クラウドネイティブアーキテクチャによるWebサービス基盤刷新のご提案

株式会社〇〇 | 2026年2月

アジェンダ

エグゼクティブサマリー → 現状と課題 → ソリューション → コストと効果 → 実施上の重要ポイント →

長期戦略

PART 1

エグゼクティブサマリー

AWSサーバーレス構成により、5つの課題すべてを解決し運用負荷ゼロを実現する

> **評価項目No.5**: 本システム実現のための最適なクラウドアーキテクチャの提案ができているか。次の観点を含むこと。 **性能 / 可用性 / コスト / 拡張性 / セキュリティ / 可視化**

表5-1 現行課題・解決策・評価観点の対応

現行の課題	解決策	評価観点
ハードウェア故障による可用性低下	Multi-AZ Fargate + Aurora Serverless v2	**可用性**
月末アクセス集中とピークコスト	Fargate/Aurora自動スケーリング + CloudFront	**性能・コスト・拡張性**
DB拡張性ボトルネック	S3オフロード + Aurora自動拡張 + S3階層化	**拡張性・コスト・性能**
日々のオペレーション負荷	マネージドサービス + Go Distroless + CI/CD	**セキュリティ・コスト**
KPI可視化の欠如	S3ログ → Athena → QuickSightダッシュボード	**可視化**

アプローチマッピング　評価観点とソリューションの参照先:

6つの評価観点すべてにおいて、現行システムから定量的に大幅改善する

~100倍
性能
ピーク時レスポンス安定

99.95%
可用性
自動フェイルオーバー

92%
コスト
従量課金+S3階層化

1TB
拡張性
S3容量無制限

~0 CVE
セキュリティ
Distroless+read-only FS

5 KPI
可視化
リアルタイムBIダッシュボード

アジェンダ

エグゼクティブサマリー → **現状と課題** → ソリューション → コストと効果 → 実施上の重要ポイント →

長期戦略

PART 2

現状分析と課題

現行オンプレミスシステムはハードウェア保守期限を迎え、抜本的な基盤刷新が必要である

1

Web層

Webサーバー（オンプレミス）

2

AP層

アプリケーションサーバー（モノリシック構成）

3

DB層

RDBMS（構造化データ＋バイナリデータを格納）

現行の課題:

- すべてのコンポーネントが単一データセンター内に配置 → **DC障害時にサービス全体が停止**
- バイナリデータ（画像・PDF等）もRDBMSに格納 → Web→AP→DBの全層を経由 → **レスポンス遅延**
- ハードウェア保守期限が到来 → 「同等HW更新」か「クラウド移行」かの判断が必要

> ハードウェアを新調しても同じアーキテクチャ上の問題は解決しない。抜本的な基盤刷新が必要。

現行システムにはインフラ起因の3つの構造的課題がある

1

課題1: 可用性低下

単一DC構成のため、HW障害がサービス停止に直結。大規模災害時のBCP対応も必要。 → **利用者の業務が完全に停止**

2

課題2: ピークアクセス

月間100万アクセスの90%が月末2日間に集中。ピーク/通常比 **約100倍**。 → **IT予算を圧迫**

3

課題3: DB拡張性

現状100GB → 5年後1TB。データの **90%がバイナリ**。RDBMSへの格納は非効率。 → **サービス継続困難のリスク**

ピーク時: 90万アクセス/8時間 ≒ 31 req/sec、通常時: 約0.3 req/sec。30日超のデータは参照頻度が極めて低いが、法定保存期限の5年間保管が必要。

