



2-P2-1-15

AWSコンサル事例でわかる

クラウド時代の移行プロジェクトの進め方

アマゾン ウェブ サービス ジャパン株式会社
プロフェッショナルサービス
鈴木 直

ネスレ日本株式会社
Eコマース本部 EC&デジタルシステム部
日比 裕介

はじめに

本セッション の概要

本セッション のゴール

特に聞いて いただきたい 方々

- クラウドの特徴を活かしたクラウド移行プロジェクトの進め方について、基本的な考え方、および事例をご紹介します
- ご紹介する移行プロジェクトの進め方についてご理解いただき、皆さまの日々の活動に活かしていただくこと
- これからクラウド移行を検討しようとしている方
- クラウド移行プロジェクトを進めているが、オンプレでのプロジェクトと同じように進めており、クラウドの効果に対して疑問をお持ちの方

本日の構成

1. 基本

構成変更の容易性やインフラのコード化による自動化などといった

クラウドの特徴を活かした、移行プロジェクトの進め方を紹介



アマゾン ウェブ サービス ジャパン株式会社 プロフェッショナルサービス
鈴木 直

2. 実践

エンタープライズ企業において、短期間で成功裏に実現したクラウド移行プロジェクトの事例を紹介



ネスレ日本株式会社 Eコマース本部 EC&デジタルシステム部
日比 裕介

1. 基本

クラウドの特徴を生かした 移行プロジェクトの進め方

自己紹介

鈴木 直（すずき ただし）

アマゾン ウェブ サービス ジャパン 株式会社
プロフェッショナルサービス
プラクティス マネージャ

エンタープライズのお客様を中心にクラウド導入を伴う、ITトランスフォーメーションをご支援しています

- ・クラウド導入の計画作成支援
- ・AWS標準化を通じたガバナンス強化支援
- ・AWSプロジェクトのマネジメント支援・技術支援



制約とプロジェクト

プロジェクトの進め方

ソフト面の制約

リソース拡張や
変更の制約

時限利用や
簡易除却の困難

生産性の限界

働き方の制約

システムリソース



マニュアルオペレーションを前提とした人的リソース



建物やスペース



ハード面の制約

制約とプロジェクト

従来と全く同じやり方をしていますか？

ソフト面の制約

リソース拡張や
変更の制約

すぐ利用できる
すぐ試せる

自動化が容易

き方の制約

クラウドの
アジリティ
フレキシビリティ
スケーラビリティ

システムリソース

拡張や変更
も容易

建物やスペース

どこからでも
利用可能

ハード面の制約

クラウドの特性とプロジェクトの進め方

1

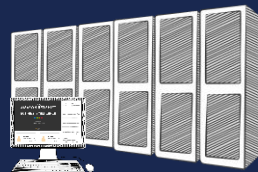
作りながら要件と設計を“固める”



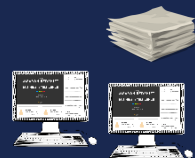
要件定義



設計



構築



テスト

2

インフラのコード化による変更容易性の実現

3

必要なときに
リソースを拡張する

4

テストの範囲を
見定める



プロジェクト管理

5

クラウドとツールで
物理的制約から解放する

1 作りながら要件と設計を“固める”



要件定義



設計



構築・テスト

机上でしっかりと要件定義・設計

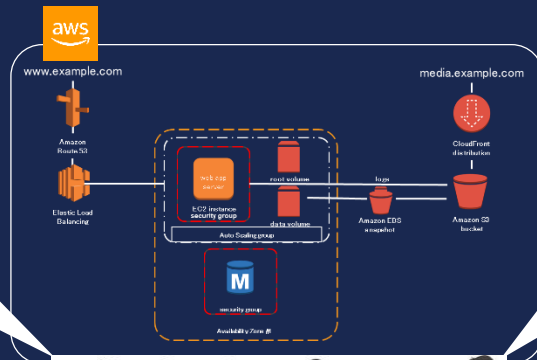
様々な制約



クラウド・AWSの特徴、活かしていますか？

1 作りながら要件と設計を“固める”

ステークホルダーと実機で確認しながら進めていくことによって真に必要な要件が定義される



サイジングやキャパシティ設計では、推測せずに計測する。
構築フェーズでクラウドの拡張性を利用する

時間がかかるんじゃないの・・・？



要件定義の詰めや机上の設計に時間をかけ過ぎない。網羅的にやればやるほど時間とコストが必要。
AWSサービスの選択肢を念頭において進める

ベストプラクティスから始める。検討の時間を短縮する

1 作りながら要件と設計を“固める”

カテゴリ	機能	サービス選択例
メール	メール送信	Amazon SES/SNS
	メール受信	3rd party tool
ネットワーク	DNS	Amazon Route53
	NTP	Amazon Time Sync Service
セキュリティ	認証	AWS SSO/Amazon Cognito/3rd party tool
	ID管理	AWS Directory Service/3rd party tool
	Firewall/IPS/AntiVirus	3rd party tool
	WAF	AWS WAF
セッション	セッション管理	Amazon ElastiCache/DynamoDB
負荷分散	アクセス負荷分散	Amazon ELB
	スケールアウト／イン	Amazon AutoScaling
データ連携	API化	Amazon API Gateway
	同期／準同期データ連携	AWS DMS/3rd party tool
	バッチファイル転送	Amazon Data Pipeline/3rd party tool

カテゴリ	機能	サービス選択例
データ保管	RDB	Amazon RDS
	NoSQL	Amazon DynamoDB
	ファイル保管	Amazon S3
ファイル管理	ファイル共有	Amazon S3/EFS
	コンテンツ管理	Amazon S3
デプロイ	デプロイ管理	AWS CloudFormation/CodeDeploy
キャッシュ	CDN	Amazon CloudFront
ログ	ログ管理	Amazon CloudWatch Logs
分析	データ加工	Amazon EMR/Athena/AWS Glue
	DWH	Amazon Redshift
	BI	Amazon QuickSight/3rd party tool
インフラ運用	障害／エラー監視	Amazon CloudWatch
	性能監視	Amazon CloudWatch /3rd party tool
	バックアップ	Amazon EBS Snapshot/3rd party tool
	ジョブ管理	AWS Systems Manager/AWS Step Functions/3rd party tool

