

Amazon ECS と Spot fleet を活用した スケーラブルで低コストな ジョブワーカーシステム

株式会社インティメート・マージャー
開発本部・SREグループ 松田和樹

今日お話しすること

- **弊社でのAWSの活用事例**
 - システム構成
 - 開発、運用で得られた知見

アジェンダ

- インティメート・マージャーについて
- ジョブワーカーシステム
 - 構築の背景
 - 第一世代 (非docker)
 - 第二世代
- Amazon ECS の運用のポイント
- 実運用してみても
- これから Amazon ECS を採用する方へ



IntimateMerger について

自己紹介



松田 和樹

■株式会社インティメート・マージャー

- ・ 開発本部
- ・ SREグループ (テックリード)

■職務内容

- ・ データ処理基盤の開発
- ・ デプロイフローの整備
- ・ パフォーマンス・チューニング
- ・ 監視・運用
- ・ 技術検証・ツール検証

株式会社インティメート・マージャー



設立 **2013年6月**

代表 **築島 亮次**

従業員数 **31名**

事業内容 **DMPの提供・構築支援**

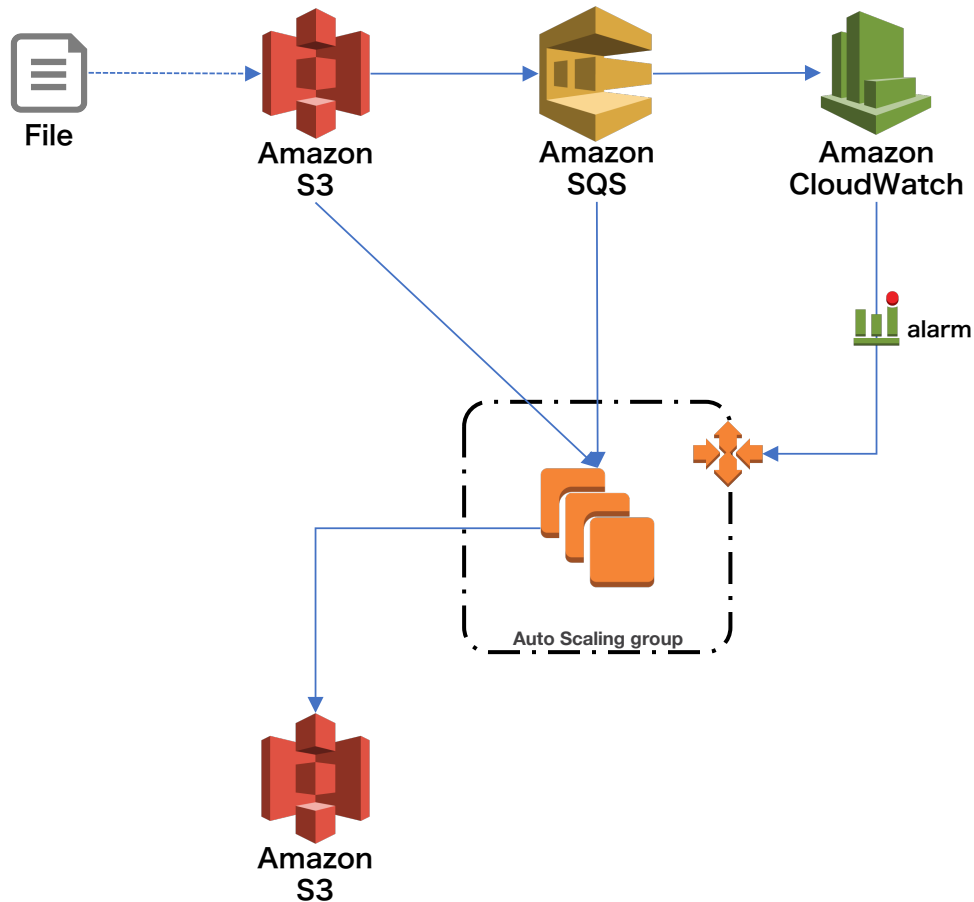
データ活用コンサルティング

ジョブワーカーシステム

構築の背景

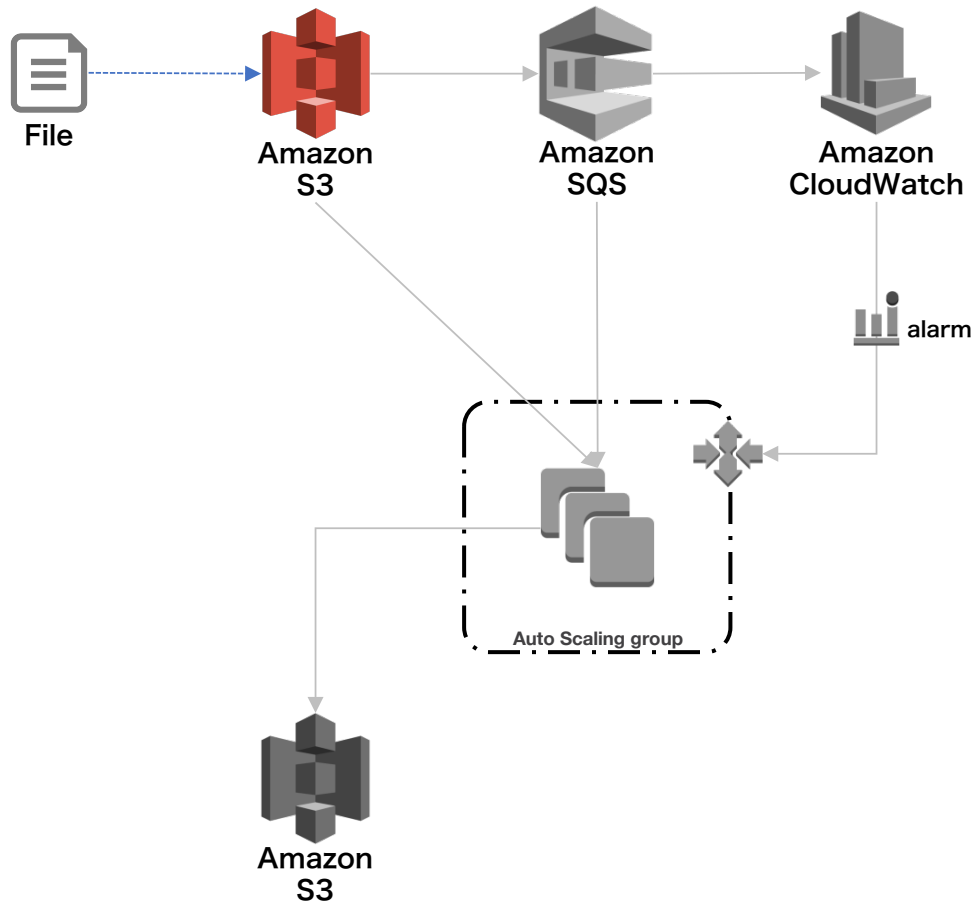
- **20以上の社外システムとデータの受け渡し**
 - **連携先毎に異なるデータ形式**
 - 異なる前処理が必要
 - **連携先毎に異なる接続方式**
 - ex) REST API, FTP, SFTP, Amazon S3, Protocol Buffers ...
- **結果、50種類ものワーカーが必要に**
- **肥大化する処理データ**
 - データの処理量に対してスケールする仕組みが必要
 - 特定のジョブのみスケールさせたい