運用・可視化にも構造的課題があり、クラウドネイティブへの刷新が必要である

4

課題4: オペレーション負荷

定常運用作業に占有され、**改善・開発に充てる余力がない**

5

課題5: KPI可視化の欠如

データに基づく意思決定ができず、**障害長期化リスク**

表5-2 課題一覧と評価観点の対応

#	課題	評価観点
1	可用性低下（HW故障）	可用性
2	アクセス集中・ピークコスト	性能・コスト・拡張性
3	DB拡張性ボトルネック	拡張性・コスト・性能

アジェンダ

エグゼクティブサマリー → 現状と課題 → **ソリューション** → コストと効果 → 実施上の重要ポイント →

長期戦略

PART 3

ソリューション提案

5つの構造的課題を、「サーバーレス・マネージド・自動化」の3方針で解消する

前パートで確認した5課題に対し、**サーバーレス・マネージド・自動化**を組み合わせ、6つの評価観点すべてに回答する。

現行の5課題

1. 可用性低下（HW故障・単一DC）
2. ピークアクセス・固定コスト
3. DB拡張性ボトルネック
4. オペレーション負荷
5. KPI可視化の欠如

→

3方針

① **サーバーレス**: Fargate/Aurora自動スケール
② **マネージド**: WAF/GuardDuty/Cognito
③ **自動化**: CI/CD・ライフサイクル・監視

表5-0 課題・アプローチ・評価観点の対応

#	課題	本提案のアプローチ	評価観点	詳細
1	可用性低下（HW故障・単一DC）	Multi-AZ + 自動フェイルオーバー	**可用性**	p.25
2	ピークアクセス・固定コスト	CloudFront + Fargate/Aurora 自動スケール	**性能・コスト・拡張性**	p.16–17
3	DB拡張性ボトルネック	S3オフロード + ライフサイクル階層化	**拡張性・コスト**	p.18–19

評価観点ごとに最適なAWSサービスを組み合わせ、6つの課題すべてを解決する

全体像 › 性能 › 拡張性・コスト › セキュリティ › 可用性・認証 › 可視化

1

全体像

アーキテクチャ設計方針と全体構成図

2

性能

CloudFront CDN + Fargate/Aurora 自動スケーリング

3

拡張性・コスト

S3オフロード + ライフサイクル管理

4

セキュリティ

Distroless + WAF多層防御 + 運用自動化

5

可用性・認証

Multi-AZ + Cognito 認証基盤

6

可視化

QuickSight リアルタイムBIダッシュボード

1. アーキテクチャ全体像

AWSサーバーレス・マネージドサービスを中心としたクラウドネイティブアーキテクチャで全課題を解決する

> [ここにアーキテクチャ全体図を挿入]

図に含めるべき要素:

- **メインフロー**: ユーザー → CloudFront（CDN + WAF） → ALB（JWT検証） → ECS on Fargate → Aurora Serverless v2
- **認証フロー**: ユーザー ↔ Amazon Cognito（MFA対応）
- **バイナリデータフロー**: ユーザー ↔ S3（Pre-signed URL、アプリ層をバイパス）
- **ダッシュボードフロー**: S3ログ → Athena → QuickSight
- **CI/CDフロー**: CodeCommit → CodeBuild → ECR → CodePipeline → CodeDeploy（Blue/Green）
- **セキュリティ層**: WAF（エッジ）、GuardDuty、CloudTrail
- **可用性**: 全コンポーネントMulti-AZ配置（2〜3 AZ）

2. CLOUDFRONT + FARGATE 自動スケーリング

CloudFront + Fargate自動スケーリングにより、月末ピーク時でもレスポンス劣化なく対応する

現行（固定構成）

- ピーク性能に合わせたHWを常時稼働
- 月の大半はリソースが過剰 → **コスト浪費**
- それでもピーク時はリソース不足のリスク

→

従量課金で最適化

提案（自動スケール）

- **CloudFront**: 静的コンテンツをエッジキャッシュ
- **Fargate**: 2タスク ↔ 20タスク自動スケール
- **従量課金**: タスク稼働時間のみ課金

ピーク時: 31 req/sec → Fargateタスク自動増加で対応。通常時: 0.3 req/sec → 最小構成で運用。ピーク/通常比 **約100倍** の差を従量課金で吸収。

3. AURORA SERVERLESS V2

Aurora Serverless v2により、データベースの容量・性能がワークロードに応じて自動調整される

現行DB

- 固定スペックのDBサーバー
- 容量拡張は手動（HW増設 or リプレース）
- **単一障害点**: DB故障時はサービス全体が停止

→

自動スケーリング

Aurora Serverless v2

- **ACU自動スケーリング**: 0.5〜128 ACU、ワークロードに応じて自動調整
- **ストレージ自動拡張**: 使用量に応じて自動拡張（最大128TB）
- **Multi-AZフェイルオーバー**: 通常30秒以内で自動切替

