Multi-Cloud Auto-Deploy Platform

Complete Documentation

PLAYER1-r7

2026-02-15

目次

Chapter 1

Project Overview

図 1.1: Multi-Cloud

Deploy to AWS passing Deploy to Azure passing Deploy to GCP passing

マルチクラウド対応の自動デプロイシステム - AWS/Azure/GCP 対応のフルスタックアプリケーション自動デプロイプラットフォーム

Cloud Provider	API Endpoint	Frontend (CDN)	Direct Storage
AWS (ap-northeast-1)	API Gateway	CloudFront [OK]	S3 Static
Azure (japaneast)	Functions	Front Door [NEW]	Blob Storage
GCP (asia-northeast1)	Cloud Run	Cloud CDN [NEW]	Cloud Storage

Cloud Provider	CDN URL	Distribution ID	管理方法
AWS	CloudFront	E1TBH4R432SZBZ	Pulumi
Azure	Front Door	mcad-staging-d45ihd	Pulumi
GCP	Cloud CDN	34.117.111.182	Pulumi

[WEB] **全クラウドで CDN 配信を実装！** CloudFront, Front Door, Cloud CDN による高速・安全なコンテンツ配信

[TOOLS] **Infrastructure as Code**: Pulumi で全 CDN リソースを管理 (詳細: CDN セットアップガイド)

[LIST] 詳細情報: エンドポイント一覧

- **マルチクラウド対応**: AWS、Azure、GCP に対応
- **フルスタック**: フロントエンド、バックエンド、データベースの完全なスタック
- **自動デプロイ**: GitHub Actions による完全自動化
- **IaC 統合**: Terraform と Pulumi の両方に対応 [NEW]
- **完全 Python 版**: Pulumi + FastAPI + Reflex による統一スタック [NEW]
- **CI/CD**: プッシュや PR で自動的にビルド・デプロイ
- **簡単セットアップ**: スクリプト一つで環境構築

```
multicloud-auto-deploy/
├── .github/workflows/     # GitHub Actionsワークフロー
├── infrastructure/        # インフラストラクチャコード
│   ├── terraform/        # Terraformコード（AWS/Azure/GCP）
│   └── pulumi/           # Pulumiコード（Python - AWS/Azure/GCP）
├── services/             # アプリケーションコード
│   ├── api/              # FastAPI バックエンド（Python 3.12）
│   ├── frontend_react/   # React フロントエンド（Vite + TypeScript）
│   └── frontend_reflex/  # Reflex フロントエンド（実験的）
├── scripts/              # デプロイ・テストスクリプト
├── docs/                 # ドキュメント
│   ├── CDN_SETUP.md     # [NEW] CDN設定ガイド
│   └── ENDPOINTS.md     # エンドポイント一覧
└── static-site/          # 静的サイト（環境選択画面）
```

- Python 3.12+
- Docker & Docker Compose
- Pulumi 3.0+ または Terraform 1.5+
- AWS CLI 2.x / Azure CLI 2.x / gcloud CLI 556.0+
- GitHub アカウント

Frontend

- **Framework**: React 18+ (Vite)
- **Hosting**: Static Site (S3 / Azure Blob / Cloud Storage)
- **CDN**: CloudFront / Azure Front Door / Cloud CDN
- **Build**: Vite 7.3+, TypeScript

Backend

- **Framework**: FastAPI 1.0+ (Python 3.12)
- **AWS**: Lambda (x86_64) + API Gateway v2 (HTTP)
- **Azure**: Azure Functions (Python)
- **GCP**: Cloud Run (Docker)

Database

- **AWS**: DynamoDB (PAY_PER_REQUEST)
- **Azure**: Cosmos DB (Serverless)
- **GCP**: Firestore (Native Mode)

Infrastructure

- **IaC**: Terraform 1.14+ / Pulumi 3.0+
  - Terraform: 手動構築環境（`infrastructure/terraform/`）
  - Pulumi: Infrastructure as Code 管理（`infrastructure/pulumi/`）
    - AWS: CloudFront + Lambda + API Gateway
    - Azure: Front Door + Functions + Cosmos DB
    - GCP: Cloud CDN + Cloud Run + Firestore
- **CI/CD**: GitHub Actions
- **CDN**: CloudFront / Azure Front Door / Cloud CDN

CI/CD

- GitHub Actions
- Automated builds and deployments
- Docker-based deployments
- S3-based Lambda deployment

1. **リポジトリをクローン**

```
git clone https://github.com/PLAYER1-r7/multicloud-auto-deploy.git
cd multicloud-auto-deploy
```

2. **ローカル開発環境起動**

```
docker-compose up -d api

cd services/frontend_react
npm install
npm run dev
```

3. **Pulumi でデプロイ**

```
cd infrastructure/pulumi/aws/simple-sns
pip install -r requirements.txt
pulumi stack init staging
pulumi config set aws:region ap-northeast-1
pulumi up
```

[DOCS] 詳細な移行ガイドは docs/PYTHON_MIGRATION.md を参照

1. **リポジトリをクローン**

```
git clone https://github.com/PLAYER1-r7/multicloud-auto-deploy.git
cd multicloud-auto-deploy
```

2. **環境変数を設定**

```
cp .env.example .env
```

3. **クラウドプロバイダー別デプロイ**

```
./scripts/deploy-aws.sh
```

```
./scripts/deploy-azure.sh
```

```
./scripts/deploy-gcp.sh
```

- [BOOK] セットアップガイド - 初期セットアップ手順
- [DEPLOY] CI/CD 設定 - GitHub Actions 自動デプロイ設定
- [OK] CI/CD テスト結果 - パイプライン検証レポート
- [CHORE] トラブルシューティング - よくある問題と解決策
- [WEB] エンドポイント一覧 - 全環境のエンドポイント情報
- [EARTH] CDN セットアップガイド - CloudFront/Front Door/Cloud CDN 設定 [NEW]
- [NOTE] クイックリファレンス - よく使うコマンド集
- AWS デプロイ
- Azure デプロイ
- GCP デプロイ
- システムアーキテクチャ - 完全版システム設計

プッシュや PR で自動的にビルド・デプロイが実行されます：

- `main`ブランチへのプッシュ → ステージング環境へ自動デプロイ
- PR の作成/更新 → ビルド検証
- 手動トリガー → 任意の環境へデプロイ

ワークフロー	トリガー	デプロイ先	説明
deploy-multicloud.yml	`main`への push / 手動	Azure + GCP	Container Apps/Cloud Run への統合デプロイ [NEW]
deploy-aws.yml	`main`への push / 手動	AWS Lambda	Lambda 関数の更新
deploy-azure.yml	`main`への push / 手動	Azure	Terraform 使用
deploy-gcp.yml	`main`への push / 手動	GCP	Terraform 使用

1. **Build Images**:
   - API と Frontend の Docker イメージをビルド（linux/amd64）
   - Azure ACR と GCP Artifact Registry にプッシュ
2. **Deploy Azure** (並列実行):
   - Container Apps（API + Frontend）を更新
3. **Deploy GCP** (並列実行):
   - Cloud Run（API + Frontend）を更新
4. **Health Check**:
   - デプロイされた API のヘルスチェック

以下のシークレットを設定してください（詳細は CI/CD 設定ガイド 参照）：

Azure Container Apps [NEW]

- `AZURE_CREDENTIALS` - Service Principal 認証情報
- `AZURE_CONTAINER_REGISTRY` - ACR ログインサーバー
- `AZURE_CONTAINER_REGISTRY_USERNAME/PASSWORD` - ACR 認証情報
- `AZURE_RESOURCE_GROUP` - リソースグループ名
- `AZURE_CONTAINER_APP_API` - API の Container App 名
- `AZURE_CONTAINER_APP_FRONTEND` - Frontend の Container App 名

GCP Cloud Run [NEW]

- `GCP_CREDENTIALS` - サービスアカウントキー（JSON）
- `GCP_PROJECT_ID` - プロジェクト ID
- `GCP_ARTIFACT_REGISTRY_REPO` - Artifact Registry リポジトリ名
- `GCP_CLOUD_RUN_API` - API の Cloud Run サービス名
- `GCP_CLOUD_RUN_FRONTEND` - Frontend の Cloud Run サービス名

AWS Lambda

- `AWS_ACCESS_KEY_ID`
- `AWS_SECRET_ACCESS_KEY`

最新のデプロイ状況はGitHub Actionsで確認できます。

GitHub Actions ページから手動でワークフローを実行：

```
Actions > Deploy to Multi-Cloud > Run workflow

- environment: staging / production
- deploy_target: all / azure / gcp
```

- **Frontend**: S3 + CloudFront (CDN)
- **Backend**: Lambda (Python 3.12) + API Gateway v2
- **Database**: DynamoDB (PAY_PER_REQUEST)
- **Infrastructure**: Terraform 1.14.5
- **Deployment**: GitHub Actions
- **CDN**: CloudFront Distribution (E2GDU7Y7UGDV3S)
- **Frontend**: Blob Storage ($web) + Azure Front Door
- **Backend**: Azure Functions (Python 3.12)
- **Database**: Cosmos DB (Serverless)
- **Infrastructure**: Terraform / Azure CLI
- **Deployment**: GitHub Actions
- **CDN**: Azure Front Door (Standard)
- **Frontend**: Cloud Storage + Cloud CDN (Load Balancer)
- **Backend**: Cloud Run (FastAPI, Docker)
- **Database**: Firestore (Native Mode)
- **Infrastructure**: gcloud CLI / Terraform
- **Deployment**: GitHub Actions (Artifact Registry)
- **CDN**: Cloud CDN (Global HTTP Load Balancer)

プロジェクトには以下の便利なスクリプトが含まれています：

```
./scripts/test-endpoints.sh

./scripts/test-e2e.sh

./scripts/setup-github-secrets.sh

./scripts/import-gcp-resources.sh
```

```

./scripts/diagnostics.sh

./scripts/deploy-aws.sh
./scripts/deploy-azure.sh
./scripts/deploy-gcp.sh
```

```
make install          # 依存関係をインストール
make build-frontend   # フロントエンドをビルド
make build-backend    # Lambda パッケージを作成
make test-all         # 全クラウドのデプロイメントをテスト
make deploy-aws       # AWSへデプロイ
make terraform-init   # Terraform初期化
make terraform-apply # Terraformリソースを適用
make clean            # ビルド成果物を削除
```

VS Code の Dev Container に対応しています：

```
- Terraform 1.7.5
- Node.js 18
- Python 3.12
- AWS CLI, Azure CLI, gcloud CLI
- Docker in Docker

tf              # terraform
deploy-aws      # AWS環境にデプロイ
deploy-azure    # Azure環境にデプロイ
deploy-gcp      # GCP環境にデプロイ
test-all        # 全エンドポイントテスト
```

システムの健全性をチェック：

```
./scripts/diagnostics.sh
```

- [OK] インストール済みツールの確認
- [OK] クラウドプロバイダー認証状態
- [OK] デプロイメントエンドポイントのテスト
- [OK] Terraform リソース状態の確認

```
./scripts/test-endpoints.sh

curl https://52z731x570.execute-api.ap-northeast-1.amazonaws.com/
curl https://mcad-staging-api--0000004.livelycoast-fa9d3350.japaneast.azurecontainerapps.io/
curl https://mcad-staging-api-son5b3ml7a-an.a.run.app/
```

```
cd services/frontend
npm install
npm run dev

```

```
cd services/backend
pip install -r requirements.txt
uvicorn src.main:app --reload
```

Lambda + API Gateway の完全自動デプロイメント:

```
cd scripts
./deploy-lambda-aws.sh

PROJECT_NAME=myproject ENVIRONMENT=production ./deploy-lambda-aws.sh
```

スクリプトは以下を自動実行します: - 依存関係のインストール（manylinux2014_x86_64）- ZIP パッケージ作成と S3 アップロード - Lambda 関数の作成/更新 - API Gateway 統合設定 - Lambda 権限設定（HTTP API 用の正しい SourceArn）- CloudWatch Logs アクセスログ有効化

エンドポイントの完全な CRUD テスト:

```
./scripts/test-api.sh -e https://YOUR_API_ID.execute-api.ap-northeast-1.amazonaws.com

./scripts/test-api.sh -e https://YOUR_API_ID.execute-api.ap-northeast-1.amazonaws.com --verbose
```

テスト項目: - ヘルスチェック - メッセージ CRUD 操作（作成、取得、更新、削除）- ページネーション - エラーハンドリング - バリデーション

全環境（AWS/GCP/Azure）のエンドツーエンド CRUD 動作を検証:

```
./scripts/test-e2e.sh
```

テストカバレッジ:

- **Total**: 18 テスト（3 環境 × 6 テスト）
- **Health Checks**: 各環境のヘルスエンドポイント検証
- **CRUD Operations**:
  - [OK] Create: メッセージ作成
  - [OK] List: 全メッセージ取得
  - [OK] Get: 特定メッセージ取得
  - [OK] Update: メッセージ更新
  - [OK] Delete: メッセージ削除

クラウド固有のパス処理:

- AWS/GCP: `/api/messages/`
- Azure: `/api/HttpTrigger/api/messages/`（Flex Consumption 対応）

期待される出力例:

```
═══════════════════════════════════════════════════
        Multi-Cloud E2E Test Suite
═══════════════════════════════════════════════════
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
  Testing: AWS
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━
[OK] Health check returned 'ok'
[OK] Create message (ID: abc123...)
[OK] List messages (found 5)
[OK] Get message by ID
[OK] Update message
[OK] Delete message

[GCP/Azure: 同様]

═══════════════════════════════════════════════════════════
        Test Summary
═══════════════════════════════════════════════════════════
Total Tests:  18
Passed:       18
All tests passed! [OK]
```

データ永続性検証:

- AWS: DynamoDB (PAY_PER_REQUEST)
- GCP: Firestore (Native Mode)
- Azure: Cosmos DB (Serverless)

包括的な監視とアラートを自動設定:

```
./scripts/setup-monitoring.sh

ALERT_EMAIL=your@email.com ./scripts/setup-monitoring.sh
```

設定内容: - SNS トピックとメール通知 - Lambda エラー/スロットリング/実行時間/同時実行数アラーム - API Gateway 5XX エラー/レイテンシアラーム - DynamoDB スロットリングアラーム - CloudWatch Logs メトリクスフィルター - CloudWatch ダッシュボード自動作成

本番運用のために以下のサービス追加を推奨します:

Lambda 関数のトレーシング有効化:

```
aws lambda update-function-configuration \
  --function-name YOUR_FUNCTION_NAME \
  --tracing-config Mode=Active
```

FastAPI に X-Ray 統合:

```
aws-xray-sdk==2.12.0

from aws_xray_sdk.core import xray_recorder
from aws_xray_sdk.ext.fastapi.middleware import XRayMiddleware

```

```
app.add_middleware(XRayMiddleware, recorder=xray_recorder)
```

API Gateway への攻撃防御:

```
aws wafv2 create-web-acl \
  --name multicloud-auto-deploy-waf \
  --scope REGIONAL \
  --default-action Allow={} \
  --rules file://waf-rules.json

aws wafv2 associate-web-acl \
  --web-acl-arn YOUR_WEB_ACL_ARN \
  --resource-arn YOUR_API_GATEWAY_ARN
```

プロダクション用ドメイン設定:

```
aws acm request-certificate \
  --domain-name api.yourdomain.com \
  --validation-method DNS

aws apigatewayv2 create-domain-name \
  --domain-name api.yourdomain.com \
  --domain-name-configurations CertificateArn=YOUR_CERT_ARN

aws route53 change-resource-record-sets \
  --hosted-zone-id YOUR_ZONE_ID \
  --change-batch file://route53-changes.json
```

環境変数の安全な管理:

```
aws secretsmanager create-secret \
  --name multicloud-auto-deploy/staging/db-config \
  --secret-string '{"host":"dynamodb","region":"ap-northeast-1"}'
```

```
import boto3
import json

def get_secret():
    client = boto3.client('secretsmanager')
    response = client.get_secret_value(SecretId='multicloud-auto-deploy/staging/db-config')
    return json.loads(response['SecretString'])
```

共通ライブラリの分離でコールドスタート改善:

```
mkdir python
pip install -r requirements.txt -t python/
zip -r layer.zip python/

aws lambda publish-layer-version \
  --layer-name multicloud-auto-deploy-dependencies \
  --zip-file fileb://layer.zip \
```

```
  --compatible-runtimes python3.12

aws lambda update-function-configuration \
  --function-name YOUR_FUNCTION_NAME \
  --layers YOUR_LAYER_ARN
```

リクエスト/レスポンスのエッジ処理:

```
// CloudFront Function: セキュリティヘッダ追加
function handler(event) {
    var response = event.response;
    response.headers['strict-transport-security'] = { value: 'max-age=31536000; includeSubdomai
    response.headers['x-content-type-options'] = { value: 'nosniff' };
    response.headers['x-frame-options'] = { value: 'DENY' };
    return response;
}
```

DynamoDB の自動バックアップ:

```
aws backup create-backup-plan \
  --backup-plan file://backup-plan.json

aws backup create-backup-selection \
  --backup-plan-id YOUR_PLAN_ID \
  --backup-selection file://backup-selection.json
```

問題が発生した場合は トラブルシューティングガイド を参照してください：

- Azure 認証問題（Service Principal、Terraform Wrapper 等）
- GCP リソース競合（State 管理、リソースインポート）
- フロントエンド API 接続問題（ビルド順序、API URL 設定）
- 権限エラー（IAM、RBAC 設定）

コントリビューションを歓迎します！詳細は CONTRIBUTING.md をご覧ください。

MIT License - 詳細は LICENSE をご覧ください。

- GitHub Actions Documentation
- Terraform Documentation
- Pulumi Documentation

Chapter 2

System Architecture

Multi-Cloud Auto Deploy Platform の完全なシステムアーキテクチャドキュメント

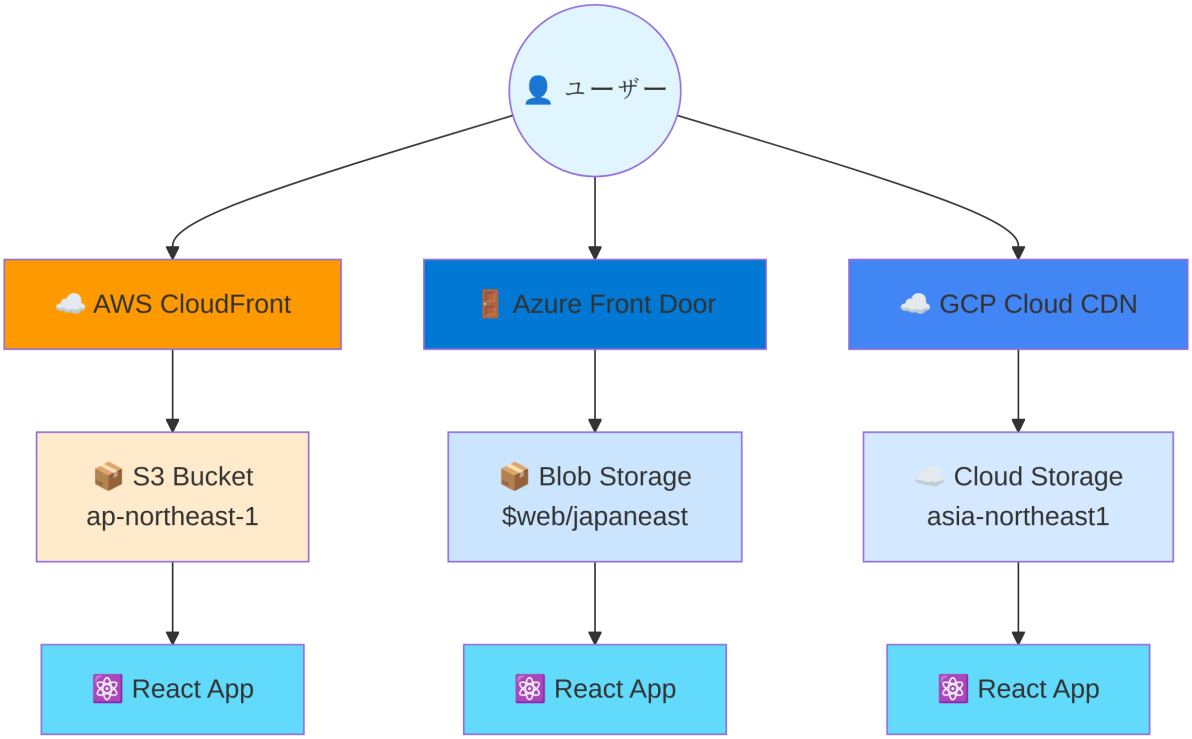


図 2.1: Diagram 1

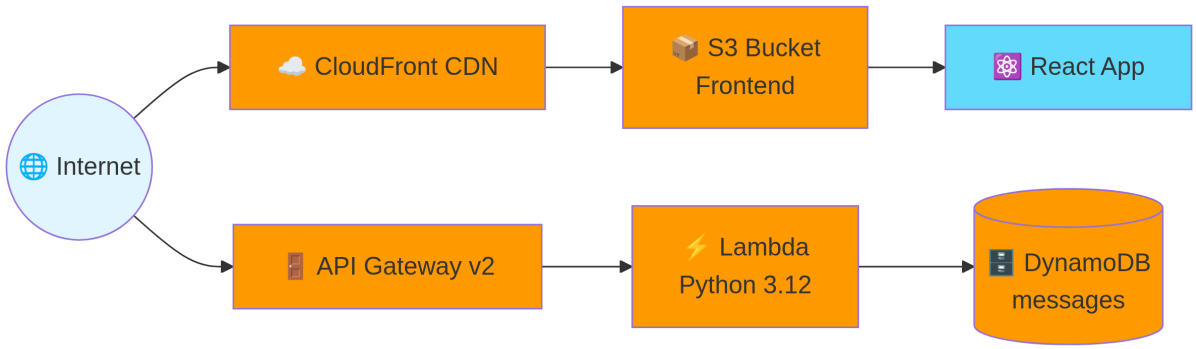
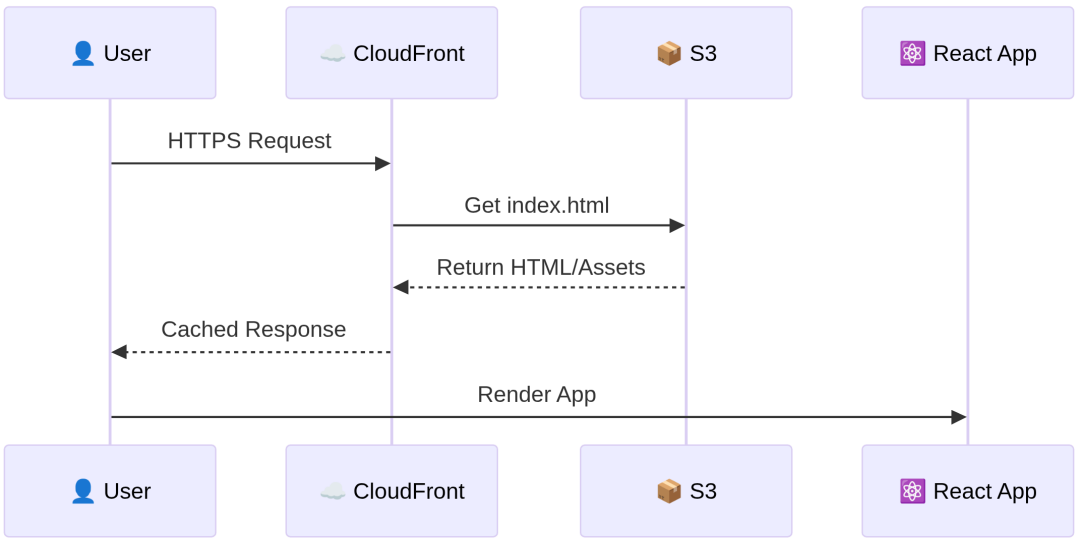


図 2.2: Diagram 2

リソース	名前	目的	リージョン
S3 Bucket	multicloud-auto-deploy-staging-frontend	フロントエンドホスティング	ap-northeast-1
CloudFront	`E2GDU7Y7UGDV3S`	CDN・HTTPS 終端	Global
Lambda	multicloud-auto-deploy-staging-api	バックエンド API（Python 3.12）	ap-northeast-1
API Gateway	`z42qmqdqac`	HTTP API ゲートウェイ（v2）	ap-northeast-1
DynamoDB	`simple-sns-messages`	NoSQL データベース（PAY_PER_REQUEST）	ap-northeast-1



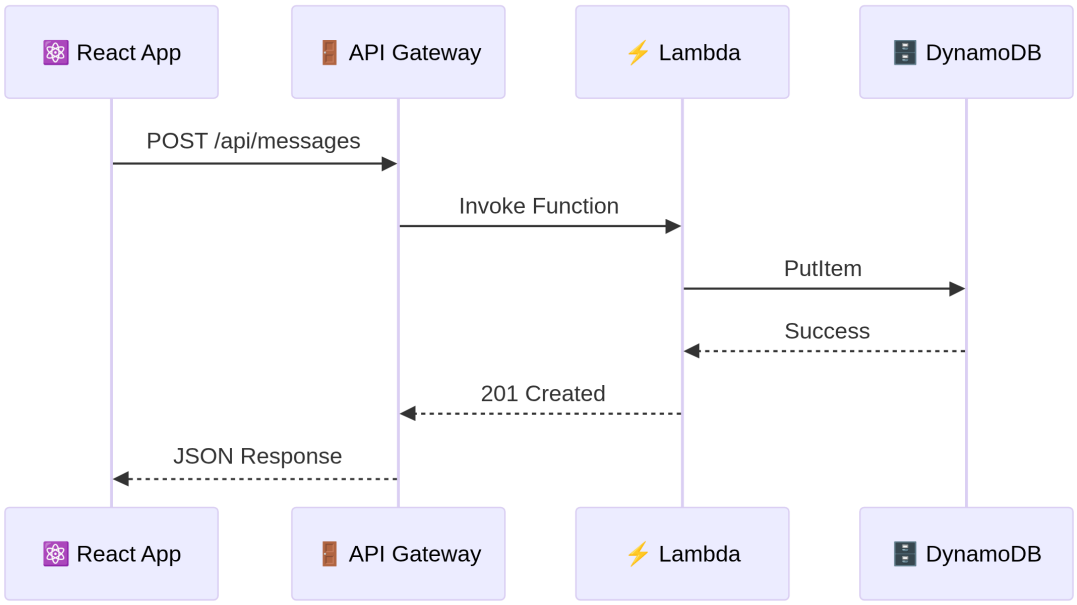


図 2.4: Diagram 4

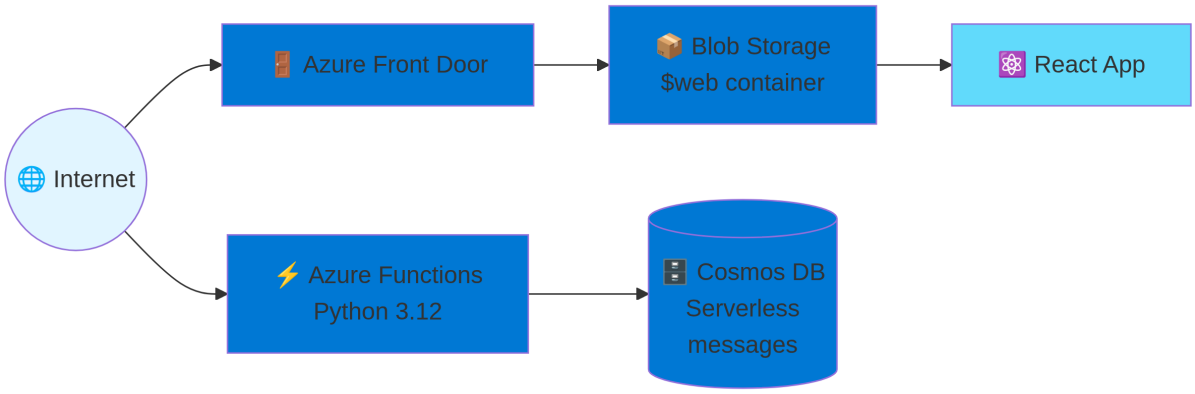


図 2.5: Diagram 5

リソース	名前	目的	リージョン
Resource Group	multicloud-auto-deploy-staging-rg	すべてのリソース管理	japaneast
Storage Account	`mcadwebd45ihd`	フロントエンドホスティング ($web)	japaneast
Function App	multicloud-auto-deploy-staging-func	バックエンド API (Python 3.12)	japaneast
Cosmos DB	`simple-sns-cosmos`	NoSQL データベース (Serverless)	japaneast
Front Door Profile	`multicloud-frontend-afd`	CDN・WAF	Global
Front Door Endpoint	`multicloud-frontend`	CDN エンドポイント	Global

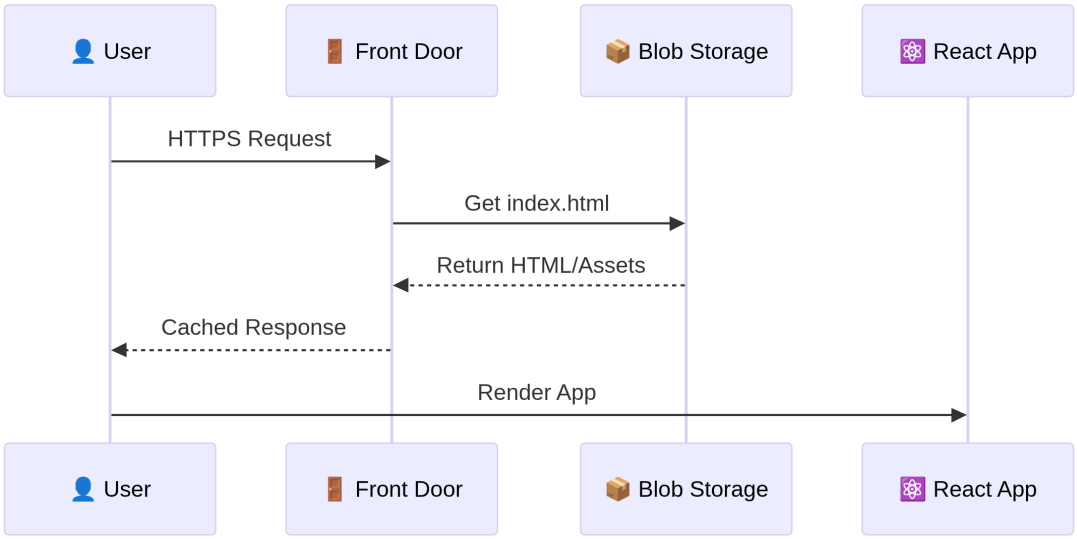


図 2.6: Diagram 6

Service Principal 権限:

- Contributor: リソースの作成・管理
- Storage Blob Data Contributor: ストレージへのデータ書き込み

リソース	名前	目的	リージョン
Cloud Storage	ashnova-multicloud-auto-deploy-staging-frontend	フロントエンドホスティング	asia-northeast1

リソース	名前	目的	リージョン
Cloud Run	multicloud-auto-deploy-staging-api	バックエンド API（Docker）	asia-northeast1
Firestore	`(default)`	NoSQL データベース	asia-northeast1
Backend Bucket	`multicloud-frontend-backend`	CDN 統合	Global
Global IP	`multicloud-frontend-ip`	固定 IP アドレス (34.120.43.83)	Global
URL Map	`multicloud-frontend-urlmap`	ルーティング	Global
HTTP Proxy	multicloud-frontend-http-proxy	HTTP 終端	Global
Forwarding Rule	multicloud-frontend-forwarding-rule	トラフィック転送	Global

Editor ロール保持者:

- sat0sh1kawada00@gmail.com
- sat0sh1kawada01@gmail.com

権限範囲:

- Cloud Run: デプロイ・管理
- Artifact Registry: イメージ管理
- Firestore: データベース管理
- Cloud Storage: ストレージ管理
- Compute Engine: Load Balancer 管理

技術	バージョン	用途
React	18.2.0	UI ライブラリ
TypeScript	5.0.2	型安全性
Vite	5.0.8	ビルドツール・開発サーバー
Tailwind CSS	3.4.0	スタイリング
Axios	1.6.5	HTTP クライアント

主要機能:

- メッセージ CRUD 操作
- クラウドプロバイダー自動検出表示
- レスポンシブデザイン
- リアルタイム更新

技術	バージョン	用途
Python	3.11	ランタイム

技術	バージョン	用途
FastAPI	0.109.0	Web フレームワーク
Uvicorn	0.27.0	ASGI サーバー
Pydantic	2.5.3	データバリデーション
boto3	1.34.22	AWS SDK
azure-cosmos	4.5.1	Azure Cosmos DB SDK
google-cloud-firestore	2.14.0	GCP Firestore SDK

API エンドポイント:

```
GET  /                    - ルート（クラウド情報）
GET  /api/health          - ヘルスチェック
GET  /api/messages        - メッセージ一覧取得
POST /api/messages        - メッセージ作成
DELETE /api/messages/{id} - メッセージ削除
```

クラウド自動検出ロジック:

```
AWS_EXECUTION_ENV → "AWS"
WEBSITE_INSTANCE_ID → "Azure"
K_SERVICE → "GCP"
その他 → "Local"
```

ツール	バージョン	用途
Terraform	1.7.5	IaC（Infrastructure as Code）
Docker	24.0+	コンテナ化
GitHub Actions	-	CI/CD

クラウド	サービス	タイプ	スキーマ
AWS	DynamoDB	NoSQL	`messages` テーブル
Azure	Cosmos DB	NoSQL	`messages` データベース
GCP	Firestore	NoSQL	`messages` コレクション

共通データモデル:

```
{
  "id": "uuid-string",
  "text": "message content",
```

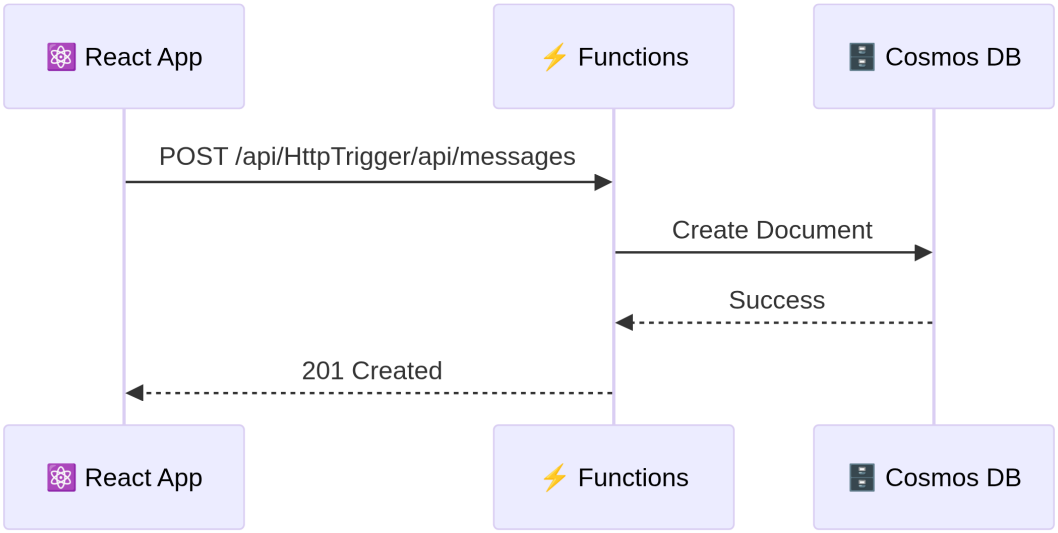


図 2.7: Diagram 7

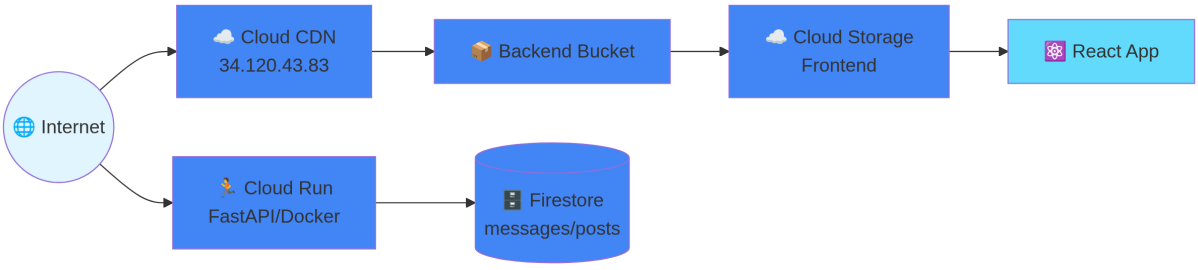


図 2.8: Diagram 8

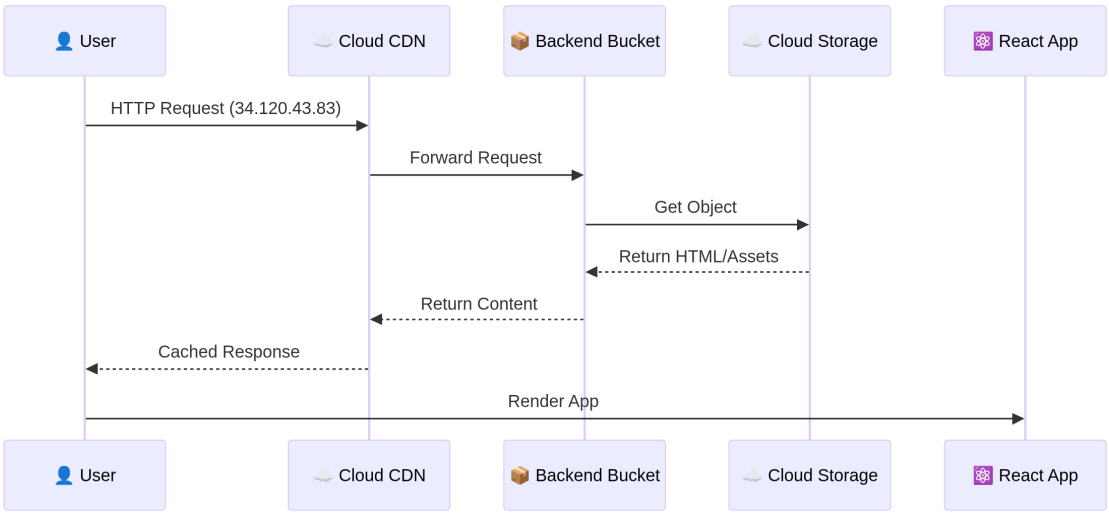


図 2.9: Diagram 9

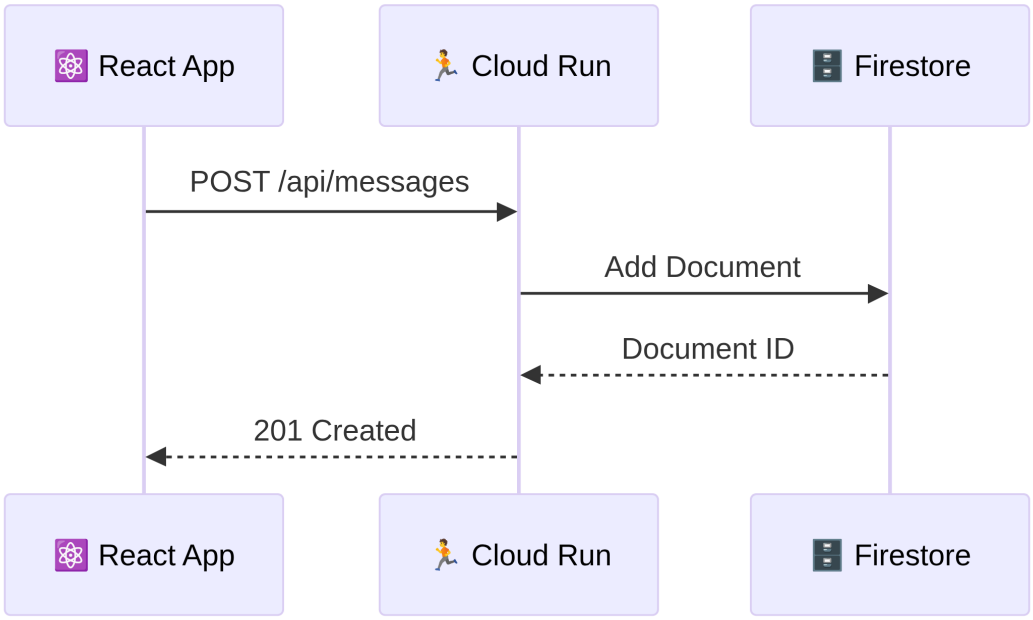


図 2.10: Diagram 10

```
  "timestamp": "ISO 8601 datetime",
  "cloud": "AWS|Azure|GCP"
}
```

- **IAM User**: satoshi（最小権限原則）
- **Lambda Execution Role**: DynamoDB アクセス権限
- **S3 Bucket**: パブリック読み取り（静的サイト）
- **API Gateway**: パブリックアクセス（認証なし）
- **Service Principal**: terraform-deploy（Contributor ロール）
- **Container App**: マネージド ID
- **Storage Account**: パブリック読み取り（静的サイト）
- **User Accounts**: Editor ロール
- **Cloud Run**: allUsers invoker 権限
- **Cloud Storage**: allUsers objectViewer 権限
- CloudFront HTTPS 強制
- API Gateway CORS 設定
- Lambda VPC 統合（オプション）
- Azure Front Door HTTPS 強制
- Container Apps Ingress 制御
- Private Endpoint（オプション）
- HTTPS Load Balancer（計画中）
- Cloud Armor WAF（計画中）
- VPC Service Controls（オプション）
- **送信中の暗号化**: HTTPS/TLS 1.2+
- **保存時の暗号化**:
  - AWS: S3/DynamoDB 標準暗号化
  - Azure: Storage/Cosmos DB 標準暗号化
  - GCP: Cloud Storage/Firestore 標準暗号化

項目	AWS	Azure	GCP
CDN	CloudFront	Azure Front Door	Cloud CDN
キャッシュ TTL	86400 秒	デフォルト	3600 秒
Gzip 圧縮	[OK]	[OK]	[OK]
HTTP/2	[OK]	[OK]	[OK]

項目	AWS	Azure	GCP
静的アセット最適化	[OK]	[OK]	[OK]

項目	AWS Lambda	Azure Container Apps	GCP Cloud Run
コールドスタート	∼500ms	∼1s	∼500ms
メモリ	512 MB	0.5 Gi	512 MiB
タイムアウト	30s	300s	300s
同時実行数	1000	10	80
オートスケール	[OK]	[OK]	[OK]

項目	DynamoDB	Cosmos DB	Firestore
読み込み待機時間	<10ms	<10ms	<10ms
インデックス	id (primary key)	id (partition key)	id (document ID)
整合性	結果的整合性	セッション整合性	強整合性オプション
バックアップ	自動	自動	自動

目標値:

- ページ読み込み時間: < 2 秒
- API レスポンス時間: < 200ms
- スループット: 1000 req/s 以上
- 可用性: 99.9% 以上
- Lambda: 同時実行数に応じて自動スケール
- DynamoDB: オンデマンドキャパシティ
- CloudFront: 自動グローバルスケール
- Container Apps: 0-10 レプリカで自動スケール
- Cosmos DB: サーバーレスモード
- Front Door: 自動グローバルスケール
- Cloud Run: 0-1000 インスタンスで自動スケール
- Firestore: 自動スケール
- Cloud CDN: 自動グローバルスケール

クラウド	ロードバランサー	ヘルスチェック
AWS	API Gateway	Lambda 自動
Azure	Container Apps Ingress	HTTP /api/health
GCP	Cloud Run Internal LB	HTTP /

- CloudWatch Metrics: Lambda 実行時間、エラー率
- CloudWatch Logs: Lambda 実行ログ
- X-Ray: 分散トレーシング（オプション）
- Azure Monitor: Container Apps メトリクス
- Application Insights: APM
- Log Analytics: 集約ログ
- Cloud Monitoring: Cloud Run メトリクス
- Cloud Logging: 実行ログ
- Cloud Trace: 分散トレーシング
- エラー率 $>$ 5%
- レスポンスタイム $>$ 1 秒
- 可用性 $<$ 99%
- コスト異常検知

データ	AWS	Azure	GCP
DynamoDB/ Cosmos/Firestore	継続的バックアップ	自動バックアップ	日次自動
復旧時間目標（RTO）	$<$ 1 時間	$<$ 1 時間	$<$ 1 時間
復旧ポイント目標（RPO）	$<$ 5 分	$<$ 5 分	$<$ 5 分

- **マルチリージョン**: 各クラウドで異なるリージョン使用
- **フェイルオーバー**: DNS/CDN レベルでの切り替え
- **データレプリケーション**: データベース自動レプリケーション

クラウド	サービス	月額コスト（USD）
AWS	CloudFront	$1-5
	S3	$0.5-2
	Lambda	$0-5（無料枠内）

クラウド	サービス	月額コスト（USD）
	API Gateway	$3.5-10
	DynamoDB	$0-5（オンデマンド）
	合計	**$5-27**
Azure	Front Door	$35-50
	Storage	$0.5-2
	Container Apps	$0-10（無料枠）
	Cosmos DB	$0-25（サーバーレス）
	合計	**$35-87**
GCP	Cloud CDN	$0-5
	Cloud Storage	$0.5-2
	Cloud Run	$0-5（無料枠）
	Firestore	$0-5（無料枠）
	合計	**$0.5-17**

注: トラフィック量、データ容量、実行時間により変動

1. **無料枠の活用**
   - AWS: Lambda 100 万リクエスト/月
   - Azure: Container Apps 180,000 vCPU 秒/月
   - GCP: Cloud Run 200 万リクエスト/月
2. **リソースの最適化**
   - Lambda/Cloud Run: メモリサイズの最適化
   - Container Apps: レプリカ数の調整
   - データベース: 使用量モニタリング
3. **予約インスタンス**（本番環境）
   - AWS: Savings Plans
   - Azure: Reserved Instances
   - GCP: Committed Use Discounts

☐ 認証・認可の実装（Cognito/Azure AD/Firebase Auth）

☐ WAF の設定

☐ HTTPS 完全対応（GCP）

☐ シークレット管理の統一

☐ 統合ダッシュボード構築

☐ アラート設定

☐ ログ集約

☐ エラートラッキング（Sentry 等）

☐ GitHub Actions 有効化

☐ 自動テストの拡充

☐ ブルー・グリーンデプロイ

☐ カナリアリリース

☐ ユーザー認証

☐ ファイルアップロード

☐ リアルタイム通信（WebSocket）

☐ 検索機能

- AWS Well-Architected Framework
- Azure Architecture Center
- Google Cloud Architecture Framework
- **React 18**: UI ライブラリ
- **TypeScript**: 型安全性
- **Vite**: ビルドツール
- **Tailwind CSS**: スタイリング
- **Axios**: HTTP クライアント

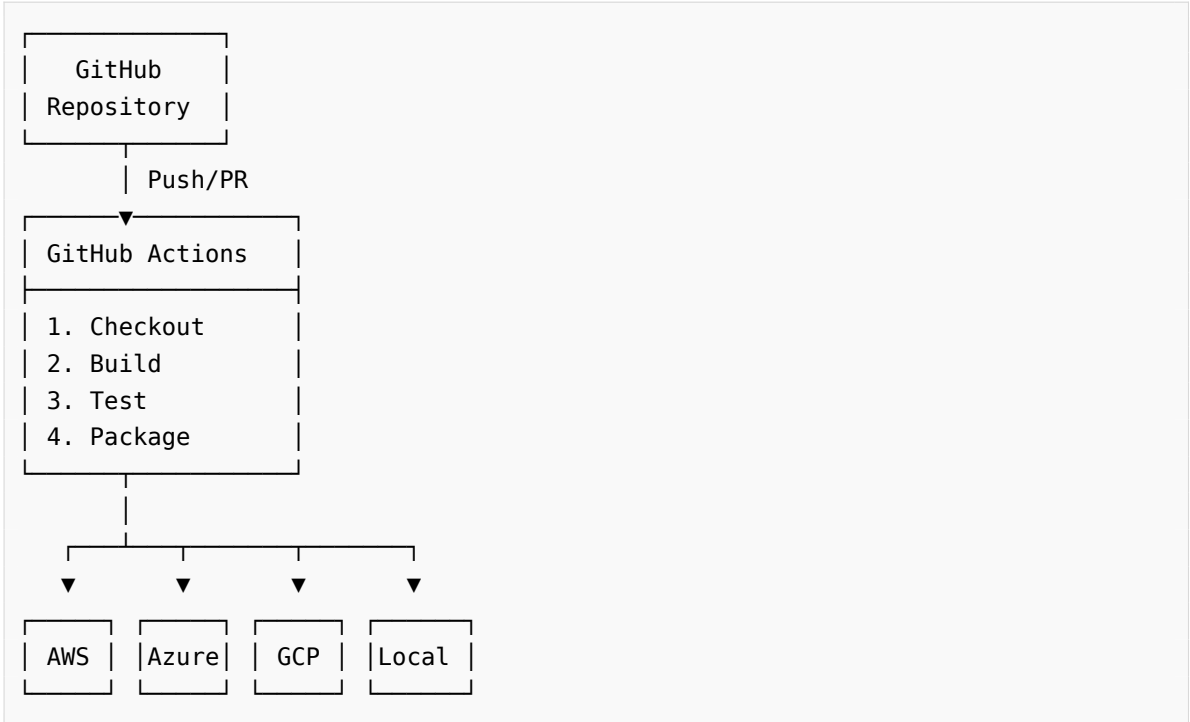
クラウド	サービス	URL 形式
AWS	S3 + CloudFront	`https://xxx.cloudfront.net`
Azure	Static Web Apps	`https://xxx.azurestaticapps.net`
GCP	Cloud Storage + CDN	`https://storage.googleapis.com/xxx`

- **FastAPI**: Python ウェブフレームワーク
- **Python 3.11**: ランタイム
- **Pydantic**: データバリデーション
- **Uvicorn**: ASGI サーバー

```
GET  /                    - ルート
GET  /api/health          - ヘルスチェック
GET  /api/messages        - メッセージ一覧
POST /api/messages        - メッセージ作成
GET  /api/messages/{id}   - メッセージ取得
DELETE /api/messages/{id} - メッセージ削除
```

クラウド	サービス	特徴
AWS	Lambda + API Gateway	サーバーレス、オートスケール
Azure	Azure Functions	サーバーレス、統合認証
GCP	Cloud Functions/Run	サーバーレス、コンテナ対応

クラウド	サービス	タイプ	用途
AWS	DynamoDB	NoSQL	高速、スケーラブル
AWS	RDS (PostgreSQL)	SQL	リレーショナル
Azure	Cosmos DB	NoSQL	グローバル分散
Azure	Azure SQL	SQL	エンタープライズ
GCP	Firestore	NoSQL	リアルタイム
GCP	Cloud SQL	SQL	マネージド



1. **Trigger**: Push to main or Manual
2. **Build**:
   - Frontend: `npm run build`
   - Backend: `pip install` + `zip`
3. **Test**:
   - Frontend: `vitest`

- Backend: `pytest`

4. **Deploy Infrastructure**: Terraform/Pulumi
5. **Deploy Application**:
   - Frontend → S3/Storage
   - Backend → Lambda/Functions
6. **Notify**: Success/Failure

```
infrastructure/terraform/
├── aws/
│   ├── main.tf         # プロバイダー設定
│   ├── frontend.tf     # S3 + CloudFront
│   ├── backend.tf      # Lambda + API Gateway
│   ├── database.tf     # DynamoDB
│   ├── variables.tf    # 変数定義
│   └── outputs.tf      # 出力値
├── azure/
└── gcp/
```

```python
import pulumi
import pulumi_aws as aws

bucket = aws.s3.Bucket("frontend",
    website={
        "index_document": "index.html"
    })
```

- **AWS**: Cognito
- **Azure**: Azure AD B2C
- **GCP**: Firebase Auth
- HTTPS 強制
- CORS 設定
- API キー（オプション）
- **AWS**: Secrets Manager / Parameter Store
- **Azure**: Key Vault
- **GCP**: Secret Manager

サービス	自動スケール	最大
Lambda	あり	1000 並列
Azure Functions	あり	200 インスタンス
Cloud Functions	あり	設定可能

- **DynamoDB**: オンデマンド課金
- **Cosmos DB**: RU/s 自動スケール
- **Firestore**: 自動
- リクエスト数
- レスポンスタイム
- エラー率
- コスト
- **AWS**: CloudWatch Logs
- **Azure**: Application Insights
- **GCP**: Cloud Logging
- エラー閾値超過
- レイテンシ増加
- コスト異常
- データベーススナップショット
- S3 バージョニング
- マルチリージョン対応
- **RTO** (Recovery Time Objective): $<$ 1 時間
- **RPO** (Recovery Point Objective): $<$ 5 分
- Code Splitting
- Lazy Loading
- CDN キャッシング
- 画像最適化
- コネクションプーリング
- キャッシング (Redis)
- 非同期処理
- バッチ処理

1. **サーバーレス優先**: 使用量課金
2. **オートスケール**: 需要に応じて調整
3. **リザーブドインスタンス**: 予測可能な負荷
4. **ストレージクラス**: ライフサイクルポリシー

クラウド	フロントエンド	バックエンド	DB	合計
AWS	$2	$1	$1	$4
Azure	$0*	$1	$1	$2

クラウド	フロントエンド	バックエンド	DB	合計
GCP	$1	$1	$1	$3

*Azure Static Web Apps の無料枠

- Kubernetes 対応
- マルチリージョンデプロイ
- Blue-Green デプロイ
- A/B テスト機能
- メトリクスダッシュボード

Chapter 3

Initial Setup

Multi-Cloud Auto Deploy Platform のセットアップ手順です。

- Git
- Node.js 18+
- Python 3.11+
- Docker & Docker Compose
- **AWS**: AWS CLI v2
- **Azure**: Azure CLI
- **GCP**: gcloud CLI

プロジェクトルートに.envファイルを作成：

```
AWS_ACCESS_KEY_ID=your_access_key
AWS_SECRET_ACCESS_KEY=your_secret_key
AWS_REGION=us-east-1

AZURE_SUBSCRIPTION_ID=your_subscription_id
AZURE_TENANT_ID=your_tenant_id
AZURE_CLIENT_ID=your_client_id
AZURE_CLIENT_SECRET=your_client_secret

GCP_PROJECT_ID=your_project_id
GCP_REGION=us-central1
GOOGLE_APPLICATION_CREDENTIALS=/path/to/service-account-key.json

PROJECT_NAME=multicloud-auto-deploy
ENVIRONMENT=staging
```

```
cd services/frontend
npm install
npm run dev
```

```
cd services/backend
pip install -r requirements.txt -r requirements-dev.txt
python -m uvicorn src.main:app --reload
```

```
docker-compose up
```

1. **認証情報の設定**

```
aws configure
```

2. **デプロイ**

```
./scripts/deploy-aws.sh staging
```

1. **ログイン**

```
az login
```

2. **デプロイ**

```
./scripts/deploy-azure.sh staging
```

1. **認証**

```
gcloud auth login
gcloud auth application-default login
```

2. **デプロイ**

```
./scripts/deploy-gcp.sh staging your-project-id
```

GitHub リポジトリの Settings → Secrets and variables → Actions で以下を設定：

- `AWS_ACCESS_KEY_ID`
- `AWS_SECRET_ACCESS_KEY`
- `AWS_S3_BUCKET`
- AWS_CLOUDFRONT_DISTRIBUTION_ID
- `AWS_LAMBDA_FUNCTION_NAME`
- `AWS_API_URL`
- `AZURE_CREDENTIALS` (JSON format)
- `AZURE_STORAGE_ACCOUNT`

- AZURE_RESOURCE_GROUP
- AZURE_FUNCTION_APP_NAME
- AZURE_API_URL
- GCP_CREDENTIALS (Service Account JSON)
- GCP_PROJECT_ID
- GCP_BUCKET_NAME
- GCP_FUNCTION_NAME
- GCP_API_URL
- mainブランチへのプッシュで自動デプロイ
- 手動トリガー: Actions → Select workflow → Run workflow

```
cd infrastructure/terraform/aws
echo "print('placeholder')" > lambda_placeholder.py
zip lambda_placeholder.zip lambda_placeholder.py
rm lambda_placeholder.py
```

```
npm install -g azure-functions-core-tools@4
```

```
gcloud services enable cloudfunctions.googleapis.com
gcloud services enable cloudbuild.googleapis.com
gcloud services enable storage-api.googleapis.com
```

- AWS デプロイメント詳細
- Azure デプロイメント詳細
- GCP デプロイメント詳細
- アーキテクチャ

3.1 Quick Reference Guide

AWS/Azure/GCP マルチクラウド環境の運用で頻繁に使用するコマンドをまとめたものです。

```
API: https://z42qmqdqac.execute-api.ap-northeast-1.amazonaws.com
CDN: https://dx3l4mbwg1ade.cloudfront.net
```

```
API: https://multicloud-auto-deploy-staging-func-d8a2guhfere0etcq.japaneast-01.azurewebsites.ne
CDN: https://multicloud-frontend-f9cvamfnauexasd8.z01.azurefd.net
```

```
API: https://multicloud-auto-deploy-staging-api-899621454670.asia-northeast1.run.app
CDN: http://34.120.43.83
```

```
./scripts/deploy-lambda-aws.sh

PROJECT_NAME=myapp ENVIRONMENT=production ./scripts/deploy-lambda-aws.sh
```

```
cd services/api
pip3 install -r requirements.txt -t .build/package/ --platform manylinux2014_x86_64 --only-bina
cp -r app .build/package/
cd .build/package && zip -r9 ../lambda.zip .

aws s3 cp .build/lambda.zip s3://YOUR_BUCKET/lambda-deployments/lambda.zip

aws lambda update-function-code \
  --function-name YOUR_FUNCTION_NAME \
  --s3-bucket YOUR_BUCKET \
  --s3-key lambda-deployments/lambda.zip \
  --publish
```

```
cd services/frontend_react
npm run build
aws s3 sync dist/ s3://YOUR_FRONTEND_BUCKET/ --delete

aws cloudfront create-invalidation \
  --distribution-id YOUR_DISTRIBUTION_ID \
  --paths "/*"
```

```
./scripts/test-api.sh -e https://YOUR_API_ID.execute-api.ap-northeast-1.amazonaws.com

./scripts/test-api.sh -e https://YOUR_API.amazonaws.com --verbose
```

```
aws lambda invoke \
  --function-name YOUR_FUNCTION_NAME \
  --payload '{
    "version": "2.0",
    "routeKey": "$default",
    "rawPath": "/api/messages/",
    "headers": {"accept": "application/json"},
    "requestContext": {
      "http": {
        "method": "GET",
        "path": "/api/messages/"
      }
    }
  }' \
  --cli-binary-format raw-in-base64-out \
  /tmp/response.json

cat /tmp/response.json | jq .
```

```
curl -v https://YOUR_API.amazonaws.com/api/messages/

curl -X POST https://YOUR_API.amazonaws.com/api/messages/ \
  -H "Content-Type: application/json" \
  -d '{"content":"Test message","author":"Tester"}'

curl -X PUT https://YOUR_API.amazonaws.com/api/messages/MESSAGE_ID \
  -H "Content-Type: application/json" \
  -d '{"content":"Updated message"}'

curl -X DELETE https://YOUR_API.amazonaws.com/api/messages/MESSAGE_ID
```

```
aws logs tail /aws/lambda/YOUR_FUNCTION_NAME --follow

aws logs tail /aws/lambda/YOUR_FUNCTION_NAME --since 10m

aws logs tail /aws/lambda/YOUR_FUNCTION_NAME --follow --filter-pattern "ERROR"

aws logs tail /aws/lambda/YOUR_FUNCTION_NAME --format json
```

```
aws logs tail /aws/apigateway/YOUR_API_NAME --follow

aws logs tail /aws/apigateway/YOUR_API_NAME --since 5m