第一世代



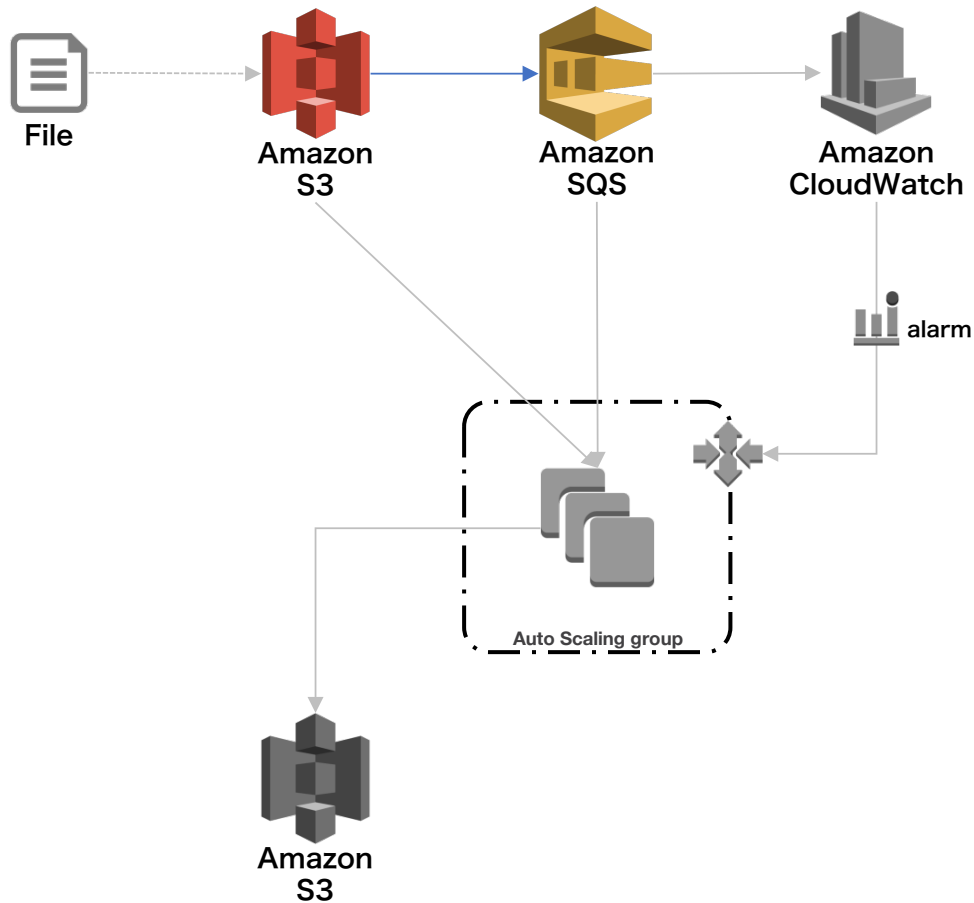
1. ファイルを Amazon S3 へアップロード
2. Amazon S3 Event Notifications で Amazon SQS へ通知
3. Amazon CloudWatch でキューの数でワーカーとなる Amazon EC2 を AutoScaling (Spot Instance)
4. ワーカーは Amazon SQS からキューを、Amazon S3 から処理対象のファイルを取得し、処理を開始
5. 後続処理がある場合は、別の Amazon S3 へ

第一世代



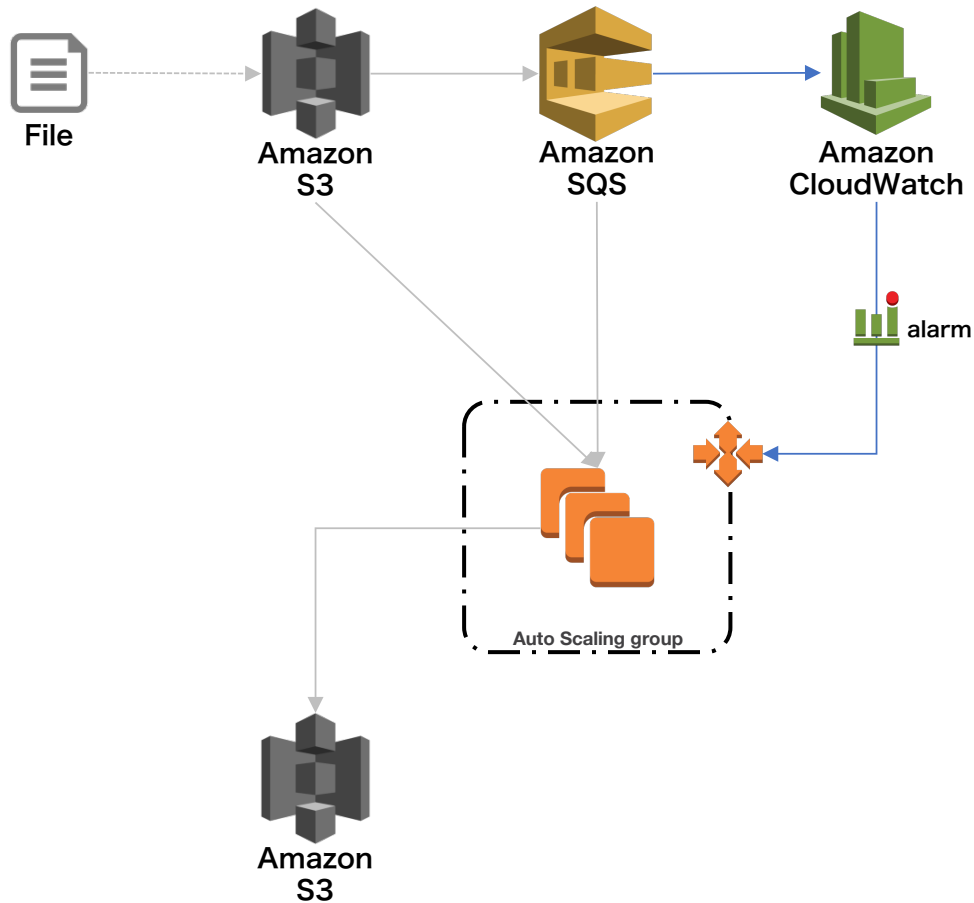
1. ファイルを Amazon S3 へアップロード
2. Amazon S3 Event Notifications で Amazon SQS へ通知
3. Amazon CloudWatch でキューの数でワーカーとなる Amazon EC2 を AutoScaling (Spot Instance)
4. ワーカーは Amazon SQS からキューを、Amazon S3 から処理対象のファイルを取得し、処理を開始
5. 後続処理がある場合は、別の Amazon S3 へ