1 作りながら要件と設計を“固める”

AWS Well-Architected

アーキテクチャに関するベストプラクティスを使用して、学習、測定、構築を行う

AWS アーキテクチャセン
ター

This is My Architecture

AWS Answers

AWS ソリューション

導入事例

クラウドセキュリティ

AWS Well-Architected

Well-Architected フレームワークは、クラウドアーキテクトがアプリケーション向けに実装可能な、最も安全かつ高パフォーマンス、障害耐性を備え、効率的なインフラストラクチャを構築することをサポートする目的で開発されました。このフレームワークでは、お客様とパートナーがアーキテクチャを評価するために一貫したアプローチを行い、アプリケーションのニーズに応じて時間の経過とともにスケールする設計を実装するのに役立つガイダンスを提供します。



より迅速な構築とデプロイ

クラウドネイティブのアーキテクチャを構築することで、キャパシティのニーズの推測をやめ、システムを大規模にテストし、オートメーションを使用して実験を容易にします。



リスクの低減と緩和

設計したアーキテクチャのリスクがある箇所を把握し、アプリケーションを本番環境に導入する前にそれらのリスクに対応します。



情報に基づいた決定

アーキテクチャの決定やトレードオフが、アプリケーションのパフォーマンスや可用性、またビジネスの成果に与える影響について判断します。



AWS のベストプラクティスについて学ぶ

AWS の何千件ものお客様のアーキテクチャを見て学んできたことを基盤としたガイダンスを含むトレーニングやホワイトペーパーが利用可能です。

ベストプラクティスから始める。検討の

ベストプラクティス site:docs.aws.amazon.com



すべて

ニュース

画像

ショッピング

地図

もっと見る

設定

ツール

約 7,310 件 (0.51 秒)

Amazon Redshift のベストプラクティス - Amazon Redshift

https://docs.aws.amazon.com/ja_jp/redshift/latest/dg/best-practices.html

この章では、最も重要な開発の原則の概要と、これらの原則を実装するために役立つ具体的なヒント、例、およびベストプラクティスを示します。1 つのベストプラクティスをあきらめる状況に適用できるわけではありません。データベース設計を確定する前に、あらゆる ...

モニタリングのベストプラクティス - Amazon Elastic Compute Cloud

https://docs.aws.amazon.com/ja_jp/AWSEC2/latest/monitoring_best_practices.html

Amazon EC2 モニタリングタスクに役立つ、モニタリングのベストプラクティスを説明します。

AWS CloudFormation のベストプラクティス - AWS CloudFormation

https://docs.aws.amazon.com/ja_jp/AWSCloudFormation/latest/best-practices.html

より効率的かつ安全に AWS CloudFormation を使用するために、ベストプラクティスを使用します。

Amazon EC2 のベストプラクティス - Amazon Elastic Compute Cloud

https://docs.aws.amazon.com/ja_jp/AWSEC2/latest/ec2-best-practices.html

Amazon EC2 のベストプラクティス。このチェックリストは、Amazon EC2 のパフォーマンスと満足度を最大限に引き出すようにするためのものです。セキュリティとネットワーク ID フレームワーク、IAM ユーザー、IAM ロールを使用して、AWS リソースおよび API への ...

18/05/14 にこのページにアクセスしました。

AWS OpsWorks スタックのベストプラクティス - AWS OpsWorks

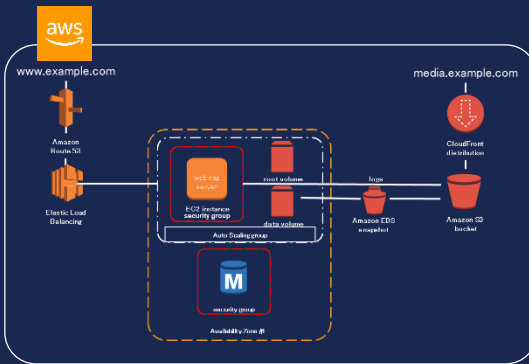
https://docs.aws.amazon.com/ja_jp/opsworks/latest/userguide/best-practices.html

AWS OpsWorks スタックを最大限に活用するために、以下のベストプラクティスに従ってください。

IAM のベストプラクティスとユースケース - AWS Identity and Access ...

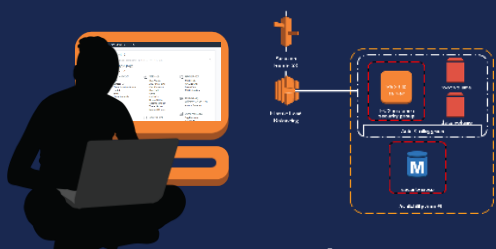
https://docs.aws.amazon.com/ja_jp/IAM/latest/IAMBestPracticesAndUseCases.html