バイナリデータ（データ全体の90%）をS3にオフロードすることで、Aurora Serverless v2にはメタデータのみを格納。5年後もDBサイズは約100GBに抑制。

4. S3 PRE-SIGNED URL によるオフロード

S3 Pre-signed URLによりバイナリデータをオフロードし、アプリケーション負荷とDBボトルネックを同時に解消する

現行フロー

- Client → Web → AP → **DB（バイナリ格納）**
- 全データがアプリケーション層を経由
- DBサイズ肥大化（100GB、うち90GBがバイナリ）

→

転送量90%削減

提案フロー

- メタデータ: Client → AP → Aurora（**10%**）
- バイナリ: Client ↔ **S3直接（90%）**
- DBサイズ: 約100GB（メタデータのみ）に抑制

AP・DBをバイナリ転送から完全解放。S3は容量無制限のため将来のデータ増にも対応可能。

5. S3 ライフサイクルポリシー

S3ライフサイクルポリシーにより、5年間のデータ保管コストを最大92%削減する

図5-1 S3ストレージクラス自動移行フロー

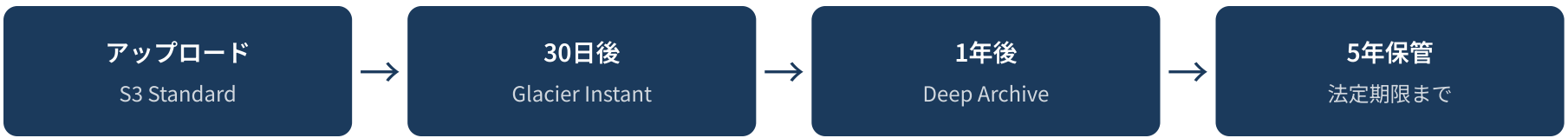


表5-3 S3ストレージクラス別コスト比較

ストレージクラス	コスト（GB/月）	取得時間
S3 Standard	$0.025	ミリ秒
S3 Glacier Instant Retrieval	$0.005	ミリ秒
S3 Glacier Deep Archive	$0.002	12時間以内

S3 Standard → Deep Archive で **約92%のコスト削減**。要件「取り出しに時間がかかっても良い」に完全合致。ライフサイクルポリシーは一度設定すれば自動実行。

6. セキュリティ — 多層防御アプローチ全体像

セキュリティを「コンテナ・多層防御・NoOps」の3アプローチで構造的に強化する

① コンテナ
Go + Distroless 攻撃面を極小化

→

② 多層防御
WAF + GuardDuty エッジ〜監査まで5層

→

③ NoOps
マネージド自動化 セキュリティ運用ゼロ

表5-3a セキュリティアプローチの概要

アプローチ	主要サービス・技術	効果	詳細
① コンテナセキュリティ	Go + Distroless + read-only FS	CVE対象〜0、イメージ1/50に縮小	p.22–23
② エッジ〜監査の多層防御	WAF + GuardDuty + CloudTrail	5層で外側から内側に自動防護	p.24
③ セキュリティ運用自動化	Fargate + ECRスキャン + CI/CD	手動監視・パッチ工数をゼロ化	p.25

> 3アプローチは独立した施策ではなく、**コンテナ単体 → ネットワーク経路 → 運用全体** へと防御範囲が広がる多層構造。

7. GO + DISTROLESS イメージ

Go言語 + Distrolessイメージにより、コンテナの攻撃面を極小化する

表5-4 コンテナイメージ比較（一般的なイメージ vs Go + Distroless）

項目	一般的なイメージ（Ubuntu+ランタイム）	Go + Distroless
イメージサイズ	約500MB	**約10〜20MB**
CVE対象パッケージ数	100以上（OS, ライブラリ, ツール）	**大幅削減（Go標準ライブラリのみ）**
シェルアクセス	あり（攻撃者に利用されるリスク）	**なし（シェル非搭載）**
パッケージマネージャー	あり	**なし**
rootファイルシステム	読み書き可能	**read-only設定**

> OS層やミドルウェアがほぼ存在しないため、CVEの検知対象が大幅に減少し攻撃面を極小化する。

8. DISTROLESS 運用効果

Distroless + read-only FSにより、セキュリティ運用工数を大幅に削減しNoOpsに貢献する

セキュリティ効果:

1
Distroless
OS・シェル非搭載で攻撃面を極小化

2
Read-only FS
マルウェアの永続化を防止。実行時にファイルを書き込めない

3
Multi-Stage Build
ビルドツールが本番イメージに含まれない

運用効果:

4
CVE工数削減
Go自体の更新のみ対応すればよい

5
デプロイ高速化
イメージサイズが小さくCI/CDが高速化

6
NoOps貢献
運用チームをセキュリティ運用から解放

9. WAF・GUARDDUTY 多層防御

WAF・GuardDuty・CloudTrailの多層防御により、エッジからデータ層まで自動的に保護する

表5-5 多層防御サービス一覧（防御層・役割）

防御層	サービス	役割
エッジ層	**CloudFront + AWS WAF（Managed Rules）**	DDoS防御、OWASP Top 10対策、不正リクエスト遮断
認証層	**ALB + Cognito JWT検証**	未認証リクエストをアプリケーション到達前に遮断
コンピュート層	**ECS on Fargate（read-only FS、SSHなし）**	コンテナ内部への侵入・改ざんを防止
監視層	**GuardDuty**	機械学習ベースの脅威検出（不正アクセス、マルウェア通信等）
監査層	**CloudTrail**	全APIコールの記録・監査証跡の保全

> すべてマネージドサービスのため、セキュリティ監視の自動化が実現する。外側から内側に向かう多層防御（オニオンモデル）を構成。

10. セキュリティ運用の自動化

マネージドサービスにより、セキュリティ運用の大半を自動化しNoOpsを実現する

表5-6 セキュリティ運用 Before/After 比較

運用項目	Before（手動）	After（マネージド）
セキュリティ監視	24/365の有人監視	**GuardDutyが自動検出・アラート**
CVE確認	手動で定期確認	**Distrolessで対象大幅削減 + ECRイメージスキャン自動化**
OSパッチ適用	手動適用・再起動	**Fargateはマネージド基盤。OS管理不要**
ログ管理	手動ローテーション・保管	**CloudWatch Logs / S3で自動管理**
バックアップ	手動実施	**Aurora自動バックアップ + S3バージョニング**

運用チームは手動のセキュリティ運用から解放され、改善・新機能開発に注力できる。

11. MULTI-AZ 可用性構成

Multi-AZ構成と自動フェイルオーバーにより、99.95%以上の可用性を確保する

表5-7 コンポーネント別 可用性設計とSLA

コンポーネント	可用性設計	SLA
CloudFront	グローバルエッジ、本質的に高可用	**99.9%**
ALB	Multi-AZ自動分散	**99.99%**
ECS on Fargate	タスクをAZ間に分散配置、異常タスク自動置換	**99.99%**
Aurora Serverless v2	Multi-AZフェイルオーバー（通常30秒以内）	**99.99%**
S3	99.999999999%の耐久性	**99.99%**

遠隔地バックアップ:

- **Aurora**: AWS Backupによるクロスリージョンバックアップ

12. COGNITO + ALB 認証フロー

Cognito + ALBのJWT検証により、不正リクエストをアプリケーション到達前に遮断する

- **有効な場合**: ALBがリクエストをFargateに転送 → 正常処理
- **無効な場合**: ALBが403 Forbiddenを返却 → **アプリケーションに到達しない**

> 認証されていないリクエストはアプリケーション到達前に遮断される。アプリケーション層の負荷軽減とセキュリティ強化を同時に実現。

13. マネージド認証基盤

マネージド認証基盤により、認証の運用負荷ゼロとピーク時のスケーラビリティを両立する

自前認証基盤

- 自前で構築・運用が必要
- 認証サーバーの増強が必要
- MFA実装は追加開発が必要
- アプリ層で認証処理

→

運用ゼロ

Cognito

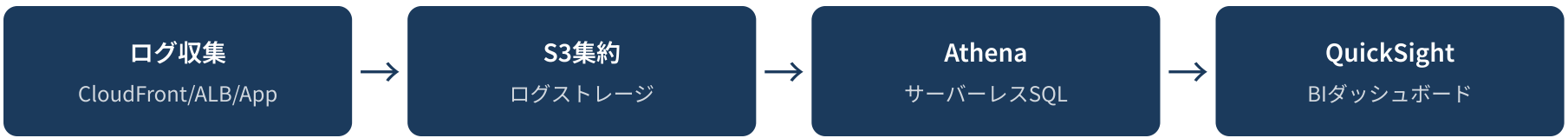
- **フルマネージド、運用ゼロ**
- **自動スケール、ピーク時も対応**
- **MFA対応が標準機能**
- **ALBで遮断、アプリ負荷軽減**