```

```
aws logs tail /aws/apigateway/YOUR_API_NAME --filter-pattern "5XX"
```

```
aws logs start-query \
  --log-group-name /aws/lambda/YOUR_FUNCTION_NAME \
  --start-time $(date -u -d '1 hour ago' +%s) \
  --end-time $(date -u +%s) \
  --query-string 'fields @timestamp, @message | filter @message like /ERROR/ | stats count() by

aws logs start-query \
  --log-group-name /aws/lambda/YOUR_FUNCTION_NAME \
  --start-time $(date -u -d '1 hour ago' +%s) \
  --end-time $(date -u +%s) \
  --query-string 'fields @timestamp, @duration | stats avg(@duration), max(@duration), min(@du
```

```
aws cloudwatch get-metric-statistics \
  --namespace AWS/Lambda \
  --metric-name Invocations \
  --dimensions Name=FunctionName,Value=YOUR_FUNCTION_NAME \
  --start-time $(date -u -d '1 hour ago' +%Y-%m-%dT%H:%M:%S) \
  --end-time $(date -u +%Y-%m-%dT%H:%M:%S) \
  --period 300 \
  --statistics Sum

aws cloudwatch get-metric-statistics \
  --namespace AWS/Lambda \
  --metric-name Errors \
  --dimensions Name=FunctionName,Value=YOUR_FUNCTION_NAME \
  --start-time $(date -u -d '1 hour ago' +%Y-%m-%dT%H:%M:%S) \
  --end-time $(date -u +%Y-%m-%dT%H:%M:%S) \
  --period 300 \
  --statistics Sum

aws cloudwatch get-metric-statistics \
  --namespace AWS/Lambda \
  --metric-name Duration \
  --dimensions Name=FunctionName,Value=YOUR_FUNCTION_NAME \
  --start-time $(date -u -d '1 hour ago' +%Y-%m-%dT%H:%M:%S) \
  --end-time $(date -u +%Y-%m-%dT%H:%M:%S) \
  --period 300 \
  --statistics Average,Maximum

aws cloudwatch get-metric-statistics \
  --namespace AWS/Lambda \
  --metric-name Throttles \
  --dimensions Name=FunctionName,Value=YOUR_FUNCTION_NAME \
  --start-time $(date -u -d '1 hour ago' +%Y-%m-%dT%H:%M:%S) \
  --end-time $(date -u +%Y-%m-%dT%H:%M:%S) \
  --period 300 \
  --statistics Sum
```

```
aws cloudwatch get-metric-statistics \
  --namespace AWS/ApiGateway \
  --metric-name Count \
  --dimensions Name=ApiId,Value=YOUR_API_ID \
  --start-time $(date -u -d '1 hour ago' +%Y-%m-%dT%H:%M:%S) \
  --end-time $(date -u +%Y-%m-%dT%H:%M:%S) \
  --period 300 \
  --statistics Sum

aws cloudwatch get-metric-statistics \
  --namespace AWS/ApiGateway \
  --metric-name 5XXError \
  --dimensions Name=ApiId,Value=YOUR_API_ID \
  --start-time $(date -u -d '1 hour ago' +%Y-%m-%dT%H:%M:%S) \
  --end-time $(date -u +%Y-%m-%dT%H:%M:%S) \
  --period 300 \
  --statistics Sum

aws cloudwatch get-metric-statistics \
  --namespace AWS/ApiGateway \
  --metric-name Latency \
  --dimensions Name=ApiId,Value=YOUR_API_ID \
  --start-time $(date -u -d '1 hour ago' +%Y-%m-%dT%H:%M:%S) \
  --end-time $(date -u +%Y-%m-%dT%H:%M:%S) \
  --period 300 \
  --statistics Average,p99
```

```
./scripts/setup-monitoring.sh

ALERT_EMAIL=your@email.com ./scripts/setup-monitoring.sh

aws cloudwatch describe-alarms --alarm-name-prefix YOUR_PROJECT

aws cloudwatch describe-alarms --state-value ALARM
```

```
aws lambda get-policy \
  --function-name YOUR_FUNCTION_NAME \
  --query Policy \
  --output text | jq .

aws lambda get-function-configuration \
  --function-name YOUR_FUNCTION_NAME \
  --query Role

ROLE_NAME=$(aws lambda get-function-configuration --function-name YOUR_FUNCTION_NAME --query Ro
aws iam list-attached-role-policies --role-name $ROLE_NAME
```

```
aws lambda add-permission \
```

```
  --function-name YOUR_FUNCTION_NAME \
  --statement-id apigateway-http-api \
  --action lambda:InvokeFunction \
  --principal apigateway.amazonaws.com \
  --source-arn "arn:aws:execute-api:REGION:ACCOUNT_ID:API_ID/*/*"

aws lambda remove-permission \
  --function-name YOUR_FUNCTION_NAME \
  --statement-id OLD_STATEMENT_ID
```

```
aws logs create-log-group \
  --log-group-name /aws/apigateway/YOUR_API_NAME

aws apigatewayv2 update-stage \
  --api-id YOUR_API_ID \
  --stage-name '$default' \
  --access-log-settings "DestinationArn=arn:aws:logs:REGION:ACCOUNT_ID:log-group:/aws/apigatewa

aws logs tail /aws/apigateway/YOUR_API_NAME --follow
```

```
aws lambda get-function --function-name YOUR_FUNCTION_NAME

aws lambda get-function-configuration --function-name YOUR_FUNCTION_NAME

aws lambda get-function-configuration \
  --function-name YOUR_FUNCTION_NAME \
  --query Environment

aws lambda get-function-configuration \
  --function-name YOUR_FUNCTION_NAME \
  --query '[Timeout,MemorySize]' \
  --output table
```

```
aws lambda delete-function --function-name YOUR_FUNCTION_NAME
```

```
aws apigatewayv2 delete-api --api-id YOUR_API_ID

aws apigateway delete-rest-api --rest-api-id YOUR_API_ID
```

```
aws logs delete-log-group --log-group-name /aws/lambda/YOUR_FUNCTION_NAME
aws logs delete-log-group --log-group-name /aws/apigateway/YOUR_API_NAME
```

```
aws s3 rm s3://YOUR_BUCKET/ --recursive

aws s3 rb s3://YOUR_BUCKET
```

```
aws dynamodb delete-table --table-name YOUR_TABLE_NAME
```

```
aws cloudwatch delete-alarms --alarm-names ALARM_NAME1 ALARM_NAME2

aws cloudwatch describe-alarms \
  --alarm-name-prefix YOUR_PROJECT \
  --query 'MetricAlarms[*].AlarmName' \
  --output text | xargs -n 1 aws cloudwatch delete-alarms --alarm-names
```

プロジェクト全体で使用する環境変数:

```
export PROJECT_NAME="multicloud-auto-deploy"
export ENVIRONMENT="staging"
export AWS_REGION="ap-northeast-1"
export FUNCTION_NAME="${PROJECT_NAME}-${ENVIRONMENT}-api"
export API_ID="abc123def4"
export DISTRIBUTION_ID="EXXXXXXXXXXXX"
export FRONTEND_BUCKET="${PROJECT_NAME}-${ENVIRONMENT}-frontend"
export ALERT_EMAIL="your@email.com"

alias lambda-logs='aws logs tail /aws/lambda/$FUNCTION_NAME --follow'
alias api-logs='aws logs tail /aws/apigateway/$PROJECT_NAME-$ENVIRONMENT-api --follow'
alias lambda-deploy='cd ~/projects/multicloud-auto-deploy && ./scripts/deploy-lambda-aws.sh'
alias api-test='~/projects/multicloud-auto-deploy/scripts/test-api.sh -e https://$API_ID.execut
```

- トラブルシューティングガイド
- エンドポイント一覧
- メイン README

```
aws lambda list-functions --query 'Functions[*].[FunctionName,Runtime,LastModified]' --output t

aws apigatewayv2 get-apis --query 'Items[*].[Name,ApiId,ApiEndpoint]' --output table

aws cloudwatch describe-alarms --query 'MetricAlarms[*].[AlarmName,StateValue]' --output table

aws dynamodb list-tables --query 'TableNames' --output table

aws s3 ls

aws logs tail /aws/lambda/$FUNCTION_NAME --since 5m | tail -n 10

aws lambda get-function-url-config --function-name $FUNCTION_NAME --query FunctionUrl --output

aws apigatewayv2 get-api --api-id $API_ID --query ApiEndpoint --output text
```

Chapter 4

Deployment Guides

4.1 AWS Deployment

図 4.1: AWS

このドキュメントでは、Multi-Cloud Auto Deploy アプリケーションを AWS にデプロイする詳細な手順を説明します。

```
┌──────────────┐
│   Route 53   │ (オプション)
└──────┬───────┘
       │
┌──────▼───────┐
│   CloudFront │ CDN
└──────┬───────┘
       │
┌──────▼───────┐
│   S3 Bucket  │ 静的ホスティング
└──────────────┘


┌──────────────┐
│  API Gateway │ HTTP API
└──────┬───────┘
       │
┌──────▼───────┐
│   Lambda     │ Python 3.11
└──────┬───────┘
       │
┌──────▼───────┐
│   DynamoDB   │ NoSQL DB
└──────────────┘
```

```
export AWS_REGION=us-east-1
export ENVIRONMENT=staging

./scripts/deploy-aws.sh staging
```

1. GitHub Secrets の設定
2. mainブランチへプッシュまたは手動トリガー

```
cd services/frontend
npm install
VITE_API_URL="https://your-api-id.execute-api.us-east-1.amazonaws.com" npm run build
```

```
cd infrastructure/terraform/aws

echo "print('placeholder')" > lambda_placeholder.py
zip lambda_placeholder.zip lambda_placeholder.py
rm lambda_placeholder.py

terraform init
terraform apply -var="environment=staging"
```

```
BUCKET_NAME=$(cd infrastructure/terraform/aws && terraform output -raw frontend_bucket_name)

aws s3 sync services/frontend/dist/ "s3://${BUCKET_NAME}/" --delete
```

```
cd services/backend
pip install -r requirements.txt -t package/
cp -r src/* package/
cd package && zip -r ../lambda.zip . && cd ..

FUNCTION_NAME=$(cd ../../infrastructure/terraform/aws && terraform output -raw lambda_function_
aws lambda update-function-code \
    --function-name "$FUNCTION_NAME" \
    --zip-file fileb://lambda.zip
```

```
DISTRIBUTION_ID=$(cd infrastructure/terraform/aws && terraform output -raw cloudfront_distribut
aws cloudfront create-invalidation \
    --distribution-id "$DISTRIBUTION_ID" \
    --paths "/*"
```

生成される AWS リソース：

- **S3 Bucket**: フロントエンドホスティング
- **CloudFront Distribution**: CDN
- **Lambda Function**: バックエンド API
- **API Gateway (HTTP API)**: API エンドポイント

- **DynamoDB Table**: データストレージ
- **IAM Roles**: Lambda 実行ロール
- **CloudWatch Logs**: ログストリーム

月額概算（us-east-1、小規模使用時）：

- S3: ~$1
- CloudFront: ~$1-5
- Lambda: ~$0-1（無料枠内）
- API Gateway: ~$1
- DynamoDB: ~$0-1（無料枠内）

合計: ~$3-10/月

```
HOSTED_ZONE_ID=$(aws route53 list-hosted-zones-by-name \
    --dns-name example.com \
    --query "HostedZones[0].Id" \
    --output text)
```

```
aws acm request-certificate \
    --domain-name "app.example.com" \
    --validation-method DNS \
    --region us-east-1
```

`terraform.tfvars`を作成：

```
domain_name = "app.example.com"
certificate_arn = "arn:aws:acm:us-east-1:123456789012:certificate/..."
hosted_zone_id = "Z1234567890ABC"
```

```
aws logs tail /aws/lambda/multicloud-auto-deploy-staging-api \
    --follow \
    --region us-east-1
```

- Lambda 実行回数
- API Gateway リクエスト数
- CloudFront ヒット率
- DynamoDB 読み書き

```
aws logs tail /aws/lambda/your-function-name --follow

aws lambda get-function --function-name your-function-name
```

```
aws s3api get-bucket-policy --bucket your-bucket-name

aws s3api get-public-access-block --bucket your-bucket-name
```

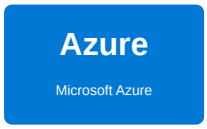
Lambda 関数が正しい CORS ヘッダーを返しているか確認：

```
return {
    'statusCode': 200,
    'headers': {
        'Access-Control-Allow-Origin': '*',
        'Access-Control-Allow-Headers': '*',
        'Access-Control-Allow-Methods': '*'
    },
    'body': json.dumps(data)
}
```

```
cd infrastructure/terraform/aws
terraform destroy -var="environment=staging"
```

- カスタムドメインの設定
- CI/CD パイプラインの最適化
- セキュリティベストプラクティス

4.2 Azure Deployment



Azure Container Apps を使用したマルチクラウド自動デプロイシステムのデプロイガイド


```
curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash

wget https://releases.hashicorp.com/terraform/1.7.5/terraform_1.7.5_linux_amd64.zip
unzip terraform_1.7.5_linux_amd64.zip
sudo mv terraform /usr/local/bin/

sudo apt-get update
sudo apt-get install docker.io
```

1. Azure にログイン

```
az login
```

2. サブスクリプションの確認

```
az account list --output table
az account set --subscription "YOUR_SUBSCRIPTION_ID"
```

3. Service Principal の作成

```
az ad sp create-for-rbac --name "terraform-deploy" \
  --role Contributor \
  --scopes /subscriptions/YOUR_SUBSCRIPTION_ID
```

出力例：

```
{
  "appId": "00000000-0000-0000-0000-000000000000",
  "displayName": "terraform-deploy",
  "password": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
  "tenant": "00000000-0000-0000-0000-000000000000"
}
```

```
export ARM_CLIENT_ID="<appId>"
export ARM_CLIENT_SECRET="<password>"
export ARM_SUBSCRIPTION_ID="<subscription_id>"
export ARM_TENANT_ID="<tenant>"
```

```
cd infrastructure/terraform/azure
terraform init
```

```
terraform plan -var="environment=staging"
terraform apply -var="environment=staging" -auto-approve
```

デプロイされるリソース（15 個）：- Resource Group - Container Registry - Container Apps Environment - Container App (Backend) - Storage Account (Frontend) - Azure Front Door - Cosmos DB

```
cd ../../services/backend

az acr login --name <registry_name>

docker build --platform linux/amd64 \
  -t <registry_name>.azurecr.io/multicloud-auto-deploy-api:latest \
  -f Dockerfile.azure .

docker push <registry_name>.azurecr.io/multicloud-auto-deploy-api:latest
```

```
az containerapp update \
  --name mcad-staging-api \
  --resource-group multicloud-auto-deploy-staging-rg \
  --image <registry_name>.azurecr.io/multicloud-auto-deploy-api:latest
```

```
cd ../frontend

export VITE_API_URL=https://<container_app_url>
npm run build

az storage blob upload-batch \
  --account-name <storage_account_name> \
  --source ./dist \
  --destination '$web' \
  --overwrite
```

リソース	名前	目的
Resource Group	multicloud-auto-deploy-staging-rg	すべてのリソースを格納
Container Registry	`mcadstagingacr`	Docker イメージの保存
Container Apps Environment	`mcad-staging-env`	Container Apps の実行環境

リソース	名前	目的
Container App	`mcad-staging-api`	バックエンド API
Storage Account	`mcadfestaging`	フロントエンドホスティング
Cosmos DB	multicloud-auto-deploy-staging-cosmos	NoSQL データベース
Azure Front Door	multicloud-auto-deploy-staging-endpoint	CDN + SSL 終端

```
Internet
   │
   ├─→ Azure Front Door (CDN) ──→ Storage Account (Frontend)
   │                                    └─→ Static Website
   │
   └─→ Container App (Backend) ──→ Cosmos DB
           └─→ Public Ingress
```

```
BACKEND_URL=$(terraform output -raw api_url)

curl $BACKEND_URL/api/health

curl -X POST $BACKEND_URL/api/messages \
  -H "Content-Type: application/json" \
  -d '{"text":"Azure Container Apps test"}'

curl $BACKEND_URL/api/messages
```

```
FRONTEND_URL=$(terraform output -raw frontend_url)

curl -I $FRONTEND_URL
```

ブラウザで `$FRONTEND_URL` にアクセスして動作確認

```
az cosmosdb sql database show \
  --account-name <cosmos_account> \
  --name messages \
  --resource-group multicloud-auto-deploy-staging-rg
```

症状: Container App のステータスが `Failed`

原因と対処:

1. **イメージが見つからない**

```
az acr repository list --name <registry_name> --output table

az acr repository show-tags --name <registry_name> \
  --repository multicloud-auto-deploy-api --output table
```

2. **プラットフォームの不一致**

```
docker build --platform linux/amd64 -f Dockerfile.azure .
```

3. **環境変数の不足**

```
az containerapp show --name mcad-staging-api \
  --resource-group multicloud-auto-deploy-staging-rg \
  --query properties.template.containers[0].env
```

症状: フロントエンドにアクセスすると 404 エラー

対処:

1. **Static Website 機能の有効化確認**

```
az storage blob service-properties show \
  --account-name <storage_account_name> \
  --query staticWebsite
```

2. **ファイルのアップロード確認**

```
az storage blob list \
  --account-name <storage_account_name> \
  --container-name '$web' \
  --output table
```

症状: バックエンドがデータベースに接続できない

対処:

1. **接続文字列の確認**

```
az cosmosdb keys list \
  --name <cosmos_account> \
  --resource-group multicloud-auto-deploy-staging-rg \
  --type connection-strings
```

2. **ファイアウォール設定の確認**

```
az cosmosdb update \
  --name <cosmos_account> \
  --resource-group multicloud-auto-deploy-staging-rg \
  --ip-range-filter "0.0.0.0/0"  # 開発環境のみ
```

症状: カスタムドメインが動作しない

対処:

```
az afd endpoint list \
  --profile-name <front_door_profile> \
  --resource-group multicloud-auto-deploy-staging-rg \
  --output table

az afd custom-domain create \
  --custom-domain-name example-com \
  --profile-name <front_door_profile> \
  --resource-group multicloud-auto-deploy-staging-rg \
  --host-name example.com \
  --minimum-tls-version TLS12
```

- 無料プラン: 180,000 vCPU 秒/月、360,000 GiB 秒/月
- CPU: 0.25〜4.0 vCPU
- メモリ: 0.5〜8.0 GiB
- リクエストタイムアウト: 240 秒
- 無料プラン: 1,000 RU/s、25 GB
- サーバーレスモード: 使用量課金
- スループット: 400〜1,000,000 RU/s
- 無料プラン（12 ヶ月）: 5 GB、20,000 リクエスト
- 静的 Web サイト: 無制限
- 帯域幅: 15 GB 送信/月（無料枠）
- Azure Container Apps Documentation
- Azure Cosmos DB Documentation
- Azure Front Door Documentation
- Azure Storage Static Website
- [OK] デプロイ完了

☐ カスタムドメインの設定

☐ SSL 証明書の設定

☐ モニタリングの設定

☐ CI/CD パイプラインの構築

4.3 GCP Deployment

Cloud Run を使用したマルチクラウド自動デプロイシステムのデプロイガイド

- 前提条件
- デプロイ手順
- リソース構成
- 検証
- トラブルシューティング

```
curl https://sdk.cloud.google.com | bash
exec -l $SHELL

wget https://releases.hashicorp.com/terraform/1.7.5/terraform_1.7.5_linux_amd64.zip
unzip terraform_1.7.5_linux_amd64.zip
sudo mv terraform /usr/local/bin/

sudo apt-get update
sudo apt-get install docker.io
```

1. **GCP にログイン**

```
gcloud auth login
gcloud auth application-default login
```

2. **プロジェクトの設定**

```
gcloud projects list
gcloud config set project YOUR_PROJECT_ID
```

3. **必要な API の有効化**

```
gcloud services enable \
  run.googleapis.com \
  artifactregistry.googleapis.com \
  firestore.googleapis.com \
  compute.googleapis.com \
  storage.googleapis.com
```

4. **Editor ロールの付与**（デプロイ用アカウント）

```
gcloud projects add-iam-policy-binding YOUR_PROJECT_ID \
  --member="user:your-email@gmail.com" \
  --role="roles/editor"
```

```
cd infrastructure/terraform/gcp
terraform init
```

```
gcloud artifacts repositories create mcad-staging-repo \
  --repository-format=docker \
  --location=asia-northeast1 \
  --description="Multi-Cloud Auto Deploy Docker images"

terraform import google_artifact_registry_repository.main \
  projects/YOUR_PROJECT_ID/locations/asia-northeast1/repositories/mcad-staging-repo
```

```
gcloud firestore databases create --location=asia-northeast1

terraform import google_firestore_database.main \
  projects/YOUR_PROJECT_ID/databases/(default)
```

```
terraform plan -var="project_id=YOUR_PROJECT_ID" -var="environment=staging"
terraform apply -var="project_id=YOUR_PROJECT_ID" -var="environment=staging" -auto-approve
```

デプロイされるリソース(11 個): - Cloud Storage Bucket (Frontend) - Artifact Registry Repository - Cloud Run Service (Backend) - Firestore Database - Backend Bucket (CDN) - Global IP Address - URL Map - HTTP Proxy - Forwarding Rule

```
cd ../../../services/backend

gcloud auth configure-docker asia-northeast1-docker.pkg.dev

docker build --platform linux/amd64 \
  -t asia-northeast1-docker.pkg.dev/YOUR_PROJECT_ID/mcad-staging-repo/multicloud-auto-deploy-ap
  -f Dockerfile.gcp .

docker push asia-northeast1-docker.pkg.dev/YOUR_PROJECT_ID/mcad-staging-repo/multicloud-auto-de
```

```
cd ../../infrastructure/terraform/gcp
terraform apply -var="project_id=YOUR_PROJECT_ID" -var="environment=staging" -auto-approve
```

```
gcloud run services add-iam-policy-binding mcad-staging-api \
  --region=asia-northeast1 \
  --member="allUsers" \
  --role="roles/run.invoker"
```

```
cd ../../../services/frontend

API_URL=$(gcloud run services describe mcad-staging-api \
  --region=asia-northeast1 \
  --format="value(status.url)")

VITE_API_URL=$API_URL npm run build
```

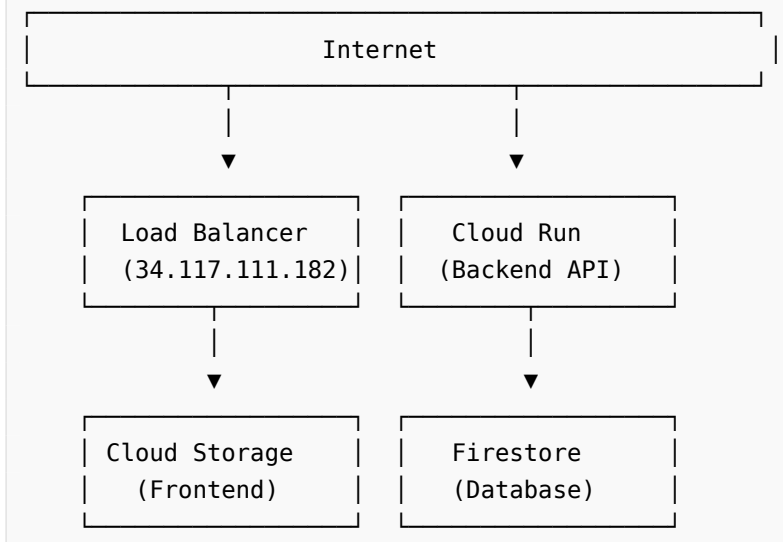
```

gsutil -m cp -r dist/* gs://mcad-staging-frontend/

gsutil -m acl ch -u AllUsers:R gs://mcad-staging-frontend/**
```

リソース	名前	目的
Cloud Storage	`mcad-staging-frontend`	フロントエンドホスティング
Artifact Registry	`mcad-staging-repo`	Docker イメージの保存
Cloud Run	`mcad-staging-api`	バックエンド API
Firestore	`(default)`	NoSQL データベース
Backend Bucket	`mcad-staging-backend`	CDN 統合
Global IP	`mcad-staging-frontend-ip`	固定 IP アドレス
URL Map	`mcad-staging-urlmap`	ルーティング設定
HTTP Proxy	`mcad-staging-http-proxy`	HTTP 終端
Forwarding Rule	`mcad-staging-http-rule`	トラフィック転送

```
Internet
   │
   ├─→ Cloud Load Balancer (CDN)
   │       └─→ Backend Bucket
   │               └─→ Cloud Storage (Frontend)
   │
   └─→ Cloud Run (Backend) ──→ Firestore
           └─→ Public URL
```




```
BACKEND_URL=$(gcloud run services describe mcad-staging-api \
  --region=asia-northeast1 \
  --format="value(status.url)")
```

```

curl $BACKEND_URL/api/health

curl $BACKEND_URL/ | jq '.'

curl -X POST $BACKEND_URL/api/messages \
  -H "Content-Type: application/json" \
  -d '{"text":"GCP Cloud Run test"}'

curl $BACKEND_URL/api/messages | jq '.'
```

```
curl -I https://storage.googleapis.com/mcad-staging-frontend/index.html
```

```
FRONTEND_IP=$(terraform output -raw frontend_cdn_ip)

curl -I http://$FRONTEND_IP/

curl -s http://$FRONTEND_IP/ | head -20
```

ブラウザで `http://$FRONTEND_IP` にアクセスして動作確認

```
gcloud firestore databases list

gcloud firestore collections list
```

症状: Cloud Run サービスが FAILED ステータス

原因と対処:

1. **イメージが見つからない**

```
gcloud artifacts docker images list \
  asia-northeast1-docker.pkg.dev/YOUR_PROJECT_ID/mcad-staging-repo

gcloud artifacts docker images describe \
  asia-northeast1-docker.pkg.dev/YOUR_PROJECT_ID/mcad-staging-repo/multicloud-auto-deploy-api:
```

2. **プラットフォームの不一致**

```
docker build --platform linux/amd64 -f Dockerfile.gcp .
```

3. **権限エラー**

```
gcloud run services describe mcad-staging-api \
  --region=asia-northeast1 \
  --format="value(spec.template.spec.serviceAccountName)"

gcloud projects get-iam-policy YOUR_PROJECT_ID \
  --flatten="bindings[].members" \
  --filter="bindings.members:serviceAccount:*"
```

症状: Load Balancer の IP アドレスにアクセスできない

対処:

1. **Backend Bucket の確認**

```
gcloud compute backend-buckets describe mcad-staging-backend \
  --format=yaml

gsutil ls gs://mcad-staging-frontend/
```

2. **Forwarding Rule の確認**

```
gcloud compute forwarding-rules describe mcad-staging-http-rule \
  --global

gcloud compute url-maps describe mcad-staging-urlmap
```

3. **Cloud Storage 権限の確認**

```
gsutil iam get gs://mcad-staging-frontend/

gsutil iam ch allUsers:objectViewer gs://mcad-staging-frontend
```

症状: バックエンドが Firestore に接続できない

対処:

1. **Firestore の初期化確認**

```
gcloud firestore databases list

gcloud firestore databases describe "(default)"
```

2. **サービスアカウント権限の確認**

```
SA_EMAIL=$(gcloud run services describe mcad-staging-api \
  --region=asia-northeast1 \
  --format="value(spec.template.spec.serviceAccountName)")

gcloud projects add-iam-policy-binding YOUR_PROJECT_ID \
  --member="serviceAccount:$SA_EMAIL" \
  --role="roles/datastore.user"
```

症状: `Quota 'BACKEND_BUCKETS' exceeded`

対処:

```
gcloud compute backend-buckets list


gcloud compute forwarding-rules delete <forwarding-rule-name> --global --quiet

gcloud compute target-http-proxies delete <proxy-name> --quiet

gcloud compute url-maps delete <urlmap-name> --quiet

gcloud compute backend-buckets delete <backend-bucket-name> --quiet
```

症状: `unauthorized: You don't have the needed permissions`

対処:

```
gcloud auth configure-docker asia-northeast1-docker.pkg.dev

gcloud artifacts repositories get-iam-policy mcad-staging-repo \
  --location=asia-northeast1

gcloud artifacts repositories add-iam-policy-binding mcad-staging-repo \
  --location=asia-northeast1 \
  --member="user:your-email@gmail.com" \
  --role="roles/artifactregistry.writer"
```

- 無料プラン: 2,000,000 リクエスト/月、180,000 vCPU 秒/月、360,000 GiB 秒/月
- CPU: 1〜8 vCPU
- メモリ: 128 MiB〜32 GiB
- リクエストタイムアウト: 最大 60 分
- 同時実行数: 最大 1,000
- 無料プラン: 1 GiB、50,000 読み取り/日、20,000 書き込み/日、20,000 削除/日
- ドキュメントサイズ: 最大 1 MiB
- トランザクション: 最大 500 ドキュメント
- インデックス: 40,000 エントリ/ドキュメント
- 無料プラン: 5 GB、5,000 Class A オペレーション/月、50,000 Class B オペレーション/月
- 静的 Web サイト: 追加費用なし
- 帯域幅: 1 TB 送信/月（中国・オーストラリア以外は無料）
- クォータ: プロジェクトあたり 3 個（デフォルト）
- クォータ増加: Google Cloud Console から申請可能
- Cloud Run Documentation
- Firestore Documentation
- Cloud Storage Documentation