第一世代



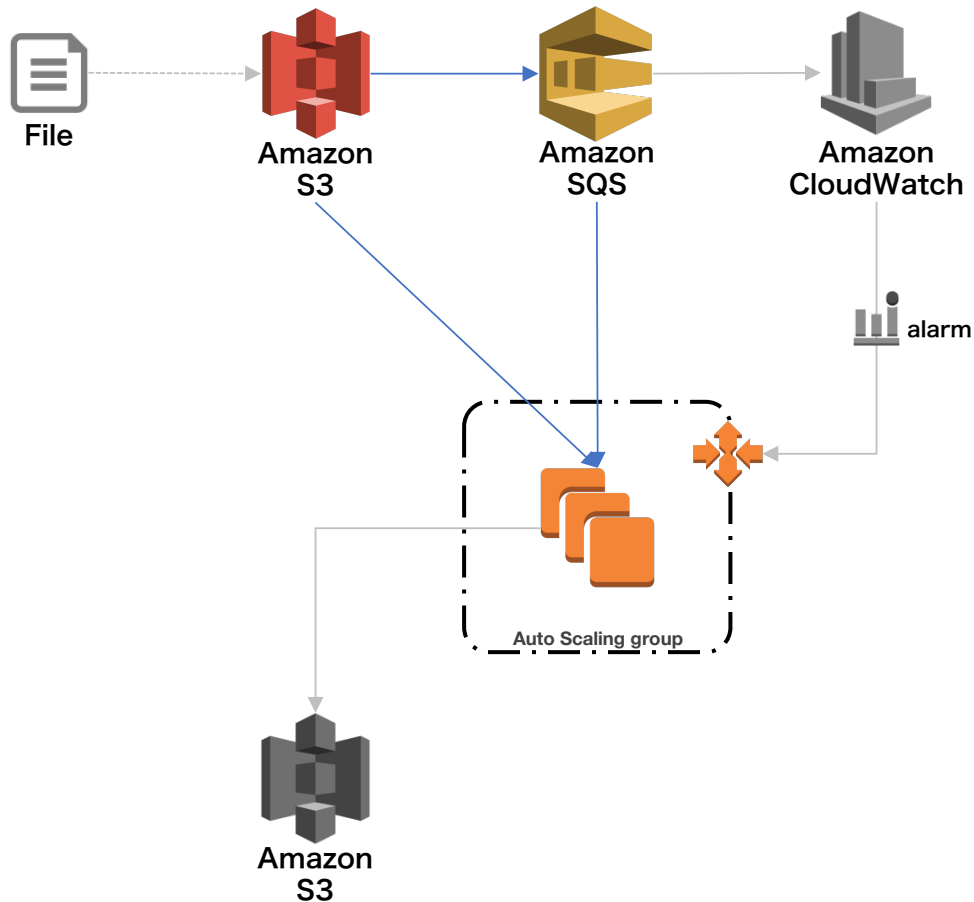
1. ファイルを Amazon S3 へアップロード
2. **Amazon S3 Event Notifications で Amazon SQS へ通知**
3. Amazon CloudWatch でキューの数でワーカーとなる Amazon EC2 を AutoScaling (Spot Instance)
4. ワーカーは Amazon SQS からキューを、Amazon S3 から処理対象のファイルを取得し、処理を開始
5. 後続処理がある場合は、別の Amazon S3 へ

第一世代



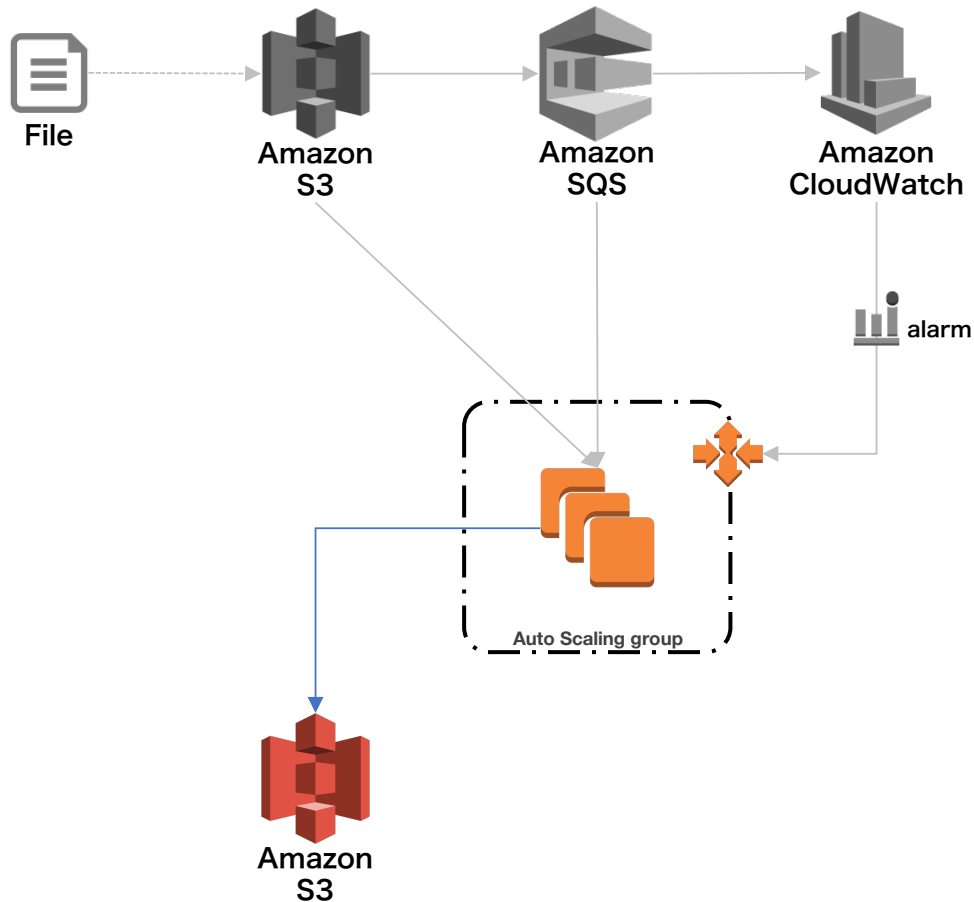
1. ファイルを Amazon S3 へアップロード
2. Amazon S3 Event Notifications で Amazon SQS へ通知
3. Amazon CloudWatch でキューの数でワーカーとなる Amazon EC2 を AutoScaling (Spot Instance)
4. ワーカーは Amazon SQS からキューを、Amazon S3 から処理対象のファイルを取得し、処理を開始
5. 後続処理がある場合は、別の Amazon S3 へ

第一世代



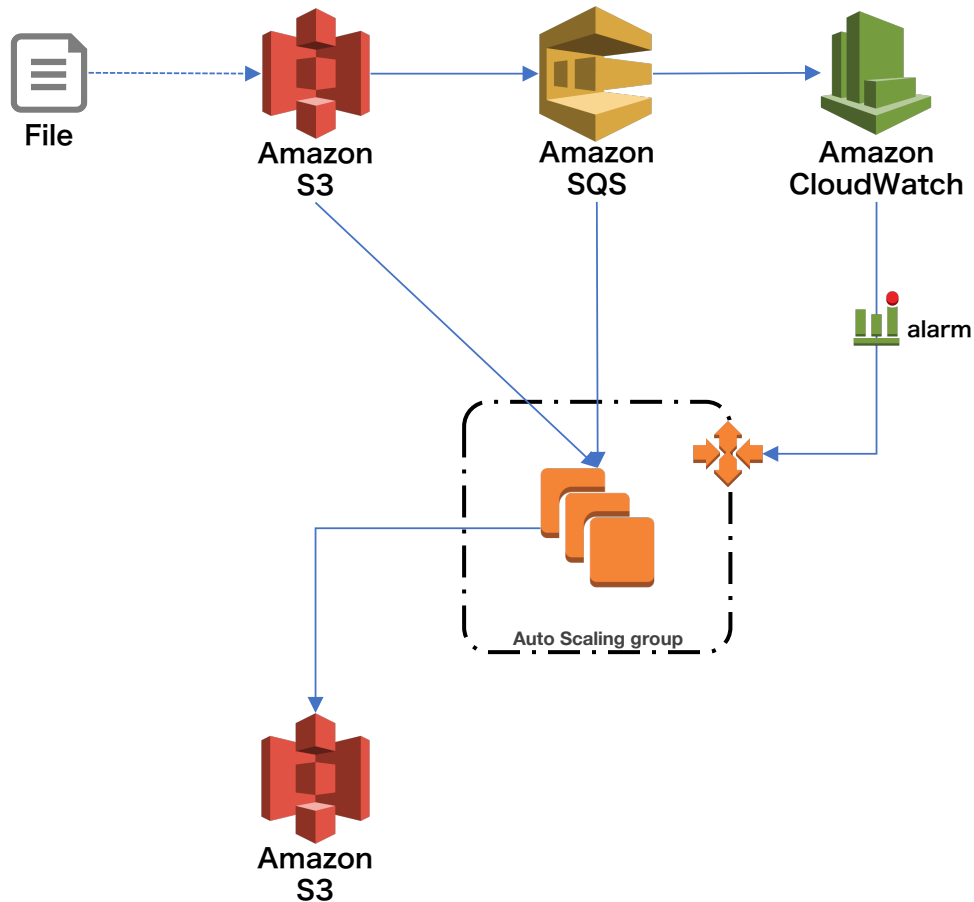
1. ファイルを Amazon S3 へアップロード
2. Amazon S3 Event Notifications で Amazon SQS へ通知
3. Amazon CloudWatch でキューの数でワーカーとなる Amazon EC2 を AutoScaling (Spot Instance)
4. ワーカーは Amazon SQS からキューを、Amazon S3 から処理対象のファイルを取得し、処理を開始
5. 後続処理がある場合は、別の Amazon S3 へ