Cognitoはフルマネージドサービスのため、認証インフラの構築・運用が不要。認証基盤自体がスケーラブルなため、月末ピーク時にも認証がボトルネックにならない。

14. QUICKSIGHT ダッシュボード

QuickSightダッシュボードにより、システムKPIをリアルタイムに可視化する

図5-3 QuickSightデータパイプライン



可視化KPI例:

- アクセス数の推移（日次・時間帯別・月末ピーク分析）
- レスポンスタイムの推移
- エラー率の推移
- リソース使用率（Fargate CPU/メモリ、Aurora ACU）
- ストレージ使用量の推移

アジェンダ

エグゼクティブサマリー → 現状と課題 → ソリューション → **コストと効果** → 実施上の重要ポイント →

長期戦略

PART 4

AWSランニングコストと効果

従量課金モデルにより、月額ランニングコストを最適化する

表5-8a コストモデル — サービス別従量課金の構造

サービス	通常時	ピーク時（月末2日間）	備考
ECS on Fargate	最小タスク構成	タスク自動増加分のみ追加	vCPU/メモリ × 稼働秒数
Aurora Serverless v2	最小ACU	ACU自動増加分のみ追加	ACU × 稼働時間
S3	格納量に応じた課金	-	ライフサイクルで階層化済み
CloudFront	転送量に応じた課金	キャッシュ効果あり	エッジキャッシュ

コスト最適化のポイント:

- Fargate/Aurora: 通常時（月の約93%）は最小構成で稼働 → ピーク構成固定に比べ大幅な削減
- S3ライフサイクル: アーカイブデータは最大92%のストレージコスト削減
- マネージドサービス: 人件費（セキュリティ監視、パッチ適用等）のコスト削減効果

6つの評価観点すべてにおいて、提案アーキテクチャにより現行システムから定量的に改善される

表5-8b 評価観点別 Before/After 比較

評価観点	Before（現行オンプレ）	After（AWS提案構成）	改善ポイント
性能	固定容量、ピーク時にリスク	**Fargate/Aurora自動スケール + CloudFront CDN**	ピーク時もレスポンス安定
可用性	単一DC、手動フェイルオーバー	**Multi-AZ自動フェイルオーバー**	SLA 99.95%以上
コスト	ピーク構成の固定費用	**従量課金 + S3階層化**	通常時コスト大幅削減
拡張性	DB 100GB上限、バイナリもDB格納	**S3容量無制限 + Aurora自動拡張**	5年後1TBに対応
セキュリティ	手動監視・手動パッチ	**マネージドWAF + GuardDuty + Distroless**	運用自動化、CVE大幅削減
可視化	KPIダッシュボードなし	**QuickSightリアルタイムダッシュボード**	データ駆動の意思決定

アジェンダ

エグゼクティブサマリー → 現状と課題 → ソリューション → コストと効果 → **実施上の重要ポイント** →

長期戦略

PART 5

実施上の重要ポイント

CI/CDパイプラインとBlue/Greenデプロイにより、ゼロダウンタイムで安全なリリースを実現する

図5-2 CI/CDパイプライン全体フロー

Blue/Greenデプロイ:

- 新バージョンをGreen環境にデプロイ → TestTrafficで検証 → 本番トラフィックをGreenに切替
- 問題発生 → Blue環境に即時ロールバック（**ダウンタイムゼロ**）
- 手動承認ステップにより本番リリースの統制を維持

