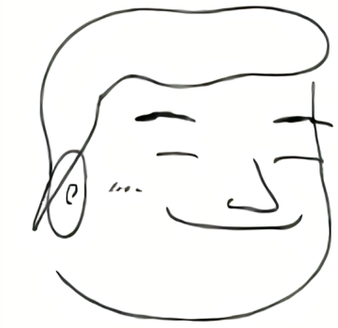
レガシーサーバーを現代の技術で再構築する

builderscon tokyo 2019

2019-08-31

@fujiwara



@fujiwara SRE (総務部)

面白法人カヤック

github.com/fujiwara

sfujiwara.hatenablog.com

面白法人カヤック

Game & Community

話すこと

とある会社にレガシーな開発支援サーバーがあり
Redmine, SVN, Git(gitolite), etc...が動いています

それをAWSのマネージドサービスを使って
近代化改修していく軌跡です

- やらないことを決める
- URLは維持する
- マネージドサービスを活用し、可用性とコスト、メンテナンス性を向上する

開発支援サーバー概要

EC2 シングル構成(Amazon Linux 1) + RDS for MySQL

Redmine : 800ユーザー 1800プロジェクト

Subversion(SVN) : 1000リポジトリ, 1TB

Gitサーバ : gitolite2 1000リポジトリ, 300GB

NoPaste : テキストスニペットを保存してURLを Slack/IRC に投稿

社内Gyazo : スクリーンショットを共有

IRC関連サービス : ircd, znc(IRCへの接続維持), tiarra(IRCクライアント ログ取得用), groonga(ログの全文検索), logviewer(ログ閲覧WebApp)

このサーバーの歴史

第1世代 2010年以前

某レンタルサーバー上に構築

ソフトウェア: Redmine, SVN
ミドルウェア: Apache HTTPd, MySQL
OS: RHEL 5?

このサーバーの歴史

第2世代 2011年

自社DC整備に伴い KVM 化

CentOS 6

SVN はデータコピーで移行

Redmine, SVN のデータは NFS サーバに保存して NFS mount

このサーバーの歴史

第3世代 2015年

自社DC撤退に伴いAWS移行

EC2 1台 (Amazon Linux 2015.03)
DB は RDS for MySQLに分離
ELB は使用せず、直接ApacheでTLS処理
データは EBS にコピーしてEC2でマウント

別のホストで動いていたIRC関連サービス, NoPaste, 社内Gyazo, Gitサーバを統合

4年ごとぐらいには再構築している

200? レンタルサーバ

2011 オンプレ

2015 AWS EC2
2019 ???

再構築は過去の問題を解消するチャンス

Redmineを便利にしていた2011年構成

SVN は htpasswd ファイルによる BASIC 認証

「Redmine のアカウントもSVNと統一したい！」

Apache で BASIC 認証を通過した REMOTE_USER

= Redmineのユーザとして扱うモンキーパッチ 🐒

プラグインを利用者の要望で気軽に追加 🔌🔌🔌

どんどん便利に...?

どんどん便利にした結果

Redmineのバージョンアップが実質不可能に 😇

手元で入れた認証まわりのモンキーパッチが都度必要
→ これは数十行なので頑張れないことはないが...

プラグインがRedmineの新バージョンに非対応(なことがある)
→ プラグインの将来にわたってのメンテを引き受けるのは不可能

「やらないこと」を決める

Redmine のバージョンアップができる状態に保つ

→ **モンキーパッチはしない** 🙈

バージョンアップの障害になるので**プラグインは入れない**

要望があっても断る。一切入れないことにする

「当時の担当者(退職)の遺言なので...」

2011 → 2015 でやったこと

⭕ DB を RDS for MySQL に分離してマネージドに

⭕ Redmine のバージョンアップ

⭕「やらないこと」を決めた

とはいえ移行前とおなじユーザ名とパスワードで認証はしたい

既存ユーザ(数百名)に以前と同一パスワードの発行は不可能

全員に新パスワードを発行 → 配布時に大混乱

モンキーパッチよりマシな方法で解消

Redmine は認証機構をコードでカスタマイズできる[1]

htpasswdを読んで認証するコードを実装することで対応

```
require 'htauth'
class AuthSourceHtpasswd < AuthSource
  def authenticate(login, password)
    r = nil
    HTAuth::PasswdFile.open("/path/to/htpasswd", HTAuth::File::ALTER) do |pf|
      user = pf.fetch(login)
      if user && user.authenticated?(password)
        r = { login: login, auth_source_id: self.id }
      end
    end
    return r
  rescue => e
    raise AuthSourceException.new(e.message)
  end
```

[1] http://www.redmine.org/projects/redmine/wiki/Alternativecustomauthentication HowTo

2011 → 2015 でやったこと

❗ **SVN や Redmine の添付ファイルのデータを EBS に保存**

Elastic Block Store(EBS): EC2用のブロックストレージ

EBS は別のホストから同時マウントできない
→ 必然的にEC2はシングル構成に

EBS は AZ を跨げない
→ AZ 障害に弱い

なんらかのリモートファイルシステムを使うべきだった...?

可用性のあるNFSサーバを自分で立てる?

注: 2015年当時、Amazon EFS はまだ存在しない

EC2 を2台、別 AZ に用意して自力でファイル同期
lsyncd とか DRBD とか

...Failover も自力でやる？

分散ファイルシステムミドルウェアを EC2 上で動かす
GlusterFS とか Ceph とか

...その維持とバージョンアップはだれが？

EBSでいくという判断はやむを得なかった部分はある

2011 → 2015 でやったこと

❌ELB を使用せず、TLS終端をApacheで実行

EBS の関係で EC2 がシングル構成

→1台しかないのにELBを入れるのはコスト的に無駄という判断

(まちがい)

当時はワイルドカード証明書(DV)を社内の各所で使用

証明書取得のコストは他と共用なので無視できた

ACMもまだない(2016年リリース)

2019年のいま考えると

TLS周辺には脆弱性が頻繁に見つかるため、定期的に対応が必要

BEAST(2011), CRIME(2012), BREACH(2013), POODLE, Heartbleed(2014), FREAK, LOGJAM(2015), DROWN(2016)..

ELB なら AWS が面倒を見てくれる

TCPを直接外に晒しているだけで影響を受ける脆弱性
(CVE-2019-11477等)

ELB だったら(ry

最近の傾向

最近のネットワークまわりの脆弱性は、OS開発者、大手クラウドベンダーや CDN 事業者間で先に検討され、クラウドサービス側が修正されてから公開されることが多い

自分でサーバのポートを外に晒す
＝ 公開後、自分らで対処するまで脆弱なまま

ELB なら AWS が既にパッチを当てている
＝ 公開時には既に影響を受けない(ことが多い)

直接EC2でグローバルに晒すのは「覚悟」が必要な時代

前回移行から4年

AWS移行からの4年間で、世間も社内事情もいろいろ変わった

社内のチャットを Slack に全面移行(2016年)
→ IRC関連がほぼ必要なくなった

SVN、社内Gitサーバをほぼ使わなくなった
→ ごく一部はまだ使用しているが、ほとんど全て GitHub に

Redmineはまだまだ使っている
3.0 → 3.3 へのバージョンアップは無事に乗り切った

そして **Amazon Linux 1 EoL (2020年6月)** で移行が必須に

「やらないこと」を決める

SVN: **今後移行してもらうことを前提に、維持する**
約1000リポジトリ、1TB程度あるがアクティブなのは数個
新規に作成はしない

⛔ **Git**: **サーバ機能を停止** ファイルだけ残す
アクティブなものは GitHub に移行する

Redmine: バージョンアップを含めて**維持する**
3.3.xはEoL。ユーザーサポート対応などでヘビーユース
即廃止や移行は困難

⛔ **IRC関連**: **全部止める**
過去ログと検索だけは残す。まだ歴史を振り返ることが...