第一世代



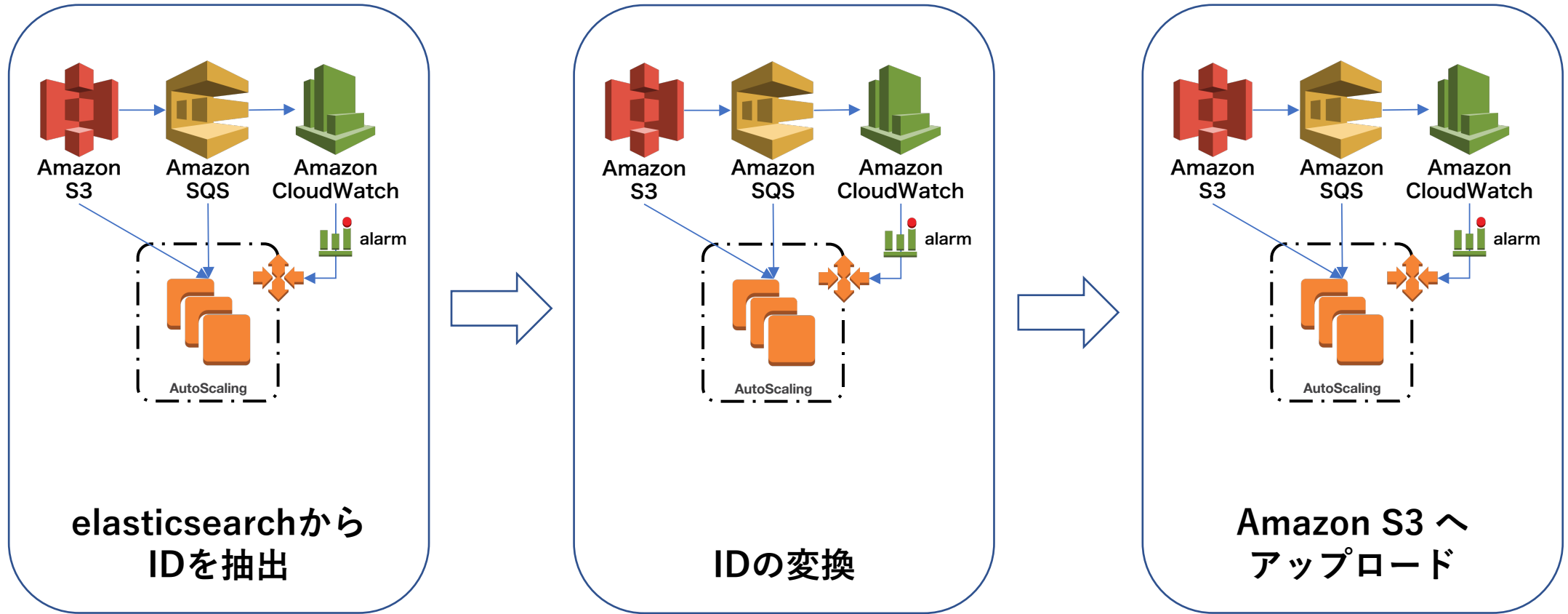
1. ファイルを Amazon S3 へアップロード
2. Amazon S3 Event Notifications で Amazon SQS へ通知
3. Amazon CloudWatch でキューの数でワーカーとなる Amazon EC2 を AutoScaling (Spot Instance)
4. ワーカーは Amazon SQS からキューを、Amazon S3 から処理対象のファイルを取得し、処理を開始
5. 後続処理がある場合は、別の Amazon S3 へ

第一世代

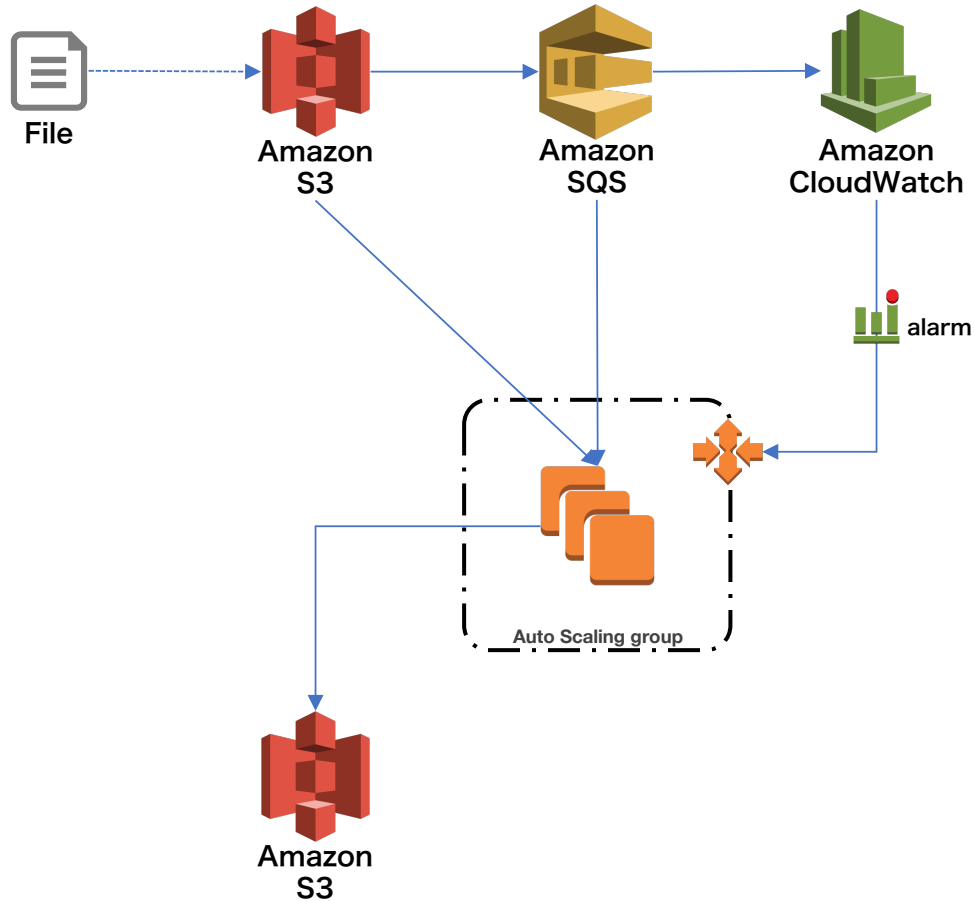


1. ファイルを Amazon S3 へアップロード
2. Amazon S3 Event Notifications で Amazon SQS へ通知
3. Amazon CloudWatch でキューの数でワーカーとなる Amazon EC2 を AutoScaling (Spot Instance)
4. ワーカーは Amazon SQS からキューを、Amazon S3 から処理対象のファイルを取得し、処理を開始
5. 後続処理がある場合は、別の Amazon S3 へ