> 完全自動化されたCI/CDパイプラインで、品質を確保しつつ迅速なリリースサイクルを実現。ロールバックは自動で数分以内に完了。

4つの移行リスクに対策を講じ、残存リスクをすべて「低」以下に抑制する

表5-8 移行リスク・対策・残存リスク一覧

リスク	影響	対策（参照先）	残存リスク
データ移行時のデータ不整合	移行後にデータ欠損・不整合が発生	移行前後のデータ件数・チェックサム照合。段階的移行で影響範囲を限定（p.35参照）	🟢 **極小: 二重検証で欠損ゼロを担保**
切り替え時のサービス停止	ユーザーがサービスを利用できない	DNS加重ルーティングによる段階的移行。Blue/Greenで即時切り戻し可能（p.35参照）	🟢 **極小: ロールバック数分以内**
運用チームの習熟不足	インシデント対応が遅延	AWSトレーニング実施。運用手順書整備。移行後3ヶ月はサポート体制強化（p.25参照）	🟡 **中: 習熟期間3ヶ月は支援継続**
想定外のコスト増	従量課金の見積もりと実績に乖離	Cost Explorerで日次モニタリング。Budgetsアラートで閾値超過を即時検知（p.31参照）	🟢 **低: アラートで即時検知・是正**

4リスク中3つは**極小〜低**に抑制済み。習熟「中リスク」は3ヶ月サポートで対応し、**中長期的にすべて低リスクに収束**する。

アジェンダ

エグゼクティブサマリー → 現状と課題 → ソリューション → コストと効果 → 実施上の重要ポイント →

長期戦略

PART 6

長期戦略

API設計により、外部システム連携やAIエージェント活用など将来の機能拡張に柔軟に対応する

提案アーキテクチャの拡張性

→ **フロント・バック分離**
CloudFront + ALB + Fargate → APIベースの設計

→ **API公開基盤**
外部システム連携が容易 → 将来的なAPI公開

→ **AIエージェント連携**
MCPを通じたシステム利活用 → 新たな活用可能性

AWSマネージドサービスの継続的進化:

- AWSが継続的にサービスを改善・機能追加
- 最新のセキュリティパッチが自動適用
- 新機能の追加はサービス設定の変更のみで対応可能

IaCによりインフラ構成をコード管理し、環境の再現性と変更管理を確保する

手動管理

- 手動で個別構築、手順書依存
- 変更履歴が不明確
- 手動変更が蓄積し構成が不明に
- 手順書に基づく手動復旧

→

コード管理

IaC（Infrastructure as Code）

- **同一コードから開発・ステージング・本番を構築**
- **コードレビュー + CI/CDで統制**
- **ドリフト検知で差分を自動検出・是正**
- **コードから環境を再構築**

CloudFormation / CDKでインフラ構成を定義し、CI/CDパイプラインと連携してインフラ変更も自動テスト・デプロイ。

まとめ

評価項目No.5 — 6つの評価観点すべてに対応し、定量的な大幅改善を実現する

> **評価項目No.5**: 最適なクラウドアーキテクチャの提案。観点: **性能 / 可用性 / コスト / 拡張性 / セキュリティ / 可視化**

表5-9a 評価観点別 要件対応サマリー

評価観点	本提案での回答	詳細
性能	Fargate/Aurora自動スケール + CloudFront CDN	p.16–17
可用性	全層Multi-AZ + 自動フェイルオーバー	p.25–27
コスト	従量課金 + S3ライフサイクル階層化	p.16,19,31
拡張性	S3オフロード + Aurora自動拡張	p.18–19
セキュリティ	WAF + GuardDuty多層防御 + Distroless	p.21–25
可視化	S3ログ → Athena → QuickSightリアルタイムBI	p.28

本提案のアーキテクチャ要素は、他の評価項目とも横断的に連携する

本提案で設計したAWS構成は評価項目No.5単独の回答にとどまらず、他の評価項目の要件にも直接貢献する。

表5-10 他評価項目との関連

関連する評価項目	本提案との関連内容	参照先
移行計画・リスク管理	Blue/Greenデプロイによるゼロダウンタイム移行、DNS段階的切替	p.35–36
運用・保守性	CI/CDパイプライン、IaC管理、マネージドサービスによるNoOps	p.35, 40–41
セキュリティ要件	WAF+GuardDuty多層防御、Distroless、CloudTrail監査証跡	p.21–25
コスト管理	従量課金モデル、S3ライフサイクル、Cost Explorer監視	p.19, 31
将来拡張性	API設計によるフロント/バック分離、AIエージェント連携対応	p.39