2 インフラのコード化による変更容易性の実現



「作りながら要件定義や設計を固めていく」というが、あとで変更が発生するのでは・・・？

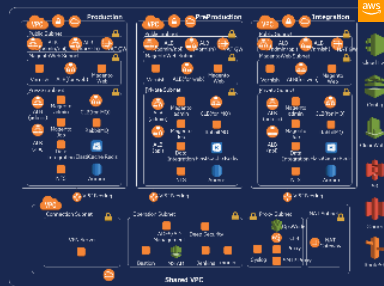
2 インフラのコード化による変更容易性の実現



Management Console
やAWS CLIからの手作業

```
1 Description: sample cfn template with YAML - Aurora
2
3 Parameters:
4   BaseStackName:
5     Description: VPC Stack Name
6     Type: String
7     MinLength: 1
8     MaxLength: 255
9     AllowedPatterns: [a-zA-Z]
10    Default: yaml-stack2
11
12   SecStackName:
13     Description: Security Stack Name
14     Type: String
15     MinLength: 1
16     MaxLength: 255
17     AllowedPatterns: [a-zA-Z]
18     Default: yaml-stack2-sec
19
20   DBUser:
21     Description: Database Master User
22     Type: String
23     MinLength: 8
24
25 Resources:
26   AuroraCluster:
27     Type: AWS::RDS::DBCluster
28     DependsOn: AuroraMonitorRole
29     DeletionPolicy: Snapshot
30     Properties:
31       DatabaseName: AuroraTest
32       Engine: aurora
33       MasterUsername: !Ref DBUser
34       MasterUserPassword: !Ref DBPassword
35       DBSubnetGroupName: !Ref AuroraSubnetGrp
36       VpcSecurityGroupIds:
37         - Fn::ImportValue: !Sub ${SecStackName}-SecGrpDB
38
39   AuroraInstanceReplica1:
40     Type: AWS::RDS::DBInstance
41     Properties:
42       DBName: AuroraTest-2
43       Engine: aurora
44       DBInstanceClass: db.t2.small
45       DBClusterIdentifier: !Ref AuroraCluster
46       MonitoringInterval: 10
47       MonitoringRoleArn: !GetAtt AuroraMonitorRole.Arn
48       AvailabilityZone: !Select [ 1, !GetAZs '' ]
```

規模の変化



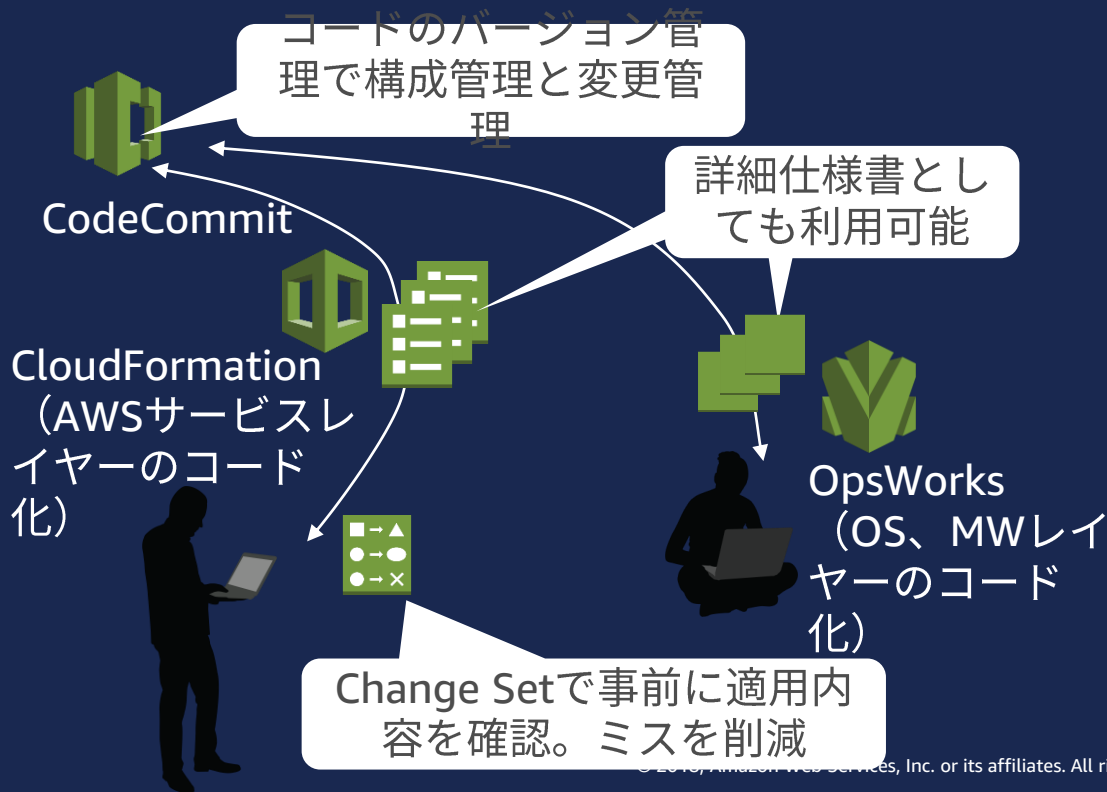
変更作業の時間増加、ミスの増加

インフラのコード化で大量構築&繰り返し構築の簡易化だけでなく、
変更作業の簡易化、迅速化、確実化を実現

```
40 DeletionPolicy: Snapshot
41 Properties:
42   DatabaseName: AuroraTest
43   Engine: aurora
44   MasterUsername: !Ref DBUser
45   MasterUserPassword: !Ref DBPassword
46   DBSubnetGroupName: !Ref AuroraSubnetGrp
47   VpcSecurityGroupIds:
48     - Fn::ImportValue: !Sub ${SecStackName}-SecGrpDB
49
50 AuroraInstanceReplica1:
51   Type: AWS::RDS::DBInstance
52   Properties:
53     DBName: AuroraTest-2
54     Engine: aurora
55     DBInstanceClass: db.t2.small
56     DBClusterIdentifier: !Ref AuroraCluster
57     MonitoringInterval: 10
58     MonitoringRoleArn: !GetAtt AuroraMonitorRole.Arn
59     AvailabilityZone: !Select [ 1, !GetAZs '' ]
```

2 インフラのコード化による変更容易性の実現

AWSでは、インフラのコード化環境自体も安価に即座に用意可能



後述事例より

- 手作業では1週間はかかる大きな環境変更作業も2時間で実現
- ミスをおかしやすい複雑な設定変更も容易に確実に実施

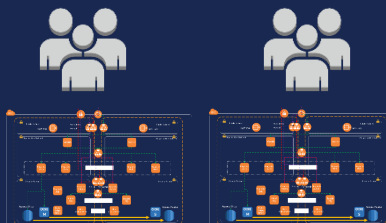
3 必要なときにリソースを拡張する

ニーズに応じてリソースを拡張・縮小させる効果は、プロジェクト期間中でも同じ。

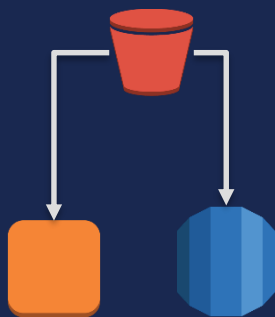
拡張させることで作業時間が短縮できるのであれば、一時的にリソースのスケールアップ、スケールアウトを検討する。