2019年移行で達成したい目標

1. TLS 終端をマネージドサービス化
脆弱性への対応向上、証明書管理からの解放

2. EC2 シングル構成を脱して堅牢に
2019-08-23 東京リージョン大障害によりダウンしてしまった

3. リポジトリで管理されてない部分を極力なくす
サーバ上で手作業が行われると管理されない設定で荒れていく

4. 現在のEC2をなくす = AmazonLinux 1 EoLに対応

今回の移行ではこの4点を達成したい

EC2 を分解しマネージドサービスで再構築

ALB (Application Load Balancer) を導入する
→ 「1. TLS 終端をマネージドサービス化」達成

EBS を Amazon EFS / S3 に置き換える
→ 「2. EC2 シングル構成を脱して堅牢に」達成のため
　複数台から共有できるストレージが必要

アプリケーションをコンテナ化してALBから振りわけ
→ 「2. EC2 シングル構成を脱して堅牢に」
　「3.リポジトリで管理されてない部分を極力なくす」
　「4.現在のEC2をなくす」達成のため

ALB でルーティングし、URL を維持したまま段階移行が可能に

Amazon ECS によるミドルウェアのコンテナ化

2015年の時点で、Redmine, SVN, gitolite 以外のものはコンテナ化済。EC2上の Docker Compose で稼働している

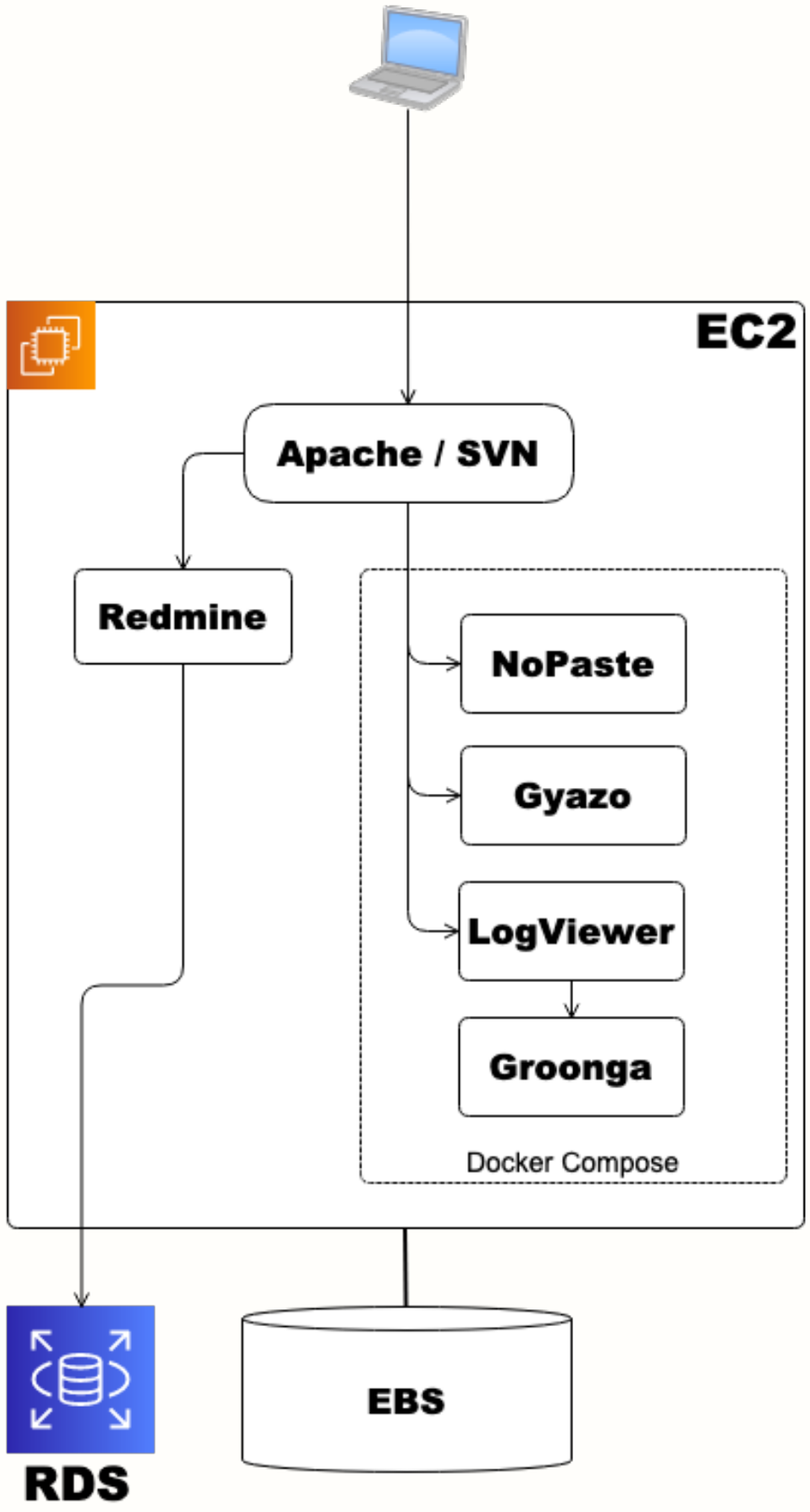
ECS のノウハウが社内に溜まって、機が熟した
ECS デプロイツール ecspresso[2] を各プロジェクトで使用
リソース管理、オペレーションもほぼ統一できている