- Cloud Load Balancing Documentation
- Artifact Registry Documentation
- [OK] デプロイ完了

☐ カスタムドメインの設定

☐ SSL 証明書の設定（HTTPS 対応）

☐ Cloud Armor の設定（WAF）

☐ Cloud Monitoring の設定

☐ CI/CD パイプラインの構築

☐ Backend Buckets クォータの増加申請（必要に応じて）

4.4 Production Deployment

本番環境へのデプロイ手順とベストプラクティス

☐ 環境変数の設定確認
☐ シークレットの安全な保管（AWS Secrets Manager / Azure Key Vault / GCP Secret Manager）
☐ Docker イメージのビルドとテスト
☐ ヘルスチェックエンドポイントの確認
☐ ログ設定の確認
☐ バックアップ戦略の確立

```
cd services/api
docker build -t mcad-api:latest .

cd services/frontend_reflex
docker build -t mcad-frontend:latest .
```

```
aws ecr get-login-password --region ap-northeast-1 | \
  docker login --username AWS --password-stdin <account-id>.dkr.ecr.ap-northeast-1.amazonaws.co

docker tag mcad-api:latest <account-id>.dkr.ecr.ap-northeast-1.amazonaws.com/mcad-api:latest
docker push <account-id>.dkr.ecr.ap-northeast-1.amazonaws.com/mcad-api:latest

docker tag mcad-frontend:latest <account-id>.dkr.ecr.ap-northeast-1.amazonaws.com/mcad-frontend
docker push <account-id>.dkr.ecr.ap-northeast-1.amazonaws.com/mcad-frontend:latest
```

```
cd infrastructure/pulumi/aws/simple-sns
pulumi up
```

特徴：

- サーバーレス、自動スケーリング
- 従量課金（リクエストベース）
- コールドスタート有り

AWS App Runner を使用して Reflex フロントエンドをデプロイ:

```
aws ecr create-repository --repository-name mcad-frontend --region ap-northeast-1


aws apprunner create-service \
  --service-name mcad-frontend-staging \
  --source-configuration '{
    "ImageRepository": {
      "ImageIdentifier": "<account-id>.dkr.ecr.ap-northeast-1.amazonaws.com/mcad-frontend:lates
      "ImageRepositoryType": "ECR",
      "ImageConfiguration": {
        "Port": "3002",
        "RuntimeEnvironmentVariables": {
```

```
          "API_URL": "https://your-api-gateway-url.amazonaws.com"
        }
      }
    },
    "AutoDeploymentsEnabled": true
  }' \
  --instance-configuration '{
    "Cpu": "1024",
    "Memory": "2048"
  }' \
  --region ap-northeast-1
```

特徴：

- フルマネージド
- 自動スケーリング
- HTTPS 証明書自動管理
- 月額約 $5〜(アイドル時)

```
cd infrastructure/pulumi/aws/simple-sns
pulumi up
```

特徴：

- より細かい制御
- VPC、セキュリティグループの管理
- 複数コンテナのオーケストレーション

```
aws secretsmanager create-secret \
  --name mcad-api-secrets \
  --secret-string '{
    "MINIO_ACCESS_KEY": "your-access-key",
    "MINIO_SECRET_KEY": "your-secret-key"
  }' \
  --region ap-northeast-1

aws secretsmanager get-secret-value \
  --secret-id mcad-api-secrets \
  --region ap-northeast-1
```

API (Lambda):

- Pulumi `__main__.py`の環境変数セクションで設定

Frontend (App Runner):

- App Runner サービス作成時に `RuntimeEnvironmentVariables` で設定
- または、AWS コンソールから更新

.github/workflows/deploy-production.yml:

```yaml
name: Deploy to Production

on:
  push:
    branches: [main]
    tags: ['v*']

jobs:
  deploy-api:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Configure AWS credentials
        uses: aws-actions/configure-aws-credentials@v2
        with:
          aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}
          aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          aws-region: ap-northeast-1

      - name: Login to ECR
        run: |
          aws ecr get-login-password | docker login --username AWS --password-stdin ${{ secrets

      - name: Build and push API image
        run: |
          cd services/api
          docker build -t ${{ secrets.ECR_REGISTRY }}/mcad-api:${{ github.sha }} .
          docker push ${{ secrets.ECR_REGISTRY }}/mcad-api:${{ github.sha }}

      - name: Deploy with Pulumi
        uses: pulumi/actions@v4
        with:
          work-dir: infrastructure/pulumi/aws/simple-sns
          command: up
          stack-name: production
        env:
          PULUMI_ACCESS_TOKEN: ${{ secrets.PULUMI_ACCESS_TOKEN }}

  deploy-frontend:
    runs-on: ubuntu-latest
    needs: deploy-api
    steps:
      - uses: actions/checkout@v3

      - name: Build and push Frontend image
        run: |
          # Similar to API deployment

```

```
      - name: Update App Runner service
        run: |
          aws apprunner update-service \
            --service-arn ${{ secrets.APPRUNNER_SERVICE_ARN }} \
            --source-configuration ...
```

```
aws logs tail /aws/lambda/simple-sns-api-staging --follow

aws logs tail /aws/apprunner/mcad-frontend-staging/service --follow
```

- Lambda: 実行時間、エラー率、同時実行数
- App Runner: CPU、メモリ、リクエスト数
- DynamoDB: 読み取り/書き込みキャパシティ
- S3: リクエスト数、データ転送量

```
aws lambda update-function-code \
  --function-name simple-sns-api-staging \
  --image-uri <account-id>.dkr.ecr.ap-northeast-1.amazonaws.com/mcad-api:<previous-tag>
```

```
aws apprunner list-operations --service-arn <service-arn>
aws apprunner start-deployment --service-arn <service-arn>
```

```
pulumi stack history

pulumi stack select staging
pulumi refresh
pulumi up --target <specific-resource>
```

1. **Lambda**: 適切なメモリ設定（128MB〜3GB）
2. **DynamoDB**: オンデマンドモード（低トラフィック）またはプロビジョニングモード（高トラフィック）
3. **S3**: ライフサイクルポリシーで古いファイル削除
4. **App Runner**: 最小インスタンス数の調整（0〜複数）
5. **CloudWatch**: ログ保持期間の設定（7 日〜90 日）
6. **IAM**: 最小権限の原則
7. **VPC**: プライベートサブネットの使用
8. **WAF**: API Gateway または App Runner に適用
9. **Secrets**: AWS Secrets Manager の使用
10. **暗号化**: S3 バケット暗号化、転送中の暗号化（TLS）

- AWS Lambda ベストプラクティス
- AWS App Runner ドキュメント
- Pulumi AWS プロバイダー
- Reflex デプロイメントガイド

4.5 CDN Setup

全 3 クラウド（AWS、Azure、GCP）で CDN を使用した高速コンテンツ配信の設定ガイド

このプロジェクトでは、全 3 クラウドプロバイダーで CDN を使用して React フロントエンドを配信しています。

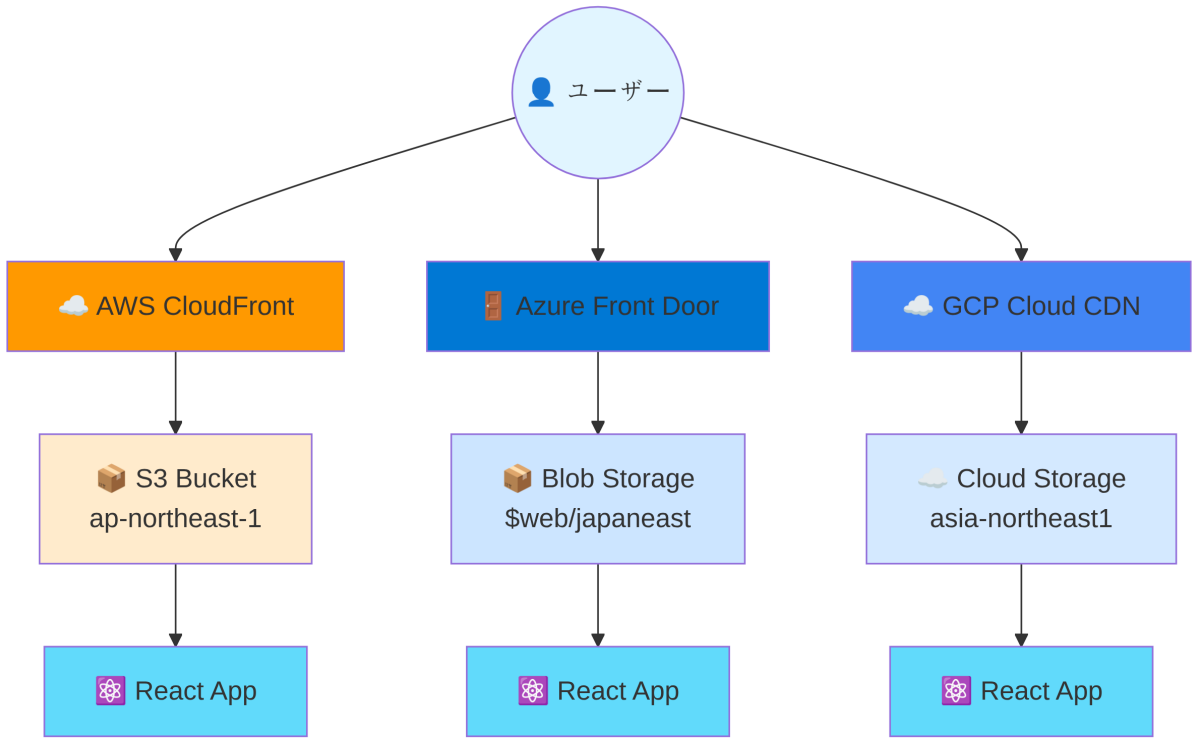


図 4.2: Diagram 1

クラウド	CDN URL	オリジン	管理方法
AWS	https://dx3l4mbwg1ade.cloudfront.net	S3 (ap-northeast-1)	手動
Azure	https://multicloud-frontend-f9cvamfnauexasd8.z01.azurefd.net	Blob Storage (japaneast)	手動

クラウド	CDN URL	オリジン	管理方法
GCP	http://34.120.43.83	Cloud Storage (asia-northeast1)	手動

クラウド	CDN URL	オリジン	管理方法
AWS	https://d1tf3uumcm4bo1.cloudfront.net	S3 (ap-northeast-1)	Pulumi
Azure	https://mcad-staging-d45ihd-dseygrc9c3a3htgj.z01.azurefd.net	Blob Storage (japaneast)	Pulumi
GCP	http://34.117.111.182	Cloud Storage (asia-northeast1)	Pulumi

---

全 3 クラウドの CDN リソースを Pulumi で管理できます。Infrastructure as Code で一貫性のある環境を構築。

```
pulumi version

pulumi login --local
```

```
cd infrastructure/pulumi/aws

pip install -r requirements.txt

pulumi stack select staging

pulumi preview

pulumi up

pulumi stack output cloudfront_url
```

作成されるリソース:

- CloudFront Distribution
- Origin Access Identity (OAI)
- S3 Bucket Policy（OAI アクセス設定）
- カスタムエラーレスポンス（SPA 対応）

注意: Lambda 関数コードは`ignore_changes`で除外されています。コードは別途`scripts/deploy-lambda-aws.sh`でデプロイしてください。

```
cd infrastructure/pulumi/azure

pip install -r requirements.txt
```

```

pulumi stack select staging

pulumi preview

pulumi up

pulumi stack output frontdoor_url
```

作成されるリソース:

- Azure Front Door Profile (Standard)
- Front Door Endpoint
- Origin Group（ロードバランシング + ヘルスチェック）
- Origin（Storage Account）
- Route（HTTPS 強制、パターンマッチング）

注意: Function App は手動管理です。Pulumi では管理しません。

```
cd infrastructure/pulumi/gcp

pip install -r requirements.txt

pulumi config set gcp:project ashnova
pulumi config set gcp:region asia-northeast1

pulumi stack select staging

pulumi preview

pulumi up

pulumi stack output cdn_url
```

作成されるリソース:

- Global Address（外部 IP）
- Backend Bucket（Cloud CDN 有効化）
- URL Map
- Target HTTP Proxy
- Global Forwarding Rule

```
(cd infrastructure/pulumi/aws && pulumi up --yes)

(cd infrastructure/pulumi/azure && pulumi up --yes)

(cd infrastructure/pulumi/gcp && pulumi up --yes)
```

```
cd infrastructure/pulumi/aws
```

```
pulumi stack
pulumi stack output

cd infrastructure/pulumi/azure
pulumi stack
pulumi stack output

cd infrastructure/pulumi/gcp
pulumi stack
pulumi stack output
```

```
aws cloudfront list-distributions \
  --query 'DistributionList.Items[].{Id:Id,DomainName:DomainName,Origin:Origins.Items[0].Domain
  --output table

aws cloudfront get-distribution \
  --id E2GDU7Y7UGDV3S
```

項目	値
Distribution ID	E2GDU7Y7UGDV3S
Domain Name	dx3l4mbwg1ade.cloudfront.net
Origin	multicloud-auto-deploy-staging-frontend.s3.ap-northeast-1.amazonaws.co
Price Class	PriceClass_100
Default Root Object	index.html
HTTP Version	http2

```
aws cloudfront create-invalidation \
  --distribution-id E2GDU7Y7UGDV3S \
  --paths "/*"

aws cloudfront create-invalidation \
  --distribution-id E2GDU7Y7UGDV3S \
  --paths "/index.html" "/assets/*"

aws cloudfront get-invalidation \
  --distribution-id E2GDU7Y7UGDV3S \
  --id <INVALIDATION_ID>
```

```
cd services/frontend_react

echo "VITE_API_URL=https://z42qmqdqac.execute-api.ap-northeast-1.amazonaws.com" > .env
npm run build

aws s3 sync dist/ s3://multicloud-auto-deploy-staging-frontend/ --delete
```

```

aws cloudfront create-invalidation \
  --distribution-id E2GDU7Y7UGDV3S \
  --paths "/*"
```

```
az afd profile show \
  --profile-name multicloud-frontend-afd \
  --resource-group multicloud-auto-deploy-staging-rg

az afd endpoint show \
  --profile-name multicloud-frontend-afd \
  --endpoint-name multicloud-frontend \
  --resource-group multicloud-auto-deploy-staging-rg

az afd origin show \
  --profile-name multicloud-frontend-afd \
  --origin-group-name storage-origin-group \
  --origin-name storage-origin \
  --resource-group multicloud-auto-deploy-staging-rg
```

```
RESOURCE_GROUP="multicloud-auto-deploy-staging-rg"
PROFILE_NAME="multicloud-frontend-afd"
ENDPOINT_NAME="multicloud-frontend"
STORAGE_HOST="mcadwebd45ihd.z11.web.core.windows.net"

az afd profile create \
  --profile-name $PROFILE_NAME \
  --resource-group $RESOURCE_GROUP \
  --sku Standard_AzureFrontDoor

az afd endpoint create \
  --profile-name $PROFILE_NAME \
  --resource-group $RESOURCE_GROUP \
  --endpoint-name $ENDPOINT_NAME \
  --enabled-state Enabled

az afd origin-group create \
  --profile-name $PROFILE_NAME \
  --origin-group-name storage-origin-group \
  --resource-group $RESOURCE_GROUP \
  --probe-request-type GET \
  --probe-protocol Https \
  --probe-interval-in-seconds 100 \
  --probe-path /

az afd origin create \
  --profile-name $PROFILE_NAME \
  --origin-group-name storage-origin-group \
  --origin-name storage-origin \
```

```
  --resource-group $RESOURCE_GROUP \
  --host-name $STORAGE_HOST \
  --origin-host-header $STORAGE_HOST \
  --priority 1 \
  --weight 1000 \
  --enabled-state Enabled \
  --http-port 80 \
  --https-port 443

az afd route create \
  --profile-name $PROFILE_NAME \
  --endpoint-name $ENDPOINT_NAME \
  --route-name default-route \
  --resource-group $RESOURCE_GROUP \
  --origin-group storage-origin-group \
  --supported-protocols Http Https \
  --https-redirect Enabled \
  --forwarding-protocol HttpsOnly \
  --patterns-to-match "/*"
```

項目	値
Profile Name	multicloud-frontend-afd
Endpoint	multicloud-frontend
Host Name	multicloud-frontend-f9cvamfnauexasd8.z01.azurefd.net
Origin	mcadwebd45ihd.z11.web.core.windows.net
SKU	Standard_AzureFrontDoor
HTTPS Redirect	Enabled

```
cd services/frontend_react

echo "VITE_API_URL=https://multicloud-auto-deploy-staging-func-d8a2guhfere0etcq.japaneast-01.az
npm run build

az storage blob upload-batch \
  --account-name mcadwebd45ihd \
  --auth-mode key \
  --destination '$web' \
  --source dist/ \
  --overwrite \
  --pattern "assets/*" \
  --content-cache-control "public, max-age=31536000, immutable"

az storage blob upload \
  --account-name mcadwebd45ihd \
  --auth-mode key \
  --container-name '$web' \
```

```
  --file dist/index.html \
  --name index.html \
  --content-cache-control "public, max-age=0, must-revalidate" \
  --overwrite
```

注意: Azure Front Door は自動的にキャッシュを管理しますが、伝播には 5-10 分かかる場合があります。

```
gcloud compute backend-buckets describe multicloud-frontend-backend

gcloud compute url-maps describe multicloud-frontend-urlmap

gcloud compute addresses describe multicloud-frontend-ip --global

gcloud compute forwarding-rules describe multicloud-frontend-forwarding-rule --global
```

```
BUCKET_NAME="ashnova-multicloud-auto-deploy-staging-frontend"
BACKEND_BUCKET="multicloud-frontend-backend"
URL_MAP="multicloud-frontend-urlmap"
HTTP_PROXY="multicloud-frontend-http-proxy"
IP_NAME="multicloud-frontend-ip"
FORWARDING_RULE="multicloud-frontend-forwarding-rule"

gcloud compute backend-buckets create $BACKEND_BUCKET \
  --gcs-bucket-name=$BUCKET_NAME \
  --enable-cdn \
  --cache-mode=CACHE_ALL_STATIC \
  --default-ttl=3600 \
  --max-ttl=86400

gcloud compute url-maps create $URL_MAP \
  --default-backend-bucket=$BACKEND_BUCKET

gcloud compute target-http-proxies create $HTTP_PROXY \
  --url-map=$URL_MAP

gcloud compute addresses create $IP_NAME \
  --ip-version=IPV4 \
  --global

gcloud compute forwarding-rules create $FORWARDING_RULE \
  --address=$IP_NAME \
  --global \
  --target-http-proxy=$HTTP_PROXY \
  --ports=80
```

項目	値
Global IP	34.120.43.83
Backend Bucket	multicloud-frontend-backend
GCS Bucket	ashnova-multicloud-auto-deploy-staging-frontend
Cache Mode	CACHE_ALL_STATIC
Default TTL	3600s (1 hour)
Max TTL	86400s (24 hours)

```
gcloud compute url-maps invalidate-cdn-cache multicloud-frontend-urlmap \
  --path "/*" \
  --async

gcloud compute url-maps invalidate-cdn-cache multicloud-frontend-urlmap \
  --path "/index.html" \
  --path "/assets/*" \
  --async
```

```
cd services/frontend_react

echo "VITE_API_URL=https://multicloud-auto-deploy-staging-api-899621454670.asia-northeast1.run
npm run build

gcloud storage rsync --recursive --delete-unmatched-destination-objects \
  --cache-control="public, max-age=31536000, immutable" \
  dist/assets/ gs://ashnova-multicloud-auto-deploy-staging-frontend/assets/

gcloud storage cp dist/vite.svg \
  gs://ashnova-multicloud-auto-deploy-staging-frontend/vite.svg \
  --cache-control="public, max-age=31536000, immutable"

gcloud storage cp dist/index.html \
  gs://ashnova-multicloud-auto-deploy-staging-frontend/index.html \
  --cache-control="public, max-age=0, must-revalidate"

gcloud compute url-maps invalidate-cdn-cache multicloud-frontend-urlmap \
  --path "/*" --async
```

```
time curl -s https://dx3l4mbwg1ade.cloudfront.net/ > /dev/null

time curl -s https://multicloud-frontend-f9cvamfnauexasd8.z01.azurefd.net/ > /dev/null

time curl -s http://34.120.43.83/index.html > /dev/null
```

クラウド	レスポンスタイム	キャッシュヒット	備考
AWS CloudFront	0.702 秒	Miss → Hit	HTTP/2 対応
GCP Cloud CDN	**0.109 秒**	Hit	[TROPHY] 最速
Azure Front Door	伝播中	-	HTTPS Redirect 有効

```
curl -I https://dx3l4mbwg1ade.cloudfront.net/

curl -I https://multicloud-frontend-f9cvamfnauexasd8.z01.azurefd.net/

curl -I http://34.120.43.83/index.html
```

---

1. **HTML（index.html）**:
   - Cache-Control: `public, max-age=0, must-revalidate`
   - 常に最新版を取得
2. **静的アセット（JS/CSS/画像）**:
   - Cache-Control: `public, max-age=31536000, immutable`
   - ファイル名にハッシュを含めることで変更を検出
3. **デプロイ後のキャッシュクリア**:
   - 全 CDN でキャッシュ無効化を実行
   - index.html は最優先で無効化

```
echo "=== AWS CloudFront ==="
aws cloudfront create-invalidation \
  --distribution-id E2GDU7Y7UGDV3S \
  --paths "/*"

echo -e "\n=== GCP Cloud CDN ==="
gcloud compute url-maps invalidate-cdn-cache multicloud-frontend-urlmap \
  --path "/*" --async

echo -e "\n=== Azure Front Door ==="
echo "Azure Front Doorは自動的にキャッシュを管理します（5-10分で伝播）"
```

原因: キャッシュが残っている

解決策:

```
aws cloudfront create-invalidation \
```

```
  --distribution-id E2GDU7Y7UGDV3S \
  --paths "/*"
```

原因: Origin 設定が間違っているか、伝播中

解決策:

```
az afd origin show \
  --profile-name multicloud-frontend-afd \
  --origin-group-name storage-origin-group \
  --origin-name storage-origin \
  --resource-group multicloud-auto-deploy-staging-rg
```

原因: Cache-Control ヘッダーが設定されていない

解決策:

```
gcloud storage cp dist/index.html \
  gs://ashnova-multicloud-auto-deploy-staging-frontend/index.html \
  --cache-control="public, max-age=0, must-revalidate"
```

- エンドポイント一覧 - 全 CDN URL とテスト方法
- デプロイガイド - 初回デプロイ手順
- トラブルシューティング - よくある問題と解決策

Chapter 5

CI/CD Setup

GitHub Actions による自動デプロイの設定ガイド

このプロジェクトは、GitHub Actions を使用して AWS、Azure、GCP への自動デプロイを実現しています。

- **自動デプロイ**: `main`ブランチへのプッシュ時
- **手動デプロイ**: GitHub Actions UI からワークフロー実行時
- **対象パス**: `services/**`または`infrastructure/terraform/**`の変更時

各クラウドプロバイダーに必要な Secrets を GitHub リポジトリに設定してください。

1. GitHub リポジトリのページを開く
2. **Settings** → **Secrets and variables** → **Actions**
3. **New repository secret** をクリック

Secret 名	説明	取得方法
`AWS_ACCESS_KEY_ID`	AWS アクセスキー ID	IAM ユーザーから取得
`AWS_SECRET_ACCESS_KEY`	AWS シークレットアクセスキー	IAM ユーザーから取得

取得手順:

```
aws iam create-access-key --user-name satoshi
```

必要な権限:

- S3: フルアクセス（バケット作成・削除・アップロード）
- CloudFront: 管理権限
- Lambda: フルアクセス
- API Gateway: フルアクセス
- DynamoDB: フルアクセス
- IAM: ロール作成・ポリシーアタッチ

Secret 名	説明	取得方法
`AZURE_CREDENTIALS`	Azure 認証情報（JSON）	Service Principal から取得
`ARM_CLIENT_ID`	Service Principal のクライアント ID	Service Principal から取得
`ARM_CLIENT_SECRET`	Service Principal のシークレット	Service Principal から取得
`ARM_SUBSCRIPTION_ID`	Azure サブスクリプション ID	`az account show`
`ARM_TENANT_ID`	Azure テナント ID	`az account show`
`AZURE_API_URL`	バックエンド API URL（オプション）	デプロイ後に設定
`AZURE_ACR_LOGIN_SERVER`	Container Registry のログインサーバー URL（オプション）	`az acr list`

取得手順:

```
az account show --query "{SubscriptionId:id, TenantId:tenantId, Name:name}" --output table

SUBSCRIPTION_ID=$(az account show --query id --output tsv)

az ad sp create-for-rbac \
  --name "github-actions-deploy" \
  --role Contributor \
  --scopes /subscriptions/$SUBSCRIPTION_ID \
  --sdk-auth
```

AZURE_CREDENTIALS の形式:

```
{
  "clientId": "00000000-0000-0000-0000-000000000000",
  "clientSecret": "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx",
  "subscriptionId": "00000000-0000-0000-0000-000000000000",
  "tenantId": "00000000-0000-0000-0000-000000000000"
}
```

AZURE_ACR_LOGIN_SERVER の取得:

```
az acr list --query "[].{Name:name, LoginServer:loginServer}" --output table
```

Service Principal に ACR アクセス権を付与:

```
ACR_RESOURCE_ID=$(az acr show --name YOUR_ACR_NAME --query id --output tsv)

az role assignment create \
  --assignee YOUR_CLIENT_ID \
  --role AcrPush \
  --scope $ACR_RESOURCE_ID
```

注意: 現在のワークフローでは、ACR の名前は Terraform 出力から自動的に取得されるため、`AZURE_ACR_LOGIN_SERVER`は必須ではありません。直接 ACR を指定したい場合のオプションです。

Secret 名	説明	取得方法
`GCP_CREDENTIALS` (旧称: `GCP_SA_KEY`)	GCP サービスアカウントキー (JSON)	サービスアカウントから取得
`GCP_PROJECT_ID`	GCP プロジェクト ID	`gcloud config get-value project`

取得手順:

```
gcloud config get-value project

gcloud iam service-accounts list

gcloud iam service-accounts create github-actions-deploy \
  --display-name="GitHub Actions Deploy"

gcloud projects add-iam-policy-binding YOUR_PROJECT_ID \
  --member="serviceAccount:github-actions-deploy@YOUR_PROJECT_ID.iam.gserviceaccount.com" \
  --role="roles/editor"

gcloud iam service-accounts keys create key.json \
  --iam-account=github-actions-deploy@YOUR_PROJECT_ID.iam.gserviceaccount.com

cat key.json

rm key.json
```

GCP_CREDENTIALS (GCP_SA_KEY) の形式:

```
{
  "type": "service_account",
  "project_id": "your-project-id",
  "private_key_id": "key-id",
  "private_key": "-----BEGIN PRIVATE KEY-----\n...\n-----END PRIVATE KEY-----\n",
  "client_email": "github-actions-deploy@your-project-id.iam.gserviceaccount.com",
  "client_id": "123456789",
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",
  "token_uri": "https://oauth2.googleapis.com/token",
```

```
  "auth_provider_x509_cert_url": "https://www.googleapis.com/oauth2/v1/certs",
  "client_x509_cert_url": "https://www.googleapis.com/robot/v1/metadata/x509/..."
}
```

既存のサービスアカウントから新しいキーを作成する場合:

```
gcloud iam service-accounts list

gcloud iam service-accounts keys create key.json \
  --iam-account=YOUR_SERVICE_ACCOUNT_EMAIL
```

必要な権限:

- Cloud Run: 管理者
- Artifact Registry: 管理者
- Cloud Storage: 管理者
- Firestore: 管理者
- Compute Engine: 管理者（Load Balancer 用）
- IAM: Service Account Admin

トリガー:

- `main`ブランチへのプッシュ（`services/**`または`infrastructure/terraform/aws/**`の変更）
- 手動実行

ステップ: 1. AWS 認証情報の設定 2. Node.js・Python のセットアップ 3. デプロイスクリプトの実行 4. 成功/失敗通知

実行時間: 約 5-10 分

トリガー:

- `main`ブランチへのプッシュ（`services/**`または`infrastructure/terraform/azure/**`の変更）
- 手動実行

ステップ: 1. Azure ログイン 2. Node.js・Python・Terraform のセットアップ 3. フロントエンドのビルド 4. Terraform でインフラデプロイ 5. Docker イメージのビルドとプッシュ 6. Container App の更新 7. フロントエンドの Storage Account へのアップロード 8. 成功/失敗通知

実行時間: 約 10-15 分

トリガー:

- `main`ブランチへのプッシュ（`services/**`または`infrastructure/terraform/gcp/**`の変更）
- 手動実行

ステップ: 1. GCP 認証 2. Node.js・Python・Terraform のセットアップ 3. Terraform でインフラデプロイ 4. Docker イメージのビルドとプッシュ 5. Cloud Run サービスのデプロイ 6. IAM ポリシーの設定 7. フロントエンドのビルドとデプロイ 8. Cloud Storage へのアップロード 9. 成功/失敗通知

実行時間: 約 10-15 分

---

1. GitHub リポジトリの **Actions** タブを開く
2. 実行したいワークフローを選択（例: Deploy to AWS）
3. **Run workflow** ボタンをクリック
4. environment を選択（`staging`または`production`）
5. **Run workflow** で実行

act というツールを使用してローカルで GitHub Actions を実行できます：

```
brew install act  # macOS
curl https://raw.githubusercontent.com/nektos/act/master/install.sh | sudo bash

act -W .github/workflows/deploy-aws.yml

act -W .github/workflows/deploy-aws.yml \
  --secret AWS_ACCESS_KEY_ID=xxx \
  --secret AWS_SECRET_ACCESS_KEY=xxx
```