ただし、コストの観点から、通常時は機能テストが実施できる最小限のリソースで起動しておく。

機能追加に基づく
変更やテストのため
の環境複製



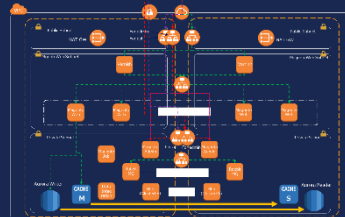
データの移行や
大量コピー



時間のかかる
Build作業や
データ変換作業



負荷テスト



4 テストの範囲を見定める

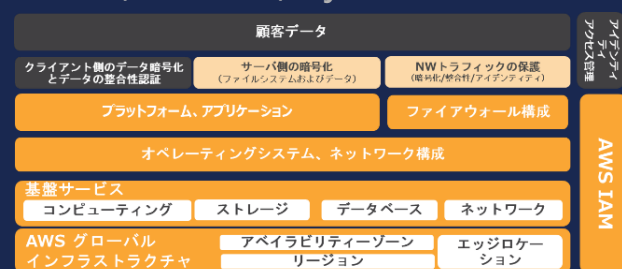
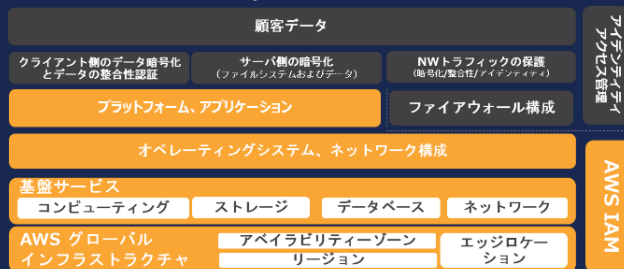
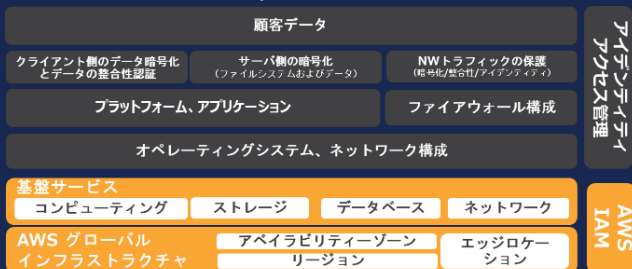
AWSのプラットフォームをテストしない

- EC2の起動/停止テストやAuto Scalingの機能テストなど、AWSが提供しているAPIの正常性を確認するようなテストは省略する
- 物理ハードウェア障害時に別のハードウェアでインスタンスを再起動するAuto Recovery機能は利用者がテストできない
- テスト工数の削減の観点からもAWSマネージドサービスの活用を検討する

Infrastructure Services (EC2, EBSなど)

Container Services (RDS, ELBなど)

Abstracted Services (S3, Lambda, DynamoDBなど)



凡例

お客様が担当する範囲

AWSが担当する範囲

オプション機能として提供されるもの

4 テストの範囲を見定める

AWSのプラットフォームをテストしない

- EC2の起動/停止テストやAuto Scalingの機能テストなど、AWSが提供しているAPIの正常性を確認するようなテストは省略する
- 物理ハードウェア障害時に別のハードウェアでインスタンスを再起動するAuto Recovery機能は利用者がテストできない
- テスト工数の削減の観点からもAWSマネージドサービスの活用を検討する

