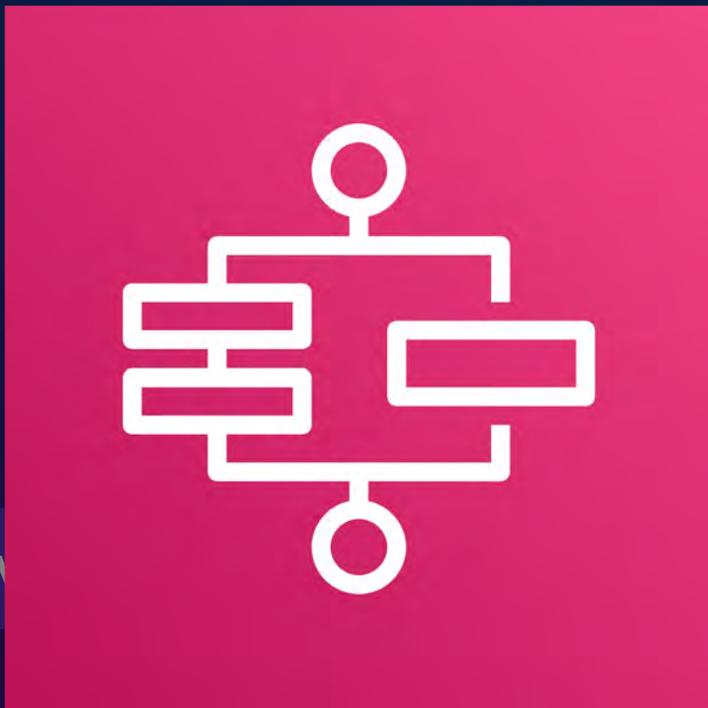
コレオグラフィ



オーケストレーションとコレオグラフィ

独自のコードを記述するのではなく、CONFIGURATIONとして組み込むことを考える

オーケストレーション



AWS Step Functions

コレオグラフィ



Amazon EventBridge

オーケストレーションと コレオグラフィの使い分け

実際のアプリケーションでどう使われるか考えてみよう



ServerlessVideo

A live video streaming application built with serverless technologies.



ServerlessVideo

<https://s12d.com/video>

ライブストリーミングとオンデマンド配信

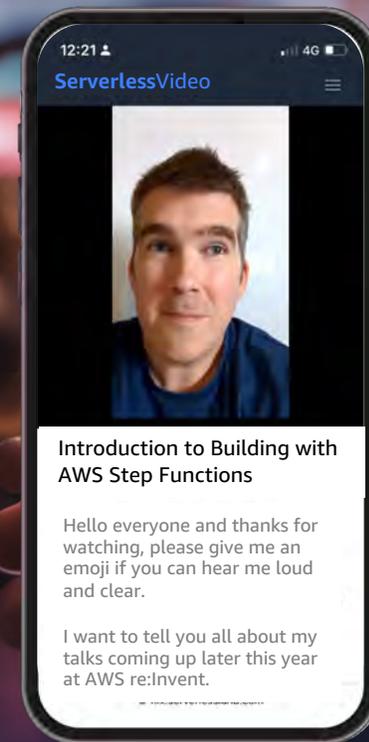
ServerlessVideo は、サーバーレス アーキテクチャで構築されたライブ ビデオ ストリーミング アプリケーションです。



ライブ配信



ライブ配信を
リアルタイムで
視聴



オンデマンドで
後から再生



ServerlessVideo



<https://s12d.com/video>

ServerlessVideo の特徴

生成 AI によるタイトル生成

Amazon Bedrock と Amazon Transcribe を使用してビデオからタイトルと説明文を生成します。

プラグインアーキテクチャ

AWS Step Functions と Amazon EventBridge によって拡張が容易でスケーラブルな構成に

選択的なサーバーレスコンピューティング

ビデオの長さに応じて、AWS Lambda や AWS Fargate を使い分けることにより、柔軟性を得る



ServerlessVideo

<https://s12d.com/video>

Frontend



Broadcast and viewing app



AWS account

チャンネル管理
サービス

ビデオ配信
ストリーミング
サービス

ビデオ管理
サービス

Frontend
通知/配信

Event bus

All
microservices
loosely
decoupled by
events

ビデオ後処理
サービス

プラグイン
マネージャー
サービス



Frontend



Broadcast and viewing app



AWS account

チャンネル管理サービス

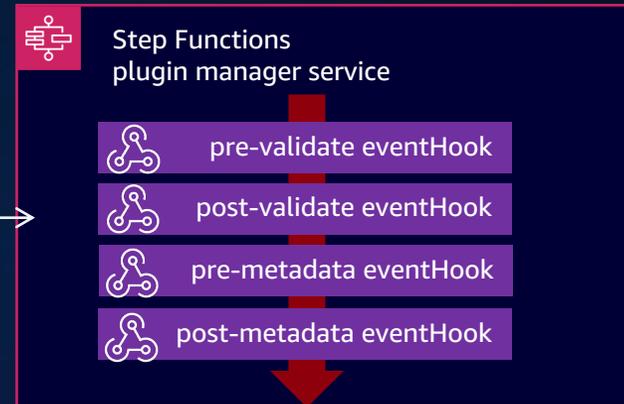
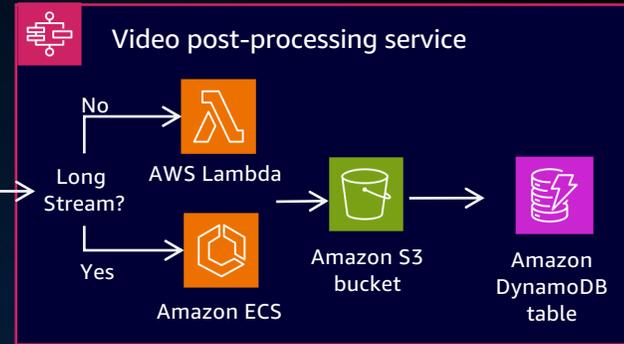
ビデオ配信
ストリーミングサービス

ビデオ管理サービス

Frontend
通知/配信

Event bus

All microservices loosely decoupled by events

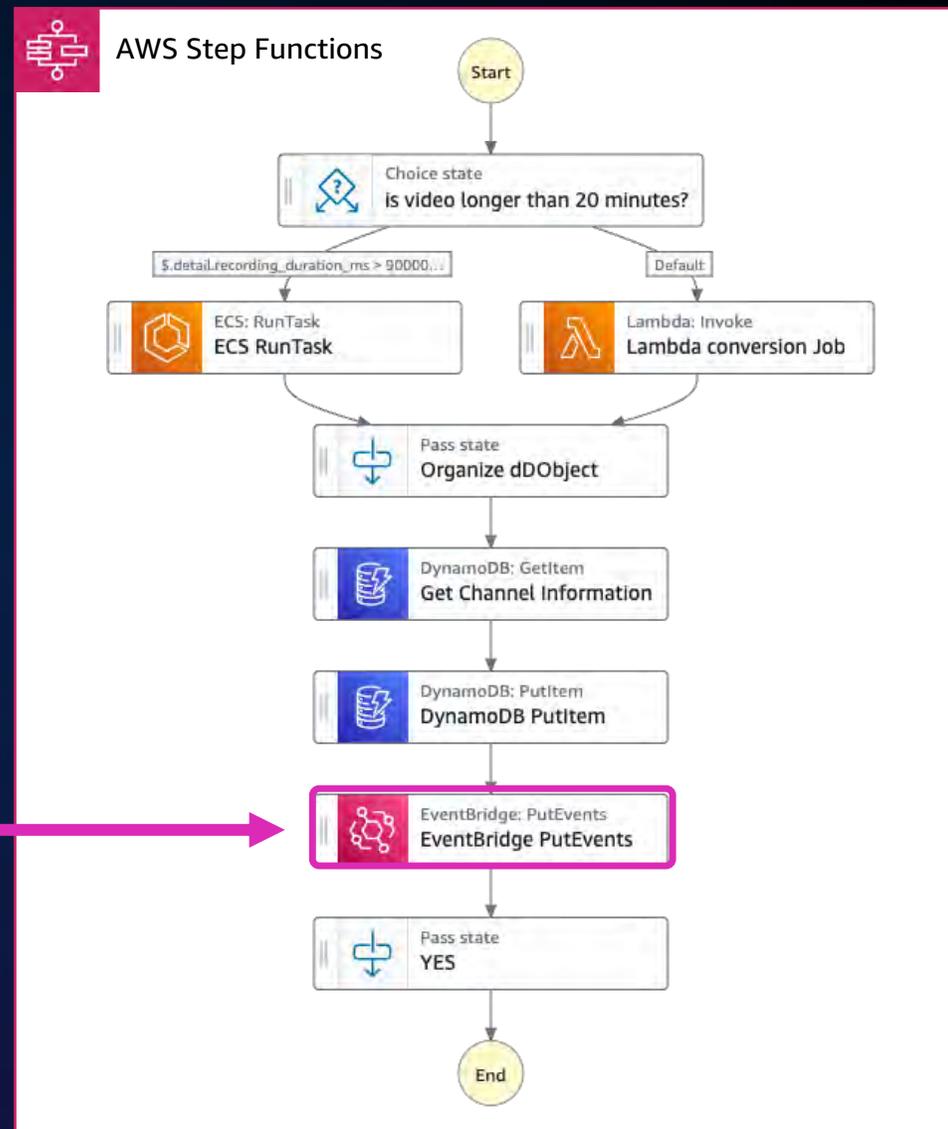


ドメインごとの処理をオーケストレーションする

VIDEO POST-PROCESSING SERVICE

- コンピューティングサービスの選択
- 可能な限りサービス統合を使用する

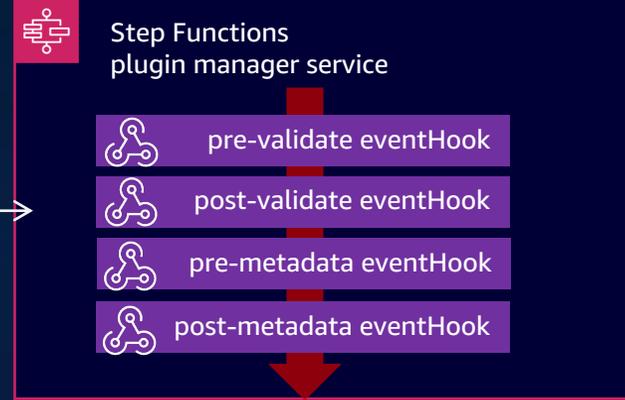
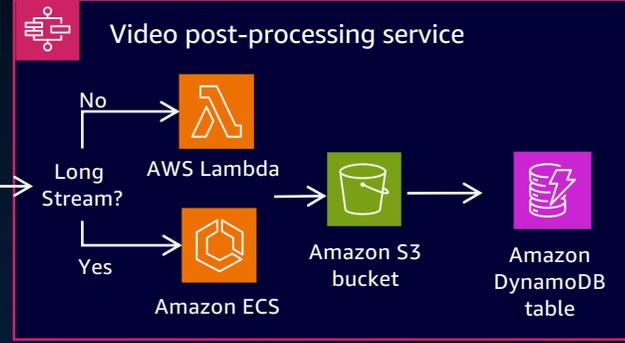
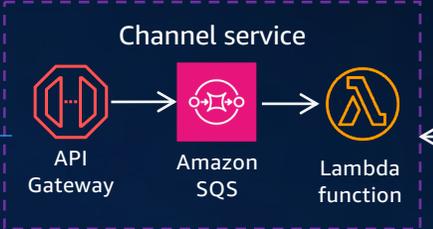
完了したら
イベントを送信