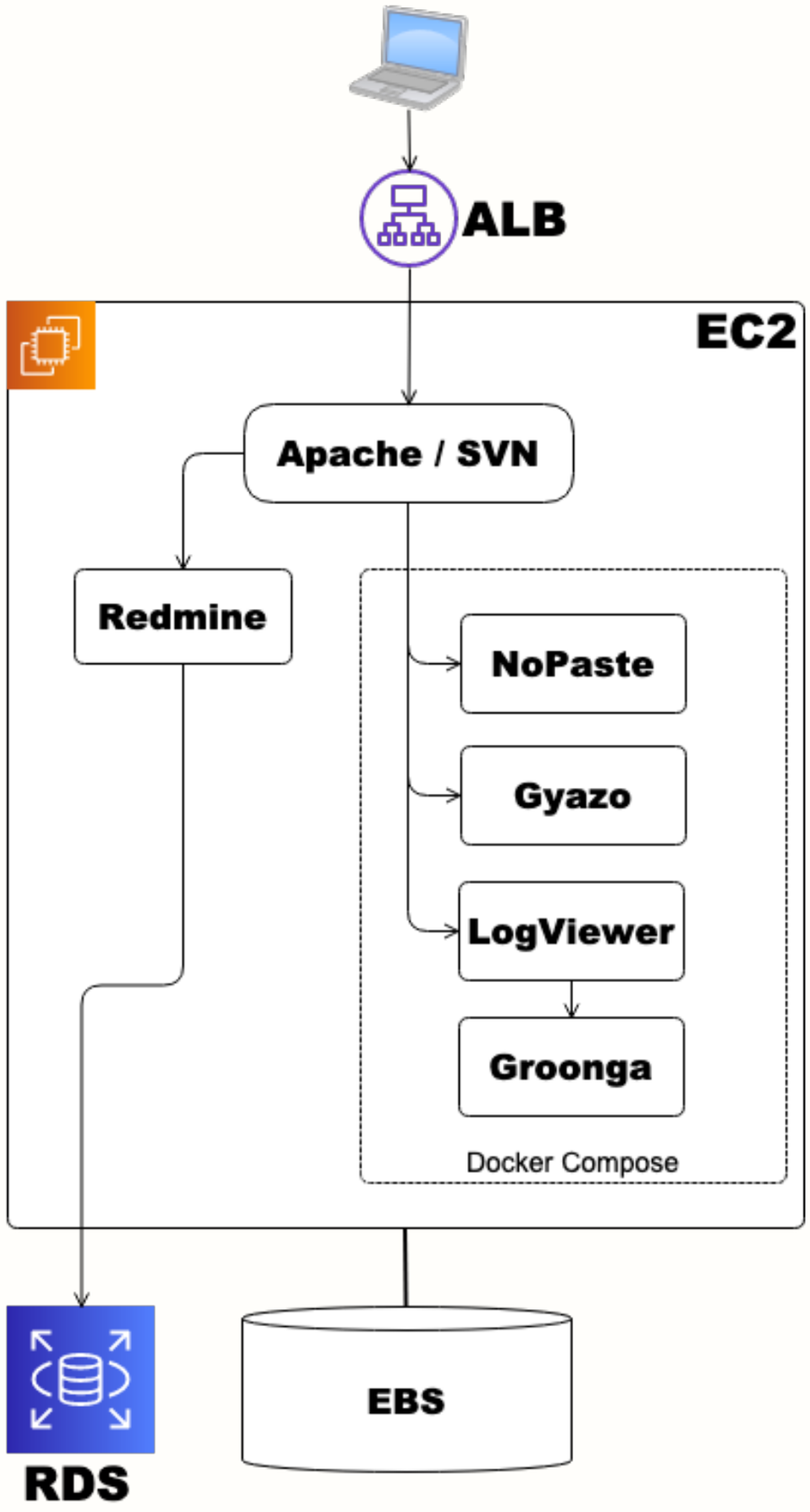
コンテナに含めるものはリポジトリで管理される

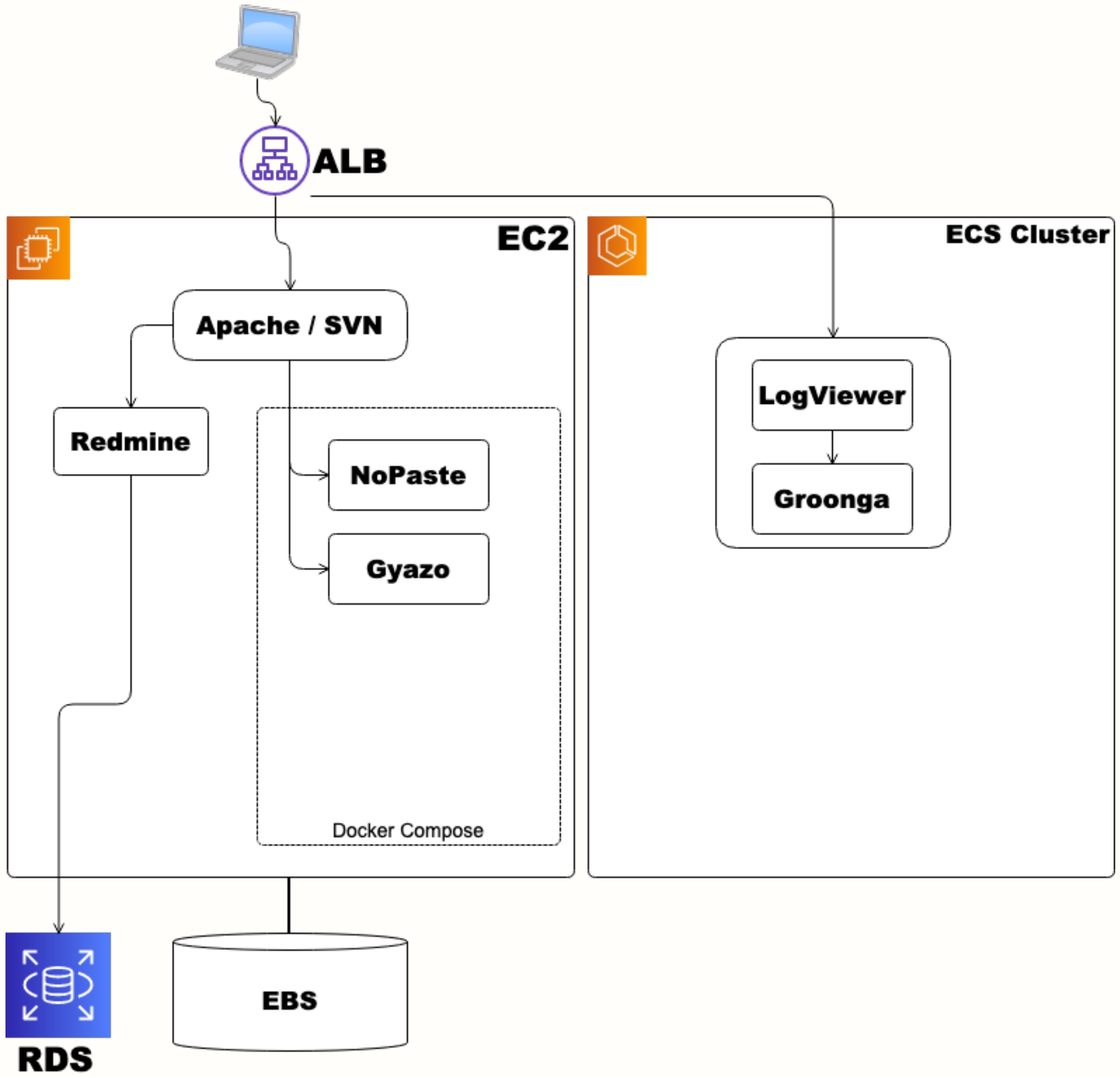
AWSのマネージドサービスを多数使用することになる

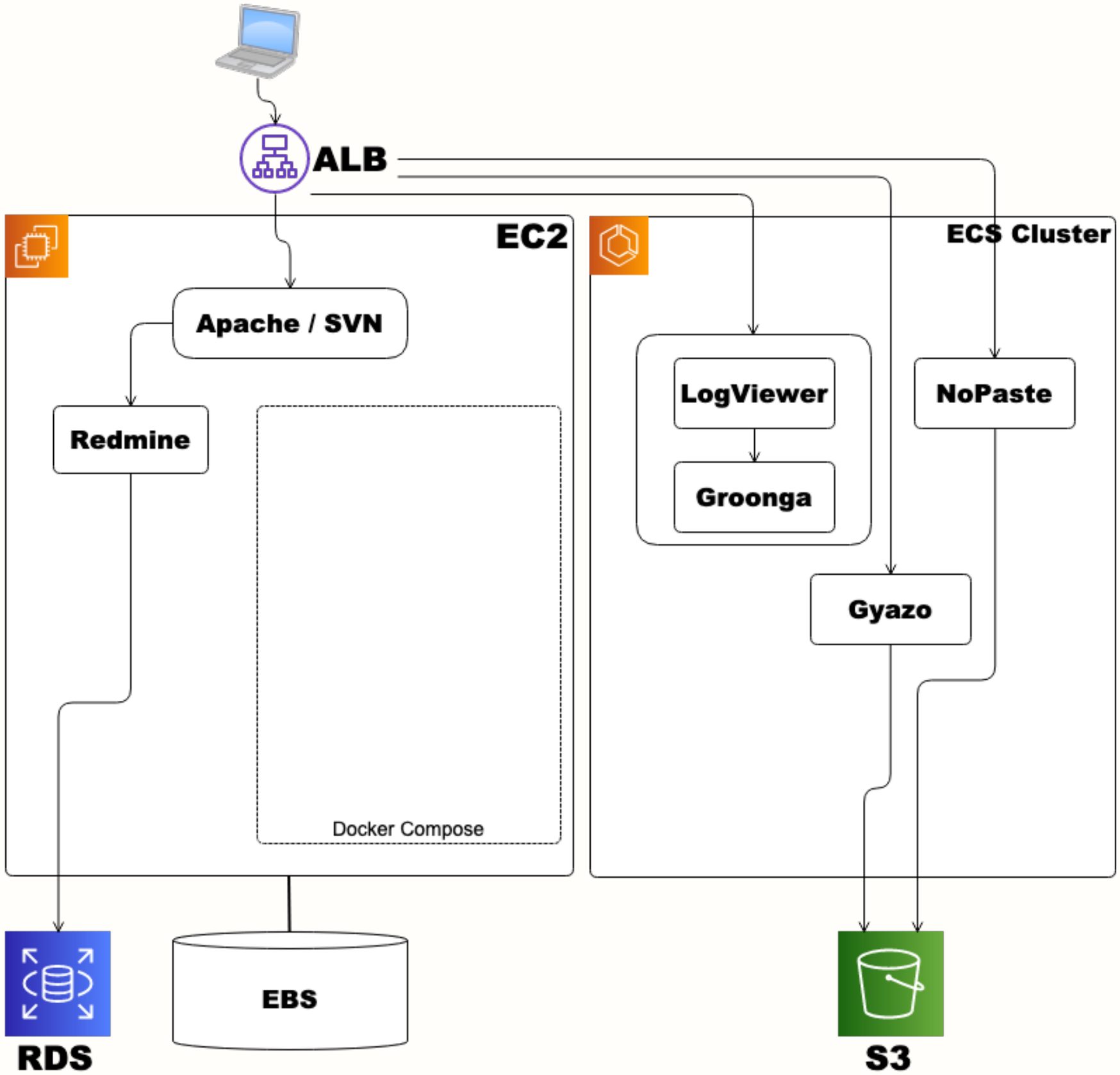
→ Terraform で管理する

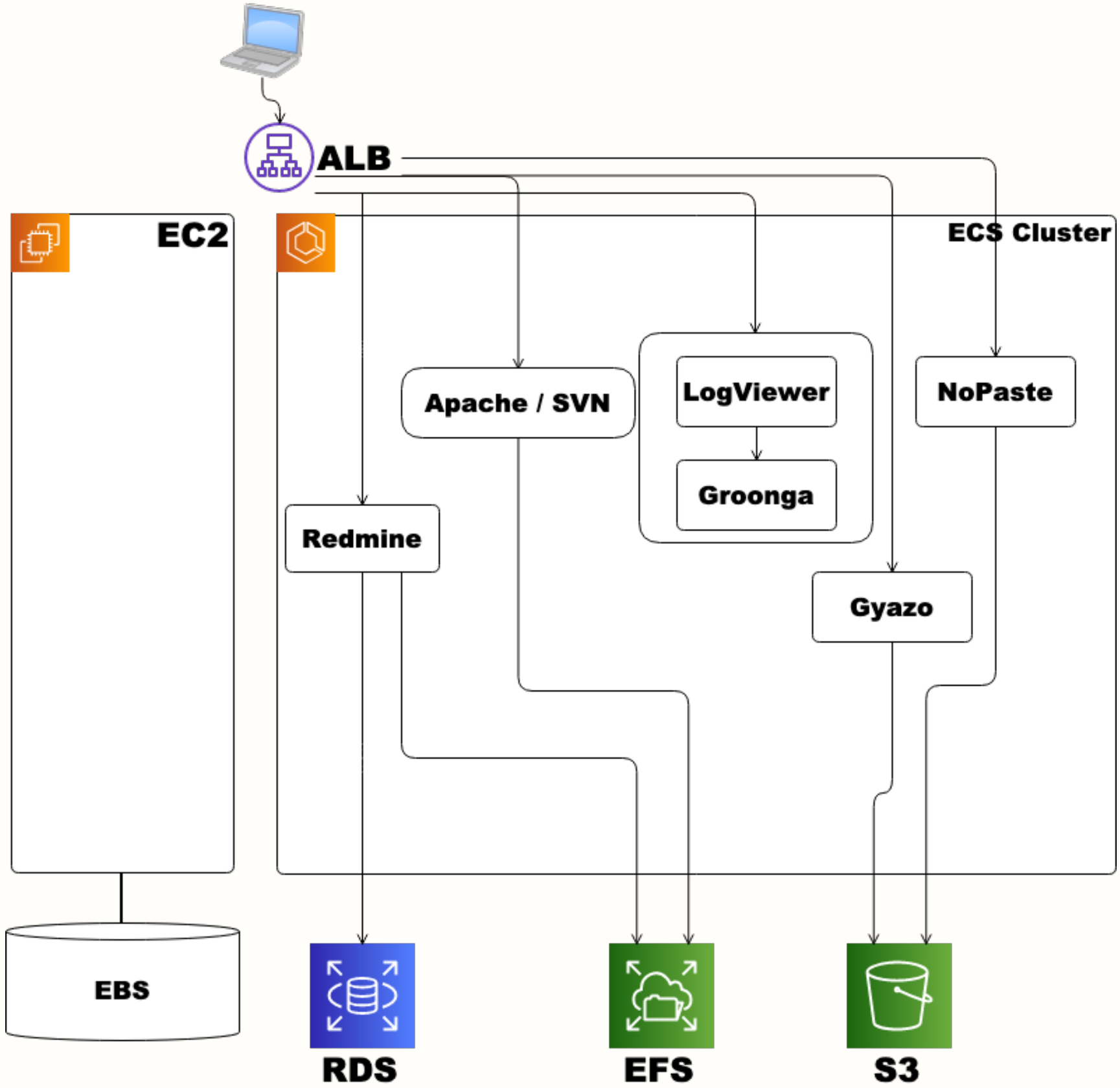
[2] https://github.com/kayac/ecspresso


EC2
Apache / SVN
Redmine
NoPaste
Gyazo
LogViewer
Groonga
Docker Compose
RDS
EBS

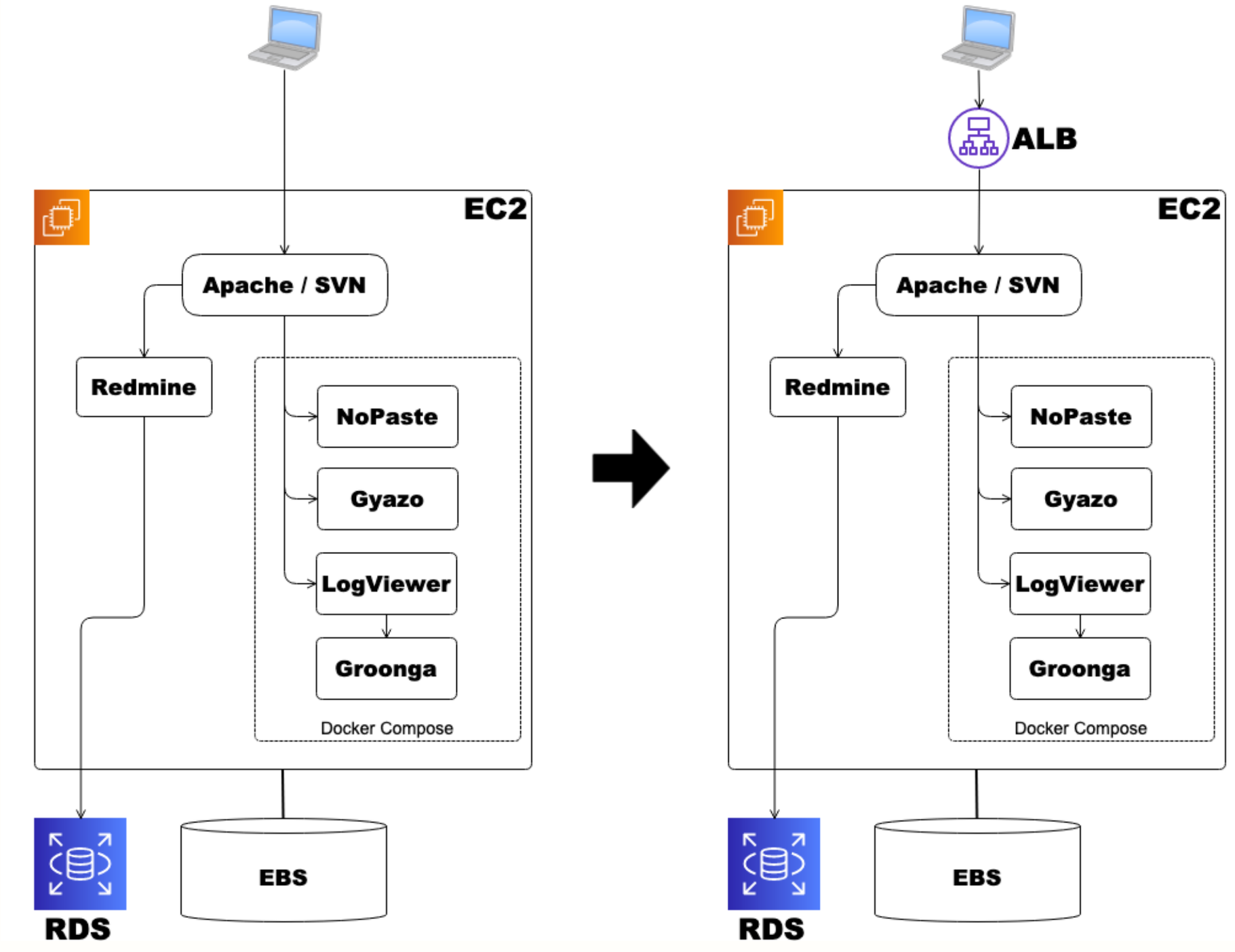

ALB
EC2
Apache / SVN
Redmine
NoPaste
Gyazo
LogViewer
Groonga
Docker Compose
RDS
EBS


ALB
EC2
ECS Cluster
Apache / SVN
Redmine
NoPaste
Gyazo
Docker Compose