第一世代



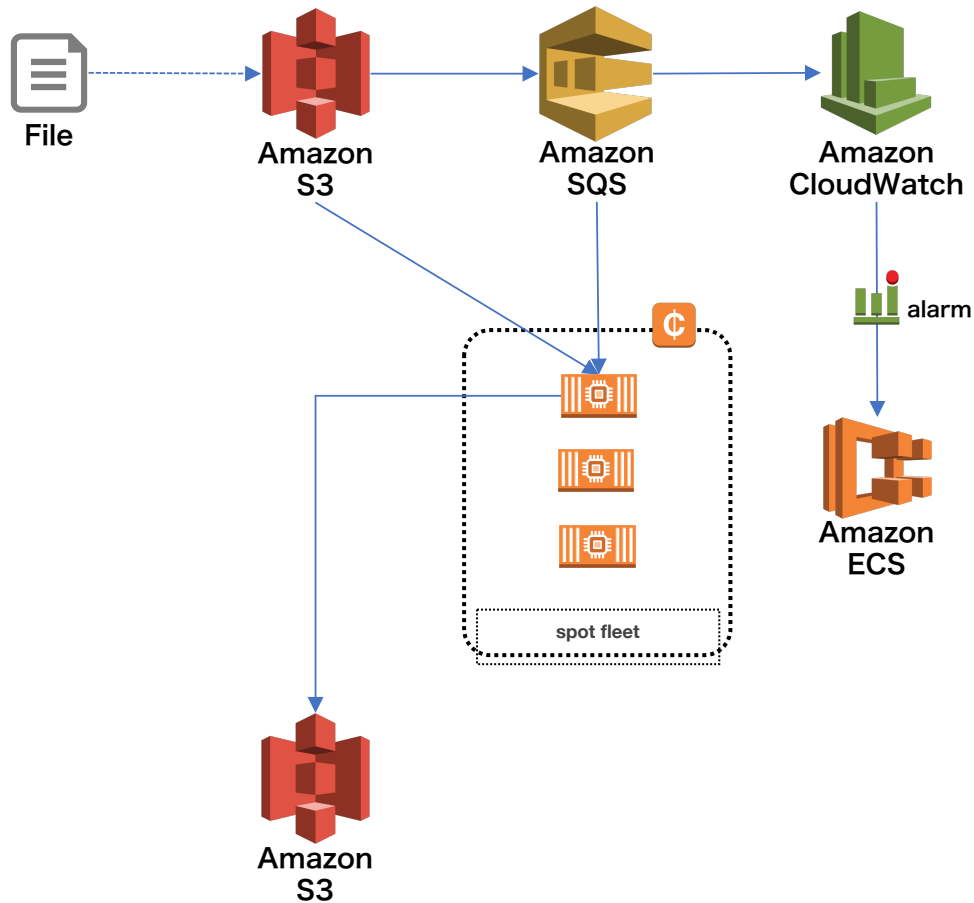
第一世代



課題

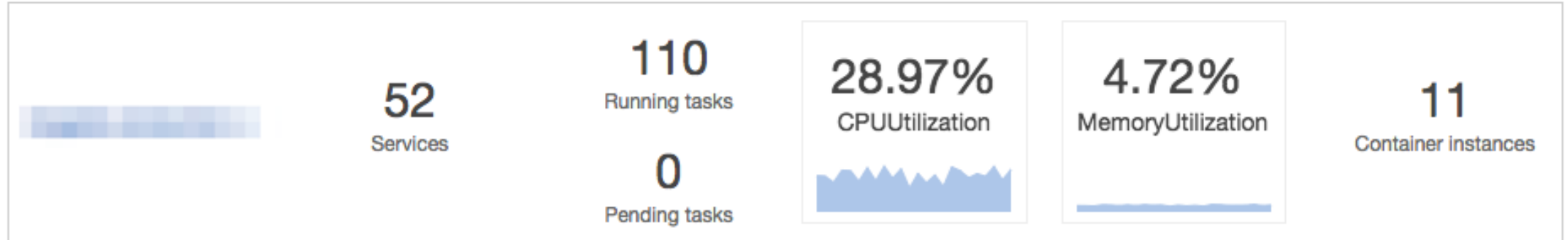
- スポット料金の高騰時など、特定のワーカーが起動不可となる
- ワーカー毎にリソース (Instance Type) の調整が必要な上、全体でのリソース効率が悪い
- 処理開始までの待機時間が長い
- AutoScaling の設定が多くなり、運用負荷が大きい

第二世代



- 基本設計は同様
- 全ワーカーの実行環境をコンテナに移行
- 全ワーカーでdocker基盤を共有
- docker基盤に Amazon ECS を採用
- docker基盤の起動にspot fleetを採用
- 各ワーカー毎にコンテナを AutoScaling

第二世代 - Amazon ECS



- 52のサービスが稼働
 - 50種類のワーカー、Mackerel（監視）、Fluentd（ログコレクタ）
- タスク数は100～250
- コンテナレジストリにはAmazon ECRを採用

第二世代 - Spot fleet

Request Id	Request type	Instance type	State	Capacity	Status	Persistence	Created	Max price
sfr-6431edfc-8761-...	fleet	c4.2xlarge,c4.4x...	active	168 of 168	fulfilled	maintain	10 days ago	\$0.069563

Request Id: sfr-6431edfc-8761-4dbd-b32e-eb7526bf3a11

Description		Instances	History	Auto Scaling
Request Id	sfr-6431edfc-8761-4dbd-b32e-eb7526bf3a11			
Request type	fleet			
Created	4/10/2017, 8:46:34 PM			
State	active			
Status	fulfilled			
Target capacity	168			
Allocation strategy	diversified			
Instance type(s)	c4.2xlarge weight=8, c4.4xlarge weight=16, c4.8xlarge weight=36, m4.2xlarge weight=8, m4.4xlarge weight=16, m4.10xlarge weight=40, m4.16xlarge weight=64			
AMI ID	ami-f63f6f91			
Subnet	subnet-8b1b1b1b			
IAM fleet role	aws-ec2-spot-fleet-role			
Max price	\$0.069563			
Persistence	maintain			
Key pair name				
IAM role	ec2-docker			
EBS-optimized	yes			
Monitoring	no			
Health check	no			
Tenancy	default			
Request valid from	4/10/2017, 8:44:08 PM			
Request valid until	-			
Terminate instances at expiration	no			