Frontend

AWS account

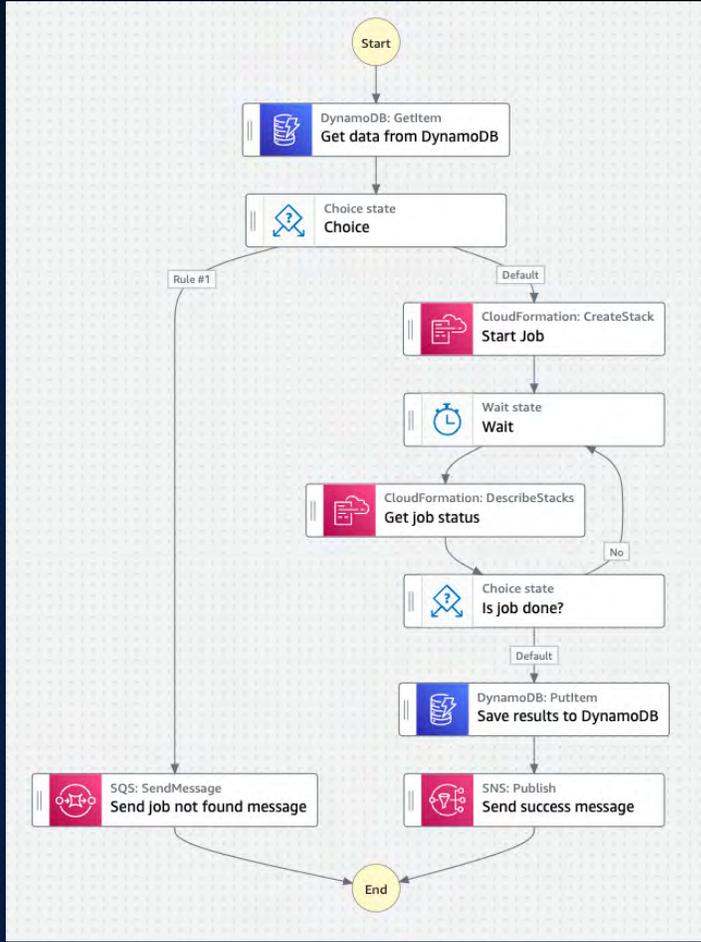
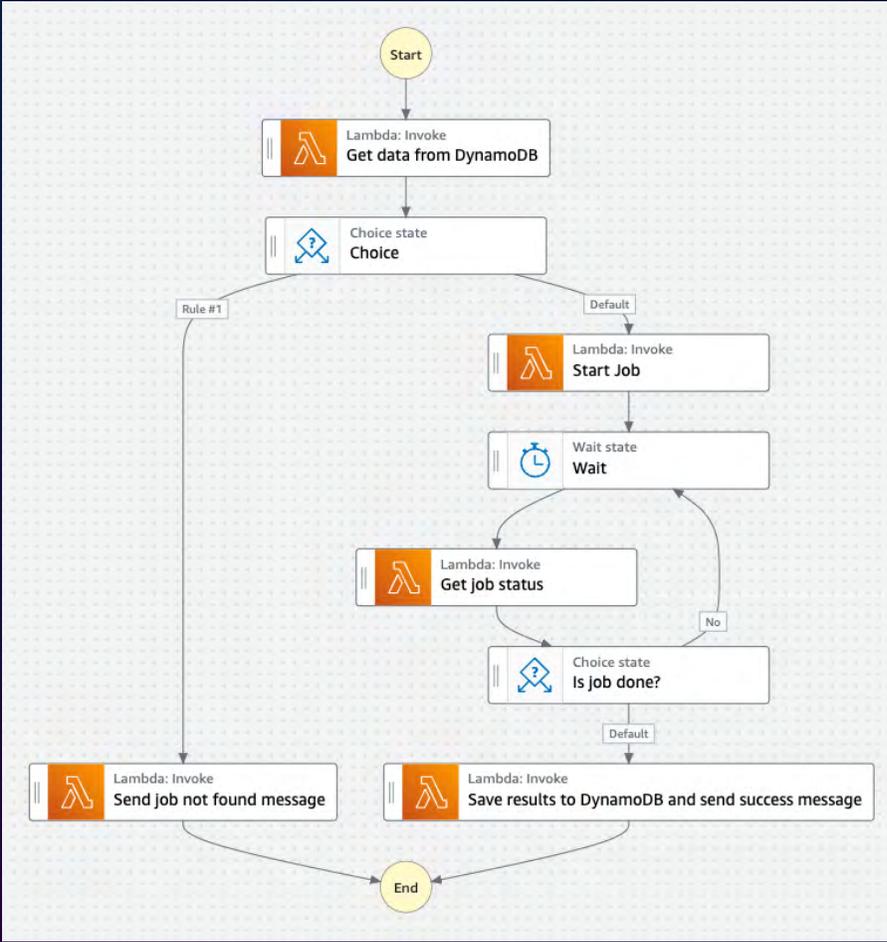
EventBridge Event bus



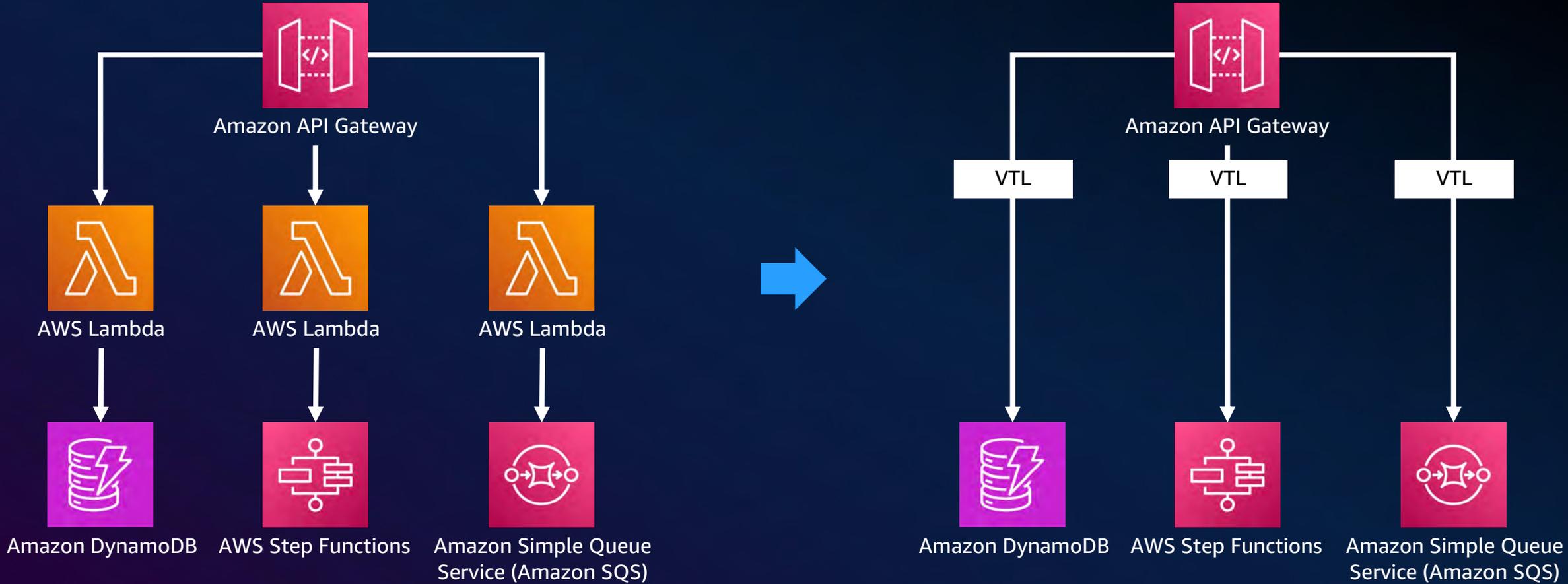
Broadcast and viewing app



Step Functions - AWS SDK integrations



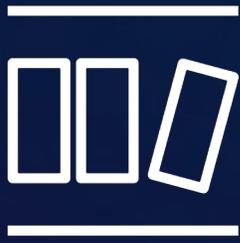
Amazon API Gateway のインテグレーション



コードによるインテグレーション



Amazon DynamoDB



Stream

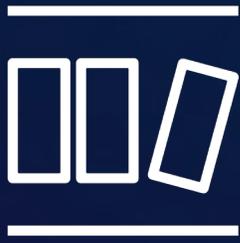
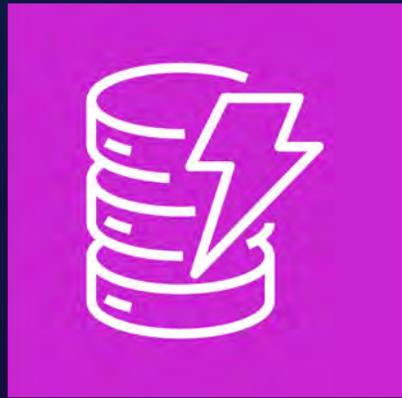