LogViewer
Groonga
RDS
EBS


ALB
EC2
Apache / SVN
Redmine
Docker Compose
ECS Cluster
LogViewer
Groonga
NoPaste
Gyazo
EBS
RDS
S3


ALB
EC2
ECS Cluster
Apache / SVN
LogViewer
Groonga
NoPaste
Redmine
Gyazo
EBS
RDS
EFS
S3

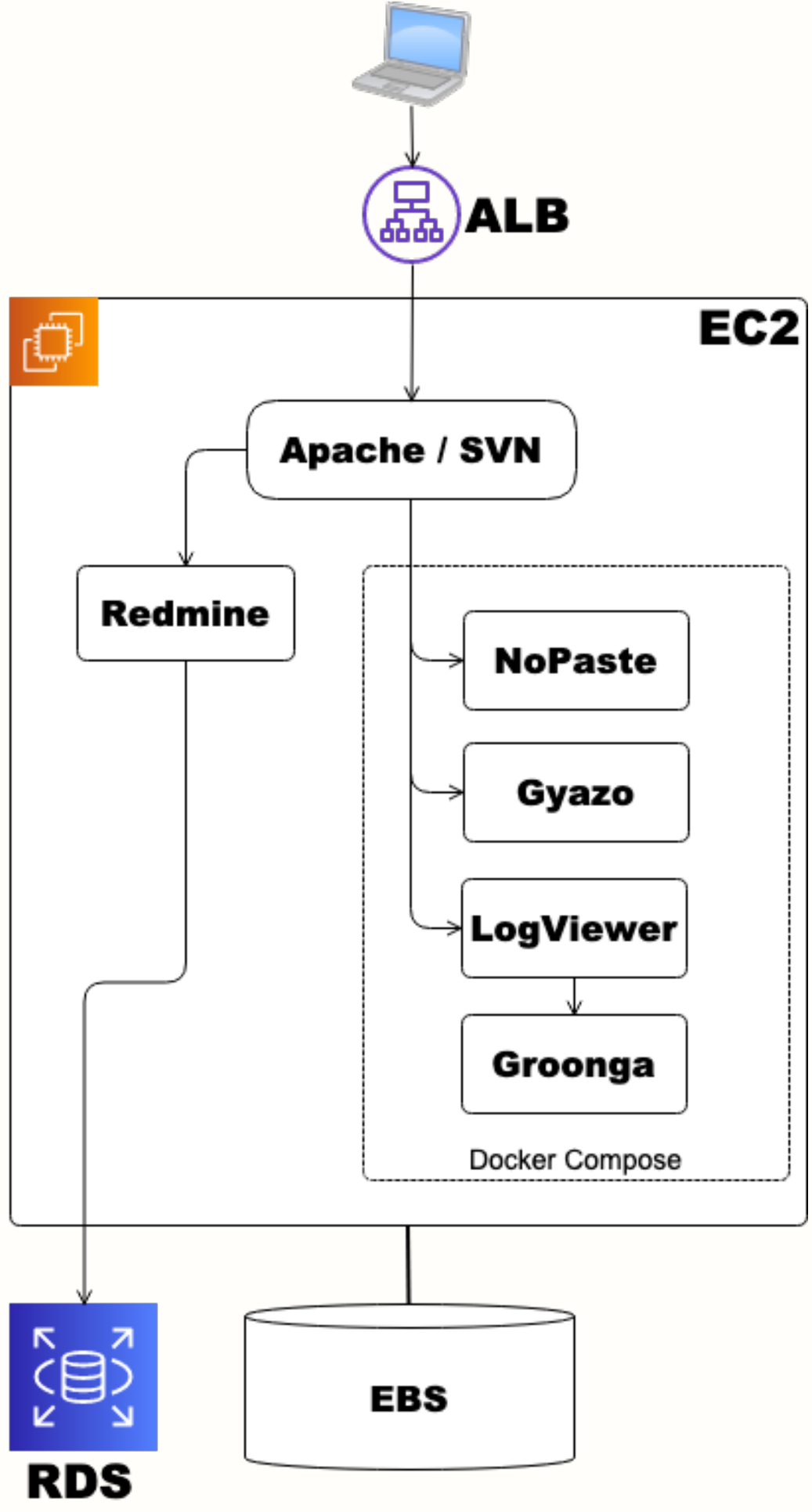
ALB (Application Load Balancer) を導入



ALB で TLS 終端

1. ALB を導入、ACM で発行した証明書を使用
2. ALB のデフォルトターゲットとして EC2 を追加
3. DNS を ALB へ向けたら完了！

簡単ですよね??



途中からALBを入れるにあたっての検討事項

ターゲットへの通信は HTTP か HTTPS か?

HTTP(L7)でproxyするため、EC2側は通常平文(HTTP)で受ける

(歴史的経緯により) httpd.conf とそこから include される設定ファイルが大量に存在

443 → 80 の VirtualHost に設定を移すところで間違うと障害に

ルーティング、ログ出力、他諸々の処理を漏れなく80に移動する必要がある 😨

💡 **「HTTPSのままなら設定を変えなくていいのでは」**

途中からALBを入れるにあたっての検討事項

EC2側はHTTPS(443)のままでいけるかどうか

ALBのターゲットは HTTPS での通信が可能⭕

ターゲット側の証明書は検証されない⭕

証明書と名前の一致、有効期限は検証されないので今のままでOK

→ そのままHTTPSで受けることにした

ALBを入れるにあたって必要なこと

リクエスト元IPアドレスを正しく認識する

なにもしないとリクエスト元が ALB のプライベートIPアドレスに
ログ記録やアクセス制限に支障が出る

ログ記録やアクセス制御を X-Forwarded-For ヘッダで行う？
既存設定を全部書き換えるのは 🤦

mod_remoteip (Apache 2.4から標準)

Apache 2.2用 github.com/ttkzw/mod_remoteip-httpd22

mod_rpaf
Apache 2.2用 github.com/ttkzw/mod_rpaf-0.6 [3]