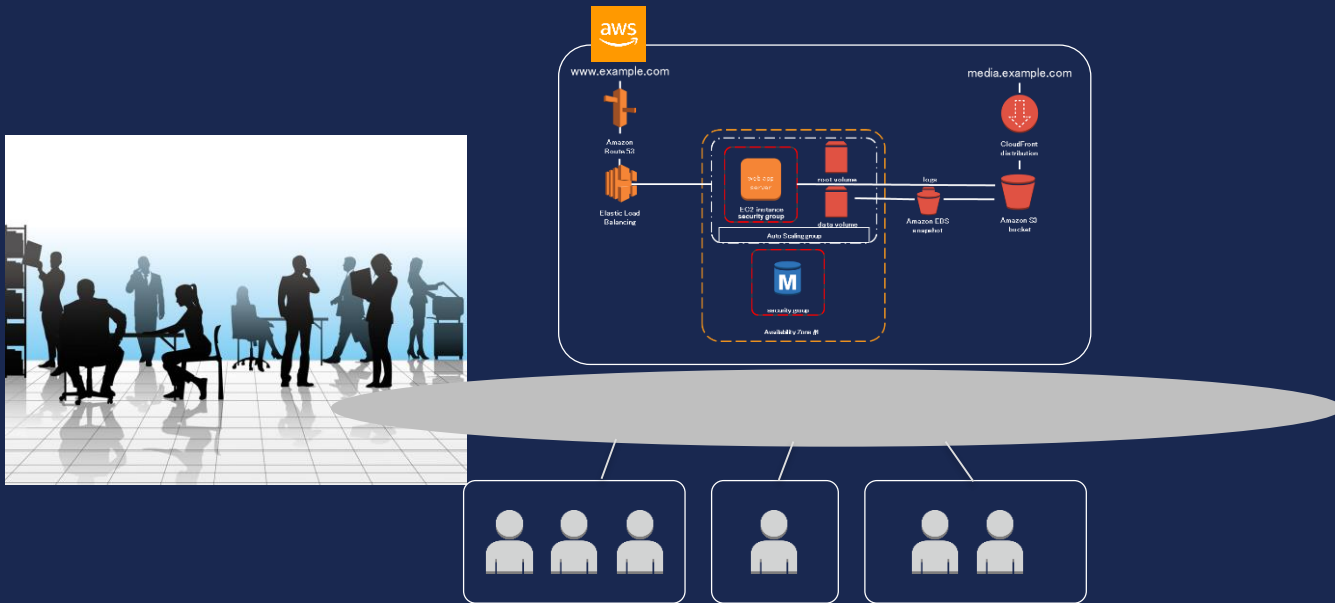
利用者

責任共有モデル

AWS

5 クラウドとツールで物理的制約から解放する

クラウドにより、物理的なスペースから解放され、
どこからでもアクセスが容易に



5 クラウドとツールで物理的制約から解放する

クラウドにより、物理的なスペースから解放され、
どこからでもアクセスが容易に

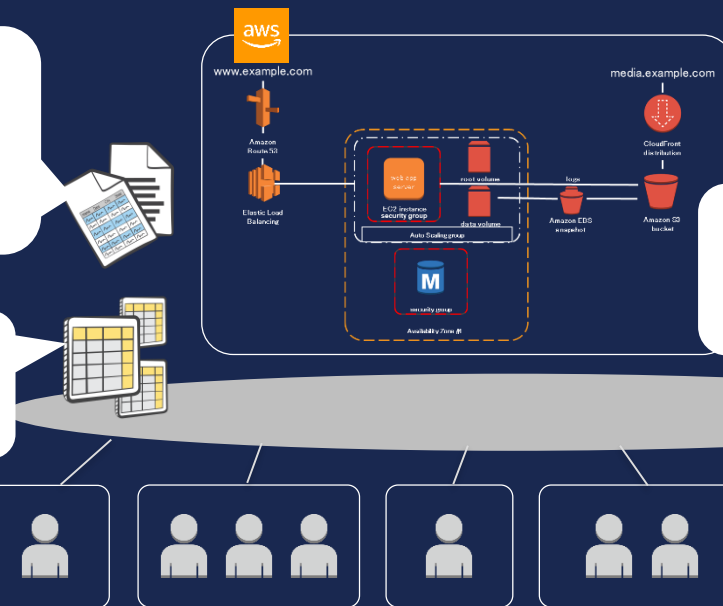


- プロジェクトルームに行かないと見れないドキュメント、紙の資料
- 口頭ベースの指示や報告
- 一か所に集まって仕事をするという暗黙的なスタイルやルール

5 クラウドとツールで物理的制約から解放する

オンラインドキュメント
ツールを使い、設計書や運
用手順書は全員が閲覧、編
集できるようにする

タスク管理もチケットベ
ースでオンライン管理し、可
視化する



コミュニケーションもツ
ール（Amazon Chime等）を
駆使して、密に連携する

クラウドの接続容易性に加え、ツールの活用、
マインドの変革でプロジェクトでの働き方にも変革を

クラウドの特性とプロジェクトの進め方

クラウドの
アジリティ
フレキシビリティ
スケーラビリティ

1

作りながら要件と設計を“固める”

PoCをやる。ただし、既に”Proof”されているものをあらためて検証しない

2

インフラのコード化による変更容易性の実現

セキュリティチームとの合意は早いフェーズで実施する

3

必要なときにリソースを拡張する

アプリチームとインフラチーム一緒に議論する

4

テストの範囲を見定める

新機能リリースの動向を常に確認する

5

クラウドとツールで物理的制約から解放する

オペレーションマニュアルの作成に時間をかけない

クラウドの特性とプロジェクトの進め方

プロジェクトにも アジリティ フレキシビリティ スケーラビリティを

1

作りながら要件と
設計を“固める”

PoCをやる。ただし、
既に“Proof”されているものを
さらためて検証しない

2

インフラのコード化に
よる変更容易性の実現

3

セキュリティチームとの合意は
早いフェーズで実施する

4

テストの範囲を
見定める

アプリチームとインフラチーム
一緒に議論する

5

クラウドとツールで
物理的制約から解放する

新機能リリースの動向を
常に確認する

5

2. 実践 クラウド移行プロジェクトの事例

自己紹介

日比 裕介（ひび ゆうすけ）

ネスレ日本 株式会社
Eコマース本部 EC&デジタルシステム部

ECサイトAWS移行プロジェクト
プロジェクトマネージャー



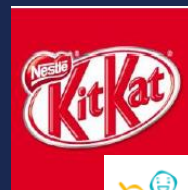
会社紹介

Nestlé Global (HQ: Switzerland)

- 2016年グループ売上高895億スイスフラン
- 社員32万8,000人（150カ国以上）
- 418工場（86カ国）
- 2,000を超えるブランド
- 1日に販売されるネスレ製品10億個

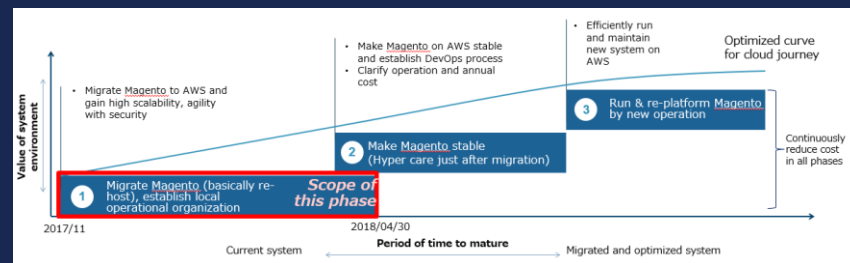
ネスレ日本株式会社（本社：神戸）

- オーガニックグロース：4.1%
(2016年/ネスレグループ：3.2% 先進国：1.7%)
- 社員2,500人
- 3工場
- 飲料、菓子、食品、ペットケアブランド
- 日本で消費されるコーヒーの約4分の1が「ネスカフェ」



プロジェクトの紹介

- ネスレ日本のECサイト
(<https://shop.nestle.jp/>) の一部を香港のRackspaceからAWS東京リージョンに移行
- 2017年11月からプロジェクトを開始し、2018年4月23日にGo Live
- フェーズ1では、EC2、RDS、ElastiCacheをベースに“Re-Host”として、AWS環境に同等の環境を構築して移行
- フェーズ2以降では、よりAWSのマネージドサービスを活用し、DevOpsプロセスの導入・クラウド最適化を図っていく