AWS Lambda



Amazon EventBridge
event bus

直接的なサービスによるインテグレーション



Stream

Amazon DynamoDB



Amazon EventBridge
pipes



Amazon EventBridge
event bus



シンプルな Lambda 関数を削除して
組み込みの統合に置き換えることができるか

生成 AI とサーバーレス



動画からタイトル・説明文を作り出す

ServerlessVideo は、サーバーレス アーキテクチャで構築されたライブ ビデオ ストリーミング アプリケーションです。



ライブ配信



ライブ配信を
リアルタイムで
視聴



タイトル・説明文を
生成 AI が作り出す

オンデマンドで
後から再生



ServerlessVideo



<https://s12d.com/video>

生成 AI アプリケーションを実装する

アプリケーションの要件

- 動画から複数のタイトル、説明を作成する
- 人間にフィードバックを依頼する
- ビデオ用のアバター画像を作成する



ServerlessVideo

AWS Step Functions のジョブ実行パターン



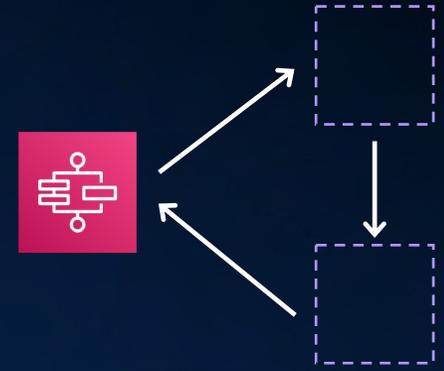
リクエスト-レスポンス

HTTP レスポンスだけを
待ってすぐに次に進みます



ジョブ実行
(.sync)

ジョブの完了を待って
次に進みます

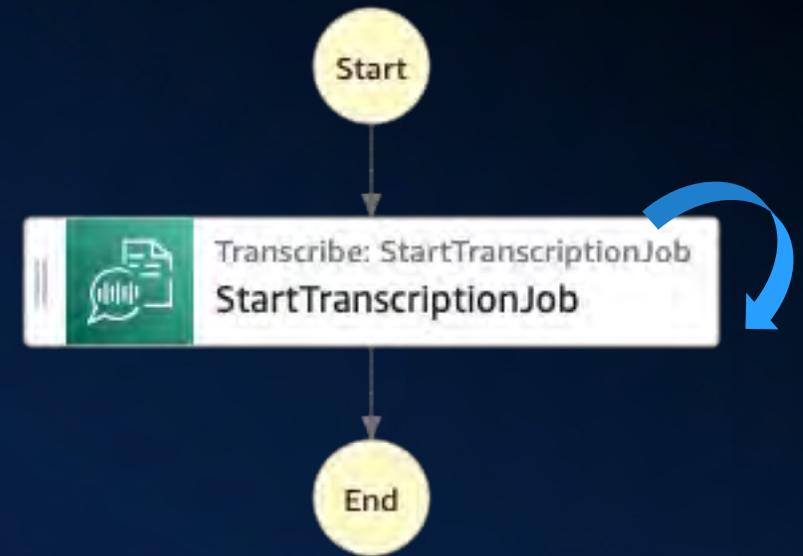


コールバック
(.waitForTaskToken)

タスクトークンが
返却されるまで待ちます

Machine Learning と生成 AI

Amazon Transcribe API と
直接統合してビデオをテキストに変換する



ServerlessVideo



<https://s12d.com/video>

サービス間の直接統合

```
app.js

const AWS = require('aws-sdk');
const docClient = new AWS.DynamoDB.DocumentClient();

var params = {
  "TableName": "reinvent2023",
  "Key": {
    "PK": {"S": "Wardrobe"},
    "SK": {"S": "shoes"}
  }
}

async function queryItems(){
  try {
    const data = await docClient.getItem(params).promise()
    return data
  } catch (err) {
    return err
  }
}

exports.handler = async (event, context) => {
  try {
    const data = await queryItems()
    return { body: JSON.stringify(data) }
  } catch (err) {
    return { error: err }
  }
}
```



AWS Lambda

クエリ



Amazon DynamoDB

サービス間の直接統合

単一タスクの「ワークフロー」であっても、組み込みのエラー処理、キャッチ、再試行、可観測性、カスタムコードの削減、ロギングが組み込まれます

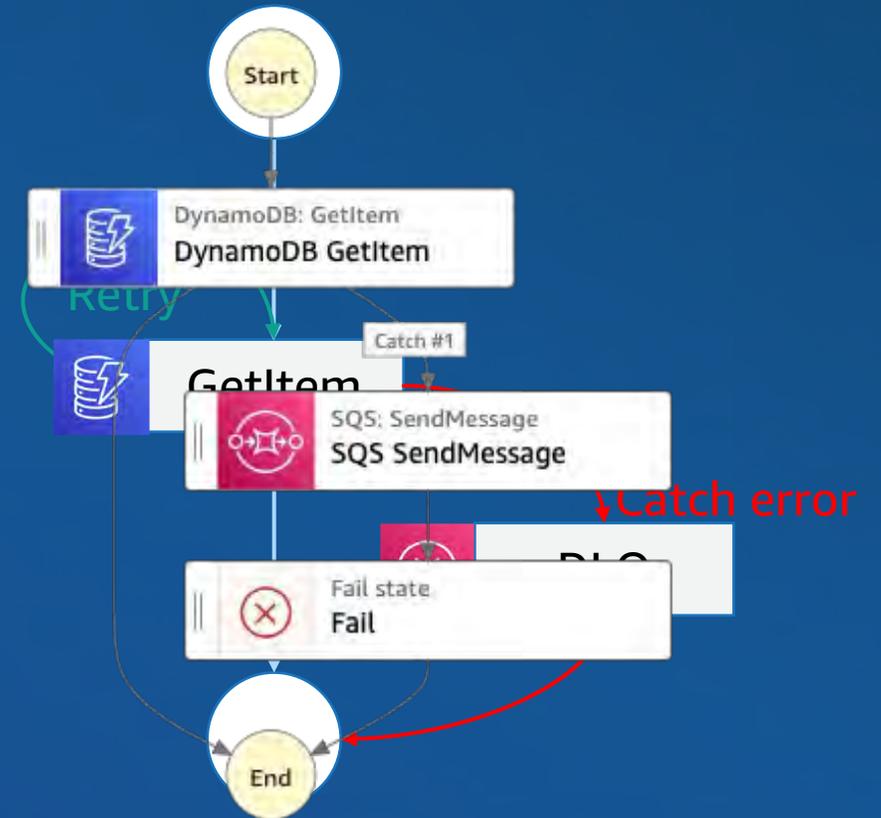
```
app.js

const AWS = require('aws-sdk');
const docClient = new AWS.DynamoDB.DocumentClient();

var params = {
  "TableName": "reinvent2023",
  "Key": {
    "PK": {"S": "Wardrobe"},
    "SK": {"S": "shoes"}
  }
}

async function queryItems(){
  try {
    const data = await docClient.getItem(params).promise()
    return data
  } catch (err) {
    return err
  }
}

exports.handler = async (event, context) => {
  try {
    const data = await queryItems()
    return { body: JSON.stringify(data) }
  } catch (err) {
    return { error: err }
  }
}
```

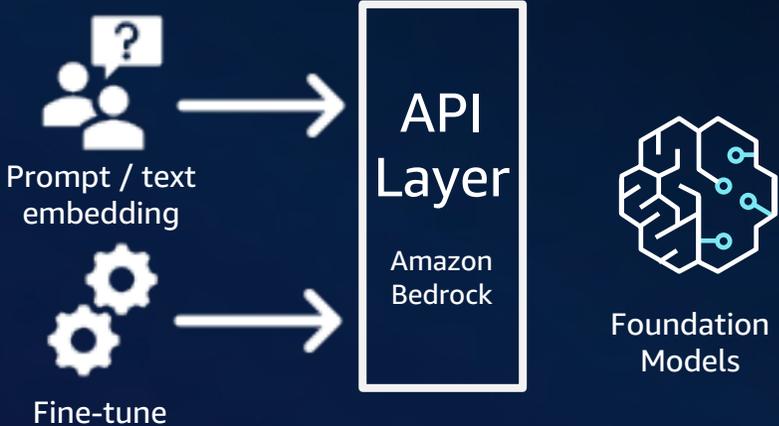


生成 AI アプリケーションを実装する

- 動画から複数のタイトル、説明を作成する
- 人間にフィードバックを依頼する
- ビデオ用のアバター画像を作成する



基盤モデルへのアクセス



Amazon Bedrock

Amazon Bedrock の基盤モデルを呼び出す

```
app.js
const AWS = require('aws-sdk');

prompt_data = get_data_from_S3(bucket, key)
request = json.dumps({
  'prompt': f'Human:{prompt_data}¥n¥nAssistant:',
  'max_tokens_to_sample': 1028,
  'temperature': 1,
  'top_k': 250,
  'top_p': 0.999,
  'stop_sequences': ['¥n¥nHuman:']
})
client = boto3.client('bedrock-runtime')
response = bedrock_client.invoke_model(
  modelId=event["ModelId"],
  body=json.dumps(event["Body"]),
)
body = json.loads(response["body"].read().decode("utf-8"))
response["body"] = body

return { body: JSON.stringify(data) }
} catch (err) {
return { error: err }
}
}
```

Introducing optimized integration for Amazon Bedrock

Amazon Bedrock との最適化された統合



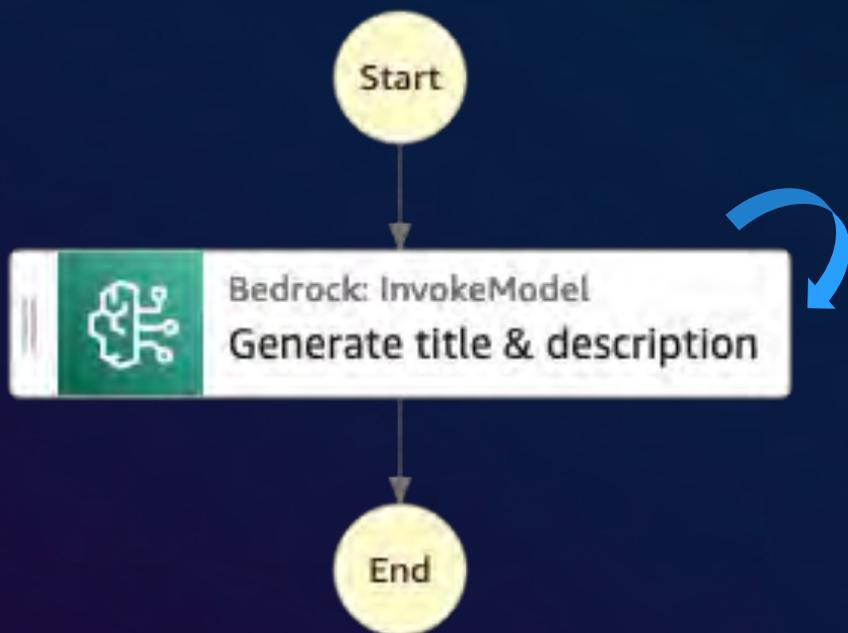
生成 AI アプリケーションの構築と拡張を シンプルにする 2 つの新しい最適化された統合

Prompt: "Write me a title..."