[3] https://heartbeats.jp/hbblog/2012/03/mod-rpaf.html

mod_remoteip or mod_rpaf ?

mod_remoteip

通常アクセスではIPアドレスが取得できたが
ALB経由で svn checkout すると ALB が 502 Bad Gateway を返す

Apache側では200を返しているが、BASIC認証通過後のアクセスログがおかしい (IPアドレス部分が空欄や "s_")

```
192.0.2.41 - - [27/Jun/2019:15:45:24 +0900] "OPTIONS /svn/xxx HTTP/1.1" 401
 - fujiwara [27/Jun/2019:15:45:24 +0900] "OPTIONS /svn/xxx HTTP/1.1" 200
s_ - fujiwara [27/Jun/2019:15:45:24 +0900] "OPTIONS /svn/xxx HTTP/1.1" 200
```

mod_rpaf

svn checkout も問題なく動作したため mod_rpaf に決定

このようなテストをどこでやるか

この EC2 インスタンスは一点もの

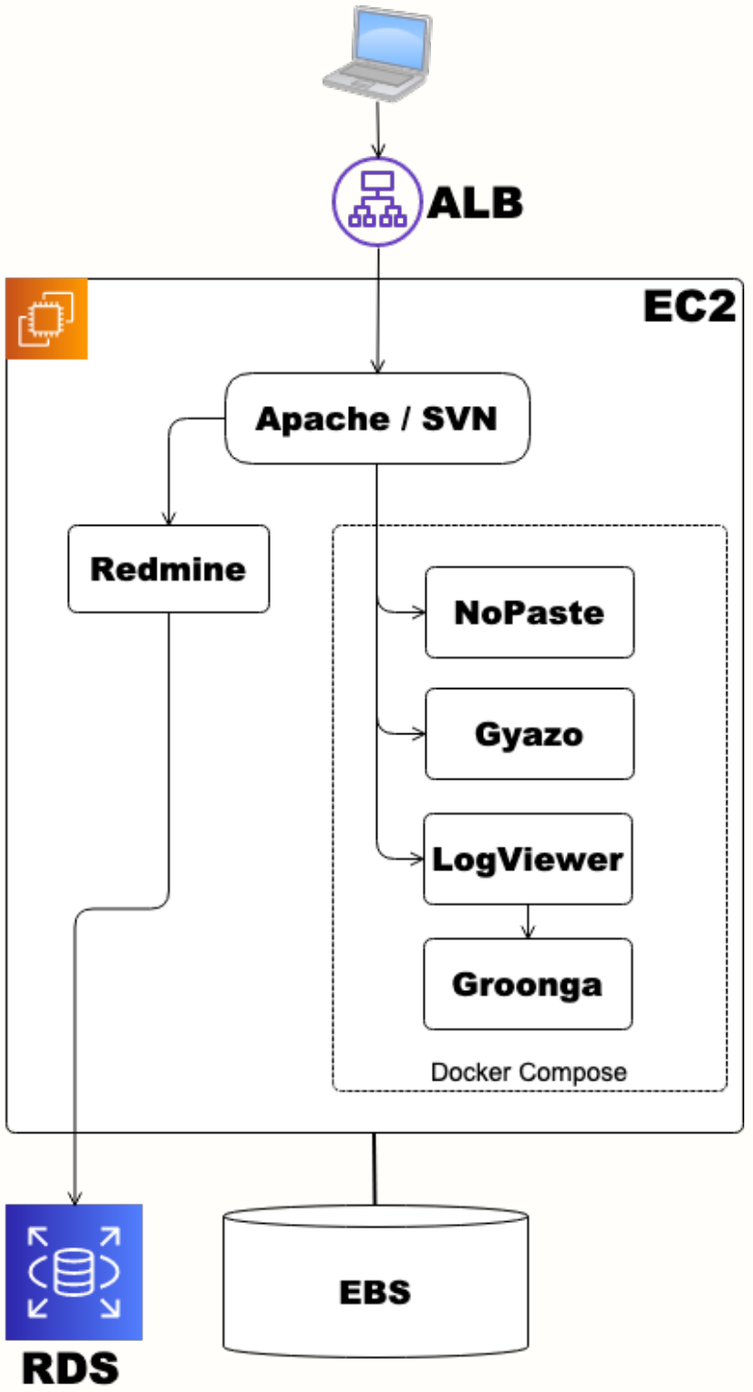
イメージを取得して別インスタンスを立ててテストしたが...

稼働中のインスタンスから作ったAMIを起動→当然 crond も起動
cron の処理が複製されたインスタンスでも同時に実行される

(例) 外部サーバーにファイルをコピーする処理が競合

教訓: 一点ものインスタンスを複製したらcrondは即停止
ユーザ定義の cron とか思わぬものが動いていたりする...

目標「1. TLS 終端をマネージドサービス化」達成 🎉



ストレージを Amazon EFS / S3 へ

これまでシングル構成だった根本原因 ＝ EBSにファイルがある

EBS 上のファイルは Amazon EFS (マネージドNFS) や S3 に移す

AZ 障害にも耐えられる

ストレージの移行先を適切に選択する

ファイルじゃないといけないもの
SVN、Redmine の添付ファイルなど

アプリケーションに手を入れられない → **EFS**

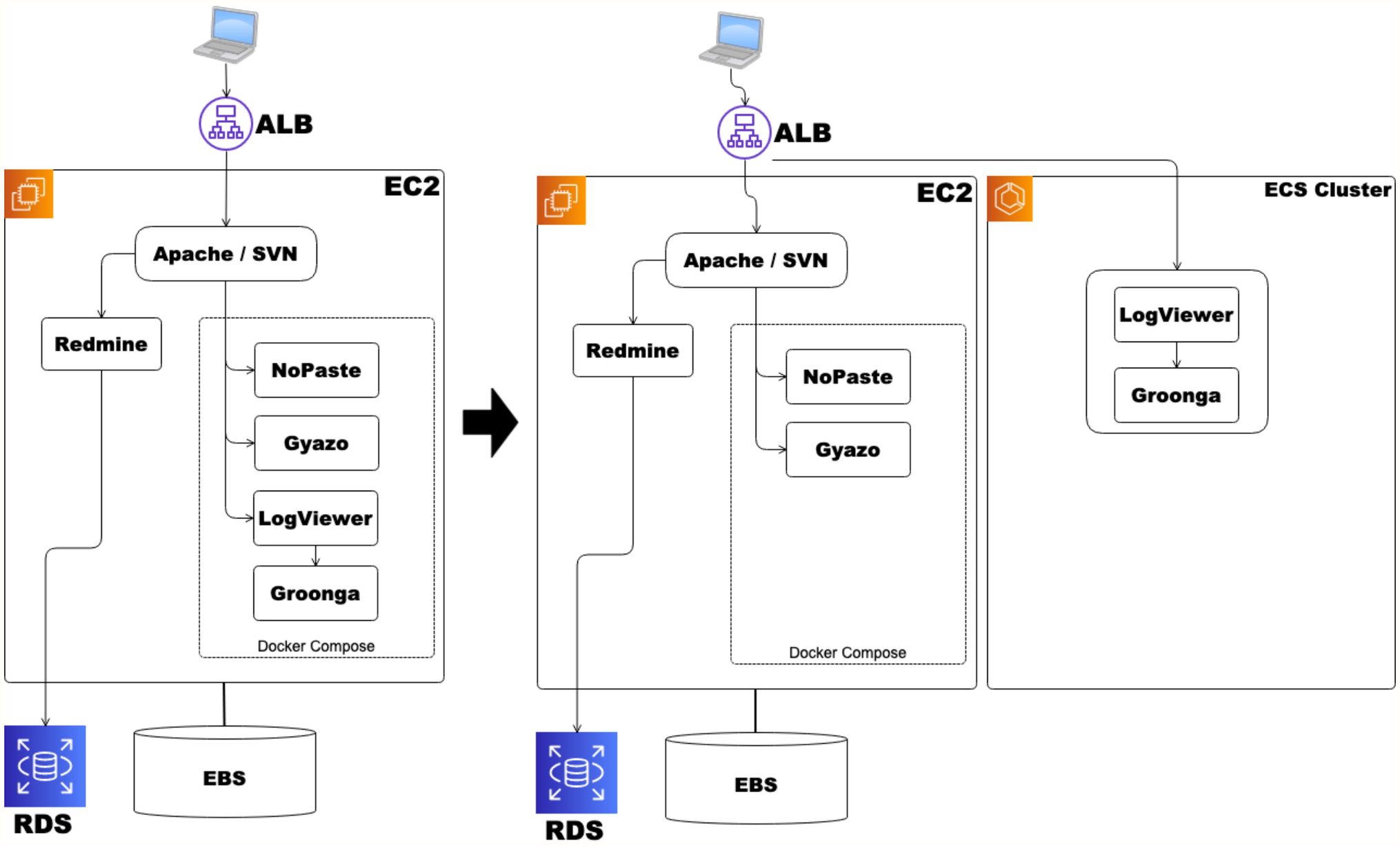
アプリケーションを書き換えられるもの
NoPaste, 社内Gyazoは自作 → **S3 を使うようにコード修正**