AWS利用に至った背景

拡張性と
スピード

- 新機能のリリースを迅速に行いたい
- 環境の設定変更を柔軟に行いたい

性能改善

- サイトパフォーマンスを改善したい
- 地理的なLatencyを最小化したい

運用効率化

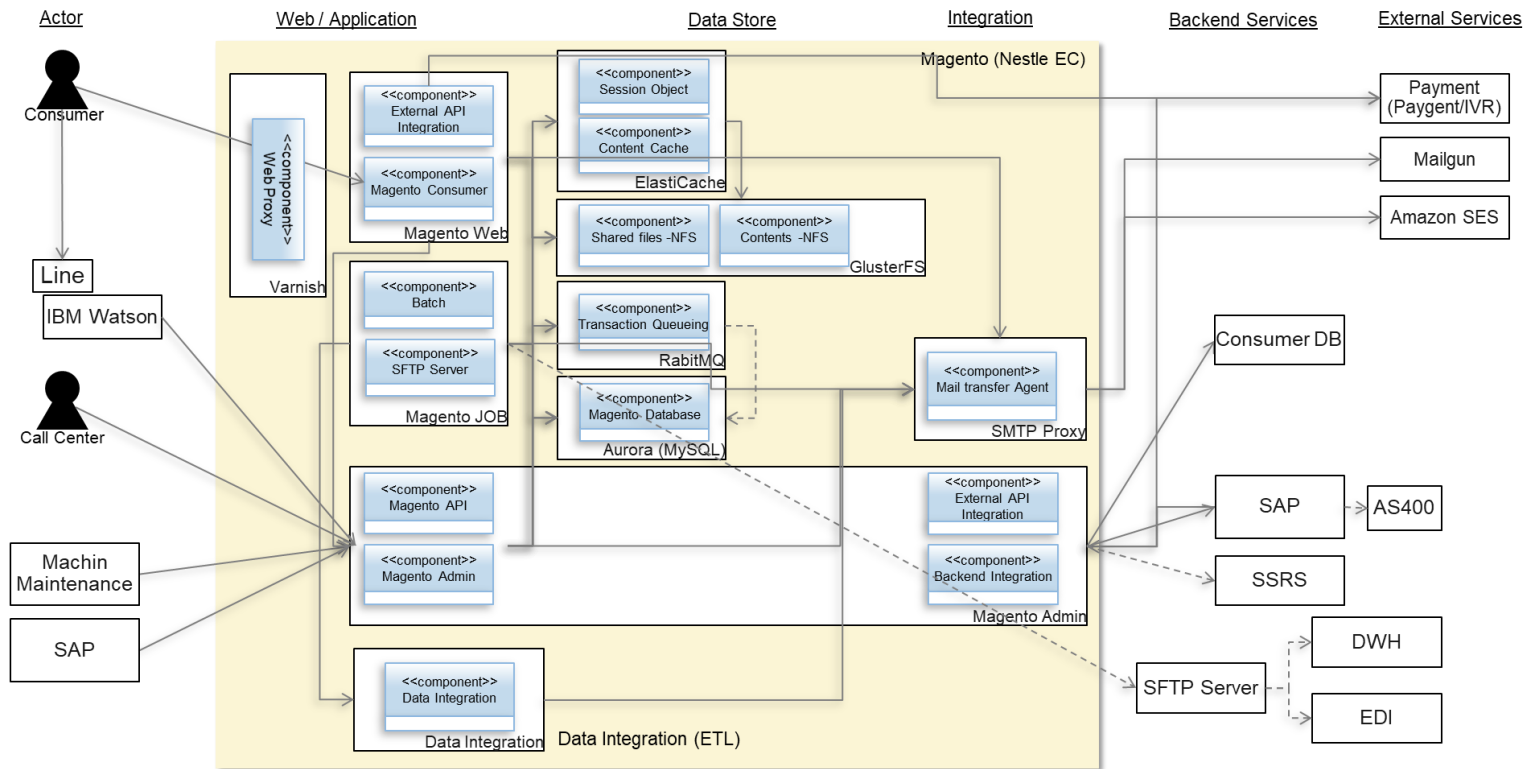
- 運用の自動化を進めていきたい
- 日本ローカルでの運用を行いたい

TCO削減

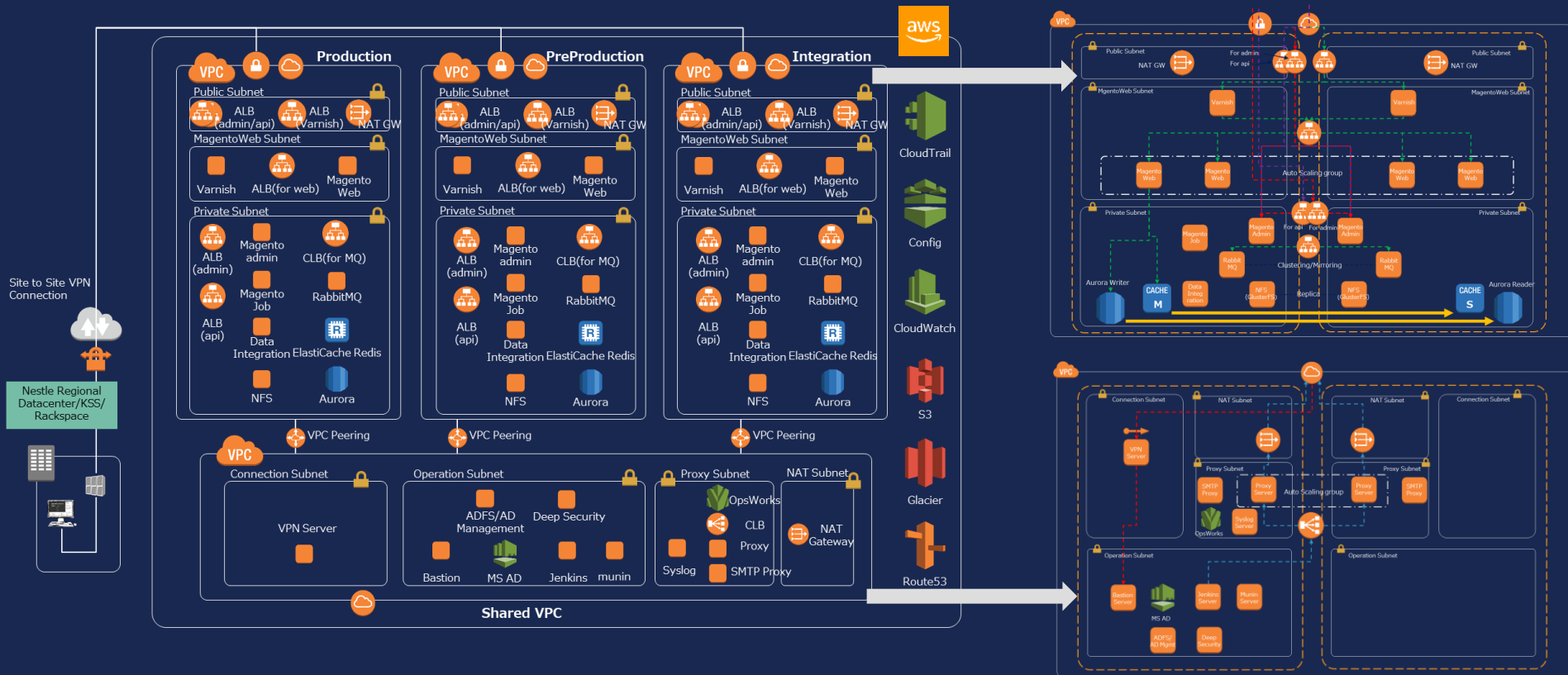
- TCOを削減したい



アーキテクチャ全体像



アーキテクチャ全体像



プロジェクトの特徴と工夫

特徴

工夫

1

変更に強い基盤の
要求

コードベースのインフラストラクチャ構築

2

継続的なシステム
改善

AWSマネージドサービスの機能を利用し