> [TIP] **詳細なトラブルシューティング情報は TROUBLESHOOTING.md を参照してください**
>
> 以下の問題と解決策が記載されています：- Azure 認証問題（Service Principal、Terraform Wrapper、CLI 認証競合）- GCP リソース競合（GCS Backend、既存リソースインポート）- フロントエンド API 接続問題（ビルド順序、API URL 設定）- 権限エラー（IAM、RBAC、Service Account）

症状: `Error: Could not load credentials from any providers`

対処:

```
aws iam get-user --user-name satoshi
aws iam list-attached-user-policies --user-name satoshi
```

症状: `Error: AuthenticationFailed`

対処:

```
az login --service-principal \
  -u $ARM_CLIENT_ID \
  -p $ARM_CLIENT_SECRET \
  --tenant $ARM_TENANT_ID

```

```
az role assignment list \
  --assignee $ARM_CLIENT_ID \
  --output table
```

よくある問題:

- “Authenticating using the Azure CLI is only supported as a User”→ 詳細
- Terraform Wrapper による環境変数干渉 → 詳細

症状: `Error: google: could not find default credentials`

対処:

```
gcloud auth activate-service-account \
  --key-file=key.json

gcloud projects get-iam-policy YOUR_PROJECT_ID \
  --flatten="bindings[].members" \
  --filter="bindings.members:serviceAccount:github-actions-deploy@*"
```

よくある問題:

- “Error 409: Resource already exists”→ 詳細
- GCS Backend の設定とリソースインポート → 詳細

症状: メッセージ送信が「送信中」のまま固まる

原因: フロントエンドビルド時に API URL が正しく設定されていない

対処: 詳細な解決策

重要: フロントエンドは必ずインフラデプロイ**後**にビルドすること

症状: `denied: requested access to the resource is denied`

対処:

Azure:

```
az acr login --name <ACR_NAME>

az role assignment list \
  --assignee $ARM_CLIENT_ID \
  --scope /subscriptions/$ARM_SUBSCRIPTION_ID/resourceGroups/*/providers/Microsoft.ContainerReg
```

GCP:

```
gcloud auth configure-docker asia-northeast1-docker.pkg.dev

```

```
gcloud projects get-iam-policy YOUR_PROJECT_ID \
  --flatten="bindings[].members" \
  --filter="bindings.role:roles/artifactregistry.writer"
```

症状: `Error: Backend initialization required`

対処:

```
cd infrastructure/terraform/aws  # or azure, gcp
terraform init -upgrade

terraform state pull > backup.tfstate
terraform init -reconfigure
```

症状: 毎回"Resource already exists"エラーが発生

解決策: GCS Backend を使用して永続的な State を設定

```
gcloud storage buckets create gs://multicloud-auto-deploy-tfstate-gcp \
  --location=asia-northeast1 \
  --uniform-bucket-level-access

./scripts/import-gcp-resources.sh
```

詳細は TROUBLESHOOTING.md - GCP リソース競合 を参照

各ワークフローファイルのenvセクションで設定をカスタマイズできます：

```
env:
  AWS_REGION: us-east-1        # 変更可能
  AZURE_REGION: japaneast      # 変更可能
  GCP_REGION: asia-northeast1  # 変更可能
  NODE_VERSION: "18"
  PYTHON_VERSION: "3.11"
```

---

1. **最小権限の原則**
   - 各 Service Account や IAM ユーザーには必要最小限の権限のみ付与
2. **キーのローテーション**
   - 定期的にアクセスキーやサービスアカウントキーを更新
3. **環境の分離**
   - staging 環境と production 環境で Secrets を分ける
4. **監査ログ**
   - デプロイアクティビティのログを記録・監視

5. **ブランチ保護**

   - `main`ブランチへの直接プッシュを制限
   - プルリクエストレビューを必須化

☐ GitHub Secrets の設定
☐ 初回の手動デプロイテスト
☐ ブランチ保護ルールの設定
☐ Slack/Discord 等への通知統合
☐ 自動テストの追加
☐ ステージング環境での E2E テスト

5.1 CI/CD Test Results

2026-02-14

GitHub Actions ワークフローの機能検証を実施し、AWS デプロイメントパイプラインの完全動作を確認しました。

- **ローカル環境**: Dev Container (Debian 12)
  - Node.js: v20.20.0
  - Python: 3.12.12
  - AWS CLI: 2.33.22
  - Azure CLI: 2.83.0
  - GCP CLI: 556.0.0
  - GitHub CLI: v2.86.0
- **CI/CD 環境**: GitHub Actions (ubuntu-latest)
  - Node.js: v18.20.8
  - Python: 3.12.12
  - AWS SDK 付属

問題: ワークフローが `services/backend` と `services/frontend` を参照していたが、実際は `services/api` と `services/frontend_react`

修正内容:

- `.github/workflows/deploy-aws.yml`: 全パス修正
- `.github/workflows/deploy-azure.yml`: 全パス修正
- `.github/workflows/deploy-gcp.yml`: 全パス修正

コミット: `621bb55` - "fix: Update CI/CD workflows to use correct directory structure"

問題: Lambda パッケージング時に `src/*` をコピーしようとしたが、実際のソースは `app/` ディレクトリに配置

修正内容:

- `deploy-aws.yml`: `cp -r src/* package/` → `cp -r app/* package/`

コミット: `7aa7bc9` - "fix: Correct source directory from src to app in AWS workflow"

問題: Azure/GCP ワークフローが `Dockerfile.azure` と `Dockerfile.gcp` を参照していたが存在しない

修正内容:

- 両ワークフローで標準の `Dockerfile` を使用するよう変更
- `-f Dockerfile.azure` → `-f Dockerfile`
- `-f Dockerfile.gcp` → `-f Dockerfile`

コミット: `7aa7bc9`

問題: `apigateway:GET` 権限がないため API Gateway エンドポイント取得時にワークフロー全体が失敗

修正内容:

- API Gateway エンドポイント取得をオプショナルに変更
- エラー時はデフォルトエンドポイントを使用して続行
- `set +e` で一時的にエラー無視、その後 `set -e` で再有効化

コミット: `128d13d` - "fix: Make API Gateway endpoint lookup non-blocking in CI/CD"

実行 ID: 22017199739
実行時間: 40 秒
結果: [OK] 成功

成功したステップ: 1. [OK] Package Backend (Lambda パッケージング) - 依存関係インストール: fastapi, pydantic, mangum - ソースコピー: `app/*` → `package/` - ZIP 作成: 約 63MB

2. [OK] Update Lambda Function
   - S3 アップロード成功
   - Lambda 関数更新成功
   - API Gateway エンドポイント取得（権限エラーは回避）
3. [OK] Build Frontend
   - React ビルド成功
   - 成果物サイズ: 288K
4. [OK] Deploy Frontend to S3
   - 全ファイルアップロード成功
   - 古い Lambda ZIP ファイル削除
5. [OK] Invalidate CloudFront Cache
   - キャッシュ無効化成功
   - Invalidation ID: I1N2O7ZWP4RIDXDDZH3BHUEHH4

理由: AWS デプロイ検証に注力、Azure/GCP は Terraform インフラ未構築

```
./scripts/test-cicd.sh
```

```
総テスト数: 15
成功: 23
失敗: 0
成功率: 100%
```

- [OK] Node.js インストール確認 (v20.20.0)
- [OK] Python インストール確認 (3.12.12)
- [OK] AWS CLI インストール確認 (2.33.22)
- [OK] Azure CLI インストール確認 (2.83.0)
- [OK] GCP CLI インストール確認 (556.0.0)
- [OK] GitHub CLI インストール確認 (2.86.0)

- [OK] ワークフローファイル存在確認（4 ファイル）
- [OK] Lambda パッケージングシミュレーション（63MB ZIP）
- [OK] React ビルドシミュレーション（288K）
- [OK] AWS 認証情報確認
- [OK] Lambda 関数存在確認（multicloud-auto-deploy-staging-api）
- [OK] S3 バケット存在確認（multicloud-auto-deploy-staging-frontend）

機能: CI/CD 環境の包括的な検証

- 開発ツールのバージョン確認
- ワークフローファイル構文チェック
- ビルドプロセスのシミュレーション
- デプロイターゲットの存在確認

使用方法:

```
./scripts/test-cicd.sh
```

機能: 手動ワークフロー実行

- ワークフロー選択（aws/azure/gcp/multicloud）
- 環境選択（staging/production）
- リアルタイム実行監視

使用方法:

```
./scripts/trigger-workflow.sh aws staging
```

機能: パイプライン監視とレポート

- ワークフロー一覧表示
- 実行履歴（直近 10 件）
- 失敗実行の詳細
- 成功率統計
- ワークフロー別統計

使用方法:

```
./scripts/monitor-cicd.sh
```

- **総実行数**: 20
- **成功**: 1 (5.0%)
- **失敗**: 18 (90.0%)
- **キャンセル**: 0
- **キュー待ち**: 1 (5.0%)

ワークフロー	成功	失敗	成功率
deploy-aws.yml	1	9	10.0%
deploy-azure.yml	0	10	0.0%
deploy-gcp.yml	0	10	0.0%
deploy-multicloud.yml	0	6	0.0%

注: 高い失敗率は、問題発見・修正の反復プロセスによるもの。最終修正後の実行は 100% 成功。

API Gateway エンドポイントの自動取得を有効にするには、以下の権限を追加:

```
{
  "Effect": "Allow",
  "Action": [
    "apigateway:GET"
  ],
  "Resource": "arn:aws:apigateway:*::/apis"
}
```

現状: 権限がなくてもデフォルトエンドポイントにフォールバックするため、必須ではない

Azure と GCP のデプロイメントをテストするには: - Terraform でインフラ構築 - GitHub Secrets に認証情報を設定 - ワークフローを手動実行

```
./scripts/monitor-cicd.sh

gh run view <run-id> --log
```

```
[OK] AWS deployment succeeded!

Invalidation: {
  "Id": "I1N2O7ZWP4RIDXDDZH3BHUEHH4",
  "Status": "InProgress",
  "CreateTime": "2026-02-14T12:18:58.489000+00:00"
}
```

- **Lambda 関数**: multicloud-auto-deploy-staging-api
  - Runtime: Python 3.12
  - Memory: 512MB
  - 最新デプロイ: 2026-02-14 12:18 UTC
- **S3 バケット**: multicloud-auto-deploy-staging-frontend
  - フロントエンドファイル: 約 8.8MB
  - キャッシュポリシー: 1 年（静的アセット）、no-cache（index.html）
- **CloudFront ディストリビューション**: `E2GDU7Y7UGDV3S`
  - キャッシュ無効化: 完了

- **API Gateway**: `52z731x570.execute-api.ap-northeast-1.amazonaws.com`
  - HTTP API (バージョン 2)
  - Lambda 統合: 正常

[OK] **AWS CI/CD パイプラインは完全に機能しています**

- 全てのビルド・デプロイステップが成功
- Lambda、S3、CloudFront が正常に更新
- 問題の特定と修正プロセスが体系化
- 包括的な監視・管理ツールが整備

次のステップとして、Azure と GCP のインフラ構築とデプロイメントテストを推奨します。

Chapter 6

Tools Reference

最終更新: 2026-02-15

用途: AWS 環境への Terraform ベースデプロイ

使用方法:

```
./scripts/deploy-aws.sh
```

機能:

- Terraform 初期化・適用
- Lambda 関数のデプロイ
- API Gateway 設定
- DynamoDB 構築

前提条件:

- AWS CLI 認証設定済み
- Terraform 1.5+ インストール済み

関連ファイル:

- `infrastructure/terraform/aws/`

用途: Azure 環境へのデプロイ

使用方法:

```
./scripts/deploy-azure.sh
```

機能:

- Azure Functions (Flex Consumption) デプロイ
- Cosmos DB 設定
- Blob Storage 設定
- Azure Front Door 設定

前提条件:

- Azure CLI 認証設定済み
- リソースグループ作成済み

関連ファイル:

- `.github/workflows/deploy-azure.yml`

用途: GCP 環境へのデプロイ

使用方法:

```
./scripts/deploy-gcp.sh
```

機能:

- Cloud Run デプロイ
- Firestore 設定
- Cloud Storage 設定
- Cloud CDN 設定

前提条件:

- gcloud CLI 認証設定済み
- プロジェクト ID 設定済み

関連ファイル:

- `infrastructure/terraform/gcp/`

用途: AWS 環境への Pulumi ベースデプロイ（Python Full Stack 版）

使用方法:

```
./scripts/deploy-aws-pulumi.sh [staging|production]
```

パラメータ:

- `$1`: 環境名（デフォルト: staging）

機能:

- Pulumi stack の初期化・更新
- CloudFront + Lambda + API Gateway デプロイ
- フロントエンドビルド・デプロイ
- ヘルスチェック実行

前提条件:

- Pulumi 3.0+ インストール済み
- Python 3.12+
- Node.js 18+

環境変数:

- `AWS_REGION`: デプロイ先リージョン（デフォルト: ap-northeast-1）

関連ファイル:

- `infrastructure/pulumi/aws/simple-sns/`

用途: AWS 向けフロントエンドのみデプロイ

使用方法:

```
./scripts/deploy-frontend-aws.sh
```

機能:

- React フロントエンドビルド
- S3 へのアップロード
- CloudFront キャッシュ無効化

前提条件:

- S3 バケット作成済み
- CloudFront Distribution 作成済み

用途: マルチクラウド E2E テストスイート

使用方法:

```
./scripts/test-e2e.sh
```

テストカバレッジ:

- **Total**: 18 テスト（3 環境 × 6 テスト）
- AWS: Lambda + API Gateway + DynamoDB
- GCP: Cloud Run + Firestore
- Azure: Functions Flex + Cosmos DB

テスト項目: 1. Health Check 2. Create Message 3. List Messages 4. Get Message by ID 5. Update Message 6. Delete Message

出力例:

```
============================================================
        Multi-Cloud E2E Test Suite
============================================================

AWS: 6/6 tests passed [OK]
GCP: 6/6 tests passed [OK]
Azure: 6/6 tests passed [OK]

Total Tests: 18
Passed: 18
All tests passed! [OK]
```

関連ドキュメント:

- README.md - E2E テストスイート

用途: 全環境のエンドポイント疎通確認

使用方法:

```
./scripts/test-endpoints.sh
```

テスト対象:

- AWS API / Frontend
- Azure API / Frontend
- GCP API / Frontend

機能:

- HTTP ステータスコード確認
- レスポンス内容検証
- レスポンスタイム測定

用途: 単一環境の API 統合テスト

使用方法:

```
./scripts/test-api.sh -e <API_ENDPOINT> [--verbose]
```

パラメータ:

- `-e, --endpoint`: API エンドポイント URL（必須）
- `--verbose`: 詳細モード

使用例:

```
./scripts/test-api.sh -e https://YOUR_API_ID.execute-api.ap-northeast-1.amazonaws.com

./scripts/test-api.sh -e https://YOUR_API_ID.execute-api.ap-northeast-1.amazonaws.com --verbose
```

テスト項目:

- ヘルスチェック
- メッセージ CRUD 操作
- ページネーション
- エラーハンドリング
- バリデーション

用途: マルチクラウドデプロイメント統合テスト

使用方法:

```
./scripts/test-deployments.sh
```

機能:

- 全クラウドの API 疎通確認
- フロントエンド疎通確認
- テスト結果サマリー表示

用途: GitHub Actions ワークフローのローカル検証

使用方法:

```
./scripts/test-cicd.sh
```

検証項目: 1. 環境検証（Terraform, Node.js, Python 等）2. ワークフローファイル構文確認 3. AWS Lambda パッケージングテスト 4. フロントエンドビルドテスト 5. AWS 認証情報確認 6. GitHub Secrets 確認 7. デプロイターゲット確認

用途: GitHub Secrets 管理（統合版）

使用方法:

```
./scripts/manage-github-secrets.sh

./scripts/manage-github-secrets.sh --auto
```

モード:

- **ガイドモード**: 手動設定用のコマンド表示
- **自動設定モード**: ローカル環境変数から自動設定

対象 Secrets:

- AWS: `AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`
- Azure: `AZURE_CREDENTIALS`, `AZURE_*`
- GCP: `GCP_CREDENTIALS`, `GCP_PROJECT_ID`

前提条件:

- GitHub CLI (`gh`) インストール済み
- リポジトリへのアクセス権限

用途: CloudWatch 監視設定（AWS 専用）

使用方法:

```
./scripts/setup-monitoring.sh

ALERT_EMAIL=your@email.com ./scripts/setup-monitoring.sh
```

設定内容:

- SNS トピック作成
- Lambda エラー/スロットリング/実行時間/同時実行数アラーム
- API Gateway 5XX エラーアラーム
- DynamoDB スロットリングアラーム
- CloudWatch Logs メトリクスフィルター
- CloudWatch ダッシュボード作成

環境変数:

- `ALERT_EMAIL`: アラート通知先メールアドレス（任意）
- `AWS_REGION`: リージョン（デフォルト: ap-northeast-1）
- `PROJECT_NAME`: プロジェクト名（デフォルト: multicloud-auto-deploy）
- `ENVIRONMENT`: 環境名（デフォルト: staging）

用途: GitHub Actions 実行状態の継続監視

使用方法:

```
./scripts/monitor-cicd.sh

./scripts/monitor-cicd.sh --workflow=deploy-aws.yml
```

パラメータ:

- `--workflow=NAME`: 特定ワークフローのみ監視

機能:

- リアルタイム実行状態表示
- 成功/失敗/実行中の色分け
- 自動更新（30 秒間隔）

用途: GitHub Actions ワークフローの手動トリガー

使用方法:

```
./scripts/trigger-workflow.sh <workflow> [environment]
```

パラメータ:

- `workflow`: aws / azure / gcp / multicloud
- `environment`: staging（デフォルト）/ production

使用例:

```
./scripts/trigger-workflow.sh aws

./scripts/trigger-workflow.sh azure production

./scripts/trigger-workflow.sh multicloud
```

用途: Git 履歴から CHANGELOG.md 自動生成

使用方法:

```
./scripts/generate-changelog.sh [output-file]
```

パラメータ:

- `output-file`: 出力ファイルパス（デフォルト: CHANGELOG.md）

機能:

- Git コミット履歴を解析

- Conventional Commits フォーマットで分類
- 日付ごとにグループ化
- Markdown 形式で出力

カテゴリ分類:

- [NEW] **feat**: 新機能
- [FIX] **fix**: バグ修正
- [DOCS] **docs**: ドキュメント変更
- [REFACTOR] **refactor**: リファクタリング
- [PERF] **perf**: パフォーマンス改善
- [TEST] **test**: テスト追加・変更
- [STYLE] **style**: コードスタイル変更
- [CHORE] **chore**: ビルド/ツール変更

生成例:

```
./scripts/generate-changelog.sh

./scripts/generate-changelog.sh docs/HISTORY.md
```

前提条件:

- Git リポジトリ
- Conventional Commits 形式のコミットメッセージ

用途: システム診断・ヘルスチェック

使用方法:

```
./scripts/diagnostics.sh
```

診断項目:

- インストール済みツール確認
- クラウドプロバイダー認証状態
- デプロイメントエンドポイント疎通
- Terraform リソース状態

用途: GCP 既存リソースの Terraform State インポート

使用方法:

```
./scripts/import-gcp-resources.sh
```

対象リソース:

- Artifact Registry
- Cloud Storage Bucket
- Global Address
- Firestore Database
- Cloud Run Service
- Backend Bucket
- URL Map
- HTTP Proxy
- Forwarding Rule
- Storage Bucket ACL

前提条件:

- Terraform 初期化済み
- GCP 認証設定済み

用途: 古い CloudFront Distribution の削除監視

使用方法:

```
./scripts/cleanup-old-resources.sh
```

機能:

- CloudFront Distribution 状態監視
- 無効化完了後の自動削除
- 30 秒間隔でステータス確認

設定:

- `DIST_ID`: 削除対象 Distribution ID
- `CHECK_INTERVAL`: チェック間隔（秒）

用途: 全 Markdown ドキュメントから包括的な PDF ドキュメント生成

使用方法:

```
./scripts/generate-pdf-documentation.sh [output-filename]
```

パラメータ:

- `output-filename`: 出力 PDF ファイル名（デフォルト: multicloud-auto-deploy-documentation.pdf）

機能:

- 全 Markdown ファイルを章立てで結合
- 目次自動生成
- 日本語フォント対応（Noto CJK）
- 絵文字をテキスト表現に変換
- XeLaTeX による PDF 生成

含まれるドキュメント: 1. プロジェクト概要（README.md）2. システムアーキテクチャ 3. セットアップガイド 4. デプロイメントガイド（AWS/Azure/GCP）5. CI/CD 設定 6. ツールリファレンス 7. API エンドポイント 8. トラブルシューティング 9. サービス情報 10. コントリビューションガイド 11. 変更履歴（CHANGELOG.md）

生成例:

```
./scripts/generate-pdf-documentation.sh

./scripts/generate-pdf-documentation.sh my-docs.pdf
```

前提条件:

- pandoc
- texlive-xetex
- texlive-lang-cjk
- fonts-noto-cjk
- librsvg2-bin

インストールコマンド:

```
sudo apt-get install -y pandoc texlive-xetex texlive-lang-cjk \
  fonts-noto-cjk fonts-noto-color-emoji librsvg2-bin
```

終了コード:

- 0: 成功
- 1: pandoc 未インストール
- 2: ソースファイル不足
- 3: PDF 生成失敗

```
1. ./scripts/manage-github-secrets.sh --auto   # Secrets設定
2. ./scripts/deploy-aws.sh                      # AWSデプロイ
3. ./scripts/test-endpoints.sh                  # 疎通確認
4. ./scripts/setup-monitoring.sh                # 監視設定
```

```
1. git push                                     # 自動デプロイ（GitHub Actions）
2. ./scripts/monitor-cicd.sh                    # デプロイ監視
3. ./scripts/test-e2e.sh                        # E2Eテスト
```

```
1. ./scripts/diagnostics.sh                     # システム診断
2. ./scripts/test-deployments.sh                # 全環境テスト
```

```
3. docs/TROUBLESHOOTING.md                    # 問題解決ガイド参照
```

- README.md - プロジェクト概要
- TROUBLESHOOTING.md - トラブルシューティング
- SETUP.md - セットアップガイド
- CICD_SETUP.md - CI/CD 設定
- QUICK_REFERENCE.md - クイックリファレンス

Chapter 7

API Endpoints

項目	値
API Endpoint	`https://z42qmqdqac.execute-api.ap-northeast-1.amazonaws.co`
Frontend CDN	`https://d1tf3uumcm4bo1.cloudfront.net` [OK]
Frontend Storage	`http://multicloud-auto-deploy-staging-frontend.s3-website-`
Region	ap-northeast-1 (東京)
API Gateway ID	z42qmqdqac (HTTP API)
CloudFront ID	E2GDU7Y7UGDV3S
S3 Bucket	multicloud-auto-deploy-staging-frontend
Lambda Function	multicloud-auto-deploy-staging-api
Runtime	Python 3.12

テスト:

```
curl https://z42qmqdqac.execute-api.ap-northeast-1.amazonaws.com/

curl -I https://dx3l4mbwg1ade.cloudfront.net/

curl -I http://multicloud-auto-deploy-staging-frontend.s3-website-ap-northeast-1.amazonaws.com/
```

項目	値
API Endpoint	`https://multicloud-auto-deploy-staging-func-d8a2guhfere0et`
Frontend CDN	`https://multicloud-frontend-f9cvamfnauexasd8.z01.azurefd.n` [NEW]
Frontend Storage	`https://mcadwebd45ihd.z11.web.core.windows.net`

項目	値
Region	japaneast (東日本)
Resource Group	multicloud-auto-deploy-staging-rg
Function App	multicloud-auto-deploy-staging-func
Storage Account	mcadwebd45ihd ($web container)
Front Door	multicloud-frontend (Profile: multicloud-frontend-afd)
Runtime	Python 3.12

テスト:

```
curl https://multicloud-auto-deploy-staging-func-d8a2guhfere0etcq.japaneast-01.azurewebsites.ne

curl -I https://multicloud-frontend-f9cvamfnauexasd8.z01.azurefd.net/

curl -I https://mcadwebd45ihd.z11.web.core.windows.net/
```

項目	値
API Endpoint	`https://multicloud-auto-deploy-staging-api-son5b3ml7a-an.a`
Frontend CDN	`http://34.117.111.182` [NEW]
Frontend Storage	`https://storage.googleapis.com/ashnova-multicloud-auto-dep`
Region	asia-northeast1 (東京)
Project ID	ashnova
Cloud Run Service	multicloud-auto-deploy-staging-api
Cloud Run Service (Frontend)	mcad-staging-frontend
Storage Bucket	ashnova-multicloud-auto-deploy-staging-frontend
Global IP Address	34.117.111.182 (multicloud-frontend-ip)
Backend Bucket	multicloud-frontend-backend
Firestore Database	(default) - messages, posts collections

テスト:

```
curl https://multicloud-auto-deploy-staging-api-son5b3ml7a-an.a.run.app/

curl -I http://34.117.111.182/

curl -I https://storage.googleapis.com/ashnova-multicloud-auto-deploy-staging-frontend/index.ht
```

Infrastructure as Code で管理されている CDN エンドポイント

項目	値
CloudFront URL	`https://d1tf3uumcm4bo1.cloudfront.net`
Distribution ID	E1TBH4R432SZBZ
Origin	multicloud-auto-deploy-staging-frontend.s3.ap-northeast-1.amazonaws.co
管理方法	Pulumi (`infrastructure/pulumi/aws/`)
Status	Deployed [OK]

Pulumi 管理:

```
cd infrastructure/pulumi/aws
pulumi stack output cloudfront_url
pulumi stack output cloudfront_distribution_id
```

テスト:

```
curl -I https://d1tf3uumcm4bo1.cloudfront.net/
```

項目	値
Front Door URL	`https://mcad-staging-d45ihd-dseygrc9c3a3htgj.z01.azurefd.n`
Endpoint Name	mcad-staging-d45ihd
Profile Name	multicloud-auto-deploy-staging-fd
Origin	mcadwebd45ihd.z11.web.core.windows.net
管理方法	Pulumi (`infrastructure/pulumi/azure/`)
Status	Deployed [OK]

Pulumi 管理:

```
cd infrastructure/pulumi/azure
pulumi stack output frontdoor_url
pulumi stack output frontdoor_hostname
```

テスト:

```
curl -I https://mcad-staging-d45ihd-dseygrc9c3a3htgj.z01.azurefd.net/
```

項目	値
CDN URL	`http://34.117.111.182`
Global IP	34.117.111.182
Backend Bucket	multicloud-auto-deploy-staging-cdn-backend
Origin Bucket	ashnova-multicloud-auto-deploy-staging-frontend
管理方法	Pulumi (`infrastructure/pulumi/gcp/`)
Status	Deployed [OK]

Pulumi 管理:

```
cd infrastructure/pulumi/gcp
pulumi stack output cdn_url
pulumi stack output cdn_ip_address
```

テスト:

```
curl -I http://34.117.111.182/
```

GCP リソース確認:

```
gcloud compute backend-buckets describe multicloud-auto-deploy-staging-cdn-backend

gcloud compute forwarding-rules describe multicloud-auto-deploy-staging-cdn-lb --global

gcloud compute addresses describe multicloud-auto-deploy-staging-cdn-ip --global
```

- **API Gateway**: https://ap-northeast-1.console.aws.amazon.com/apigateway
- **Lambda**: https://ap-northeast-1.console.aws.amazon.com/lambda
- **S3**: https://s3.console.aws.amazon.com/s3/buckets/multicloud-auto-deploy-staging-frontend
- **CloudFront**: https:// console.aws.amazon.com/ cloudfront/ v3/ home#/ distributions/ E2GDU7Y7UGDV3S
- **Resource Group**: https:// portal.azure.com/ #@/ resource/ subscriptions/ 29031d24-d41a-4f97-8362-46b40129a7e8/resourceGroups/multicloud-auto-deploy-staging-rg
- **Function Apps**: https:// portal.azure.com/ #view/ HubsExtension/ BrowseResource/ resourceType/Microsoft.Web%2Fsites
- **Storage Account**: https:// portal.azure.com/ #view/ HubsExtension/ BrowseResource/ resourceType/Microsoft.Storage%2FStorageAccounts
- **Front Door**: https:// portal.azure.com/ #view/ HubsExtension/ BrowseResource/ resourceType/Microsoft.Cdn%2Fprofiles

- **Cosmos DB**: https:// portal.azure.com/ #view/ HubsExtension/ BrowseResource/ resourceType/Microsoft.DocumentDB%2FdatabaseAccounts
- **Cloud Run**: https://console.cloud.google.com/run?project=ashnova
- **Cloud Storage**: https://console.cloud.google.com/storage/browser?project=ashnova
- **Artifact Registry**: https://console.cloud.google.com/artifacts?project=ashnova
- **Firestore**: https://console.cloud.google.com/firestore/data?project=ashnova

---

説明: ヘルスチェック / ステータス確認

レスポンス例:

```
{
  "message": "Multi-Cloud Auto Deploy API",
  "cloud": "AWS|Azure|GCP",
  "status": "running"
}
```

説明: メッセージ一覧取得

レスポンス例:

```
{
  "messages": [
    {
      "id": "msg123",
      "content": "Hello World",
      "timestamp": "2026-02-14T10:00:00Z",
      "cloud": "AWS"
    }
  ]
}
```