Create Model Customization Job
.sync

要件 1: タイトルと説明文の生成



Generate title & description Definition Test state >

Configuration | Input | Output | Error handling

State name
Generate title & description

API
Bedrock: InvokeModel

API Parameters

Foundation Models Provisioned Models

Bedrock model identifier
Determines which foundation model will be invoked
Enter model
arn:aws:bedrock:us-east-1::foundation-model/amazon.titan-tg1-large
Must be a valid model.

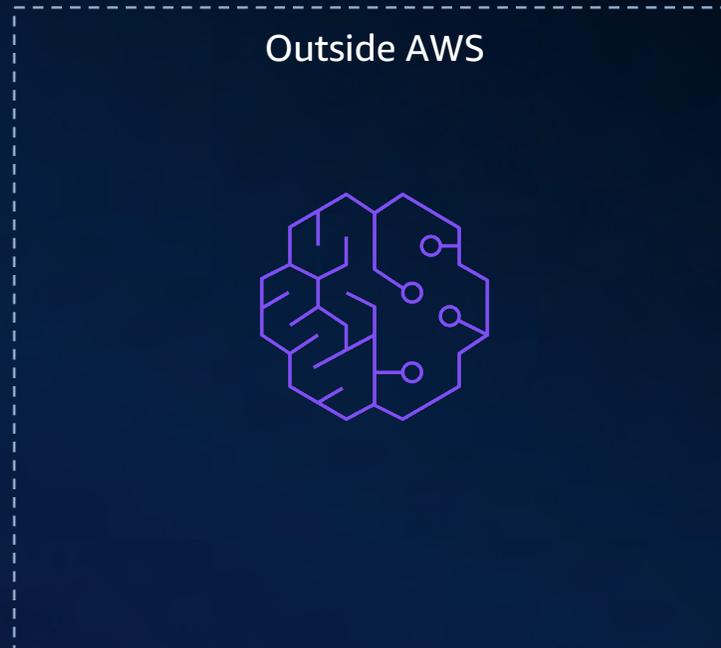
Bedrock Model Parameters
JSON object containing inference parameters for the selected Bedrock model. Contains sample values. Update the JSON with your own parameter values.
 Enter model inference parameters Load model inference parameters from S3

生成 AI アプリケーションを実装する

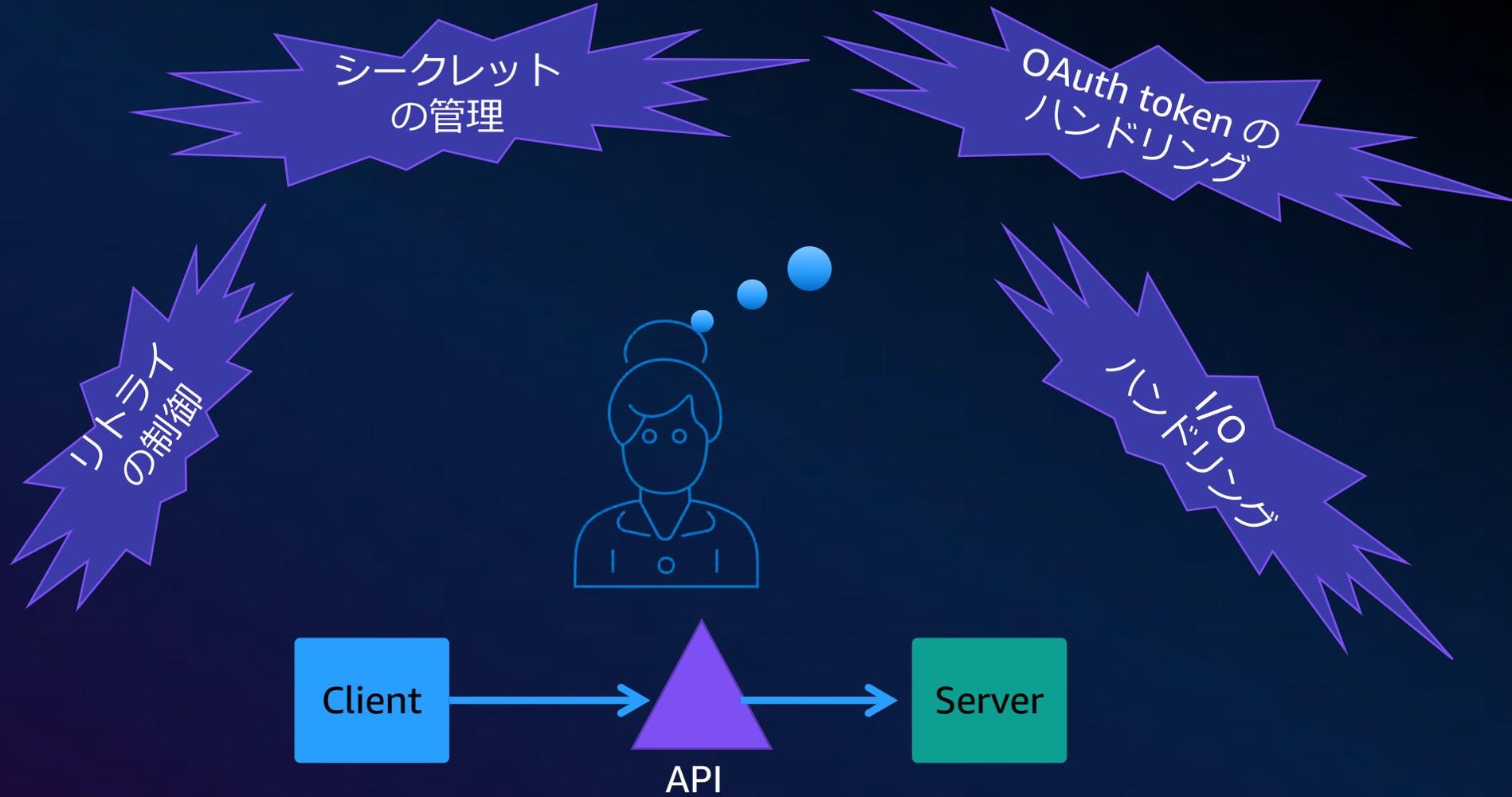
- 動画から**複数の**タイトル、説明を作成する
- 人間にフィードバックを依頼する
- ビデオ用のアバター画像を作成する



AWS 外にあるモデルへのアクセス



Public API へのアクセス



Public API へのアクセス



Undifferentiated Heavy Lifting

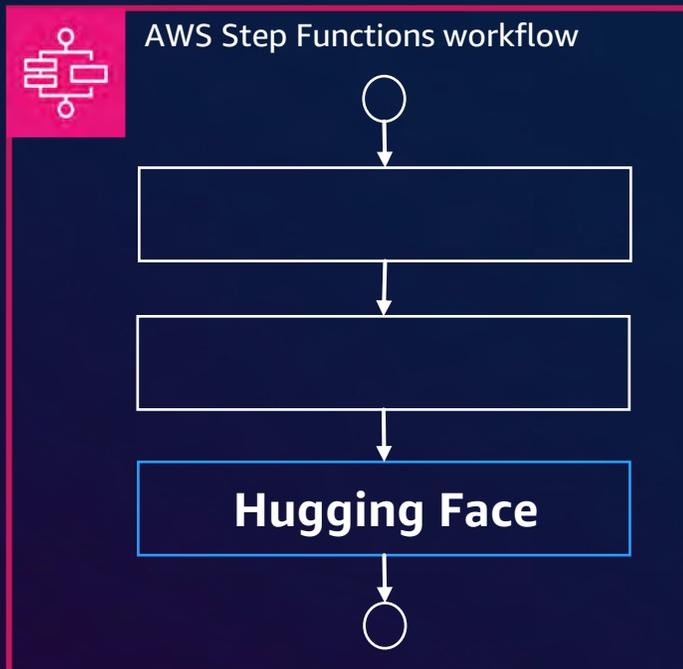
～ビジネスの差別化とならない重労働～

Public HTTPS API Integration

API Endpoint への直接統合



Public API へのアクセス – Step Functions



The screenshot shows the AWS Step Functions console interface for a workflow named "MyStateMachine-j9pem2g56". The interface includes a top navigation bar with "Design", "Code", and "Config" tabs, and a toolbar with "Undo", "Redo", "Zoom in", "Zoom out", "Center", "Duplicate", and "Delete" options. A search bar is located below the toolbar. The main content area is divided into three sections: "MOST POPULAR" (listing actions like AWS Lambda Invoke, Amazon SNS Publish, Amazon ECS RunTask, AWS Step Functions StartExecution, and AWS Glue StartJobRun), "THIRD-PARTY API" (listing HTTP Endpoint Call third-party API), and "COMPUTE" (listing Amazon Data Lifecycle Manager, Amazon EBS, Amazon EC2, and AWS EC2 Instance Connect). The central canvas displays a workflow diagram with a "Start" node, a dashed box labeled "Drag first state here", and an "End" node. The right sidebar shows the "Workflow" definition editor, including a "Comment" field and a "TimeoutSeconds" field set to 600.