た作業期間の短縮

3

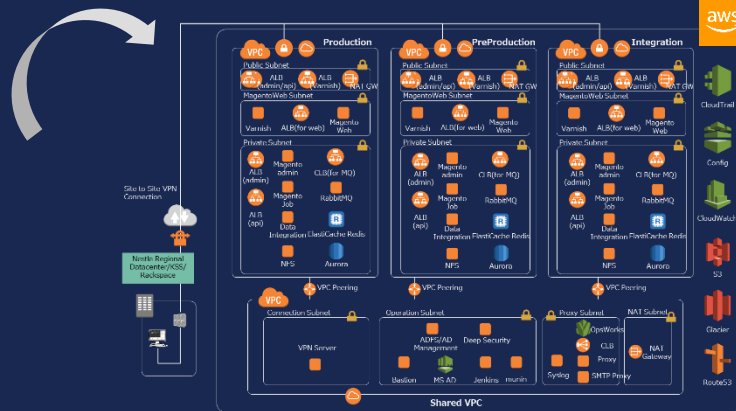
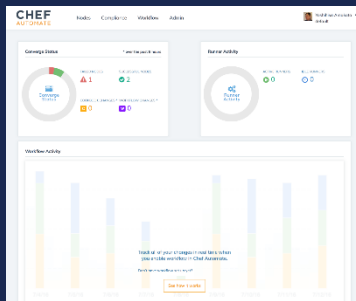
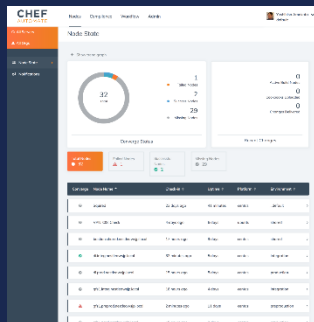
ステークホルダーが
地理的に分散

働く場所を制限しないコミュニケーション

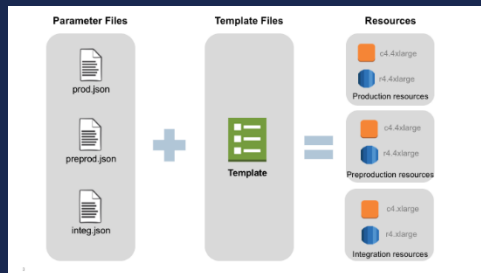
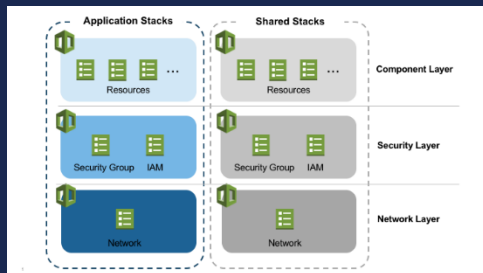
コードベースのインフラストラクチャ構築

環境の構築、複製、変更は原則コードベースですべて実施

OS以上の設定や
ミドルウェアの導
入にはChef
(AWS OpsWorks
for Chef
Automate)