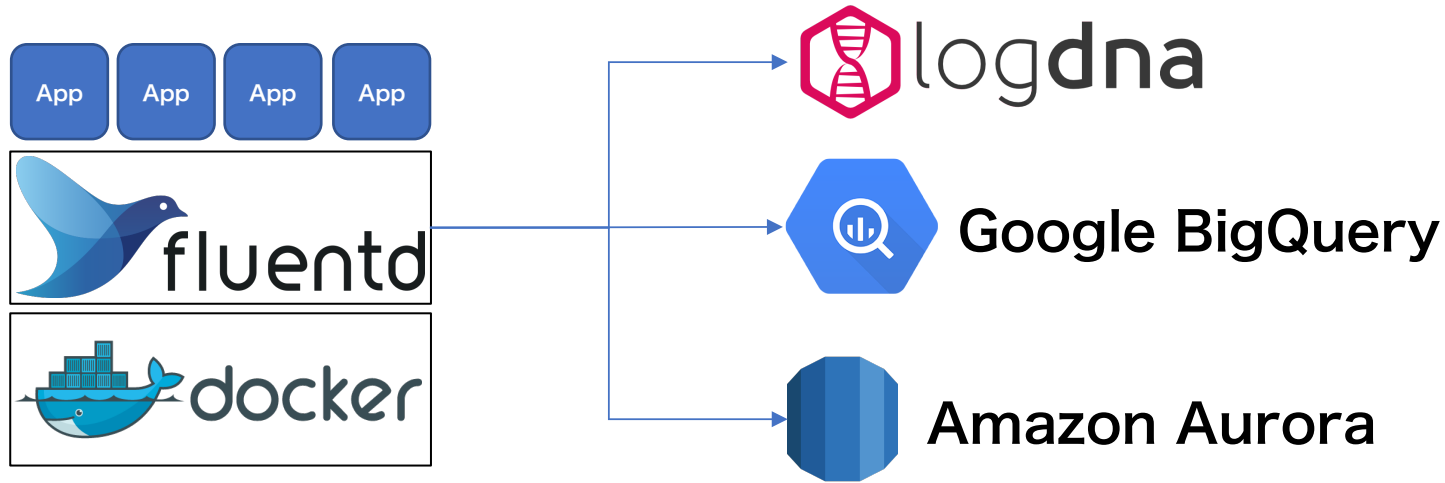
- 容量は 168 vCPUs
- 入札は 自動入札
- Instance Typeは 7種
 - C4系、M4系

Amazon ECS の運用のポイント

Amazon ECS の運用のポイント

- ログのハンドリング
- 環境構築
- デプロイフロー
- 強制Terminate (Spot Instance)
- AMI
- cloud-init

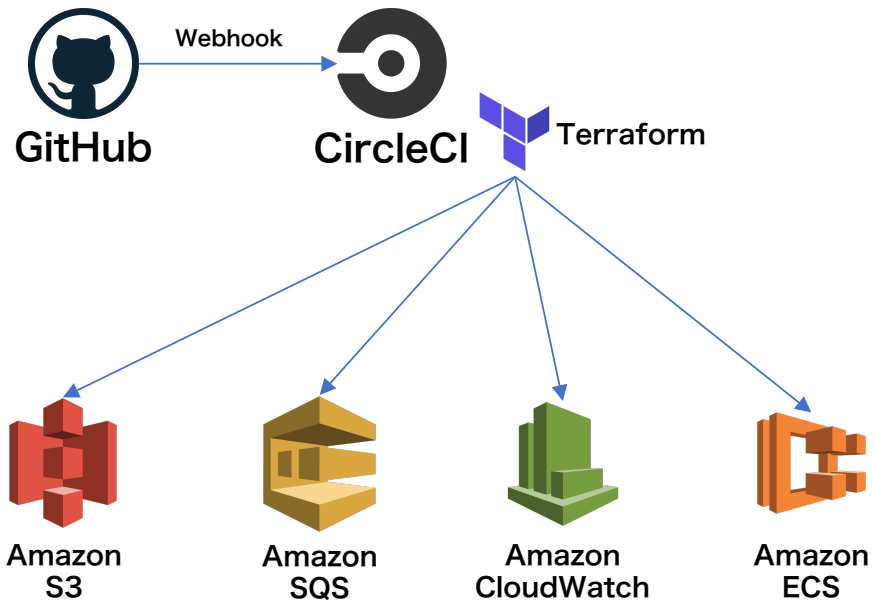
ログのハンドリング



- 各Instanceで稼働するfluentdコンテナでログを集約
 - ワーカーからは `tcp://127.0.0.1:24223` で参照
- logdna : ログのLive Tail & テキストマッチによるアラート
- BigQuery : 長期に渡る分析用
- Amazon Aurora : 直近データの分析、参照用

- ログ収集のクラウドサービス
- リアルタイムに参照することが可能 (tail -f)
- テキストマッチによるアラート機能
 - Slack, Email, Webhook ...
- 低コスト
 - \$1.25/GB/month ~

環境構築



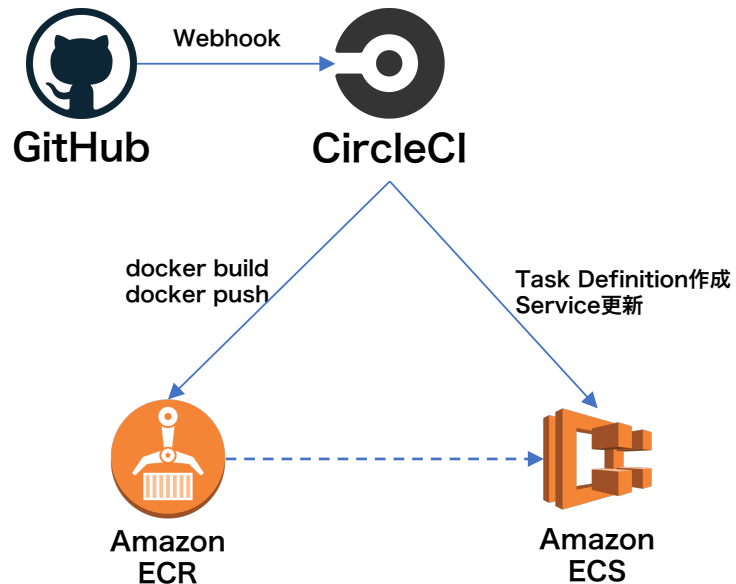
- Terraformを採用

- GitHubで管理
- CircleCI上で実行

- 各ワーカー毎にコンポーネントを作成

- Amazon S3
 - Bucket
 - Event
- Amazon SQS
- Amazon CloudWatch
- Amazon ECS
 - AutoScaling

デプロイフロー



- CircleCIからAPI経由でデプロイ
 - docker build
 - docker push (Amazon ECR)
 - Task Definitionの作成 (Amazon ECS)
 - Serviceの更新 (Amazon ECS)
- コードはPython製(boto3)
- dockerが利用可能なCIサービスの活用がおすすめ

強制Terminate (Spot Instance)

- 価格高騰時など強制Terminate時は、正常なシャットダウンプロセスが行われない
 - コンテナが強制停止される
 - Webサーバの場合は特に影響が大きい
- Terminateの検知
 - 5秒おきにメタデータを参照
 - <http://169.254.169.254/latest/meta-data/spot/termination-time>
 - 検知後、Amazon ECS Clusterから退役
 - コンテナは他のInstanceにオフロードされる
 - amazon-ecs-agent の機能に入れて欲しい・・・

AMIについて

- **Amazon ECS-Optimized AMI を採用**
 - Amazon ECS に最適化されている
- **最新版をそのまま利用**
 - yum update しない
 - AMIのカスタマイズ作成は行わない
- **必要な設定は cloud-init から行う**
 - 必要最低限の設定のみ行い、セキュアに保つため
 - Amazon S3に配置して参照

cloud-init

```
#include
```

```
https://s3-ap-northeast-1.amazonaws.com/<Bucket>/cloud-init.yml
```