Public API へのアクセス – Step Functions



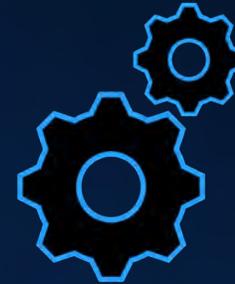
エラーハンドリング

HTTP Status Code で
ハンドリング



認証／認可

OAuth, Basic,
API キー認証



データ加工

URLエンコーディング
Request Body の加工



テストステート

ステップを独立してテストする
ビジネスロジックを検証する

要件 1: 複数のタイトル、説明文の生成

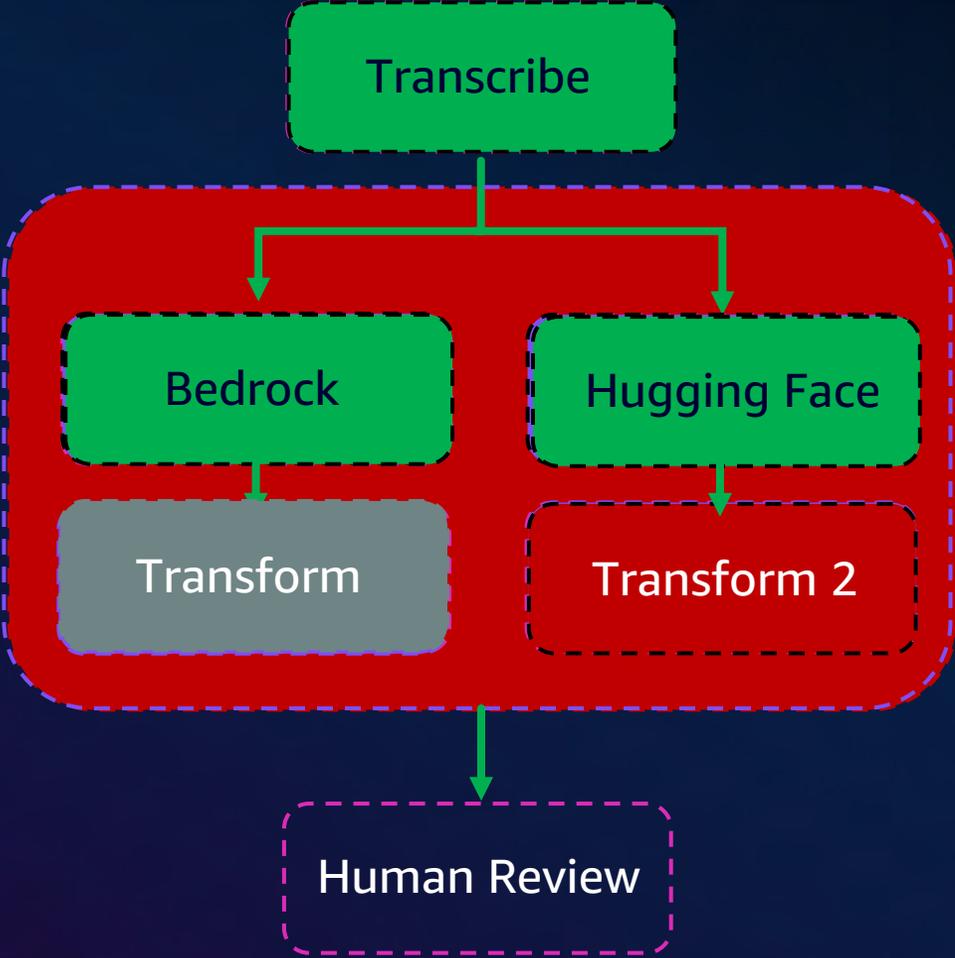
The screenshot displays the AWS Step Functions console interface. The main area shows a workflow diagram with a 'Start' node leading to a 'Parallel state' box. This box contains two parallel tasks: 'Bedrock: InvokeModel Generate title & description' and 'HTTP Endpoint Hugging Face'. Both tasks lead to an 'End' node.

The right-hand panel is the configuration for the 'Hugging Face' action. It includes the following fields and options:

- State name:** Hugging Face
- State type:** Task: HTTP endpoint
- API Parameters:** Edit as JSON (checked), API endpoint: `https://api-inference.huggingface.co/models/google/tapas-base-finetun`
- Method:** GET
- Authentication:** Enter connection ARN

The left-hand sidebar shows a search bar and a list of actions under categories: MOST POPULAR (AWS Lambda Invoke, Amazon SNS Publish, Amazon ECS RunTask, AWS Step Functions StartExecution, AWS Glue StartJobRun), THIRD-PARTY API (HTTP Endpoint Call third-party API), and COMPUTE (Amazon Data Lifecycle Manager).

失敗した時点からどうリカバリするのか

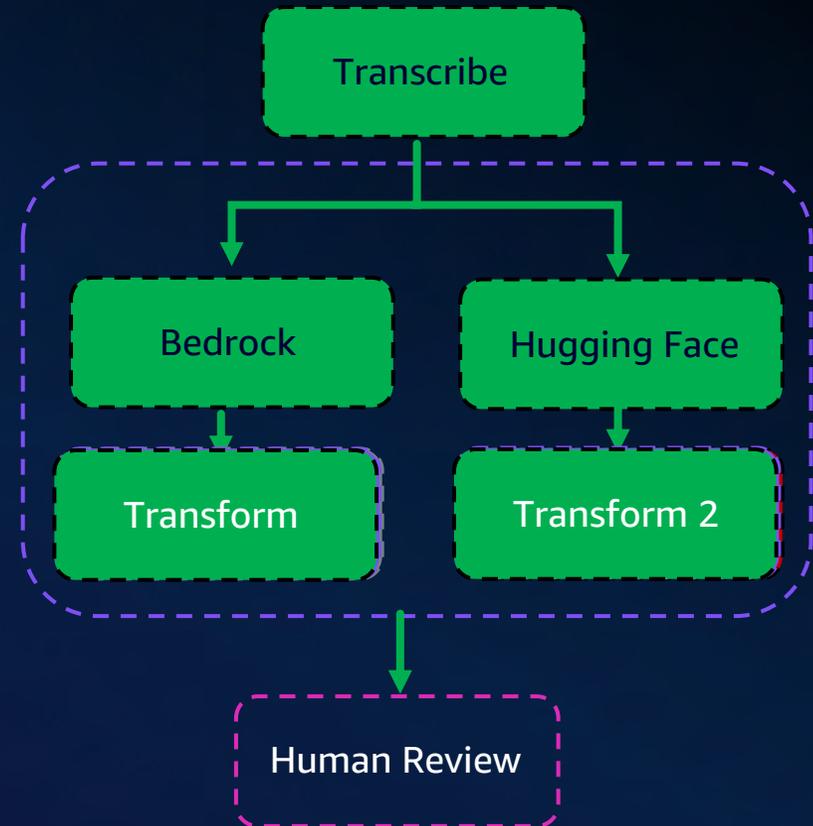
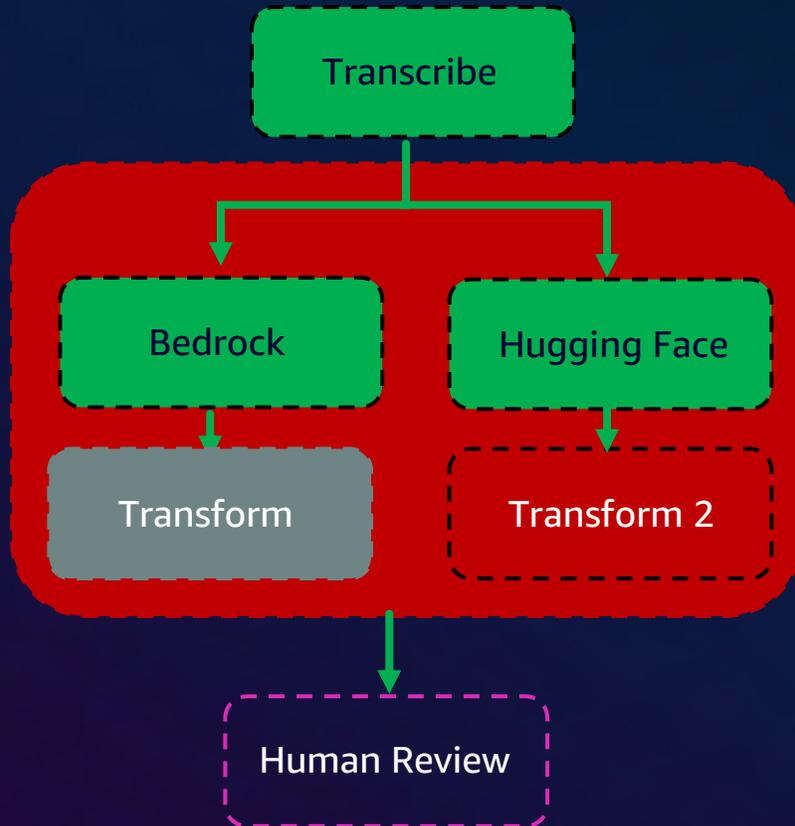