各評価項目で**本提案の該当ページを横断参照**することで、提案全体の一貫性と設計の統合性を確認できる。

補足資料（Appendix）

補足A: AWS各サービスのSLA一覧

サービス	SLA（月間稼働率）	補足
Amazon CloudFront	99.9%	グローバルエッジネットワーク
Elastic Load Balancing (ALB)	99.99%	Multi-AZ自動分散
Amazon ECS on Fargate	99.99%	Multi-AZタスク配置
Amazon Aurora	99.99%	Multi-AZフェイルオーバー
Amazon S3	99.99%（可用性）/ 99.999999999%（耐久性）	イレブンナインの耐久性
Amazon Cognito	99.9%	マネージド認証
AWS WAF	CloudFrontまたはALBのSLAに準ずる	-
Amazon GuardDuty	リージョン内の高可用性	マネージド脅威検出

補足B: S3ストレージクラス別コスト詳細

ストレージクラス	保管コスト（GB/月）	取得コスト	最低保管期間	取得時間
S3 Standard	$0.025	無料	なし	ミリ秒
S3 Standard-IA	$0.0138	$0.01/GB	30日	ミリ秒
S3 Glacier Instant Retrieval	$0.005	$0.03/GB	90日	ミリ秒
S3 Glacier Flexible Retrieval	$0.0045	$0.01/GB（標準）	90日	数分〜数時間
S3 Glacier Deep Archive	$0.002	$0.02/GB（標準）	180日	12時間以内

料金は東京リージョン（ap-northeast-1）の参考値。最新の料金はAWS公式サイトを参照。

補足C: AWSランニングコスト試算の前提条件と全サービス内訳

前提条件:

- リージョン: 東京（ap-northeast-1）
- 月間アクセス数: 100万リクエスト
- データ量: 初年度100GB、5年後1TB
- バイナリデータ比率: 90%
- ピーク時トラフィック: 月末2営業日の営業時間（8時間）に月間アクセスの90%が集中

サービス	サイジング前提	概算月額
ECS on Fargate	通常時: 0.5vCPU/1GBメモリ × 2タスク、ピーク時: × 10〜20タスク	[要見積]
Aurora Serverless v2	通常時: 0.5 ACU、ピーク時: 4〜8 ACU	[要見積]
S3	初年度: 100GB（Standard + ライフサイクル移行）	[要見積]
CloudFront	月間データ転送量に依存	[要見積]
ALB	固定費 + LCU課金	[要見積]

補足D: セキュリティ対策の詳細マッピング

脅威カテゴリ	対策	AWSサービス
DDoS攻撃	エッジでの遮断、レート制限	CloudFront + AWS Shield Standard
SQLインジェクション / XSS	WAFルールによる検知・遮断	AWS WAF（Managed Rules）
不正アクセス	JWT認証、MFA	Cognito + ALB
コンテナへの侵入	read-only FS、シェルなし、SSHなし	Fargate + Distroless
内部脅威・異常行動	機械学習ベースの異常検知	GuardDuty
監査証跡	全APIコールの記録	CloudTrail
データ暗号化	保管時・転送時の暗号化	S3 SSE、Aurora暗号化、ACM(TLS)
脆弱性管理	コンテナイメージスキャン	ECRイメージスキャン

WAF SOCオプション: AWS Managed Rulesで主要な攻撃パターンに対応。より高度なWAF運用（カスタムルール管理、24/365監視、インシデント対応）が必要な場合、WafCharm等のSOCサービスの導入を検討。

補足E: Go + Distrolessのベンチマーク比較

イメージサイズ・セキュリティ比較:

ベースイメージ	サイズ	CVE検知数目安	備考
Ubuntu 22.04 + Go	約800MB	50〜200件	フルOS + ツール群
Alpine + Go	約300MB	10〜30件	軽量だがmusl libc脆弱性あり
golang:alpine (multi-stage)	約15MB	10〜30件	ビルドステージ分離
gcr.io/distroless/static	**約10MB**	**極少**	**OS/シェルなし、最小構成**

Multi-Stage Build: Builder(golang:alpine)でgo build → Production(distroless/static)に単一バイナリのみコピー

> Distrolessにより**イメージサイズ1/80、CVE検知ほぼゼロ**を同時に実現。シェル非搭載でコンテナ侵入後の攻撃も防止。