- Amazon S3にymlを配置して参照
 - cloud-init 変更時にSpot fleetの作り直しが必要になるため
- VPC Endpointで権限を付与
 - httpで参照するために必要
- cloud-config 形式で記述
 - 実行タイミングを制御できるため

cloud-init で行っていること

- 強制Terminateを検知するcron (前述)
- カーネルパラメータの変更
- Amazon ECSの設定 (/etc/ecs/ecs.config)
- Instance Storeのマウント
 - Docker Engine起動前にマウントしないと利用できない
 - bootcmd ステップでのマウントが必須 (cloud-config 推奨)

実運用してみてください

Amazon ECS にして良かった点

- **可用性の向上**

- Spot fleetにより、スポット価格の高騰の影響を受けにくくなった

- **コスト効率の向上**

- 可用性の向上により、積極的に新世代のInstanceを選択可能
- OS分のリソースが不要になり、同時起動数が向上
- Uptimeを気にせず、コンテナの上げ下げが可能

Amazon ECS にして良かった点

- **スピンアップ時間の短縮**
 - Spot Instanceは起動に時間がかかる
 - dockerがSpot Instanceのデメリットを補完
- **AWSの他サービスとの連携**
 - AWSと統合されているため、既存の仕組みを大きく変えずに
AutoScaling 等が利用可能

Amazon ECS の課題

- クラスタ管理という点では課題も多い
 - agent方式なので、詰まったりラグが生じたりすることも多々
 - Amazon EC2 的にTerminateされているが、Clusterに残っていることも
- dockerの全機能が使えるわけではない
 - agent経由でdockerを管理しているため、ecs-agentに実装されていない機能は使えない

aws / amazon-ecs-agent Watch 136 Star 872 Fork 239

Code Issues 114 Pull requests 8 Projects 0 Insights

Support Google Cloud Logging Driver #734

Open mats116 opened this issue on 22 Mar · 0 comments

mats116 commented on 22 Mar

I think It is not cool to reject "gcplogs" functionally.

https://github.com/aws/amazon-ecs-agent/blob/master/agent/engine/dockerclient/logging_drivers.go#L18-L26

liwenwu-amazon added the feature request label on 23 Mar

aaithal added scope/ECS Agent scope/ECS Service and removed scope/ECS Agent scope/ECS Service labels on 28 Apr

Assignees: No one assigned

Labels: feature request

Projects: None yet

Milestone: No milestone

Notifications: Unsubscribe

You're receiving notifications because you authored the thread.

3 participants

Write Preview AA B i " < > ☰ ☷ ☰ ☷ ☰ ☷ ↶ @ 📎

Leave a comment

Attach files by dragging & dropping, selecting them, or pasting from the clipboard.

Styling with Markdown is supported

Close issue Comment

全Serviceを表示できない

Clusters > [Cluster Name]

Cluster : [Cluster Name] Delete Cluster

Get a detailed view of the resources on your cluster.

Status **ACTIVE**

Registered container instances 13

Pending tasks count 0

Running tasks count 137

Services | Tasks | ECS Instances | Metrics

Create Update Delete Last updated on April 24, 2017 10:05:07 AM (1m ago) Refresh Help

Filter in this page < 1-10 >

<input type="checkbox"/>	Service Name	Status	Task Definition	Desired tasks	Running tasks
<input type="checkbox"/>	[Service Name]	ACTIVE	[Task Definition]	0	0
<input type="checkbox"/>	[Service Name]	ACTIVE	[Task Definition]	0	0
<input type="checkbox"/>	[Service Name]	ACTIVE	[Task Definition]	0	0
<input type="checkbox"/>	[Service Name]	ACTIVE	[Task Definition]	2	2
<input type="checkbox"/>	[Service Name]	ACTIVE	[Task Definition]	0	0
<input type="checkbox"/>	[Service Name]	ACTIVE	[Task Definition]	0	0
<input type="checkbox"/>	[Service Name]	ACTIVE	[Task Definition]	0	0
<input type="checkbox"/>	[Service Name]	ACTIVE	[Task Definition]	1	1
<input type="checkbox"/>	[Service Name]	ACTIVE	[Task Definition]	0	0
<input type="checkbox"/>	[Service Name]	ACTIVE	[Task Definition]	1	1

- Amazon ECS の管理画面に全Serviceを表示できない
- 上限が10
- ページ送りが必要

全Serviceを表示できない

実行環境一覧

ECSサーブिसー覧

status	serviceName	desiredCount	runningCount	pendingCount
ACTIVE	[redacted]	0	0	0
ACTIVE	[redacted]	0	0	0
ACTIVE	[redacted]	0	0	0
ACTIVE	[redacted]	2	2	0
ACTIVE	[redacted]	0	0	0
ACTIVE	[redacted]	3	3	0
ACTIVE	[redacted]	0	0	0
ACTIVE	[redacted]	2	2	0
ACTIVE	[redacted]	0	0	0
ACTIVE	[redacted]	0	0	0
ACTIVE	[redacted]	0	0	0
ACTIVE	[redacted]	0	0	0
ACTIVE	[redacted]	0	0	0
ACTIVE	[redacted]	0	0	0
ACTIVE	[redacted]	0	0	0
ACTIVE	[redacted]	0	0	0
ACTIVE	[redacted]	1	1	0
ACTIVE	[redacted]	0	0	0
ACTIVE	[redacted]	0	0	0
ACTIVE	[redacted]	0	0	0

ECSタスク詳細: [redacted]

実行環境(ECS,EC2)の一覧に戻る

タスク一覧 (running/desired = 2/2)

stop all tasks

taskId	operation	taskDefinition	desiredStatus	lastStatus	startedAt	createdAt
64f1d03b-bed9-4ca9-8aab-da24d98baeae	stop	[redacted]:24	RUNNING	RUNNING	2017年4月24日11:07	2017年4月24日11:07
7a4fc42d-5ded-466f-a7c4-9ebc8c324d6f	stop	[redacted]:24	RUNNING	RUNNING	2017年4月24日11:22	2017年4月24日11:22

logdna

Unsaved View

All Sources

All Apps

All Levels

Help

VIEWS

- Dashboard
- Everything
- aerospike transaction...
- brigade err
- BIRCH

Usage: 2.95GB

Results in 22 days

2017-04-24 11:24:29	ip-10-3-5-187	info	[redacted]
2017-04-24 11:24:33	ip-10-3-5-187	info	[redacted]
2017-04-24 11:24:39	ip-10-3-5-187	info	[redacted]
2017-04-24 11:24:44	ip-10-3-5-187	info	[redacted]
2017-04-24 11:24:49	ip-10-3-5-187	info	[redacted]
2017-04-24 11:24:51	ip-10-3-5-187	info	[redacted]
2017-04-24 11:24:54	ip-10-3-5-187	info	[redacted]
2017-04-24 11:24:59	ip-10-3-5-187	info	[redacted]
2017-04-24 11:25:04	ip-10-3-5-187	info	[redacted]
2017-04-24 11:25:09	ip-10-3-5-187	info	[redacted]
2017-04-24 11:25:14	ip-10-3-5-187	info	[redacted]
2017-04-24 11:25:19	ip-10-3-5-187	info	[redacted]
2017-04-24 11:25:21	ip-10-3-5-187	info	[redacted]
2017-04-24 11:25:24	ip-10-3-5-187	info	[redacted]
2017-04-24 11:25:30	ip-10-3-5-187	info	[redacted]
2017-04-24 11:25:34	ip-10-3-5-187	info	[redacted]
2017-04-24 11:25:34	ip-10-3-5-187	info	[redacted]
2017-04-24 11:25:34	ip-10-3-5-187	info	[redacted]
2017-04-24 11:25:40	ip-10-3-5-187	info	[redacted]
2017-04-24 11:25:41	ip-10-3-5-187	info	[redacted]

ecs_task:7a4fc42d-5ded-466f-a7c4-9ebc8c324d6f

Jump to timeframe...