Redrive from failure

失敗した時点からの再実行

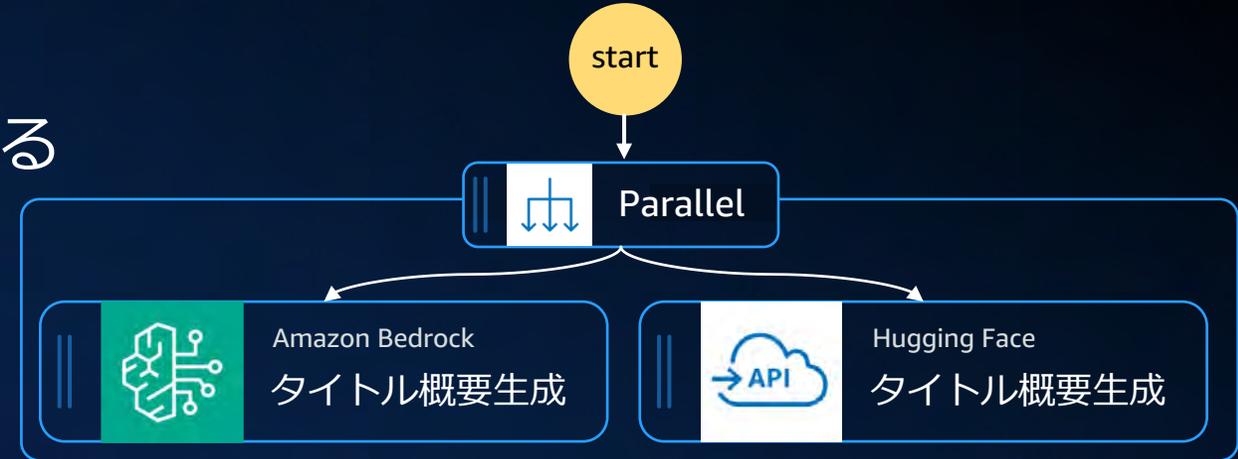


Redrive from failure – 失敗した時点からの再実行



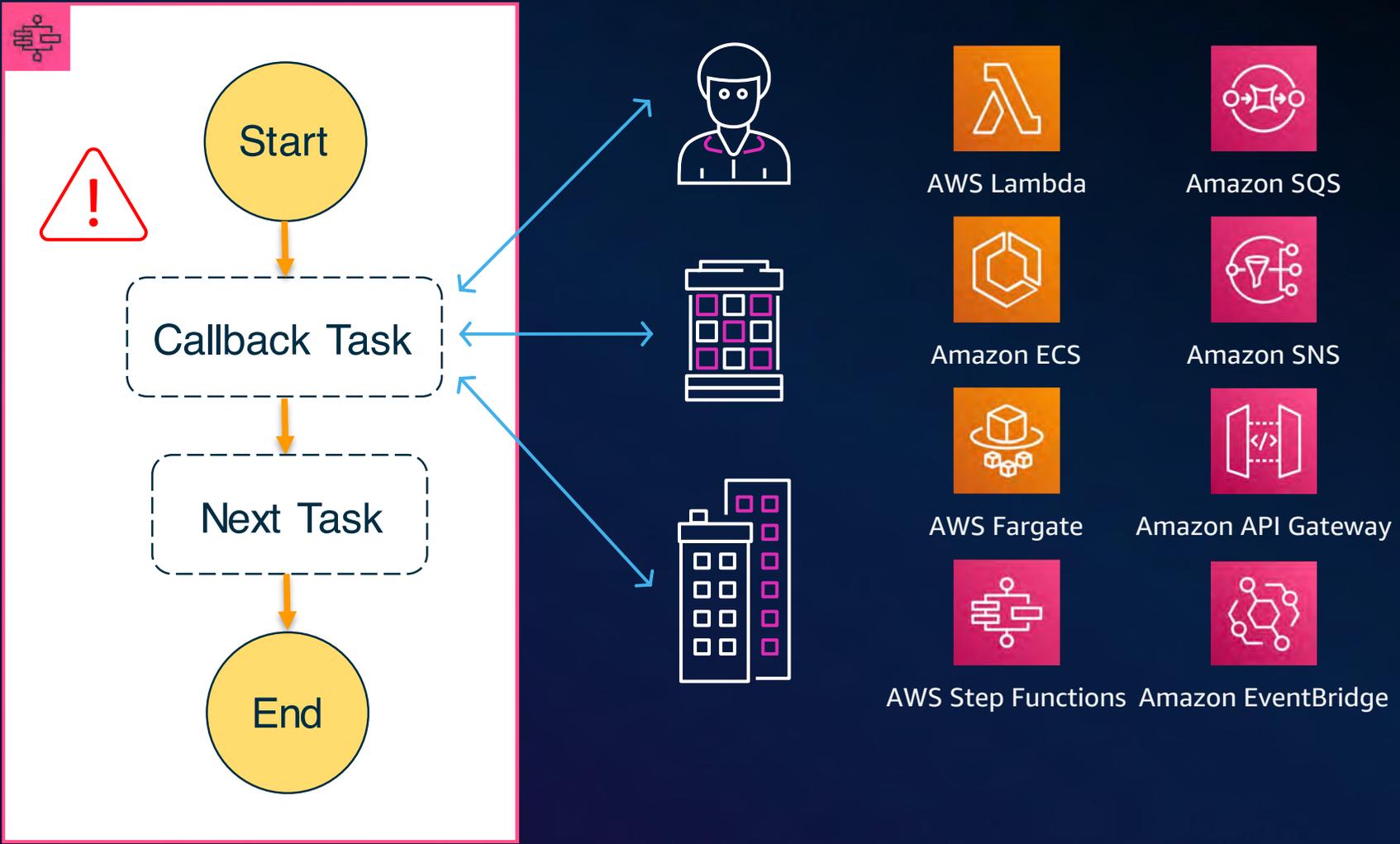
生成 AI アプリケーションを実装する

- 動画から複数のタイトル、説明を作成する
- 人間にフィードバックを依頼する
- ビデオ用のアバター画像を作成する



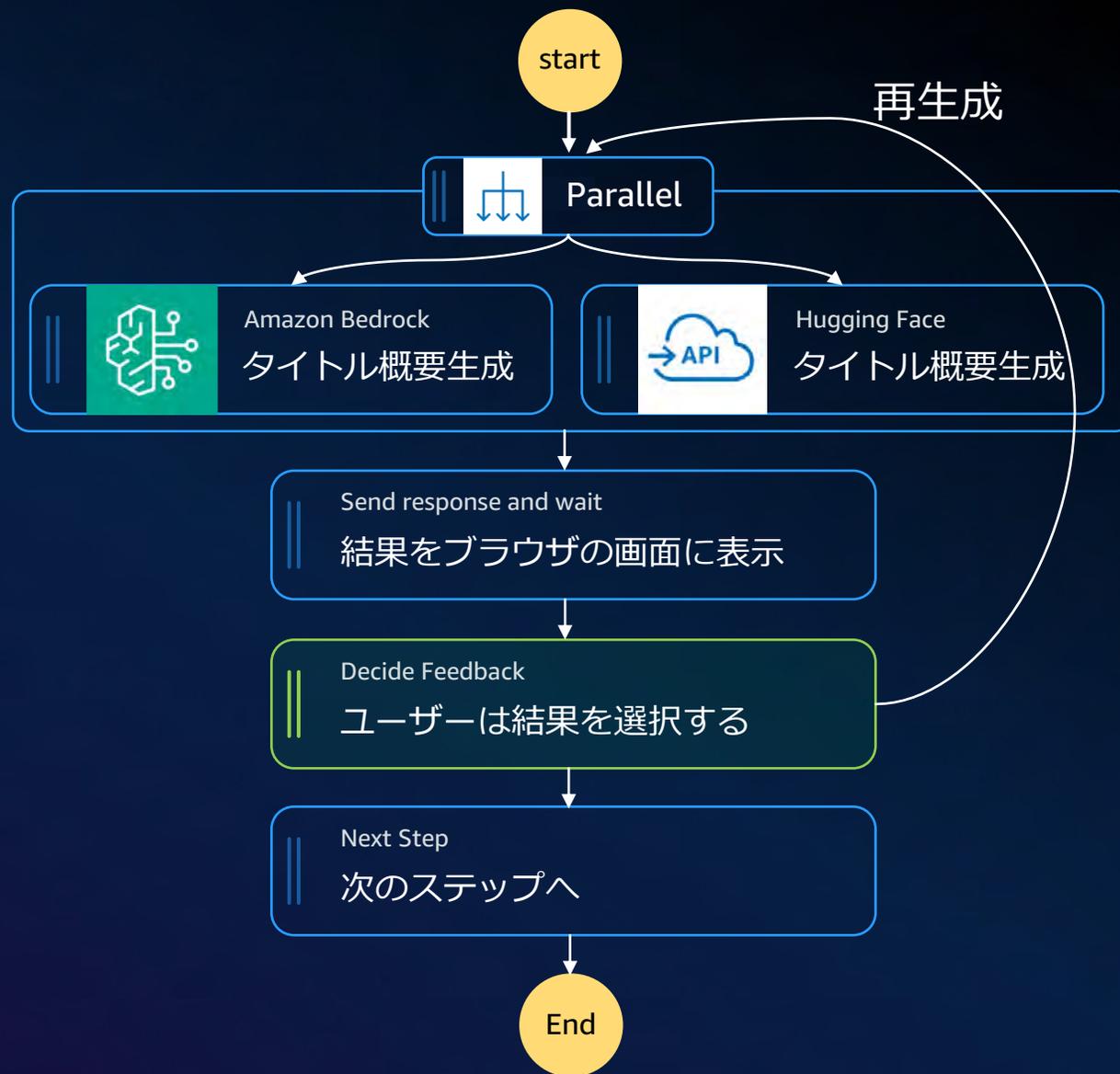
Callback pattern

人や外部システムにおける処理が介在する場合、トークンを用いてタスクに待機状態を実現する



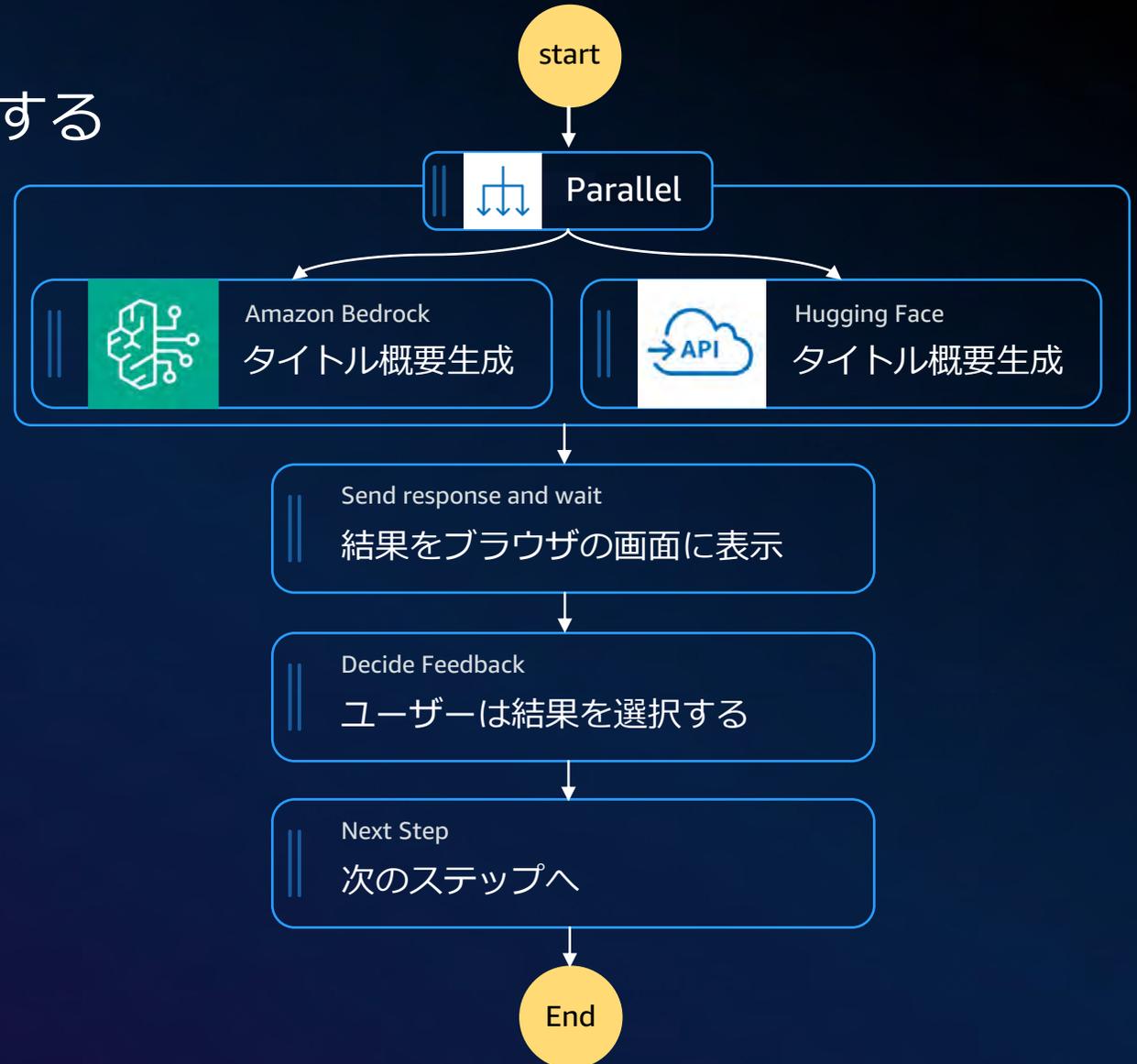
要件 2: 人間によるフィードバックを入れる

- フィードバックループ
- 非同期チャネル
- メールや WebSocket
- レスポンスを返す API