AWS環境は
CloudFormation



インフラのコード
管理はCodeCommit



コードベースのインフラストラクチャ構築

環境の構築、複製、変更は原則コードベースですべて実施

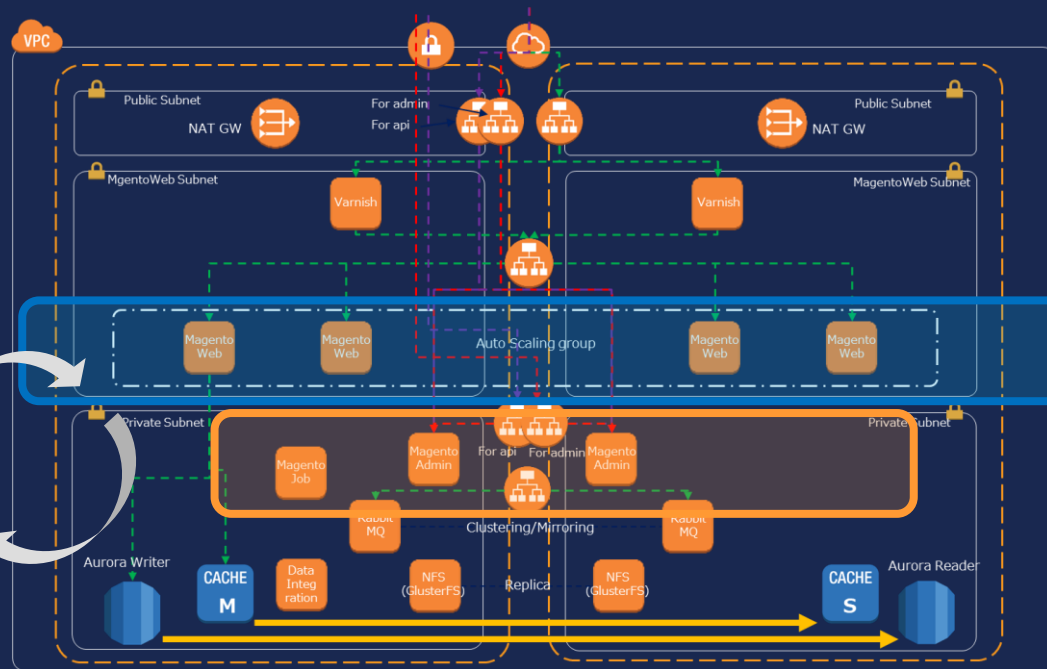
- **環境の変更作業の時間短縮と運用への影響度減少**
プロジェクトの途中で環境毎にAWSアカウントを分割。
CloudFormationを使っていたため、担当者1名で約1時間の作業で完結
- **環境の複製構築が短時間で実現**
同一構成（EC2 20台/ELB/RDS/ElastiCache等）の別環境の用意を約2時間で実現
- **作業ミスも減少**
セキュリティグループのルール数が約250以上。
CloudFormationを使っていたため、変更が容易に行えた

AWSマネージドサービスを活用した作業期間の短縮

Rolling UpdateによりAuto Scaling
グループのEC2を
順次更新



Magento Web



運用管理機能自体も
マネージドサービス
を活用

AWSマネージドサービスを活用した作業期間の短縮

- **Auto Scalingの機能を使用することで機能開発/テスト時間を短縮**
マネージドサービスとして提供される機能を使うことでプロジェクトで個別に機能開発/テストを行う必要なくリリース作業に使用できた
- **サービス停止なしでアプリケーションのコード更新**
AWS移行前はアプリケーションのコード更新毎にサービスを停止していたが、サービス停止なしでコード更新を実現。運用作業の効率化を実現
- **マネージドサービスを活用することで、運用管理自体の構築期間を短縮**
ログの統合やシステム監視機能も開発やテストを省略することができ、かつそれ自体の運用も不要になった

働く場所を制限しないコミュニケーション



働く場所を制限しないコミュニケーション

コミュニケーションには
Chime/Slackを随時使用

プロジェクトマネージャ（神戸）

PMO（東京）



タスク管理は全てJIRAで
実施し状況把握

セキュリティ担当
（オーストラリア）

JIRA



ドキュメントは全て
Confluenceでオンライン管
理し、運用設計（東京）
全員が閲覧・編集可能

Confluence

どこからでもセキュアにアクセス
可能

パッケージアプリケーション
（ウクライナ）

アプリケーション開発（ベトナム）

基盤構築（東京）

働く場所を制限しないコミュニケーション



- ロケーション/ベンダーを跨いだチームの一体感の醸成
- コミュニケーションコストの減少
- 構築作業の短縮への寄与
- 移動時間やスペース賃料などの間接費用も削減

パッケージアプリケーション
(ウクライナ)



アプリケーション開発 (ベトナム)

基盤構築 (東京)

プロジェクトの特徴と工夫

	特徴	工夫	効果
1	変更に強い基盤の要求	コードベースのインフラストラクチャ構築	<ul style="list-style-type: none">環境の変更作業時間短縮と運用への影響度減少環境の複製構築が短時間で実現作業ミスも減少
2	継続的なシステム改善	AWSマネージドサービスの機能を利用した作業期間の短縮	<ul style="list-style-type: none">Auto Scalingの機能を使用することで機能開発/テスト時間を短縮サービス停止なしでコード更新マネージドサービスを活用することで、運用管理自体の構築期間を短縮
3	ステークホルダーが地理的に分散	働く場所を制限しないコミュニケーション	<ul style="list-style-type: none">ロケーション/ベンダーを跨いだチームの一体感の醸成コミュニケーションコストの減少構築作業の短縮への寄与移動時間やスペース賃料などの間接費用も削減

まとめ

制約とプロジェクト

プロジェクトの進め方

ソフト面の制約

リソース拡張や
変更の制約

時限利用や
簡易除却の困難

生産性の限界

働き方の制約

システムリソース



マニュアルオペレーションを前提とした人的リソース



建物やスペース



ハード面の制約

制約とプロジェクト

クラウドの特徴を活かしたプロジェクトの進め方

ソフト面の制約

リソース拡張
変更の制約

すぐ利用できる
すぐ試せる

クラウドの
アジリティ
フレキシビリティ
スケーラビリティ

自動化が容易
スキルの制約

システムリソース

拡張や変更
も容易

建物やスペース

どこからでも
利用可能

ハード面の制約

クラウドの特性とプロジェクトの進め方

クラウドの
アジリティ
フレキシビリティ
スケーラビリティ

1

作りながら要件と設計を“固める”

PoCをやる。ただし、既に”Proof”されているものをあらためて検証しない

2

インフラのコード化による変更容易性の実現

セキュリティチームとの合意は早いフェーズで実施する

3

必要なときにリソースを拡張する

アプリチームとインフラチーム一緒に議論する

4

テストの範囲を見定める

新機能リリースの動向を常に確認する

5

クラウドとツールで物理的制約から解放する

オペレーションマニュアルの作成に時間をかけない

クラウドの特性とプロジェクトの進め方

扱っているものの特徴が変わった
既存の進め方がそのままいいのか疑うこと

Thank You !