もう更新されないけど参照だけしたいもの
IRCの過去ログと全文検索。合計数GB程度

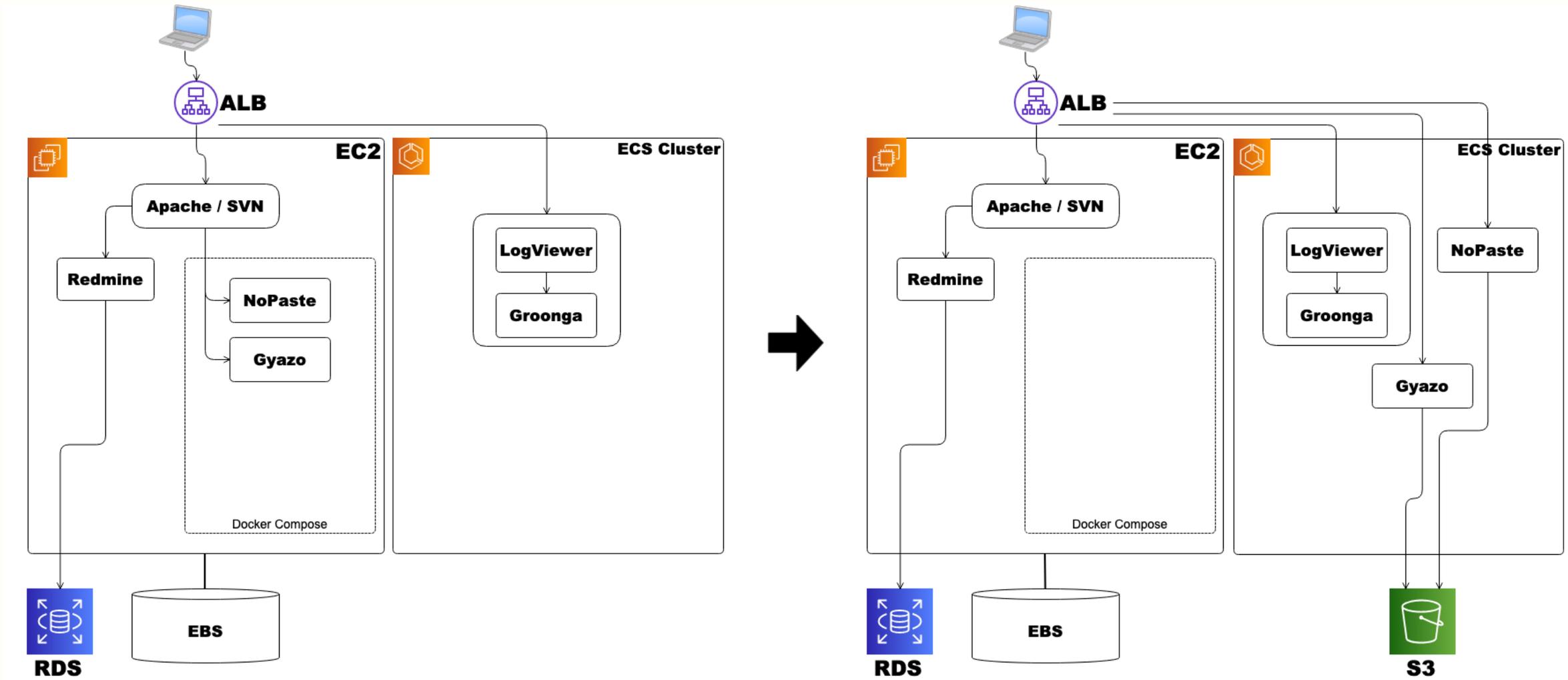
→ **コンテナイメージにデータを焼き込んでしまう**

ストレージが必要ないアプリケーションをECS化

ALB のリスナールールで ECS に振り分ける



ストレージにS3を使用するアプリケーションをECS化



ファイルからS3へ

NoPaste, Gyazo はファイル読み書き部分をS3に変更

Before

POST: ファイルに保存

GET: ファイルから読み出して返す

After

POST: S3に保存

GET: S3に存在したら返す

S3に存在しなかったらファイルにfallback

新規投稿はS3、既存の投稿はファイル参照になるので

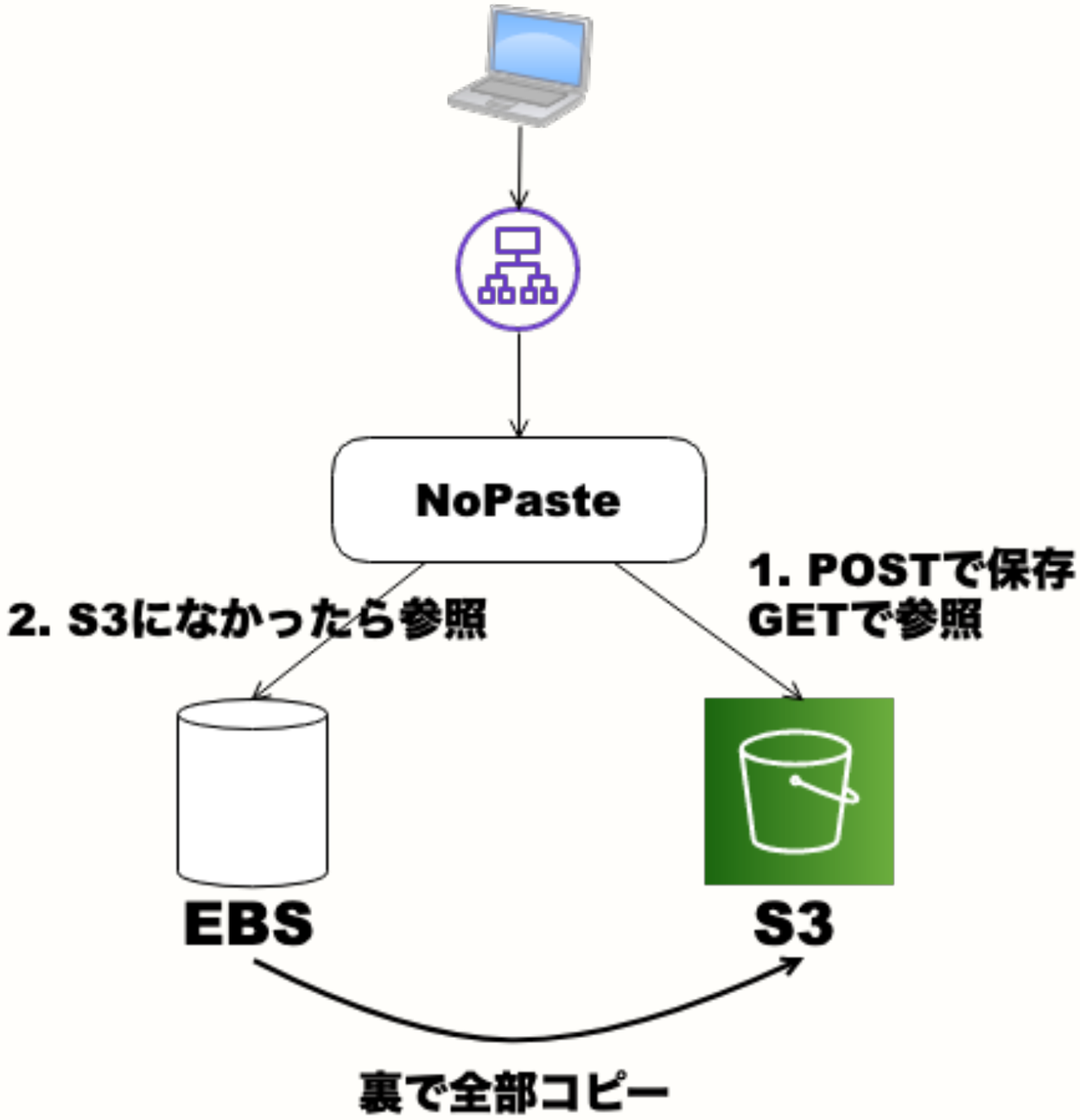
まず EC2 上で稼働中のアプリケーションを差し替える

ファイルからS3へ

新規投稿はS3、既存の投稿はEBS上のファイル参照になっている

この状態でファイルをS3にコピーすれば、全てのリクエストをS3で処理できるようになる

S3へのコピーが終われば、アプリケーションをEC2からECSに移動できる



薄々気づいていたこと

ファイルが1ディレクトリに階層を切らずに保存されている(約100万個)

元々は古いファイルは時限で消す運用だったのでフラットでよかった

→ 不便なので消すのをやめたら... 💥

ファイルがディレクトリに大量にありすぎると

ls ができない

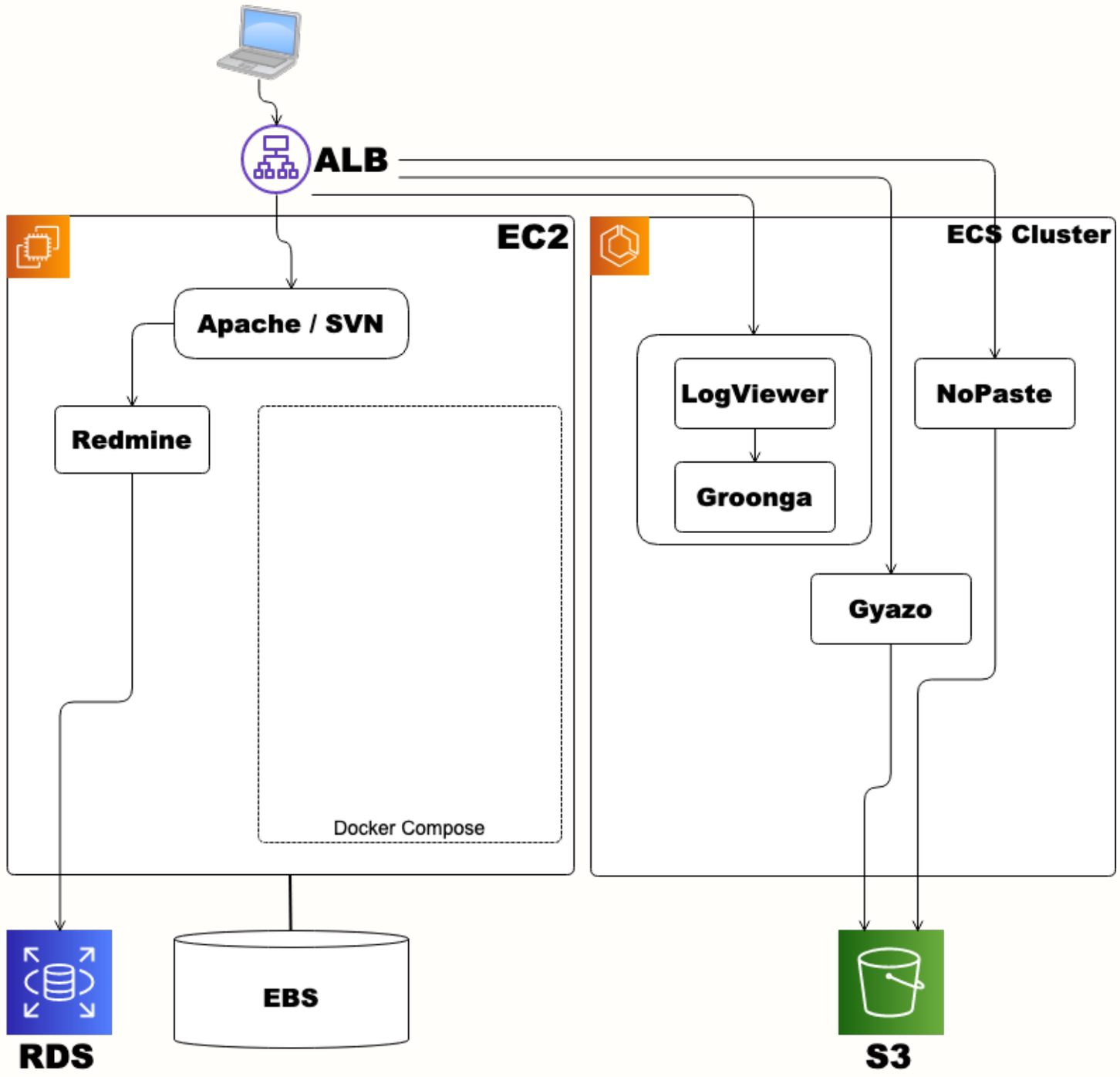
`ls` コマンドはファイルの一覧をソートして返す
全ファイルのメタデータを読み出してからでないと帰ってこない