- Choice State を使用した分岐
- 再生成もループ処理で

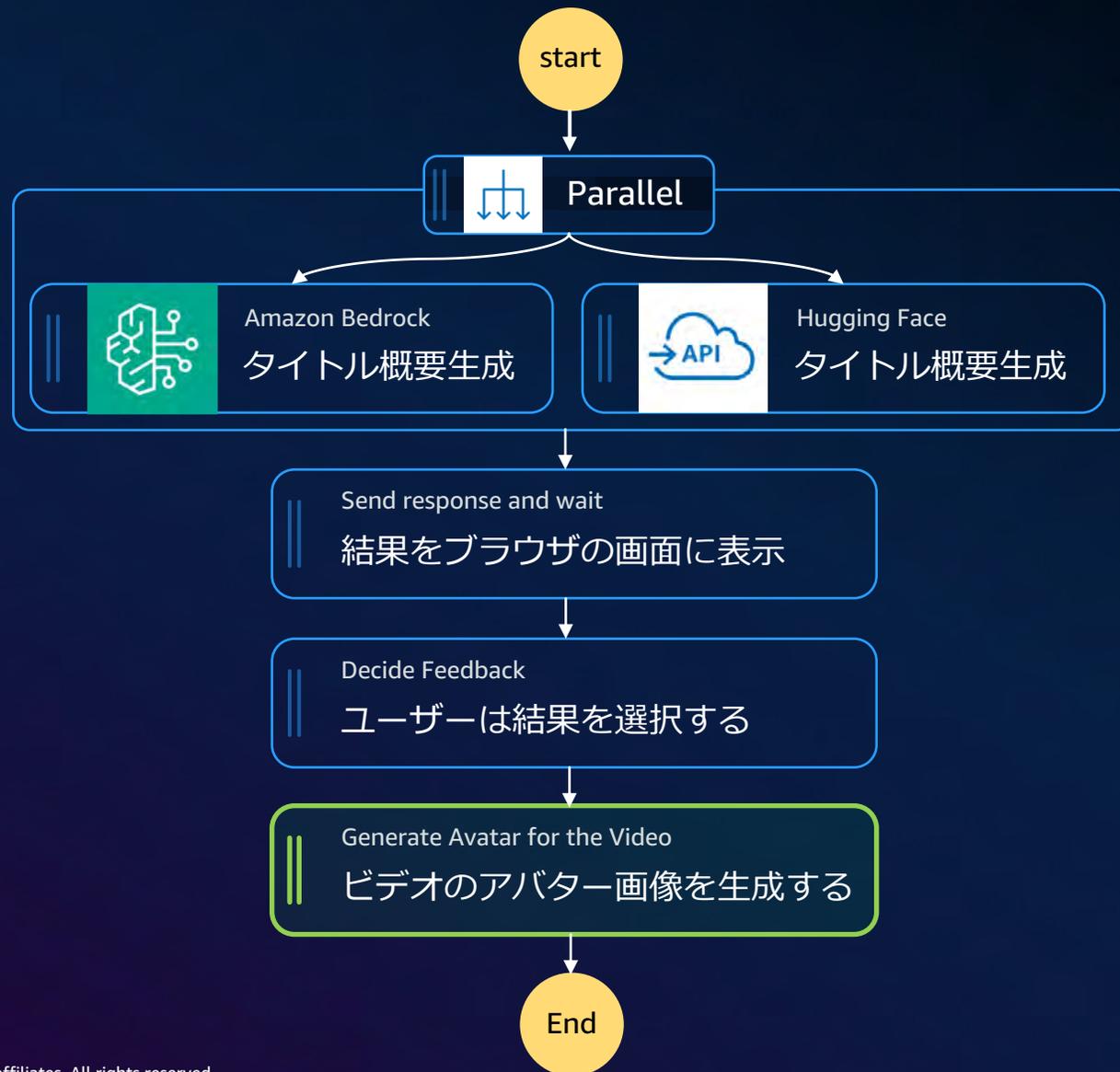


生成 AI アプリケーションを実装する

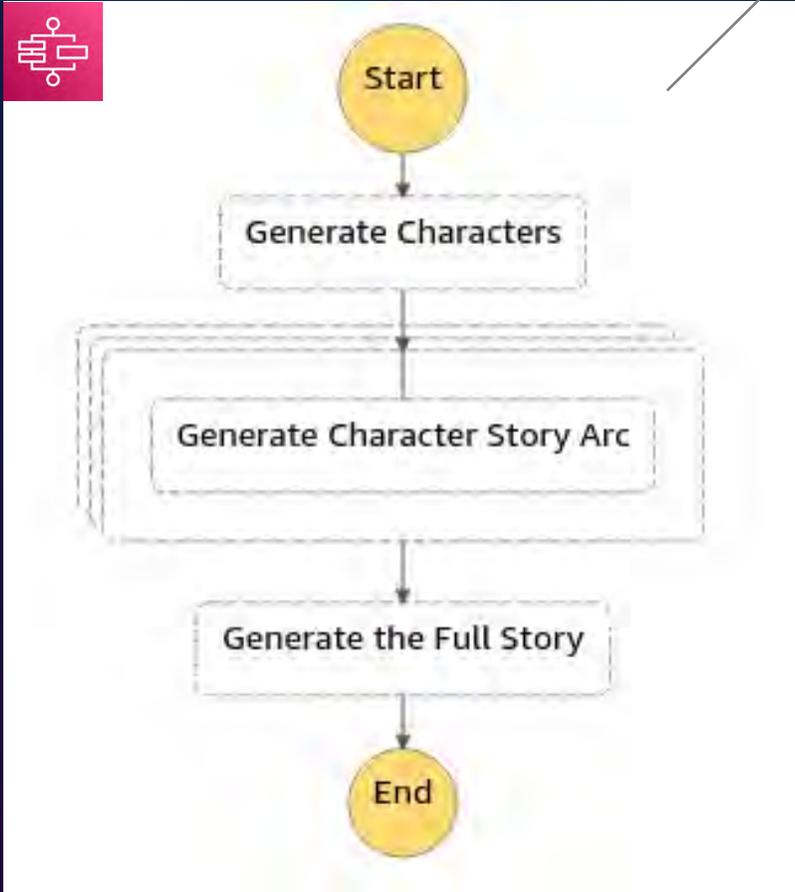
- 動画から複数のタイトル、説明を作成する
- 人間にフィードバックを依頼する
- **ビデオ用のアバター画像を作成する**



要件 3: アバター画像の生成



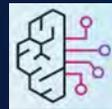
Prompt chaining



あなたは受賞歴のあるフィクション作家で、{story_description}に関する新しい物語を執筆しています。

物語を書く前に、物語に登場する5人のキャラクターについて説明します。応答は JSON 配列としてフォーマットする必要があります。配列内の各要素には、キャラクターの名前を表す「name」キーと、キャラクターの説明を含む「description」キーが含まれます。