LIVE

一部の設定はServiceの作り直しが必要

Clusters > [Cluster Name] > [Service Name]

Service : [Service Name] [Update](#) [Delete](#)

Details

Cluster [Cluster Name]
Status **ACTIVE**

Task Definition [Task Definition Name]
Desired count 2
Pending count 0
Running count 2

Load Balancing

Load Balancer Name	Container Name	Container Port
No load balancers		

Deployment Options

Minimum healthy percent 0 ⓘ
Maximum percent 100 ⓘ

Task Placement

Strategy No strategies
Constraint No constraints

Tasks Events Deployments Auto Scaling Metrics

Last updated on April 24, 2017 10:25:55 AM (0m ago) [Refresh](#) ⓘ

Task status: **Running** Stopped

Filter in this page < 1-2 > Page size 50

Task	Task Definition	Group	Last status	Desired status
678f255a-b91c-498d-818...	[Task Definition Name]	[Group Name]	RUNNING	RUNNING
929e2ccf-667c-4007-ac2...	[Task Definition Name]	[Group Name]	RUNNING	RUNNING

- Elastic Load Balancer
- Taskの配置戦略

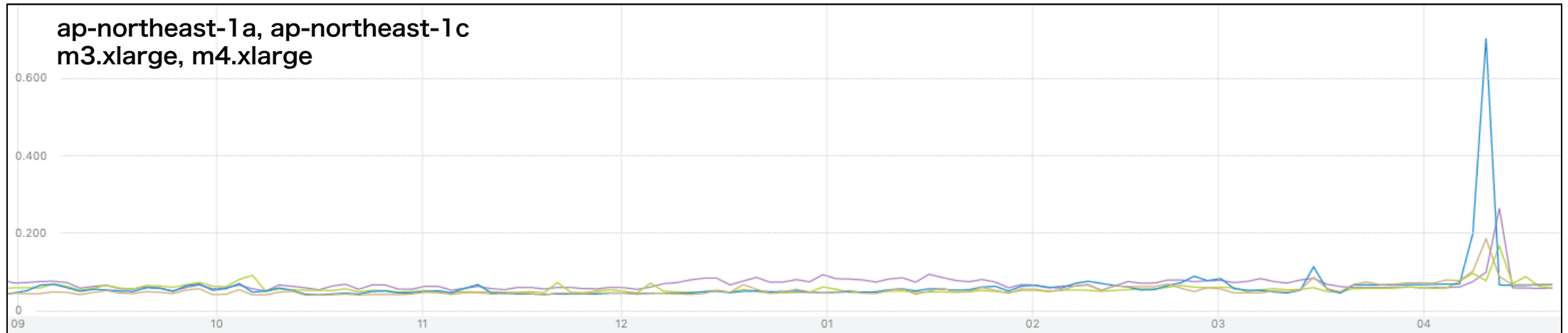
Spot fleetは設定変更が不可

Request Id: sfr-6431edfc-8761-4dbd-b32e-eb7526bf3a11

Description	Instances	History	Auto Scaling
Request Id	sfr-6431edfc-8761-4dbd-b32e-eb7526bf3a11		Max price \$0.069563
Request type	fleet		Persistence maintain
Created	4/10/2017, 8:46:34 PM		Key pair name
State	active		IAM role ec2-docker
Status	fulfilled		EBS-optimized yes
Target capacity	168		Monitoring no
Allocation strategy	diversified		Health check no
Instance type(s)	c4.2xlarge weight=8, c4.4xlarge weight=16, c4.8xlarge weight=36, m4.2xlarge weight=8, m4.4xlarge weight=16, m4.10xlarge weight=40, m4.16xlarge weight=64		Tenancy default
AMI ID	ami-f63f6f91		Request valid from 4/10/2017, 8:44:08 PM
Subnet			Request valid until -
IAM fleet role	aws-ec2-spot-fleet-role		Terminate instances at expiration no

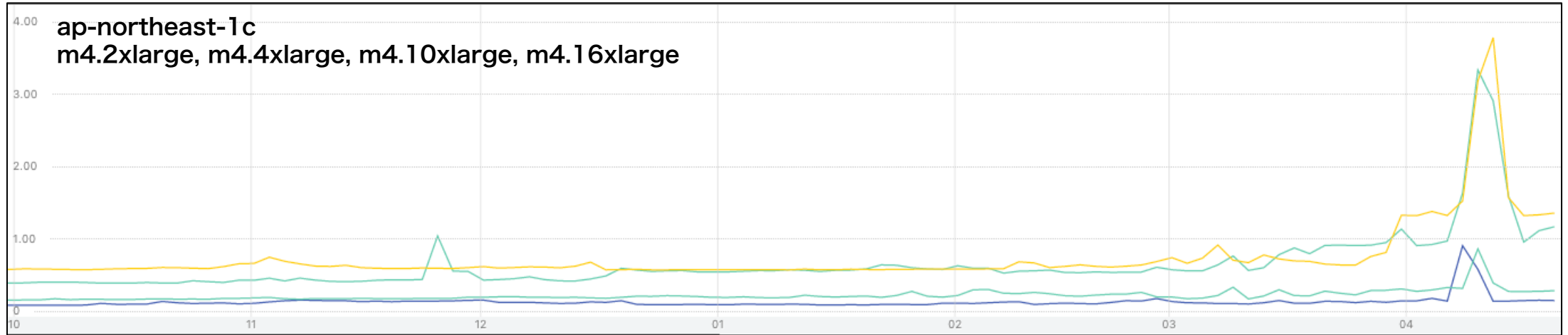
- Capacityの変更のみ可
- 下記の変更はできない
 - AMI
 - Instance Type
 - User data (cloud-init)

Spot Priceのトレンド



- 複数AZで同時に高騰することもある
- そのレアケースが異なるInstance Typeで同時に起こることもある

Spot Priceのトレンド



- 同一Instance Familyが同時に高騰することもある
- 過去に相関がなくても、同時に高騰する可能性は大いにある

これから Amazon ECS を採用する方へ

これから Amazon ECS を採用する方へ

- クラスタ管理としての Amazon ECS
 - 楽な面も多いが、独自の概念も多く、学習コストが低いわけではない
 - 薄いラッパーであるが故にやや緩慢
 - Kubernetes等の方が高機能かつ機敏
 - GitHubの [amazon-ecs-agent](#) のページは見るべき
- Amazon ECS の強みはAWSの他のサービスとの連携
 - Elastic Load Balancer
 - AutoScaling
 - Amazon CloudWatch

これから Amazon ECS を採用する方へ

- Spot fleet はおすすめだが、発展途上
 - Amazon ECS 利用時は使うべき
 - 安い、運用が楽
 - 細かくハンドリングしていない分、実運用してみないと分からないことも多い
- 意外と低レイヤーの知識が必要
 - kernel
 - init (起動関連)
 - cloud-init

ご清聴ありがとうございました