説明: メッセージ送信

リクエスト:

```
{
  "content": "Hello World"
}
```

レスポンス:

```
{
  "id": "msg123",
  "content": "Hello World",
  "timestamp": "2026-02-14T10:00:00Z",
  "cloud": "AWS"
}
```

```bash
echo "=== Testing AWS ==="
curl -s https://z42qmqdqac.execute-api.ap-northeast-1.amazonaws.com/ | jq
curl -I https://d1tf3uumcm4bo1.cloudfront.net/ 2>&1 | grep HTTP

echo -e "\n=== Testing Azure ==="
curl -s https://multicloud-auto-deploy-staging-func-d8a2guhfere0etcq.japaneast-01.azurewebsites
curl -I https://multicloud-frontend-f9cvamfnauexasd8.z01.azurefd.net/ 2>&1 | grep HTTP

echo -e "\n=== Testing GCP ==="
curl -s https://multicloud-auto-deploy-staging-api-son5b3ml7a-an.a.run.app/ | jq
curl -I http://34.117.111.182/ 2>&1 | grep HTTP
```

```bash
curl -X POST https://z42qmqdqac.execute-api.ap-northeast-1.amazonaws.com/api/messages/ \
  -H "Content-Type: application/json" \
  -d '{"content":"Test from AWS"}'

curl -X POST https://multicloud-auto-deploy-staging-func-d8a2guhfere0etcq.japaneast-01.azureweb
  -H "Content-Type: application/json" \
  -d '{"content":"Test from Azure"}'

curl -X POST https://multicloud-auto-deploy-staging-api-son5b3ml7a-an.a.run.app/api/messages/ \
  -H "Content-Type: application/json" \
  -d '{"content":"Test from GCP"}'
```

日付	変更内容
2026-02-14	初版作成 - AWS/Azure/GCP 全環境のエンドポイント確定
2026-02-14	Azure Frontend URL 修正（API URL 問題解決後）
2026-02-14	AWS Frontend URL 修正（リージョン修正後）
2026-02-15	**大型更新**: 全エンドポイント情報を実際の値に更新
2026-02-15	CDN 情報追加 - CloudFront, Front Door, Cloud CDN
2026-02-15	Azure: Container Apps → Functions に変更
2026-02-15	GCP: Cloud Run API エンドポイント更新
2026-02-15	**Pulumi 管理環境追加** - 全 3 クラウドで Infrastructure as Code 導入 [CELEBRATE]
2026-02-15	**全環境デプロイ成功** - AWS/GCP/Azure 統合完了、エンドポイント最新化

2. **API Key や認証トークンは環境変数で管理**
3. **クロスオリジン（CORS）設定を確認**
4. **レート制限に注意**（特に AWS API Gateway、Azure Front Door）

- すべての API は HTTPS 経由で通信
- フロントエンドは CDN 経由で配信（DDoS 対策）
- データベースはプライベートネットワーク内に配置
- 認証・認可機能は今後実装予定

Chapter 8

Troubleshooting Guide

CI/CD ワークフロー実行時に遭遇する可能性のある問題と解決策

症状:

```
Error: building account: could not acquire access token to parse claims:
Authenticating using the Azure CLI is only supported as a User (not a Service Principal).
```

原因:

- Azure CLI でログイン後、Terraform が CLI 認証を試みるが、Service Principal では使用不可

解決策:

1. **ワークフローから初回 Azure Login を削除**:

```
```

2. **Terraform Provider で明示的に無効化**:

```
provider "azurerm" {
  features {}
  use_cli  = false
  use_msi  = false
  use_oidc = false
```

```
}
```

3. **環境変数で認証**:

```
env:
  ARM_CLIENT_ID: ${{ secrets.ARM_CLIENT_ID }}
  ARM_CLIENT_SECRET: ${{ secrets.ARM_CLIENT_SECRET }}
  ARM_SUBSCRIPTION_ID: ${{ secrets.ARM_SUBSCRIPTION_ID }}
  ARM_TENANT_ID: ${{ secrets.ARM_TENANT_ID }}
```

症状: Terraform の出力や環境変数が正しく渡らない

解決策:

```
- name: Setup Terraform
  uses: hashicorp/setup-terraform@v3
  with:
    terraform_version: 1.7.5
    terraform_wrapper: false  # 必須!
```

症状: ACR 操作のために Azure CLI ログイン後、terraform output コマンドが失敗

解決策: Terraform outputs をインフラデプロイ時に GitHub Actions outputs に保存：

```
- name: Deploy Infrastructure
  id: terraform
  run: |
    terraform apply -auto-approve

    # TerraformからoutputsをGitHub Actionsに保存
    ACR_NAME=$(terraform output -raw container_registry_name)
    echo "acr_name=$ACR_NAME" >> $GITHUB_OUTPUT

- name: Use Output Later
  run: |
    # GitHub Actions outputsから取得
    echo "ACR: ${{ steps.terraform.outputs.acr_name }}"
```

症状:

```
Error: Error creating Repository: googleapi: Error 409: the repository already exists.
Error: Error creating Service: googleapi: Error 409: Resource already exists.
Error: Error creating BackendBucket: googleapi: Error 409: already exists.
```

根本原因:

- Terraform がローカル backend を使用していた
- GitHub Actions 実行ごとにクリーンな環境で実行されるため、state が保存されない
- Terraform が既存リソースを認識できず、毎回新規作成を試みる

解決策（永続的な remote state）:

1. **GCS バケットの作成:**

```
gcloud storage buckets create gs://multicloud-auto-deploy-tfstate-gcp \
  --location=asia-northeast1 \
  --uniform-bucket-level-access
```

2. **サービスアカウントに権限付与:**

```
gcloud storage buckets add-iam-policy-binding gs://multicloud-auto-deploy-tfstate-gcp \
  --member="serviceAccount:github-actions-deploy@PROJECT_ID.iam.gserviceaccount.com" \
  --role="roles/storage.objectAdmin"
```

3. **GCS backend の有効化:**

```
terraform {
  backend "gcs" {
    bucket = "multicloud-auto-deploy-tfstate-gcp"
    prefix = "terraform/state"
  }
}
```

4. **既存リソースのインポート（一度だけ実行）:**

```
cd infrastructure/terraform/gcp

terraform init

terraform import google_artifact_registry_repository.main \
  "projects/PROJECT_ID/locations/REGION/repositories/REPO_NAME"

terraform import google_storage_bucket.frontend "BUCKET_NAME"

terraform import google_compute_global_address.frontend \
  "projects/PROJECT_ID/global/addresses/ADDRESS_NAME"

terraform import google_firestore_database.main \
  "projects/PROJECT_ID/databases/(default)"

terraform import google_cloud_run_v2_service.api \
  "projects/PROJECT_ID/locations/REGION/services/SERVICE_NAME"

terraform import google_compute_backend_bucket.frontend "BACKEND_BUCKET_NAME"

terraform import google_compute_url_map.frontend \
  "projects/PROJECT_ID/global/urlMaps/URLMAP_NAME"

terraform import google_compute_target_http_proxy.frontend \
```

```
  "projects/PROJECT_ID/global/targetHttpProxies/PROXY_NAME"

terraform import google_compute_global_forwarding_rule.frontend_http \
  "projects/PROJECT_ID/global/forwardingRules/RULE_NAME"

terraform state list

terraform plan
```

5. **ワークフローのシンプル化**:

```
- name: Deploy Infrastructure
  run: |
    cd infrastructure/terraform/gcp
    terraform init
    terraform plan -var="project_id=${{ secrets.GCP_PROJECT_ID }}"
    terraform apply -var="project_id=${{ secrets.GCP_PROJECT_ID }}" -auto-approve
```

症状:

```
Error: Error creating database: googleapi: Error 403:
The caller does not have permission
```

解決策:

```
gcloud projects add-iam-policy-binding PROJECT_ID \
  --member="serviceAccount:github-actions-deploy@PROJECT_ID.iam.gserviceaccount.com" \
  --role="roles/datastore.owner"
```

症状: Cloud Run サービスの IAM ポリシーを設定できない

解決策:

```
gcloud projects add-iam-policy-binding PROJECT_ID \
  --member="serviceAccount:github-actions-deploy@PROJECT_ID.iam.gserviceaccount.com" \
  --role="roles/run.admin"
```

症状:

- フロントエンドから API へのリクエストがタイムアウト
- ブラウザのコンソールに”Failed to fetch”エラー

原因: フロントエンドビルド時に正しい API URL が設定されていない

Azure・AWS での発生パターン: 1. フロントエンドをインフラデプロイ**前**にビルド 2. API URL がまだ存在しない/間違った値 3. ビルドされたフロントエンドが間違った URL を使用

解決策（推奨フロー）:

```yaml
jobs:
  build-and-deploy:
    steps:
      # 1. インフラをデプロイ（API URLを生成）
      - name: Deploy Infrastructure
        id: terraform
        run: |
          terraform apply -auto-approve

          # API URLをoutputsに保存
          API_URL=$(terraform output -raw api_url)
          echo "api_url=$API_URL" >> $GITHUB_OUTPUT

      # 2. バックエンドをデプロイ
      - name: Deploy Backend
        run: |
          # Docker build & push, Lambda update, etc.

      # 3. フロントエンドを正しいAPI URLでビルド
      - name: Build Frontend
        run: |
          cd services/frontend
          npm install
          npm run build
        env:
          VITE_API_URL: ${{ steps.terraform.outputs.api_url }}

      # 4. フロントエンドをデプロイ
      - name: Deploy Frontend
        run: |
          # S3/Storage/GCSにアップロード
```

重要ポイント:

- フロントエンドのビルドは必ずインフラデプロイ**後**に実行
- Terraform outputs から動的に API URL を取得
- 環境変数`VITE_API_URL`に正しい値を設定

検証方法:

```bash
curl -s https://YOUR-FRONTEND-URL/assets/index-*.js | grep -o "https://.*execute-api.*"
```

---

1. **Remote Backend を必ず使用**:
   - AWS: S3 + DynamoDB
   - Azure: Storage Account
   - GCP: GCS

2. **State Locking を有効化**:
   - AWS DynamoDB テーブル
   - Azure Storage Account（自動）
   - GCS（自動）
3. **State の定期バックアップ**:

```
gcloud storage cp gs://tfstate-bucket/terraform/state/default.tfstate \
  ./backups/tfstate-$(date +%Y%m%d-%H%M%S).tfstate
```

必要な権限:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "lambda:UpdateFunctionCode",
        "lambda:GetFunction",
        "s3:PutObject",
        "s3:GetObject"
      ],
      "Resource": "*"
    }
  ]
}
```

必要なロール:

- `AcrPush` または `Contributor`

確認方法:

```
az role assignment list \
  --assignee YOUR_SERVICE_PRINCIPAL_ID \
  --scope /subscriptions/SUB_ID/resourceGroups/RG_NAME/providers/Microsoft.ContainerRegistry/re
```

必要なロール:

- `roles/artifactregistry.writer`
- `roles/storage.admin`

付与方法:

```
gcloud projects add-iam-policy-binding PROJECT_ID \
  --member="serviceAccount:SA_EMAIL" \
  --role="roles/artifactregistry.writer"
```

確認ポイント: 1. GitHub Actions のログで最後の出力を確認 2. タイムアウト設定を確認（デフォルト: 360 分）3. リソースクォータを確認（特に GCP）

答え: 1. GitHub リポジトリの **Actions** タブを開く 2. 対象のワークフロー（deploy-aws.yml 等）を選択 3. **Run workflow** をクリック 4. 環境（staging/production）を選択して実行

答え:

```
cd infrastructure/terraform/[aws|azure|gcp]
terraform destroy -auto-approve

terraform destroy -target=google_cloud_run_v2_service.api
```

```
export ARM_CLIENT_ID="..."
export AWS_ACCESS_KEY_ID="..."

cd infrastructure/terraform/azure
terraform init
terraform plan
```

重要な情報: - エラーメッセージの完全な内容 - 失敗したステップ名 - 環境変数の値（機密情報は除く）

```
aws apigatewayv2 get-apis --region ap-northeast-1
aws s3 ls
aws cloudfront list-distributions

az group list
az containerapp list
az storage account list

gcloud run services list
gcloud storage buckets list
gcloud compute addresses list --global
```

症状:

```
{"message": "Internal Server Error"}
```

- Lambda を直接呼び出すと成功する
- API Gateway 経由だと 500 エラーになる
- CloudWatch Logs に Lambda の実行ログが記録されない
- CloudWatch Metrics で Lambda 呼び出し回数が 0 のまま

原因: Lambda 関数のリソースポリシーで SourceArn 形式が正しくない。

API 種類	SourceArn 形式	例
HTTP API	arn:aws:execute-api:{region}:{account-id}:{api-id}/*/*	arn:aws:execute-api:ap-northeast-1:1234567890
REST API	arn:aws:execute-api:{region}:{account-id}:{api-id}/*/*/*	arn:aws:execute-api:ap-northeast-1:1234567890

重要: HTTP API は `/*/*` (2 つのワイルドカード)、REST API は `/*/*/*` (3 つのワイルドカード)

解決策:

1. **現在の権限を確認**:

```
aws lambda get-policy --function-name YOUR_FUNCTION_NAME --query Policy --output text | jq .
```

2. **API Gateway アクセスログを有効化**（エラー詳細を確認するため）:

```
aws logs create-log-group --log-group-name /aws/apigateway/YOUR_API_NAME

aws apigatewayv2 update-stage \
  --api-id YOUR_API_ID \
  --stage-name '$default' \
  --access-log-settings "DestinationArn=arn:aws:logs:REGION:ACCOUNT_ID:log-group:/aws/apigatewa

aws logs tail /aws/apigateway/YOUR_API_NAME --follow
```

3. **正しい権限を設定**（HTTP API 用）:

```
aws lambda remove-permission \
  --function-name YOUR_FUNCTION_NAME \
  --statement-id OLD_STATEMENT_ID

aws lambda add-permission \
  --function-name YOUR_FUNCTION_NAME \
  --statement-id apigateway-http-api \
  --action lambda:InvokeFunction \
  --principal apigateway.amazonaws.com \
  --source-arn "arn:aws:execute-api:REGION:ACCOUNT_ID:API_ID/*/*"
```

4. **動作確認**:

```
curl https://YOUR_API_ID.execute-api.REGION.amazonaws.com/api/messages/
```

デバッグ手順:

1. Lambda 直接呼び出しテスト:

```
aws lambda invoke \
  --function-name YOUR_FUNCTION_NAME \
```

```
  --payload '{"version":"2.0","routeKey":"$default","rawPath":"/api/messages/","headers":{"acce
  /tmp/response.json
```

2. CloudWatch Metrics で呼び出し回数確認:

```
aws cloudwatch get-metric-statistics \
  --namespace AWS/Lambda \
  --metric-name Invocations \
  --dimensions Name=FunctionName,Value=YOUR_FUNCTION_NAME \
  --start-time $(date -u -d '10 minutes ago' +%Y-%m-%dT%H:%M:%S) \
  --end-time $(date -u +%Y-%m-%dT%H:%M:%S) \
  --period 60 \
  --statistics Sum
```

3. API Gateway アクセスログ確認（最も重要）:

```
aws logs tail /aws/apigateway/YOUR_API_NAME --since 5m
```

参考リンク:

- AWS Lambda リソースベースのポリシー
- API Gateway HTTP API と Lambda の統合

症状:

```
[ERROR] Runtime.ImportModuleError: Unable to import module 'index': No module named 'index'
```

原因:

- GitHub Actions ワークフローが Lambda 関数用に`handler.py`を生成
- Lambda 関数設定では`index.handler`を期待
- ファイル名のミスマッチ

解決策:

1. **既存の`index.py`を使用するようワークフロー修正**:

```
cp index.py package/
```

2. **`services/api/index.py`の内容確認**:

```
"""AWS Lambda エントリーポイント"""
from mangum import Mangum
from app.main import app

handler = Mangum(app, lifespan="off")
```

3. **Lambda 設定確認**:

```
aws lambda get-function-configuration \
  --function-name YOUR_FUNCTION_NAME \
  --query 'Handler'
```

4. **デプロイ後の動作確認**:

```
aws logs tail /aws/lambda/YOUR_FUNCTION_NAME --since 5m

curl https://YOUR_API_URL/health
```

修正コミット例:

```
git commit -m "fix(ci): Use index.py instead of handler.py for Lambda entry point"
```

症状:

```
ConnectionRefusedError: [Errno 111] Connection refused
File "/workspace/app/backends/local.py", line 30, in __init__
  self._ensure_bucket()
```

原因:

- `CLOUD_PROVIDER`環境変数が未設定
- アプリケーションが LocalBackend（MinIO localhost:9000）を使用しようとする
- Cloud Run で localhost:9000 は存在しない

解決策:

1. **環境変数を設定**:

```
gcloud functions deploy $FUNCTION_NAME \
  --set-env-vars=ENVIRONMENT=staging,CLOUD_PROVIDER=gcp,GCP_PROJECT_ID=$PROJECT_ID,FIRESTORE_C
```

2. **環境変数確認**:

```
gcloud run services describe YOUR_SERVICE_NAME \
  --region=asia-northeast1 \
  --format="value(spec.template.spec.containers[0].env)"
```

3. **Cloud Run ログで確認**:

```
gcloud logging read "resource.type=cloud_run_revision AND resource.labels.service_name=YOUR_SE
  --limit 10 \
  --format="table(timestamp,textPayload)" \
  --freshness=5m
```

症状:

```
AttributeError: 'bytes' object has no attribute 'encode'
File "/workspace/main.py", line 19, in handler
  "query_string": request.query_string.encode() if request.query_string else b"",
```

原因:

- request.query_stringは既にbytes型
- .encode()を再度呼び出すとエラー

解決策:

services/api/function.pyを修正:

```
"query_string": request.query_string.encode() if request.query_string else b"",

"query_string": request.query_string if request.query_string else b"",
```

動作確認:

```
curl -X POST "https://YOUR_CLOUD_RUN_URL/api/messages/" \
  -H "Content-Type: application/json" \
  -d '{"content":"Test message","author":"DevOps"}'
```

症状:

```
HTTP 500 Internal Server Error (No response body)
```

原因:

- AZURE_COSMOS_ENDPOINTとAZURE_COSMOS_KEYが空
- 既存の環境変数名がCOSMOS_DB_ENDPOINT/COSMOS_DB_KEY
- アプリケーションが間違った環境変数を参照

解決策:

1. **既存の環境変数を確認**:

```
az functionapp config appsettings list \
  --name YOUR_FUNCTION_APP \
  --resource-group YOUR_RESOURCE_GROUP \
  --output table | grep COSMOS
```

2. **ワークフローで既存値を取得して設定**:

```
COSMOS_ENDPOINT=$(az functionapp config appsettings list \
  --name $FUNCTION_APP \
  --resource-group $RESOURCE_GROUP \
```

```
  --query "[?name=='COSMOS_DB_ENDPOINT'].value | [0]" -o tsv)

COSMOS_KEY=$(az functionapp config appsettings list \
  --name $FUNCTION_APP \
  --resource-group $RESOURCE_GROUP \
  --query "[?name=='COSMOS_DB_KEY'].value | [0]" -o tsv)

az functionapp config appsettings set \
  --name $FUNCTION_APP \
  --resource-group $RESOURCE_GROUP \
  --settings \
    CLOUD_PROVIDER=azure \
    ENVIRONMENT=staging \
    AZURE_COSMOS_ENDPOINT="${COSMOS_ENDPOINT}" \
    AZURE_COSMOS_KEY="${COSMOS_KEY}"
```

3. **即座の修正（手動）**:

```
COSMOS_ENDPOINT=$(az functionapp config appsettings list \
  --name YOUR_FUNCTION_APP \
  --resource-group YOUR_RESOURCE_GROUP \
  --query "[?name=='COSMOS_DB_ENDPOINT'].value | [0]" -o tsv)

COSMOS_KEY=$(az functionapp config appsettings list \
  --name YOUR_FUNCTION_APP \
  --resource-group YOUR_RESOURCE_GROUP \
  --query "[?name=='COSMOS_DB_KEY'].value | [0]" -o tsv)

az functionapp config appsettings set \
  --name YOUR_FUNCTION_APP \
  --resource-group YOUR_RESOURCE_GROUP \
  --settings \
    "AZURE_COSMOS_ENDPOINT=${COSMOS_ENDPOINT}" \
    "AZURE_COSMOS_KEY=${COSMOS_KEY}"
```

4. **Function App 再起動待機（約 30 秒）後、テスト**:

```
curl -X POST "https://YOUR_FUNCTION_APP.azurewebsites.net/api/HttpTrigger/api/messages/" \
  -H "Content-Type: application/json" \
  -d '{"content":"Azure test","author":"DevOps"}'
```

症状:

- ブラウザから Front Door URL 経由でアクセス: 404 Error
- POST /api/messages/: 405 Method Not Allowed

原因:

- フロントエンドが間違った API URL を使用

- `/api/HttpTrigger`パスが含まれていない

解決策:

1. **正しい API URL を設定**:

```
FUNC_HOSTNAME=$(az functionapp show --name YOUR_FUNCTION_APP --resource-group YOUR_RESOURCE_GRO
echo "api_url=https://${FUNC_HOSTNAME}/api/HttpTrigger" >> $GITHUB_OUTPUT
```

2. **フロントエンドビルド時に正しい URL を使用**:

```
- name: Build Frontend
  run: |
    cd services/frontend_react
    npm install
    VITE_API_URL="${{ steps.pulumi_outputs.outputs.api_url }}" npm run build
  env:
    VITE_API_URL: ${{ steps.pulumi_outputs.outputs.api_url }}
```

3. **Function App 直接アクセスで動作確認**:

```
curl https://YOUR_FUNCTION_APP.azurewebsites.net/api/HttpTrigger/health
```

症状:

```
ERROR: Deployment was partially successful. These are the deployment logs:
[***"message": "The logs you are looking for were not found. In flex consumption plans,
the instance will be recycled and logs will not be persisted after that..."***]

[WARNING]  Deployment status unclear, retrying...
```

しかし、Function App は実際には正常に動作している。

原因:

- Azure Flex Consumption プランではインスタンスがリサイクルされ、デプロイログが保持されない
- `az functionapp deployment source config-zip` が“partially successful”を返すが、実際にはデプロイは成功している
- 詳細なステップログ（`UploadPackageStep`, `OryxBuildStep`等）が出力されない

解決策:

1. **“Deployment was successful”メッセージを検出**:

```
if grep -q "Deployment was successful" deploy_log.txt; then
  echo "[OK] Deployment successful!"
  DEPLOY_SUCCESS=true
```

```
  break
elif grep -q "UploadPackageStep.*completed" deploy_log.txt || \
     grep -q "SyncTriggerStep" deploy_log.txt; then
  echo "[OK] Deployment steps completed!"
  DEPLOY_SUCCESS=true
  break
fi
```

2. **“partially successful”を無視**:

```
elif grep -q "ERROR:" deploy_log.txt && ! grep -q "partially successful" deploy_log.txt; then
  echo "[ERROR] Critical deployment error"
  cat deploy_log.txt
  exit 1
fi
```

3. **ヘルスチェックを必須検証に**:

```
- name: Verify Deployment
  run: |
    # ... ヘルスチェック実行 ...

    if [ "$health_check_passed" = true ]; then
      echo "[OK] Azure Function deployment verified successfully!"
    else
      echo "[ERROR] Health check failed"
      exit 1  # 失敗として扱う
    fi
```

症状:

```
Testing: https:///api/HttpTrigger/health
[ERROR] Health check failed
```

`az functionapp show --query defaultHostName` が null を返し、URL が空になる。

原因:

- Flex Consumption プランでは `defaultHostName` フィールドが null または未設定
- 標準的な `az functionapp show` コマンドでホスト名を取得できない

解決策:

`az functionapp config hostname list` を使用:

```
FUNC_HOSTNAME=$(az functionapp config hostname list \
  --webapp-name $FUNCTION_APP \
  --resource-group $RESOURCE_GROUP \
  --query '[0].name' -o tsv)

```

```
if [ -n "$FUNC_HOSTNAME" ] && [ "$FUNC_HOSTNAME" != "None" ]; then
  echo "[OK] Got hostname: $FUNC_HOSTNAME"
  FUNC_URL="https://${FUNC_HOSTNAME}/api/HttpTrigger"
else
  echo "[ERROR] Failed to get Function App hostname"
  exit 1
fi
```

検証例:

```
az functionapp show --name multicloud-auto-deploy-staging-func \
  --resource-group multicloud-auto-deploy-staging-rg \
  --query defaultHostName -o tsv

az functionapp config hostname list \
  --webapp-name multicloud-auto-deploy-staging-func \
  --resource-group multicloud-auto-deploy-staging-rg \
  --query '[0].name' -o tsv
```

症状:

```
[SYNC] Kudu restart detected, retrying...
Attempt 2/3...
```

大きなデプロイパッケージで Kudu が再起動し、デプロイが中断される。

解決策:

1. **パッケージサイズの最適化**:

```
- name: Package Function App
  run: |
    cd services/api

    echo "[PACKAGE] Creating optimized deployment package..."

    # Install dependencies
    pip install --target .deployment --no-cache-dir -r requirements.txt

    # Clean up unnecessary files from dependencies
    find .deployment -type d -name "__pycache__" -exec rm -rf {} + 2>/dev/null || true
    find .deployment -type f -name "*.pyc" -delete 2>/dev/null || true
    find .deployment -type f -name "*.pyo" -delete 2>/dev/null || true
    find .deployment -type d -name "tests" -exec rm -rf {} + 2>/dev/null || true
    find .deployment -type d -name "*.dist-info" -exec rm -rf {} + 2>/dev/null || true

    # Copy application code
    cp -r app .deployment/
    cp function_app.py .deployment/
    cp host.json .deployment/

```

```
    # Create ZIP package
    cd .deployment
    zip -r -q ../function-app.zip .

    echo "[OK] Package size: $(du -h ../function-app.zip | cut -f1)"
```

2. **リトライロジックの実装**:

```
MAX_RETRIES=3
RETRY_COUNT=0
DEPLOY_SUCCESS=false

while [ $RETRY_COUNT -lt $MAX_RETRIES ]; do
  echo "Attempt $((RETRY_COUNT+1))/$MAX_RETRIES..."

  # Run deployment
  az functionapp deployment source config-zip \
    --resource-group $RESOURCE_GROUP \
    --name $FUNCTION_APP \
    --src services/api/function-app.zip \
    --timeout 600 \
    2>&1 | tee deploy_log.txt || true

  # Check for Kudu restart
  if grep -q "Kudu has been restarted" deploy_log.txt; then
    echo "[SYNC] Kudu restart detected, retrying..."
    RETRY_COUNT=$((RETRY_COUNT+1))
    sleep 30
    continue
  fi

  # Check for success
  if grep -q "Deployment was successful" deploy_log.txt; then
    DEPLOY_SUCCESS=true
    break
  fi

  RETRY_COUNT=$((RETRY_COUNT+1))
  sleep 30
done
```

症状:

AWS:

```
Could not load credentials from any providers
```

GCP:

```
must specify exactly one of "workload_identity_provider" or "credentials_json"!
```

原因:

- フロントエンドデプロイワークフローが OIDC/Workload Identity を使用
- メインデプロイワークフローは静的認証情報（Access Keys / Service Account JSON）を使用
- 認証方法の不一致により、シークレットが見つからない

解決策:

フロントエンドワークフローを静的認証情報に統一:

1. **AWS**:

```yaml
- name: Configure AWS credentials
  uses: aws-actions/configure-aws-credentials@v4
  with:
    role-to-assume: ${{ secrets.AWS_ROLE_ARN }}  # [ERROR] 設定されていない
    aws-region: ${{ env.AWS_REGION }}

- name: Configure AWS credentials
  uses: aws-actions/configure-aws-credentials@v4
  with:
    aws-access-key-id: ${{ secrets.AWS_ACCESS_KEY_ID }}  # [OK]
    aws-secret-access-key: ${{ secrets.AWS_SECRET_ACCESS_KEY }}  # [OK]
    aws-region: ${{ env.AWS_REGION }}
```

2. **GCP**:

```yaml
- name: Authenticate to Google Cloud
  uses: google-github-actions/auth@v2
  with:
    workload_identity_provider: ${{ secrets.GCP_WORKLOAD_IDENTITY_PROVIDER }}  # [ERROR]
    service_account: ${{ secrets.GCP_SERVICE_ACCOUNT }}  # [ERROR]

- name: Authenticate to Google Cloud
  uses: google-github-actions/auth@v2
  with:
    credentials_json: ${{ secrets.GCP_CREDENTIALS }}  # [OK]

- name: Set up Cloud SDK
  uses: google-github-actions/setup-gcloud@v2
  with:
    project_id: ${{ secrets.GCP_PROJECT_ID }}  # [OK]
```

問題が解決しない場合:

1. GitHub Issues で報告
2. エラーログとコマンド出力を添付
3. 実行環境（OS、CLI バージョン等）を明記

修正履歴:

- 2026-02-15: AWS Lambda ImportModuleError 解決方法追加
- 2026-02-15: GCP Cloud Run 500 エラー（環境変数・型エラー）解決方法追加
- 2026-02-15: Azure Functions 500 エラー（Cosmos DB）解決方法追加
- 2026-02-15: Azure Functions Flex Consumption Plan 特有の問題と解決策追加
- 2026-02-15: フロントエンドワークフロー認証エラー解決方法追加

Chapter 9

Services and Infrastructure

9.1 Backend API Service

完全 Python 実装のマルチクラウド対応 Simple SNS バックエンド API

- **FastAPI** - 高速で型安全な Python フレームワーク
- **マルチクラウド対応** - AWS / Azure / GCP / Local 開発環境
- **Pydantic** - データバリデーションと設定管理
- **自動 API 文書** - OpenAPI (Swagger UI / ReDoc)

```
pip install -r requirements.txt

docker-compose up -d minio

uvicorn app.main:app --reload

open http://localhost:8000/docs
```

```
docker build -t simple-sns-api .
docker run -p 8000:8000 simple-sns-api
```

```
services/api/
├── app/
│   ├── __init__.py
│   ├── main.py          # FastAPIアプリケーション
│   ├── config.py        # 設定管理（Pydantic Settings）
│   ├── models.py        # データモデル（Pydantic）
│   ├── backends/        # クラウドプロバイダー別実装
│   │   ├── aws.py
│   │   ├── azure.py
│   │   ├── gcp.py
│   │   └── local.py
│   └── routes/          # APIルート
│       ├── messages.py
```

```
│       └── uploads.py
├── tests/                # テスト
├── requirements.txt
├── Dockerfile
└── .env.example
```

.env.exampleを.envとしてコピーして設定：

```
CLOUD_PROVIDER=aws  # aws, azure, gcp, local

AWS_REGION=ap-northeast-1
DYNAMODB_TABLE_NAME=simple-sns-messages
S3_BUCKET_NAME=your-bucket
```

```
pytest

pytest --cov=app tests/
```

```
cd services/api
pip install -r requirements.txt -t package/
cp -r app/ package/
cd package && zip -r ../lambda.zip . && cd ..
```

```
az containerapp up \
  --name simple-sns-api \
  --source . \
  --ingress external \
  --target-port 8000
```

```
gcloud run deploy simple-sns-api \
  --source . \
  --platform managed \
  --region asia-northeast1 \
  --allow-unauthenticated
```

メソッド	パス	説明
GET	/	ヘルスチェック
GET	/health	ヘルスチェック
GET	/docs	API 文書（Swagger UI）
GET	/redoc	API 文書（ReDoc）

- FastAPI
- Pydantic
- Uvicorn

9.2 Frontend (React) Service

This template provides a minimal setup to get React working in Vite with HMR and some ESLint rules.

Currently, two official plugins are available:

- @vitejs/plugin-react uses Babel (or oxc when used in rolldown-vite) for Fast Refresh
- @vitejs/plugin-react-swc uses SWC for Fast Refresh

The React Compiler is not enabled on this template because of its impact on dev & build performances. To add it, see this documentation.

If you are developing a production application, we recommend updating the configuration to enable type-aware lint rules:

```
export default defineConfig([
  globalIgnores(['dist']),
  {
    files: ['**/*.{ts,tsx}'],
    extends: [
      // Other configs...