有効な応答の例を以下に示します。



Amazon Bedrock
ANTHROPIC Claude

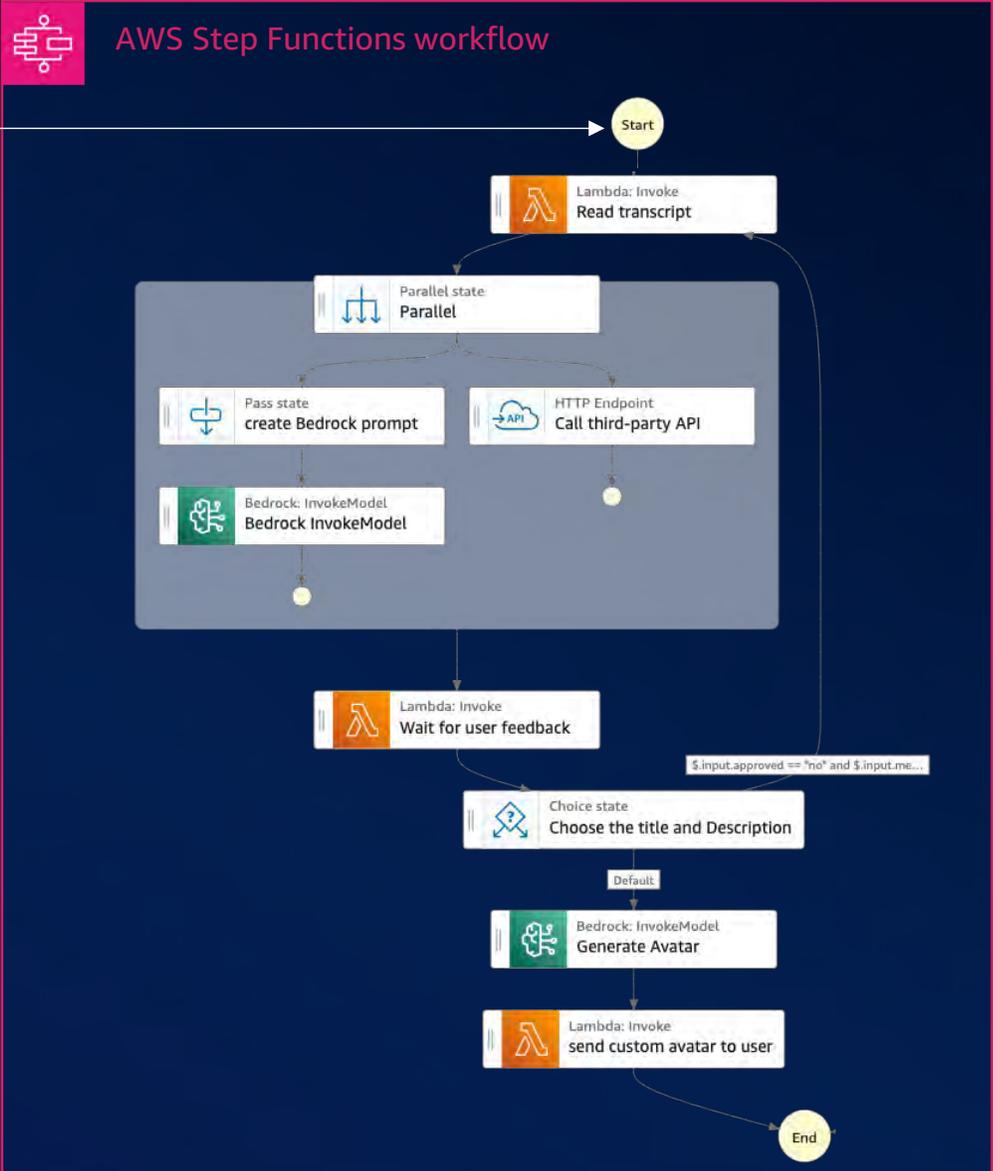
Prompt chaining

- 複数のプロンプトを接続し複雑なコンテンツを生成する
- あるモデルから次のモデルに応答をフィードする

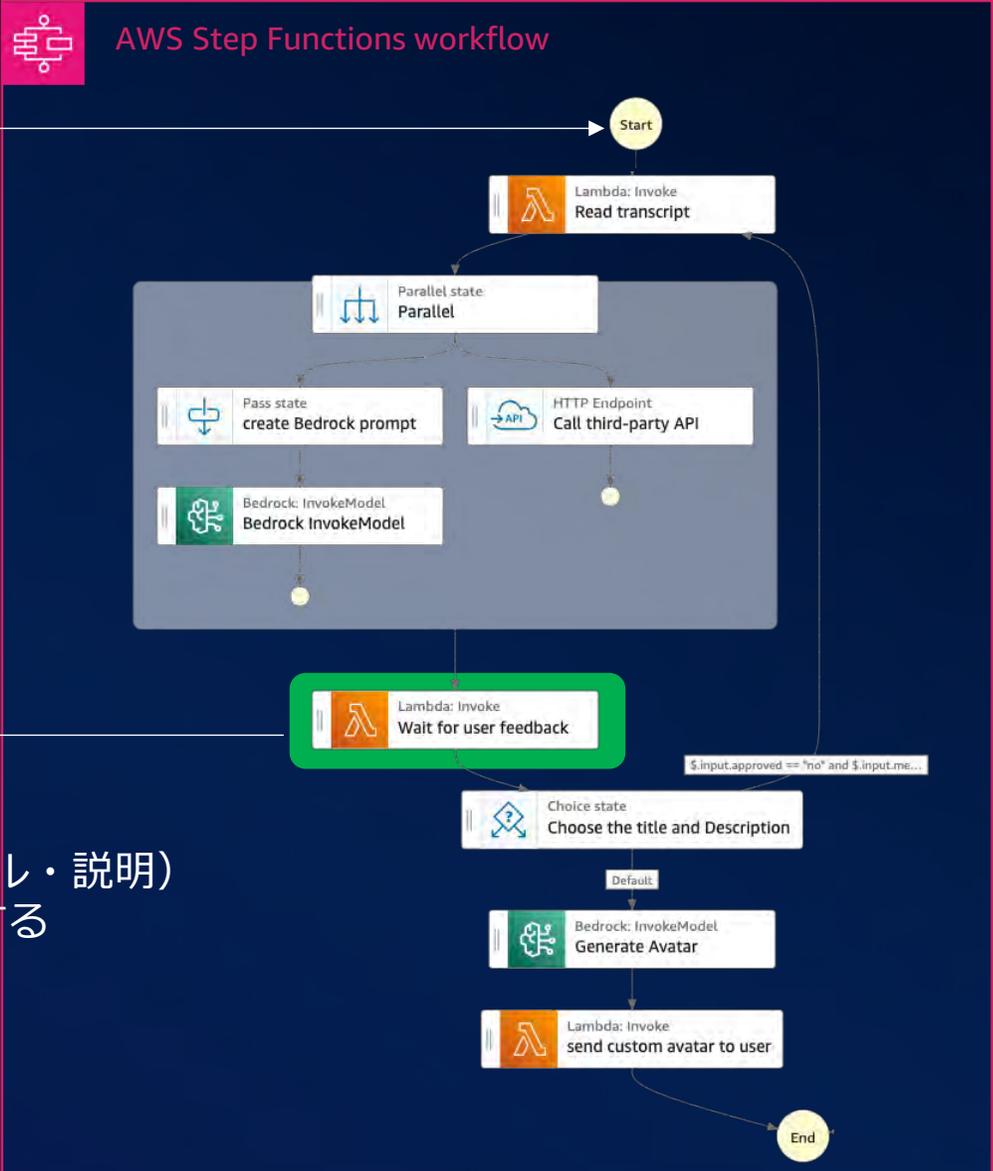
Use cases

- ブログや記事を書く
- 応答の検証
- 会話型 LLM

Architecture



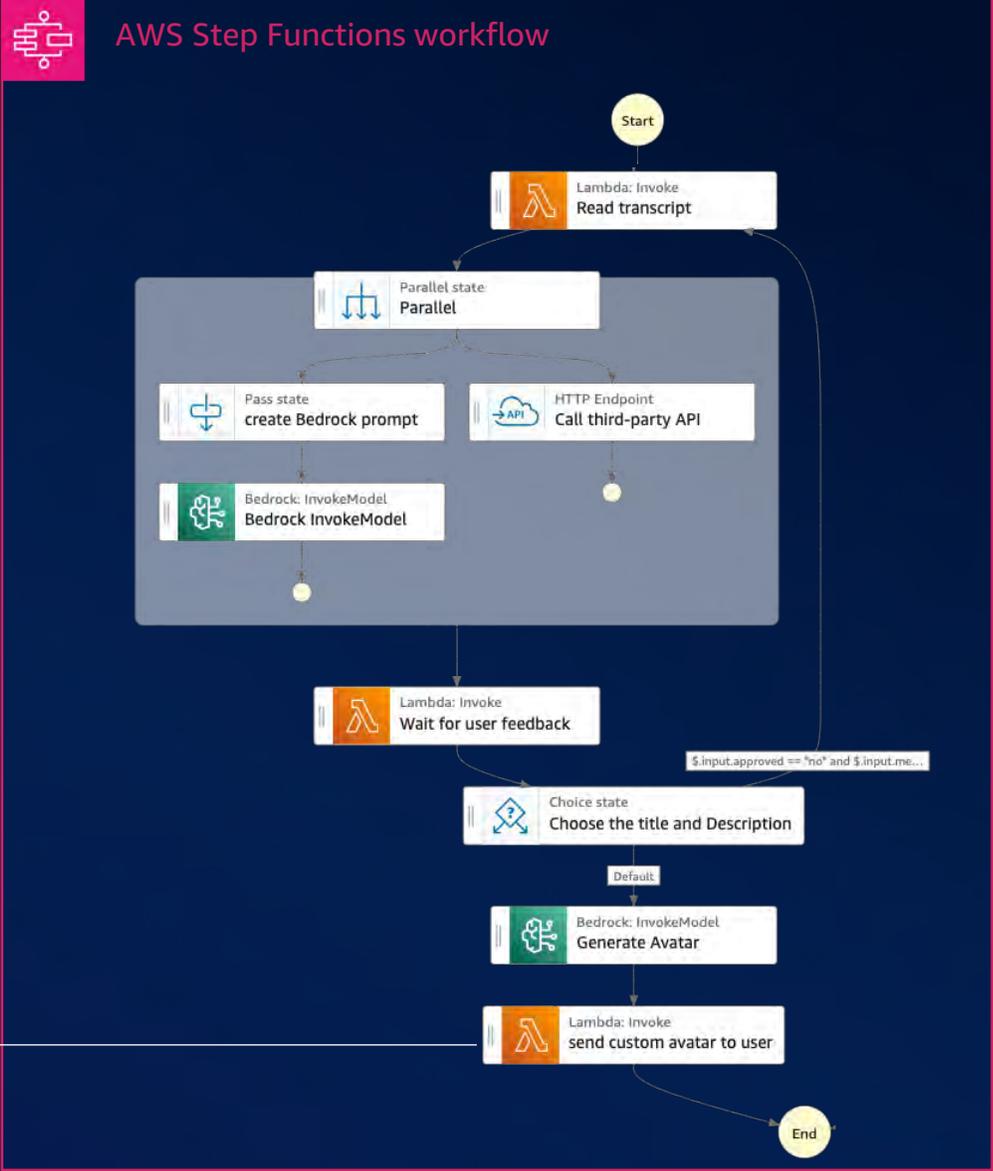
Architecture



Architecture



5 アバター画像の署名付きURL



まとめ

より効果的に、より賢く使いこなすために

- サービスフルなサーバーレスによって Lambda の処理を移譲し、クラウドの中でアプリケーションを組み合わせていく
- オーケストレーションとコレオグラフィを組み合わせ
自律的で、拡張容易性を高めるサーバーレスアーキテクチャ
- 生成系AI アプリケーションのために
AWS Step Functions をより効果的に、より賢く使いこなす

Thank you!

Daisuke Awaji

Solutions Architect

