■AWS Well-Architected Summary
Project: take1-project（AWS WordPress/WooCommerce Infrastructure）
Author: Kita
Date: 2025-11

■ Operational Excellence（運用上の優秀性）

観点	実装例	補足
IaC 管理	Terraform（モジュール構成 ＋ remote_state S3+DynamoDB）	バージョン固定・Lint/Tfsec 対応設計
CI/CD	GitHub Actions + OIDC + ECS Deploy	Secret 不要、環境分離（dev/stage/prod）
ロギング	CloudWatch Logs + ECS Exec	コンテナ単位ログ＋運用トラブル時の即時対応
設計可視化	draw.io + architecture PNG	フロー・依存関係をチーム共有可能に

■ Security（セキュリティ）

観点	実装例	補足
IAM 最小権限	OIDC Role / TaskRole / ExecutionRole 分離	least privilege 実践
通信経路	CloudFront – ALB – ECS 間全 HTTPS	ACM + ALB TLS 1.2/1.3
データ保護	SSM Parameter Store（SecureString）	DB 認証情報暗号化管理
WAF 統合	AWS Managed Rules ＋ IPSet ＋ Rate-based Rule	攻撃防御・ログ → Athena 解析基盤

■ Reliability（信頼性）

観点	実装例	補足
可用性	ECS Fargate Multi-AZ + ALB HealthCheck	タスク障害時の自動復旧
スケーリング	TargetTrackingPolicy (CPU 70%)	トラフィック増加時の自動スケール
RDS 層	Aurora MySQL (Multi-AZ) + RDS Proxy（設計中）	接続安定化・再デプロイ時の接続保持
Redis 層	ElastiCache for Redis	WooCommerce キャッシュ／セッション安定化

■ Performance Efficiency（パフォーマンス効率）

観点	実装例	補足
アプリ性能	nginx + php-fpm（Unix ソケット接続）	TCP より低レイテンシ／軽量通信
キャッシュ	CloudFront / Redis	静的・動的両方で高速化

観点	実装例	補足
ECS 設定	512MB/0.25vCPU × オートスケーリング	負荷に応じて動的最適化
EFS 最適化	wp-content のみ永続化	不要な I/O 削減、コスト最適化

■ Cost Optimization（コスト最適化）

観点	実装例	補足
NAT 構成	NAT Instance（検証中）	NAT Gateway より低コスト運用
Fargate	コンテナ単位課金	無駄な EC2 維持コスト削減
ストレージ	S3 Logging / S3 Media 分離	バケット用途別にライフサイクル管理
開発コスト	Terraform モジュール再利用	環境追加・修正の工数削減

■ Sustainability（持続可能性）

観点	実装例	補足
インフラ効率	Fargate + AutoScaling	必要リソースのみ稼働（無駄ゼロ）
ログ保存	S3 Lifecycle（90 日→Glacier）	ストレージ削減
運用自動化	CI/CD + Terraform	手動操作を最小限化
構成の可搬性	再利用可能モジュール設計	他案件展開が容易

■ 今後の展開

RDS Proxy 導入による接続安定化検証
Redis キャッシュ TTL 設計チューニング
Stripe Webhook Terraform 化
CloudWatch Alarm → SNS 通知連携
ECS AutoScaling Module の共通化
WAF ログ解析を Athena ダッシュボード化

■総括

本構成は、AWS Well-Architected Framework の 6 柱を満たすことを目的に、学習環境を越えた本番運用相当の設計基盤 として構築。
Terraform モジュール化と CI/CD 連携により、再現性・自動化・セキュリティ・コスト最適化のバランスを実現。