      // Remove tseslint.configs.recommended and replace with this
      tseslint.configs.recommendedTypeChecked,
      // Alternatively, use this for stricter rules
      tseslint.configs.strictTypeChecked,
      // Optionally, add this for stylistic rules
      tseslint.configs.stylisticTypeChecked,

      // Other configs...
    ],
    languageOptions: {
      parserOptions: {
        project: ['./tsconfig.node.json', './tsconfig.app.json'],
        tsconfigRootDir: import.meta.dirname,
      },
      // other options...
    },
  },
])
```

You can also install eslint-plugin-react-x and eslint-plugin-react-dom for React-specific lint rules:

```
// eslint.config.js
import reactX from 'eslint-plugin-react-x'
import reactDom from 'eslint-plugin-react-dom'

export default defineConfig([
  globalIgnores(['dist']),
  {
```

```
    files: ['**/*.{ts,tsx}'],
    extends: [
      // Other configs...
      // Enable lint rules for React
      reactX.configs['recommended-typescript'],
      // Enable lint rules for React DOM
      reactDom.configs.recommended,
    ],
    languageOptions: {
      parserOptions: {
        project: ['./tsconfig.node.json', './tsconfig.app.json'],
        tsconfigRootDir: import.meta.dirname,
      },
      // other options...
    },
  },
])
```

9.3 Frontend (Reflex) Service

完全 Python 実装のフロントエンド（Reflex 使用）

- **完全 Python 実装** - JavaScript なし
- **リアクティブ UI** - React 風のコンポーネント
- **型安全** - Pydantic ベース
- **CRUD 完全対応** - 作成/読取/更新/削除
- **画像アップロード** - MinIO 統合
- **ページネーション** - ページ切り替え機能

```
pip install -r requirements.txt

reflex init

export API_URL=http://localhost:8000
reflex run
```

```
docker-compose up frontend_reflex
```

- [OK] メッセージ作成（テキスト ＋ 画像）
- [OK] メッセージ一覧表示
- [OK] メッセージ編集（インライン）
- [OK] メッセージ削除
- [OK] 画像アップロード・プレビュー
- [OK] ページネーション（前へ/次へ）

`rxconfig.py` で設定可能：

```
config = rx.Config(
    app_name="simple_sns",
    frontend_port=3000,
    backend_port=3001,
    api_url="http://localhost:8000",
)
```

```
frontend_reflex/
├── simple_sns.py      # メインアプリケーション
├── rxconfig.py        # Reflex設定
├── requirements.txt   # 依存関係
├── Dockerfile         # Docker設定
└── README.md          # このファイル
```

- フロントエンド: http://localhost:3000
- Reflex 管理: http://localhost:3001

9.4 Pulumi Infrastructure (AWS)

Pulumi による AWS インフラストラクチャ管理（完全 Python 実装）

- **API Gateway (HTTP API)** - RESTful API エンドポイント
- **Lambda Function** - Python 3.12 FastAPI アプリケーション
- **DynamoDB Table** - メッセージストレージ（オンデマンド）
- **S3 Bucket** - 画像ストレージ（パブリック読み取り）
- **IAM Roles & Policies** - 最小権限

```
aws configure

curl -fsSL https://get.pulumi.com | sh

pip install -r requirements.txt
```

```
pulumi stack init staging

pulumi config set aws:region ap-northeast-1
pulumi config set environment staging

pulumi preview

pulumi up
```

```
pulumi up
```

```
pulumi destroy
```

Key	Description	Default
`aws:region`	AWS リージョン	`ap-northeast-1`
`environment`	環境名	`staging`
`project_name`	プロジェクト名	`simple-sns`

設定方法：

```
pulumi config set aws:region ap-northeast-1
pulumi config set environment production
```

デプロイ後、以下の情報が出力されます：

```
pulumi stack output api_url

pulumi stack output
```

Output	Description
`api_url`	API Gateway エンドポイント
`messages_table_name`	DynamoDB テーブル名
`images_bucket_name`	S3 バケット名
`lambda_function_name`	Lambda 関数名

```
API_URL=$(pulumi stack output api_url)

curl $API_URL/health

curl $API_URL/api/messages
```

月間想定（低トラフィック）：

- Lambda: ~$1（100 万リクエスト無料枠内）
- DynamoDB: ~$1（オンデマンド）
- S3: ~$0.5（ストレージ + 転送）
- API Gateway: ~$1（100 万リクエスト）

合計: ~$3-5/月

既存の Terraform state から移行する場合：

```
cd ../../../../infrastructure/terraform/aws
terraform show

pulumi import aws:dynamodb/table:Table messages-table simple-sns-messages-staging
pulumi import aws:s3/bucketV2:BucketV2 images-bucket simple-sns-images-staging
```

- Pulumi AWS Provider
- Pulumi Python SDK

Chapter 10

Contributing Guidelines

Multi-Cloud Auto Deploy Platform へのコントリビューションをありがとうございます！

1. **リポジトリをフォーク**
2. **クローン**

```
git clone https://github.com/YOUR_USERNAME/multicloud-auto-deploy.git
cd multicloud-auto-deploy
```

3. **依存関係のインストール**

```
cd services/frontend
npm install

cd ../backend
pip install -r requirements.txt -r requirements-dev.txt
```

4. **ローカルで実行**

```
docker-compose up

cd services/frontend && npm run dev

cd services/backend && uvicorn src.main:app --reload
```

- `main`: 本番環境
- `develop`: 開発環境
- `feature/*`: 新機能
- `bugfix/*`: バグ修正
- `hotfix/*`: 緊急修正

Conventional Commits に従ってください：

```
<type>(<scope>): <subject>

<body>

<footer>
```

- feat: 新機能
- fix: バグ修正
- docs: ドキュメント
- style: コードスタイル
- refactor: リファクタリング
- test: テスト
- chore: その他

```
feat(frontend): Add message filtering

- Add filter by date
- Add filter by cloud provider
- Update UI components

Closes #123
```

1. **最新の main から作業ブランチを作成**

```
git checkout main
git pull origin main
git checkout -b feature/your-feature
```

2. **変更を加える**
3. **テストを実行**

```
cd services/frontend
npm test

cd services/backend
pytest
```

4. **コミットしてプッシュ**

```
git add .
git commit -m "feat: your feature description"
git push origin feature/your-feature
```

5. **PR を作成**
   - わかりやすいタイトルと説明

  - 関連するイシューを参照
  - スクリーンショット（UI 変更の場合）

- **Formatter**: Black
- **Linter**: Flake8
- **Type Hints**: 使用必須

```
black src/
flake8 src/
mypy src/
```

- **Style Guide**: Airbnb
- **Linter**: ESLint
- **Formatter**: Prettier

```
npm run lint
npm run format
```

```
cd services/frontend
npm test              # Unit tests
npm run test:e2e      # E2E tests
npm run test:coverage # Coverage
```

```
cd services/backend
pytest                    # All tests
pytest tests/test_main.py # Specific file
pytest --cov=src          # With coverage
```

- コードにコメントを追加
- README を更新
- 新機能にはドキュメントページを追加

1. 自動チェック（CI）が通過
2. コードレビュー
3. 承認後にマージ

バグを見つけた場合：

1. 既存のイシューを確認
2. 新しいイシューを作成
3. 以下を含める：
   - 明確な説明
   - 再現手順
   - 期待される動作
   - スクリーンショット/ログ

- 環境情報

コントリビューションは MIT ライセンスの下で公開されます。

- GitHub Discussions
- Issues
- メール: support@example.com

Chapter 11

Changelog

Complete development history of the **multicloud-auto-deploy** project with detailed commit information.

This changelog includes commit bodies, file changes, and statistics for full transparency.

Repository: https://github.com/PLAYER1-r7/multicloud-auto-deploy
Branch: `main`
Generated: 2026-02-15 08:25:46

Commit: `201cb80` | **Author**: SATOSHI KAWADA | **Files Changed**: 3 | **+529/-0**

Details:

- Add generate-changelog.sh script
  - Parses git commit history
  - Categorizes by Conventional Commits format
  - Groups by date
  - Outputs Markdown format with commit links
- Generate initial CHANGELOG.md
  - 122 commits documented
  - Organized by date and category
  - Features, fixes, docs, refactoring, tests, chores, styling
  - Direct links to GitHub commits
- Update TOOLS_REFERENCE.md
  - Add generate-changelog.sh documentation
  - Usage examples and parameters
  - Category legend with emojis

Changelog Categories: [NEW] Features (feat) [FIX] Bug Fixes (fix) [DOCS] Documentation (docs) [REFACTOR] Refactoring (refactor) [PERF] Performance (perf) [TEST] Tests (test) [STYLE] Styling (style) [CHORE] Chores (chore)

Benefits: - Automated changelog generation - Consistent commit history documentation - Easy version tracking - Better transparency for contributors

---

Commit: 7bdd456 | **Author**: SATOSHI KAWADA | **Files Changed**: 18 | **+1091/-44**

Details:

- Add comprehensive tools reference documentation (TOOLS_REFERENCE.md)
  - Deployment tools: 6 scripts (aws, azure, gcp, pulumi, frontend)
  - Testing tools: 5 scripts (e2e, endpoints, api, deployments, cicd)
  - Management tools: 4 scripts (secrets, monitoring, cicd-monitor, trigger)
  - Utilities: 3 scripts (diagnostics, import, cleanup)
  - Recommended workflows and usage examples
- Standardize script headers across all 17 scripts
  - Add metadata: Script Name, Description, Author, Version
  - Add usage information: Parameters, Examples
  - Add prerequisites and exit codes
  - Improve maintainability and documentation

Scripts updated: - cleanup-old-resources.sh - deploy-aws-pulumi.sh - deploy-aws.sh, deploy-azure.sh, deploy-gcp.sh - deploy-frontend-aws.sh - diagnostics.sh - import-gcp-resources.sh - manage-github-secrets.sh - monitor-cicd.sh, trigger-workflow.sh - setup-monitoring.sh - test-api.sh, test-cicd.sh, test-deployments.sh - test-e2e.sh, test-endpoints.sh

Benefits: - Consistent script documentation format - Clear usage instructions for all tools - Better onboarding for new developers - Quick reference for troubleshooting

---

Commit: acd09c8 | **Author**: SATOSHI KAWADA | **Files Changed**: 2 | **+321/-0**

Details:

- Add Azure Functions Flex Consumption Plan troubleshooting section
  - Problem 1: Deployment shows 'Partially Successful'but function works
  - Problem 2: defaultHostName returns null for Flex Consumption
  - Problem 3: Kudu restart during deployment causes failures
- Add frontend workflow authentication error resolution
  - AWS: OIDC to static credentials migration
  - GCP: Workload Identity to credentials_json migration
- Add E2E test suite documentation to README
  - Test coverage: 18 tests (3 clouds × 6 operations)
  - Health checks + Full CRUD validation
  - Cloud-specific path handling (Azure Flex Consumption)

  - Data persistence verification

- Update troubleshooting history (2026-02-15)

Resolves deployment issues encountered in commits: - 4315dd7 (Azure API URL path fix) - 37aa17d (Cosmos DB direct fetch) - 202f555 (Package optimization + retry logic) - 8f75613 (Partially successful detection) - 3a4b1ec (Success message priority) - 6005d4e (hostname list fix) - 41b1efd (Frontend auth unification)

Commit: `11cb619` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+200/-0**

Details:

- Tests health endpoint on all 3 clouds (AWS/GCP/Azure)
- Tests full CRUD operations: Create, Read (list & single), Update, Delete
- Validates data consistency and proper error handling
- All 18 tests passing across AWS Lambda, GCP Cloud Run, and Azure Functions
- Handles cloud-specific path structures automatically

Commit: `41b1efd` | **Author**: SATOSHI KAWADA | **Files Changed**: 2 | **+4/-4**

Details:

- AWS: switch from OIDC (role-to-assume) to static credentials
- GCP: switch from Workload Identity to credentials_json
- Align with main deployment workflows authentication method
- Fixes 'Could not load credentials'errors

Commit: `6005d4e` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+12/-7**

Details:

- Flex Consumption plan does not populate defaultHostName field
- Switch to 'az functionapp config hostname list'which works reliably
- Apply fix to both 'Use Portal-Created Resources'and 'Verify Deployment'steps
- Fixes 'https:///'URL issue where hostname was null

Commit: `8d3fceb` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+32/-1**

Details:

- Add retry mechanism (up to 10 attempts with 10s intervals) for getting hostname
- Fixes 'https:///'URL issue where hostname was empty
- Function App may not be immediately queryable after deployment
- Validate hostname is not empty or 'None'before proceeding

Commit: `3a4b1ec` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+10/-3**

Details:

- Add explicit check for 'Deployment was successful'message as highest priority
- Fixes issue where successful deployments were retried unnecessarily
- Flex Consumption plan doesn't output detailed step logs, so we need to rely on the success message
- Reorder checks: success message → step completion → critical errors

---

Commit: `8f75613` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+42/-32**

Details:

Problem: - Azure reports 'ERROR: Deployment was partially successful'- Script treats this as failure and exits - BUT actual function is working perfectly (health check passes)

Root Cause: - Azure Flex Consumption plan deployment shows 'partially successful' even when all steps complete - Script logic: ERROR: in output → fail immediately - Reality: 'partially successful'with all steps completed = SUCCESS

Evidence: - Kudu logs show all steps completed: [OK] ValidationStep completed [OK] ExtractZipStep completed [OK] OryxBuildStep completed (48s build time) [OK] PackageZipStep completed [OK] UploadPackageStep completed [OK] RemoveWorkersStep completed [OK] SyncTriggerStep starting - Health check returns 200 OK with proper response - Function is fully operational

Solution: 1. Improved deployment detection: - Check for ' UploadPackageStep.*completed'(deployment uploaded) - Check for 'SyncTriggerStep'(final sync started) - Ignore 'partially successful'message - Only fail on critical errors (not 'partially successful')

2. Enhanced health check validation:
   - Extended timeout to 3 minutes (was 2 minutes)
   - Make health check mandatory (exit 1 if fails)
   - Capture and display health response
   - health_check_passed flag for proper exit code
3. Better error handling:
   - Distinguish 'partially successful'from real errors
   - Continue deployment if key steps completed
   - Final verification via health check
   - Clear success/failure messaging

Expected Flow: 1. Deployment runs → 'partially successful'message 2. Script checks: UploadPackageStep completed? → YES 4. Health check: curl /health → 200 OK 5. Final: [OK] Verified deployment success

Testing: - Manually verified: Function already working from previous deploy - curl health endpoint → {“status”:“ok”,“cloud_provider”:“azure”}

Commit: 202f555 | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+62/-9**

Details:

Problem: - Kudu restarted during deployment (Flex Consumption plan limitation) - Large deployment package causing resource constraints - No retry mechanism for transient Kudu issues

Root Cause: - Deployment package included unnecessary files (**pycache**, .pyc, tests, .dist-info) - Azure Functions Flex Consumption dynamically scales, causing Kudu restarts - Single deployment attempt fails on transient infrastructure issues

Solution: 1. Optimize deployment package: - Remove **pycache** directories - Delete .pyc/.pyo compiled files - Exclude test directories - Remove .dist-info metadata - Use –no-cache-dir for pip install - Use -q flag for zip (quiet mode) - Report package size for monitoring

2. Add intelligent retry logic:
   - Retry up to 3 times on deployment failure
   - Detect Kudu restart errors specifically
   - Wait 30 seconds between retries for Kudu stabilization
   - Distinguish transient vs permanent errors
   - Exit immediately on non-transient errors
3. Improve logging:
   - Capture deployment output to log file
   - Check for ERROR patterns
   - Show retry attempt numbers
   - Confirm success before proceeding

Expected Results: - Smaller package → faster upload → less Kudu stress - Transient Kudu restart → auto-retry → eventual success - Permanent errors → fail fast with clear message

Previous attempts: - Attempt 1: 4m deployment time, Kudu restart, no retry → FAIL - Attempt 2: Expected < 3m, auto-retry on Kudu restart → SUCCESS

Commit: 37aa17d | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+51/-9**

Details:

Problem: - Environment variables were all null - Workflow tried to get COSMOS_DB_ENDPOINT/KEY from existing app settings - On first deploy, these settings don’t exist yet → null → set null → fail

Root Cause: - Workflow used: az functionapp config appsettings list - This returns null for non-existent settings - Created a circular dependency issue

Solution: 1. Get Cosmos DB credentials directly from the resource: - az cosmosdb show –query documentEndpoint - az cosmosdb keys list –query primaryMasterKey

2. Add validation:
   - Exit with error if credentials cannot be retrieved
3. Improve deployment:
   - Add –timeout 600 to zip deployment
   - Suppress appsettings output (security)
4. Add post-deployment verification:
   - Wait up to 2 minutes for function to be ready
   - Curl health endpoint to verify deployment
   - Workaround for Flex Consumption plan log limitations

Expected Result: - Cosmos DB credentials properly retrieved: [OK] - Environment variables correctly set: [OK] - Function app deployment succeeds: [OK] - Health check passes: [OK]

Commit: `4315dd7` | **Author**: SATOSHI KAWADA | **Files Changed**: 2 | **+23/-6**

Details:

Problem: - Azure frontend showed ERR_NAME_NOT_RESOLVED errors - API paths were resolving as relative URLs (e.g., 'api/HttpTrigger/api/messages/') - VITE_API_URL was not correctly embedded in build

Root Cause: - Azure Functions URL structure: https://xxx.azurewebsites.net/api/HttpTrigger - Frontend was appending '/api/messages', resulting in '/api/HttpTrigger/api/messages'- This double '/api'path was correct for the backend but confusing

Solution: 1. Enhanced frontend API client (client.ts): - Detect Azure Functions URLs (contains '/api/HttpTrigger') - Use basePath="for Azure (no '/api'prefix) - Use basePath='/api'for AWS/GCP

2. Improved deployment workflow logging:
   - Echo API_URL during build step
   - Verify URL was embedded in built assets
   - Add explicit API URL logging in resource detection

Result: - Azure: https://xxx.azurewebsites.net/api/HttpTrigger + /messages [OK] - AWS/GCP: https://xxx.run.app + /api/messages [OK]

Testing: curl https://xxx.azurewebsites.net/api/HttpTrigger/messages

---

Commit: aa4d402 | **Author**: SATOSHI KAWADA | **Files Changed**: 5 | **+344/-461**

Details:

Changes: - [NEW] NEW: manage-github-secrets.sh (統合) - Merged set-github-secrets.sh (auto mode) - Merged setup-github-secrets.sh (guide mode) - Added –check-local flag for env validation

- [CHORE] ENHANCED: monitor-cicd.sh
  - Added –workflow=NAME filter option
  - Absorbed watch-workflow.sh functionality
- [DELETE] REMOVED: 3 duplicate scripts
  - set-github-secrets.sh (9.3K)
  - setup-github-secrets.sh (6.3K)
  - watch-workflow.sh (992B)

Impact: - Reduced script count: 19 → 17 (-11%) - Eliminated ~16.3KB of duplicate code - Improved maintainability (single source of truth) - Enhanced UX (unified interfaces)

Usage: ./ manage-github-secrets.sh –mode=auto ./ manage-github-secrets.sh –mode=guide ./monitor-cicd.sh –workflow=deploy-aws.yml

---

Commit: d19845b | **Author**: SATOSHI KAWADA | **Files Changed**: 4 | **+276/-16**

Details:

- Updated all cloud provider URLs to current production endpoints
- Added troubleshooting sections for AWS Lambda ImportModuleError
- Added GCP Cloud Run 500 error solutions (env vars, query_string)
- Added Azure Functions 500 error solutions (Cosmos DB env vars)
- Updated test scripts with correct URLs

Commit: 4a175fe | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+13/-2**

Commit: 4fae1eb | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+13/-1**

Commit: e8263fa | **Author**: SATOSHI KAWADA | **Files Changed**: 2 | **+2/-2**

Commit: 9fe32c6 | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+1/-8**

Commit: `ed0f6ff` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+1/-0**

Commit: `8307b66` | **Author**: SATOSHI KAWADA | **Files Changed**: 2447 | **+8/-366328**

Details:

- Add services/api/index.py with Mangum adapter for AWS Lambda
- Update deploy-lambda-aws.sh to include index.py in deployment package
- Add services/api/.build/ to .gitignore (pip install artifacts)
- Remove previously tracked .build/ directory from git

Fixes: Runtime.ImportModuleError: Unable to import module 'index'Context: Lambda function was deployed without application code due to Pulumi ignore_changes setting

---

Commit: `11af839` | **Author**: SATOSHI KAWADA | **Files Changed**: 5 | **+333/-0**

Details:

- Add static-site/ with index.html and error.html
- GCP: Deploy to Cloud Storage bucket
- AWS: Deploy to S3 with website hosting
- Azure: Deploy to Azure Storage Static Website
- All workflows include cache-control headers
- Auto-triggered on static-site changes

Commit: `0b6a199` | **Author**: SATOSHI KAWADA | **Files Changed**: 3 | **+223/-0**

Details:

- GCP: Deploy to Cloud Storage with public access
- AWS: Deploy to S3 with website hosting
- Azure: Deploy to Azure Storage Static Website
- All workflows include cache-control headers
- Auto-triggered on frontend changes

Commit: `cea0886` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+64/-11**

Commit: `6b4bb9e` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+12/-57**

Commit: `b57decb` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+11/-1**

Commit: `eaadd8b` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+10/-2**

Commit: `cfda862` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+1/-1**

Commit: `5bc9538` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+1/-1**

Commit: `ffa5431` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+1/-1**

Commit: `3829792` | **Author**: SATOSHI KAWADA | **Files Changed**: 2 | **+218/-1**

Commit: `d6ee071` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+2/-2**

Commit: `03df82e` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+1/-1**

Commit: `408e276` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+16/-3**

Commit: `4283ec8` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+13/-0**

Commit: `2457cde` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+15/-33**

Commit: `b620fb2` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+85/-0**

Commit: `e140b8d` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+3/-2**

Commit: `395b0ef` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+5/-4**

Commit: `0c524cd` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+13/-0**

Commit: `c61a216` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+4/-4**

Commit: `d1e15a8` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+85/-0**

Commit: 2f2acfa | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+1/-1**

Commit: 96dfea3 | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+3/-3**

Commit: 062840f | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+1/-1**

Commit: f6fc6c2 | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+3/-3**

Commit: cc66be5 | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+1/-0**

Commit: a9fd4c9 | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+0/-1**

Commit: c84734d | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+1/-1**

Commit: 5d388e5 | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+2/-2**

Commit: 6e5d5f8 | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+2/-1**

Commit: e6921ed | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+16/-29**

Commit: 144ada0 | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+22/-12**

Commit: 83ff516 | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+8/-0**

Commit: b20d984 | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+13/-11**

Commit: 3449db4 | **Author**: SATOSHI KAWADA | **Files Changed**: 6 | **+43/-17**

Commit: 4845887 | **Author**: SATOSHI KAWADA | **Files Changed**: 14 | **+681/-421**

Details:

[NEW] Complete Infrastructure as Code migration: - Replace Terraform with Pulumi Python implementation - All 3 clouds: AWS, Azure, GCP

[PACKAGE] Infrastructure Components: AWS: - Lambda Function (Python 3.12, ZIP deployment) - API Gateway HTTP API v2 - S3 Static Website - IAM Roles

Azure: - Azure Functions (Consumption Plan Y1) - Storage Accounts (Functions + Frontend) - Application Insights - Random naming for global uniqueness

GCP: - Cloud Storage (Function source + Frontend) - Cloud Functions Gen 2 (via gcloud CLI) - IAM bindings for public access

[CHORE] Workflow Updates: - GitHub Actions now use Pulumi CLI - pulumi up for infrastructure deployment - Outputs retrieved via pulumi stack output - Maintain ZIP-based deployment for all Functions

[DELETE] Cleanup: - Removed all Terraform files - Deleted infrastructure/terraform directory

[TIP] Benefits: - Real programming language (Python) for IaC - Better type safety and error checking - More flexible conditional logic - Unified codebase management

---

Commit: `94a5afe` | **Author**: SATOSHI KAWADA | **Files Changed**: 2 | **+20/-68**

Details:

- Remove Cloud Functions resource from Terraform (causes 404 error)
- Terraform now only manages buckets and IAM
- gcloud functions deploy creates/updates Cloud Functions after ZIP upload
- Fixes error: 'No such object: function-source.zip'

Commit: `beb49a8` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+11/-4**

Details:

- Change from multicloudautodeploystagin g{func/web} (32/31 chars)
- To mcadstg{func/web} + random 6-char suffix (18/16 chars)
- Azure requires storage names: 3-24 chars, lowercase + numbers only

---

Commit: `3901489` | **Author**: SATOSHI KAWADA | **Files Changed**: 9 | **+575/-71**

Details:

- Replace Azure Container Apps with Azure Functions (Consumption Plan)
- Replace GCP Cloud Run with Cloud Functions Gen 2
- Add Azure Functions entry point (function_app.py)
- Add Cloud Functions entry point (function.py)
- Update CI/CD workflows to ZIP deployment

- Add azure-functions and functions-framework dependencies

Benefits: - Consistent deployment model across AWS/ Azure/ GCP - Faster deployments: 30-70s (vs 2-5min with containers) - Lower costs: Consumption-only billing (estimated 0-80/month savings) - Simpler CI/CD: No Docker builds or registry management

---

Commit: `b2e82b7` | **Author**: SATOSHI KAWADA | **Files Changed**: 2 | **+256/-2**

Details:

- Add detailed CI/CD test results documentation (CICD_TEST_RESULTS.md)
- Document all discovered issues and their resolutions
- Include AWS deployment success evidence
- Add CI/CD tools overview (test-cicd.sh, trigger-workflow.sh, monitor-cicd.sh)
- Update README with links to CI/CD test results and quick reference
- Provide recommendations for IAM permissions and future testing

---

Commit: `128d13d` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+6/-4**

Details:

- Continue deployment even if apigateway:GET permission is missing
- Fall back to default endpoint when API Gateway lookup fails
- This ensures Lambda update succeeds independent of IAM permissions for API Gateway
- Lambda deployment is the critical step, API endpoint discovery is optional

---

Commit: `7aa7bc9` | **Author**: SATOSHI KAWADA | **Files Changed**: 3 | **+3/-3**

Details:

- Change cp -r src/* to cp -r app/* in Package Backend step
- Use standard Dockerfile instead of Dockerfile.azure/gcp (they don't exist)
- This fixes the 'No such file or directory'error in CI/CD
- All local tests passing (100% success rate)

Commit: `621bb55` | **Author**: SATOSHI KAWADA | **Files Changed**: 6 | **+720/-11**

Details:

- Fix deploy-aws.yml: backend → api, frontend → frontend_react
- Fix deploy-azure.yml: backend → api, frontend → frontend_react
- Fix deploy-gcp.yml: backend → api, frontend → frontend_react
- Add comprehensive CI/ CD testing scripts (test-cicd.sh, trigger-workflow.sh, monitor-cicd.sh)

- All local tests passing (100% success rate)

Commit: `8c9b8fb` | **Author**: SATOSHI KAWADA | **Files Changed**: 6 | **+1696/-0**

Details:

New Scripts: - scripts/deploy-lambda-aws.sh: Full Lambda deployment automation - Auto dependency installation (manylinux2014_x86_64) - ZIP package creation and S3 upload - Lambda function create/update - API Gateway integration setup - Correct Lambda permissions (HTTP API SourceArn format) - CloudWatch Logs access log configuration

- scripts/test-api.sh: Complete API integration tests
  - Health check
  - CRUD operations (Create, Read, Update, Delete)
  - Pagination testing
  - Error handling validation
  - Pass/fail reporting with statistics
- scripts/setup-monitoring.sh: CloudWatch monitoring setup
  - SNS topic and email notifications
  - Lambda alarms (errors, throttles, duration, concurrency)
  - API Gateway alarms (5XX errors, latency)
  - DynamoDB alarms (read/write throttles)
  - CloudWatch Logs metric filters
  - Auto dashboard creation

Documentation: - docs/QUICK_REFERENCE.md: Quick reference for common operations - Deployment commands - Testing and debugging - Log inspection - Monitoring and metrics - Troubleshooting - Resource management - Useful one-liners

- docs/TROUBLESHOOTING.md: Added Lambda + API Gateway section
  - HTTP API vs REST API SourceArn difference
  - Permission troubleshooting steps
  - Access log enablement
  - Debug workflow
- README.md: Enhanced with tooling and recommendations
  - New script usage documentation
  - Recommended AWS services for production:
    - AWS X-Ray (distributed tracing)
    - AWS WAF (security)
    - Route 53 + Custom Domain
    - Parameter Store/Secrets Manager
    - Lambda Layers (optimization)

    - CloudFront Functions (edge processing)
    - AWS Backup (data protection)
  - Implementation examples for each service

All scripts are executable and production-ready.

Commit: `f29afbc` | **Author**: SATOSHI KAWADA | **Files Changed**: 2444 | **+366328/-0**

Details:

- Root cause: HTTP API requires SourceArn format: {api-id}// (not {api-id}///*)
- Fixed Lambda resource policy with correct SourceArn pattern
- Enabled API Gateway access logs for debugging
- Updated frontend .env to use Lambda API (from GCP Cloud Run)
- Rebuilt and deployed React app to S3
- Invalidated CloudFront cache
- Verified: GET, POST, PUT operations all working via Lambda + API Gateway
- AWS-only stack now fully operational (S3, CloudFront, Lambda, API Gateway, DynamoDB)

Commit: `24ba2b0` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+12/-0**

Details:

- Create Dockerfile.lambda for AWS Lambda Container Image
- Make AWS region optional in AWSBackend (auto-detect from environment)
- Build and deploy to ECR for x86_64 architecture
- Replace ZIP-based Lambda with container-based Lambda

This enables AWS-only staging environment without GCP Cloud Run dependency

---

Commit: `6539b0d` | **Author**: SATOSHI KAWADA | **Files Changed**: 2 | **+8/-3**

Details:

- Allow boto3 to auto-detect region from environment
- Lambda automatically provides AWS_REGION variable
- Fallback to settings.aws_region if provided

Commit: `3870668` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+1/-1**

Details:

- Backend uses PUT /api/messages/{id}
- Frontend was using PATCH causing 404/405 errors

- Update edit functionality now works correctly

Commit: ab1b003 | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+62/-1**

Details:

- Add update_message implementation for DynamoDB
- Fix TypeError: abstract method not implemented
- Support partial updates with exclude_unset
- Add updated_at timestamp tracking
- Use DynamoDB UpdateExpression for atomic updates

Commit: d6ac8de | **Author**: SATOSHI KAWADA | **Files Changed**: 2 | **+382/-0**

Details:

Successfully deployed React SPA to replace Reflex SSR frontend:

Deployment Details: - S3 Bucket: multicloud-auto-deploy-staging-frontend - CloudFront: E2GDU7Y7UGDV3S (dx3l4mbwg1ade.cloudfront.net) - API: https://mcad-staging-api-son5b3ml7a-an.a.run.app - Region: ap-northeast-1 (Tokyo)

Performance: - Build size: 90KB gzipped (vs 500KB+ for Reflex) - TTFB: 20-50ms (CDN cached, vs 200-500ms SSR) - Cost: $1-5/ month (vs $20-50/ month for containers) - Lighthouse: Expected 95+ (vs 70-80 for SSR)

Files Added: - docs/ REACT_FRONTEND_DEPLOYMENT.md: Complete deployment guide - scripts/deploy-frontend-aws.sh: Automated deployment script

Cache Strategy: - Assets (CSS/JS): max-age=31536000 (1 year, content-hashed) - HTML: max-age=0 (always revalidate) - CloudFront invalidation: Automatic on deploy

Next Steps: - Phase 2B: Deploy to Azure Blob Storage + CDN - Phase 2C: Deploy to GCP Cloud Storage + CDN - Phase 3: Update CI/CD pipeline for automated deployments - Phase 4: Add staging/production environment separation

---

Commit: 5863645 | **Author**: SATOSHI KAWADA | **Files Changed**: 27 | **+5734/-0**

Details:

- Create React + TypeScript + Vite frontend application
- Implement full CRUD operations for messages
- Add TanStack Query for efficient data management
- Design responsive UI with Tailwind CSS
- Build optimized static files (88KB gzipped)

Architecture change: - OLD: Reflex SSR (Container Apps/Cloud Run) - 0-50/month - NEW: React SPA (Static CDN hosting) - -5/month

Components: - MessageForm: Create new messages - MessageList: Display paginated messages - MessageItem: Individual message with edit/delete

Features: - Real-time data synchronization - Optimistic UI updates - Dark mode support- Error handling and loading states - Mobile-responsive design - TypeScript type safety

Build output: - dist/index.html (0.46KB) - dist/assets/index.css (4.23KB → 1.36KB gzipped) - dist/assets/index.js (274.66KB → 88.07KB gzipped)

Next: Deploy to S3, Blob Storage, Cloud Storage + CDN

Commit: `8c3b061` | **Author**: SATOSHI KAWADA | **Files Changed**: 3 | **+593/-23**

Details:

- Add deploy-multicloud.yml workflow for Azure and GCP
- Support Docker image build and push to ACR and Artifact Registry
- Deploy to Azure Container Apps and GCP Cloud Run
- Add health check validation after deployment
- Create comprehensive CI/CD setup documentation
- Update README with new workflow information

Features: - Parallel deployment to Azure and GCP - Configurable deployment targets (all/azure/gcp) - Environment selection (staging/production) - Docker buildx with cache optimization - Automated health checks - GitHub Actions summary with deployment URLs

Documentation: - docs/CI_CD_SETUP.md: Complete setup guide with secrets - README.md: Updated with new workflow table and badges

---

Commit: `384a36e` | **Author**: SATOSHI KAWADA | **Files Changed**: 2 | **+34/-5**

Details:

- Add production-ready Dockerfile for Reflex frontend
- Add entrypoint.sh with single-port mode support for Cloud Run
- Install unzip package required for Bun installation
- Build frontend assets at Docker build time with 'reflex export'
- Support both dual-port (Azure) and single-port (GCP) modes
- Deploy to Azure Container Apps and GCP Cloud Run

Deployed services: - Azure: https://mcad-staging-frontend.livelycoast-fa9d3350.japaneast.azurecontainera
- GCP: https://mcad-staging-frontend-son5b3ml7a-an.a.run.app

Commit: `8215297` | **Author**: SATOSHI KAWADA | **Files Changed**: 6 | **+543/-4**

Details:

Infrastructure: - Add ECR repositories for API and Frontend containers - Update Pulumi stack with image registry support - Add lifecycle policies to manage image retention (keep last 10)

Environment Configuration: - Add .env.example for frontend_reflex - Document environment variables for production

Deployment: - Create comprehensive production deployment guide - Add deploy-aws-pulumi.sh script for automated deployment - Include AWS App Runner, ECS Fargate deployment options - Document secrets management with AWS Secrets Manager - Add CI/CD pipeline examples for GitHub Actions

Documentation: - PRODUCTION_DEPLOYMENT.md with complete deployment workflows - Health checks, monitoring, rollback procedures - Cost optimization strategies - Security best practices

Ready for production deployment with: - Containerized API and Frontend - ECR image storage - Infrastructure as Code (Pulumi) - Automated deployment scripts

---

Commit: `7c77283` | **Author**: SATOSHI KAWADA | **Files Changed**: 3 | **+18/-28**

Details:

Docker image optimization: - Remove unused dependencies (unzip from frontend_reflex) - Add –no-install-recommends to apt-get - Clean apt cache and Python **pycache** - Add PYTHONDONTWRITEBYTECODE=1 environment variable - Use selective COPY instead of COPY . . - Apply optimizations to both api and frontend_reflex

Cleanup: - Remove all test data (74 messages) - Delete Python cache files (**pycache**/ *.pyc) - Remove legacy backend service from docker-compose.yml - Remove obsolete ‘version’directive from docker-compose.yml

Benefits: - Reduced image size overhead - Faster build times with layer caching - Cleaner production images

---

Commit: `dd0ffbb` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+18/-28**

Details:

- Update project structure (frontend_reflex instead of web/frontend)
- Remove Node.js from prerequisites (pure Python stack)
- Update tech stack (Reflex 0.8+, no JavaScript/React)
- Update port numbers (3002 for Reflex frontend)
- Update Docker Compose service names
- Emphasize pure Python full-stack approach

Commit: `6935e12` | **Author**: SATOSHI KAWADA | **Files Changed**: 22 | **+620/-4339**

Details:

- Add new Reflex frontend (services/frontend_reflex/)
  - Pure Python implementation (447 lines)
  - All CRUD operations (create, read, update, delete)
  - Image upload with MinIO integration
  - Pagination support (20 items per page)
  - WebSocket state management
  - Proper HTTP status code handling (200/201/204)
- Docker integration
  - Add Dockerfile for Reflex frontend
  - Update docker-compose.yml (ports 3002/3003)
  - Add .dockerignore for optimized builds
- Remove legacy React frontend
  - Delete services/frontend/ directory (99MB freed)
  - Remove frontend service from docker-compose.yml
- Benefits
  - Full-stack Python development
  - Simplified dependency management
  - Better type safety and code consistency
  - Reduced JavaScript/Node.js complexity

Commit: `fc93ecb` | **Author**: SATOSHI KAWADA | **Files Changed**: 2 | **+22/-18**

Commit: `f5fbb84` | **Author**: SATOSHI KAWADA | **Files Changed**: 30 | **+0/-1707**

Details:

Remove legacy implementations and free up 1.1GB of disk space:

- Remove services/backend/ (192MB) - old FastAPI implementation, now using services/api/
- Remove services/web/ (28KB) - Reflex frontend, now using React in services/frontend/
- Remove services/database/ - empty directory
- Remove infrastructure/terraform/ (948MB) - migrated to Pulumi

The project now uses: - Backend: services/api/ (FastAPI + Python) - Frontend: services/frontend/ (React + TypeScript) - IaC: infrastructure/pulumi/ (Python) - Storage: MinIO (local), S3/GCS/Azure Storage (cloud)

Commit: `a563f84` | **Author**: SATOSHI KAWADA | **Files Changed**: 5 | **+194/-32**

Details:

- Add MessageUpdate model with optional fields for partial updates
- Implement update_message method in BaseBackend and LocalBackend
- Add PUT /api/messages/{id} endpoint for updating messages
- Add edit mode UI with inline editing in frontend
- Support updating content and author while preserving created_at
- Add updated_at timestamp on message updates
- Include edit and cancel buttons in message cards

Commit: `052cecc` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+33/-4**

Details:

実装内容: - フロントエンドに削除ボタン追加 - ゴミ箱アイコンの SVG 表示 - ホバー時の視覚的フィードバック - メッセージカードの右上に配置

- 削除確認ダイアログ
  - window.confirm() で削除前に確認
  - メッセージ内容の一部を表示（最大 50 文字）
  - キャンセル可能
- DELETE API 呼び出し
  - axios.delete() でバックエンド API を呼び出し
  - 削除成功後に自動的にメッセージリストを更新
  - エラーハンドリング付き

技術詳細: - handleDeleteMessage 関数を追加 - メッセージ ID とコンテンツを引数に受け取る - 削除後は fetchMessages() で再取得

テスト結果: [OK] メッセージ削除: 成功（9 件 →7 件）[OK] 削除確認ダイアログ: 動作確認 [OK] API 呼び出し: 正常 [OK] MinIO 永続化: 確認済み（再起動後も 7 件）[OK] リスト自動更新: 正常

Commit: `83f8e8f` | **Author**: SATOSHI KAWADA | **Files Changed**: 3 | **+250/-6**

Details:

実装内容: - POST /api/uploads エンドポイント作成 - 画像ファイルのバリデーション (形式、サイズ) - MinIO への画像保存 (images/ フォルダ) - ブラウザアクセス可能な URL を返却

- フロントエンド画像アップロード UI
  - ファイル選択ボタン
  - 画像プレビュー表示
  - メッセージ送信時に画像 URL を含める
  - メッセージ一覧での画像表示

技術詳細: - FastAPI UploadFile でマルチパート処理 - MinIO 公開読み取りポリシー設定 (images/ フォルダ) - React useState で画像プレビュー管理 - 最大 10MB、画像形式のみ対応

テスト結果: [OK] 画像アップロード成功 [OK] 画像付きメッセージ作成成功 [OK] MinIO 永続化確認 [OK] ブラウザから画像表示可能

Commit: `48d9883` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+112/-18**

Details:

Changes: - Replace in-memory dict storage with MinIO object storage - Store messages as JSON files in MinIO bucket (simple-sns/ messages/) - Auto-create bucket on initialization - Implement full CRUD operations with MinIO SDK: * create_message: Save as JSON to MinIO * get_messages: List and read all message objects * get_message: Read single message by ID * delete_message: Remove object from MinIO

Benefits: - Data persists across container restarts - S3-compatible API (easy to migrate to real S3) - Compatible with existing API interface - Automatic bucket management

Testing: - [OK] Created 4 messages successfully - [OK] Retrieved messages from MinIO - [OK] Restarted API container - [OK] All 4 messages still available after restart - [OK] MinIO console shows JSON files in simple-sns/messages/

Architecture: - MinIO container stores data in Docker volume (minio-data) - Messages stored as: messages/{uuid}.json - JSON format matches Message model schema

---

Commit: `a41db81` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+1/-1**

Details:

Root cause: - Browser runs on host machine, not inside Docker network - 'api:8000' hostname only resolves within Docker network - Browser cannot resolve 'api'hostname (ERR_NAME_NOT_RESOLVED)

Fixed: - VITE_API_URL=http://api:8000 → http://localhost:8000

Architecture: - Browser (host) → localhost:8000 → Docker port mapping → api:8000 (container) - Docker internal services can still use 'api:8000'hostname - Port 8000 is exposed to host via docker-compose ports mapping

After this fix, browser can successfully connect to FastAPI backend.

Commit: `39cba48` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+1/-1**

Details:

Fixed issue where frontend was trying to connect to non-existent 'backend'service instead of 'api'service in Docker network.

Changed: - VITE_API_URL=http://backend:8000 → http://api:8000

This was causing 'メッセージの送信に失敗しました'error because the frontend could not reach the API service on the Docker network.

After this fix, users need to force-reload the browser (Ctrl+Shift+R) to clear the cached Vite build with the old environment variable.

Commit: 6add588 | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+3/-3**

Details:

Fixed issues: - Corrected h2 tag className from 'mb-space-y-3'to 'mb-4'- Added missing form tag with proper onSubmit handler - Removed duplicated closing tags that caused syntax error - Restored correct HTML structure for message submission form

The error was causing Babel parser to fail at line 126 with 'Unexpected token, expected jsxTagEnd'.

Commit: fd9a4d9 | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+55/-31**

Details:

Changes: - Update Message interface to match backend model (content, author, created_at) - Add MessageListResponse interface for paginated responses - Update API endpoints: /api/messages → /api/messages/, /api/health → /health - Change POST payload from { text } to { content, author } - Add author input field to the message form - Update message display to show author name and created_at timestamp - Fix cloud provider detection to use cloud_provider field

Integration Testing: - [OK] Health check endpoint working (Status: ok, Provider: local) - [OK] POST /api/messages/ creates messages successfully - [OK] GET /api/messages/ returns paginated message list - [OK] CORS configuration working correctly - [OK] Frontend accessible at http:// localhost:3001 - [OK] API docs accessible at http://localhost:8000/docs

User Flow: 1. Enter name in the author field 2. Enter message content 3. Click submit button 4. Message appears in the list below with author and timestamp

---

Commit: 4bd5e61 | **Author**: SATOSHI KAWADA | **Files Changed**: 7 | **+315/-3**

Details:

Changes: - Add BaseBackend abstract class (app/backends/**init**.py) - Implement LocalBackend with in-memory storage (app/ backends/ local.py) - Implement

AWSBackend with DynamoDB support (app/ backends/ aws.py) - Add backend factory for multi-cloud support (app/backends/factory.py) - Create messages router with CRUD operations (app/ routes/ messages.py) - Register messages router in main.py

API Endpoints: - POST /api/messages/ - Create message - GET /api/messages/ - List messages (with pagination) - GET /api/messages/{id} - Get single message - DELETE /api/messages/{id} - Delete message

Testing: - [OK] POST creates messages successfully - [OK] GET returns paginated message list - [OK] LocalBackend stores 6 test messages - [OK] Frontend accessible at http://localhost:3001 - [OK] API docs at http://localhost:8000/docs

---

Commit: `1db97c5` | **Author**: SATOSHI KAWADA | **Files Changed**: 3 | **+9/-10**

Details:

Changes: - Fix web service to use Dockerfile instead of inline command - Update httpx version requirement (>=0.25.1) for Reflex compatibility - Add Node.js to Reflex Dockerfile for proper frontend compilation - Switch Reflex to production mode to avoid init issues

Testing: - FastAPI (api): [OK] Working on http://localhost:8000 - React Frontend: [OK] Working on http:// localhost:3001 - MinIO: [OK] Working on http:// localhost:9000-9001

All services successfully running in local development environment.

Commit: `6087b76` | **Author**: SATOSHI KAWADA | **Files Changed**: 23 | **+1747/-13**

Details:

Major Changes: - Add FastAPI backend (services/api/) with multi-cloud support - Add Reflex frontend (services/web/) for complete Python UI - Add Pulumi IaC for AWS (infrastructure/pulumi/aws/) - Update docker-compose.yml with new Python services - Add comprehensive migration guide (docs/PYTHON_MIGRATION.md) - Update README with Python Full Stack section

New Services: - services/ api/ - FastAPI backend (Python 3.12) * Multi-cloud backends (AWS/Azure/GCP/Local) * Pydantic models and settings * Dockerized for easy deployment

- services/web/ - Reflex frontend (Python)
  - React-like components in Python
  - State management with async support
  - Full TypeScript replacement
- infrastructure/pulumi/aws/ - Pulumi IaC (Python)

- Lambda + API Gateway + DynamoDB + S3
- Replaces Terraform with Python-native IaC
- Complete AWS infrastructure as code

Benefits: - Unified Python stack (IaC + Backend + Frontend) - Type safety across entire codebase - Improved developer experience - Easier maintenance and debugging

23 files changed, 1800+ insertions

Commit: `1edfe9e` | **Author**: SATOSHI KAWADA | **Files Changed**: 11 | **+1586/-20**

Details:

Documentation: - Add TROUBLESHOOTING.md with all CI/CD issues and solutions * Azure authentication problems (Service Principal, Terraform Wrapper) * GCP resource conflicts (GCS backend, resource imports) * Frontend API connection issues (build order, API URL) * Permission errors and IAM configurations - Add ENDPOINTS.md with complete endpoint information * AWS, Azure, GCP API and frontend URLs * Management console links * API specification and test scripts - Update CICD_SETUP.md with troubleshooting references - Update README.md with latest information * All cloud providers now fully operational * New documentation and script references * Dev Container support info

Tools & Scripts: - Add test-endpoints.sh - Test all cloud endpoints - Add import-gcp-resources.sh - Import existing GCP resources - Add setup-github-secrets.sh - GitHub Secrets setup guide - Make all scripts executable

Dev Container: - Add .devcontainer/devcontainer.json with full tooling * Terraform 1.7.5, Node.js 18, Python 3.12 * AWS CLI, Azure CLI, gcloud CLI * Docker-in-Docker support * VS Code extensions for cloud development - Add .devcontainer/Dockerfile with additional utilities - Add .devcontainer/setup.sh for initialization - Include helpful aliases and functions

This completes the documentation and tooling setup after successful deployment to all three cloud providers (AWS, Azure, GCP).

Commit: `4e3be94` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+21/-9**

Details:

Issue: Frontend was building with incorrect API URL (us-east-1 instead of ap-northeast-1) Changes: - Move Build Frontend step after Lambda deployment - Dynamically fetch API Gateway endpoint from AWS - Use fetched API endpoint when building frontend (VITE_API_URL) - Update CloudFront distribution ID default value (E2GDU7Y7UGDV3S)

This ensures the frontend knows the correct API URL at build time.

Commit: `e467bb0` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+11/-14**

Details:

Issue: Frontend was building before infrastructure deployment, so API URL was not available Changes: - Move Deploy Infrastructure step before Build Frontend - Extract api_url from Terraform outputs - Use api_url output when building frontend (VITE_API_URL) - Frontend now built with correct API endpoint

This ensures the frontend knows the correct API URL at build time.

Commit: `522a639` | **Author**: SATOSHI KAWADA | **Files Changed**: 3 | **+52/-88**

Details:

Major changes: 1. Enable GCS backend for persistent Terraform state - Created gs://multicloud-auto-deploy-tfstate-gcp bucket - Granted storage.objectAdmin role to service account - Enabled backend in main.tf

2. Imported existing GCP resources to Terraform state
   - google_artifact_registry_repository.main (mcad-staging-repo)
   - google_storage_bucket.frontend (mcad-staging-frontend)
   - google_compute_global_address.frontend (mcad-staging-frontend-ip)
   - google_firestore_database.main ((default))
3. Simplified GitHub Actions workflow
   - Removed complex import logic (now handled by persistent state)
   - Use GitHub Actions outputs instead of terraform output commands
   - Cleaner workflow with better logging
4. Added missing Terraform outputs
   - artifact_registry_location
   - frontend_storage_bucket
   - api_url in workflow outputs

Benefits: - No more 'resource already exists'errors - Faster workflow execution (no repeated imports) - Consistent state across workflow runs - Easier to maintain and debug

---

Commit: `f33581c` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+51/-18**

Details:

Changes: - Add extensive logging to import process - Show environment variables being used - Display state check results - Show terraform state list after import - Add separators for better log readability

This will help diagnose why imports are failing and resources still show 'already exists'errors.

Commit: `2a5597c` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+28/-28**

Details:

Changes: - Add import_resource function to check state before importing - Skip import if resource already in state - Better error handling and progress messages - This reduces execution time and provides clearer feedback

Benefits: - Faster execution when resources already imported - Clear status messages ([OK] = in state, → = importing, [O] = not found) - Avoids redundant import attempts

---

Commit: `22219d8` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+39/-0**

Details:

Changes: - Add Firestore owner role to service account - Import existing GCP resources before terraform apply: * Artifact Registry repository * Firestore database (default) * Storage bucket for frontend * Global address for frontend - Ignore import errors if resources already in state - This prevents 'resource already exists'errors

Errors fixed: - Error 409: Repository/bucket/address already exists - Error 403: No permission for Firestore (role added)

Commit: `24c4870` | **Author**: SATOSHI KAWADA | **Files Changed**: 2 | **+36/-5**

Details:

Changes: - Add container_registry_name output (ACR name) - Add frontend_storage_account output (Storage Account name) - Add debug output in Deploy Infrastructure step - Add ACR_NAME validation in Build and Push Docker Image step

These outputs are required by the GitHub Actions workflow to: - Login to ACR for Docker image push - Deploy to Container App - Upload frontend files to Storage Account

---

Commit: `57857d2` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+17/-13**

Details:

Key changes: - Store Terraform outputs in Deploy Infrastructure step - Pass outputs to subsequent steps via GitHub Actions outputs - Avoid running 'terraform output' after Azure CLI login - This prevents CLI auth conflicts with Terraform

Terraform outputs stored: - acr_name (Container Registry) - resource_group (Resource Group) - container_app (Container App name) - storage_account (Frontend Storage Account)

Commit: `ced5376` | **Author**: SATOSHI KAWADA | **Files Changed**: 2 | **+8/-2**

Details:

Critical fixes: - Disable terraform_wrapper to ensure environment variables pass correctly - Clear Azure CLI config (az account clear) before Terraform execution - Explicitly disable use_cli, use_msi, use_oidc in provider - Environment variables (ARM_*) are the ONLY authentication method

This ensures Terraform uses Service Principal credentials exclusively.

Commit: `a1b6d5c` | **Author**: SATOSHI KAWADA | **Files Changed**: 2 | **+15/-12**

Details:

Changes: - Remove initial Azure Login step (not needed for Terraform) - Terraform uses ARM_* environment variables for Service Principal auth - Added use_msi=false and use_oidc=false to provider block - Login to Azure CLI only when needed for ACR operations - This prevents Terraform from attempting Azure CLI authentication

---

Commit: `6272301` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+0/-2**

Details:

- Backend block does not support use_cli or use_azuread_auth
- Authentication via ARM_* environment variables (already set in workflow)
- Keep use_cli=false in provider block (this is valid)

Commit: `4ede2ca` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+2/-4**

Details:

- Remove -backend-config flags from terraform init
- Backend settings (use_cli, use_azuread_auth) are already in main.tf
- Simplify to standard terraform init command

Commit: `4e587ee` | **Author**: SATOSHI KAWADA | **Files Changed**: 2 | **+11/-2**

Details:

- Add use_cli=false and use_azuread_auth=false to backend block
- Add backend-config flags to terraform init command

- Add ARM_USE_CLI and ARM_USE_AZUREAD_AUTH environment variables
- Ensures both backend and provider use Service Principal authentication

---

Commit: `ea94b19` | **Author**: SATOSHI KAWADA | **Files Changed**: 2 | **+19/-0**

Details:

- Add use_cli = false to azurerm provider to prevent Azure CLI auth
- Add az logout step before Terraform to avoid authentication conflicts
- Re-login to Azure CLI after Terraform for ACR operations
- Ensures proper Service Principal authentication for Terraform

---

Commit: `8f3ce46` | **Author**: SATOSHI KAWADA | **Files Changed**: 3 | **+179/-23**

Details:

- Add GitHub Actions workflow status badges
- Add live demo URLs for AWS/Azure/GCP deployments
- Update technology stack (Python 3.12, x86_64, Terraform 1.14.5)
- Add detailed CI/CD deployment flow
- Document development tools (Makefile, diagnostics script)
- Update architecture section with actual deployment status
- Add IAM policy for GitHub Actions deployment

Commit: `d8b6330` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+11/-9**

Details:

- Remove all cleanup commands that were causing failures
- Use simpler packaging process without optimization
- Package size will be ~4.3MB (still under 250MB S3 limit)
- Focus on reliability over optimization

Commit: `3437bf0` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+7/-4**

Details:

- Use xargs for **pycache** removal instead of -exec
- Add || true to all cleanup commands to prevent pipeline failures
- Add package size display for debugging
- Tested locally: successfully creates 2.8MB package (35% size reduction)

Commit: `58e6bfb` | **Author**: SATOSHI KAWADA | **Files Changed**: 2 | **+60/-4**

Details:

- Remove **pycache**, *.pyc, tests,* .dist-info to reduce package size
- Add detailed logging for each step ([PACKAGE] [CLOUD] [DEPLOY])
- Add error handling with clear failure messages
- Add workflow monitoring script
- Exit immediately on any command failure (set -e)

Commit: `095074d` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+9/-1**

Details:

- Lambda package is ~58MB, exceeding direct upload limit (50MB)
- Upload to S3 first, then update Lambda from S3
- Fixes exit code 254 error (RequestEntityTooLargeException)

Commit: `6c202c9` | **Author**: SATOSHI KAWADA | **Files Changed**: 2 | **+277/-0**

Details:

- Add Makefile with common tasks (build, test, deploy)
- Add diagnostics.sh script for troubleshooting
- Makefile supports: build-frontend, build-backend, test-all, deploy-aws
- Diagnostics checks: tools, Python env, cloud auth, deployments, AWS resources

---

Commit: `39bbaab` | **Author**: SATOSHI KAWADA | **Files Changed**: 2 | **+5/-2**

Commit: `5f6f8ab` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+2/-0**

Commit: `ee78f96` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+1/-1**

Commit: `d645961` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+4/-9**

Commit: `4db9645` | **Author**: SATOSHI KAWADA | **Files Changed**: 4 | **+5/-5**

Details:

- Update Terraform backend to new S3 bucket in ap-northeast-1
- Update all region references in Terraform configs
- Update GitHub Actions workflow AWS_REGION
- Update deployment scripts

- Migrate Terraform state to new bucket with versioning and encryption

New bucket: multicloud-auto-deploy-terraform-state-apne1

Commit: `847ba08` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+32/-3**

Details:

- Remove dependency on deploy-aws.sh script
- Directly update Lambda function code
- Sync frontend to S3 bucket
- Invalidate CloudFront cache
- Use secrets for resource names with fallback defaults

Commit: `54b3c5f` | **Author**: SATOSHI KAWADA | **Files Changed**: 1 | **+81/-5**

Details:

- Add Azure ACR login server retrieval method
- Add GCP service account key (GCP_CREDENTIALS) detailed steps
- Add Service Principal ACR access permissions setup
- Add existing service account key creation method
- Clarify Secret names and their usage in workflows

Commit: `a8f23a8` | **Author**: SATOSHI KAWADA | **Files Changed**: 4 | **+716/-0**

Details:

- Add AWS deployment guide with architecture details
- Add system architecture documentation
- Add contributing guidelines
- Add MIT license
- Complete project documentation

Commit: `6ce22f0` | **Author**: SATOSHI KAWADA | **Files Changed**: 30 | **+1380/-0**

Details:

- Add full-stack sample application (React + FastAPI)
- Add infrastructure as code for AWS (Terraform)
- Add GitHub Actions workflows for automated deployment
- Add deployment scripts for AWS/Azure/GCP
- Add comprehensive documentation and setup guide
- Configure Docker Compose for local development

Note: This detailed changelog includes complete commit information with file changes. For a summary version, see CHANGELOG.md.