Protip: find を使う

`find` コマンドはディレクトリエントリを辿って、見つかったものから表示するのでファイル名を順に表示できる

`/21604557d4` → `/21/60/4557d4` のように階層を作ってコピーする Goのツールを書いて S3 へコピー

だんだんEC2が軽くなってきた



ところで… 認証はどうする？

社内ツールなので認証が必須
最近新規に作られたアプリケーションは G Suite のアカウントを使った認証をしている

SVN は htpasswd での BASIC 認証
Redmine は htpasswd を使用する認証プラグイン(自作)
NoPaste, Gyazo の閲覧も BASIC 認証

htpasswd(ファイル) が大統一パスワードデータベース

全社的に G Suite を導入しているので、寄せるならここだが…
Redmine のごく一部を使うだけの人もいるため、全員に G Suite アカウント発行は難しい… 💸

認証をどうする？

社員(G Suiteアカウントを持っている)を前提としてよいところは
ALB の機能で OIDC 認証ができる
→ IRC 過去ログに適用

それ以外は htpasswd を当面使い続けるしかない...

htpasswdファイルを EC2, ECS で同期する仕組みを考える

htpasswdの管理

現状の htpasswd アカウント管理マニュアル(要旨)

1. EC2 に ssh します
2. 作業前にファイルを日付をつけた名前でバックアップします！
   (例) `htpasswd.20190831`
3. アカウント作成 : `htpasswd` コマンドでIDとハッシュ化されたパスワードを追加します！！
   アカウント削除 : `vi` でファイルの当該行を削除します！！！

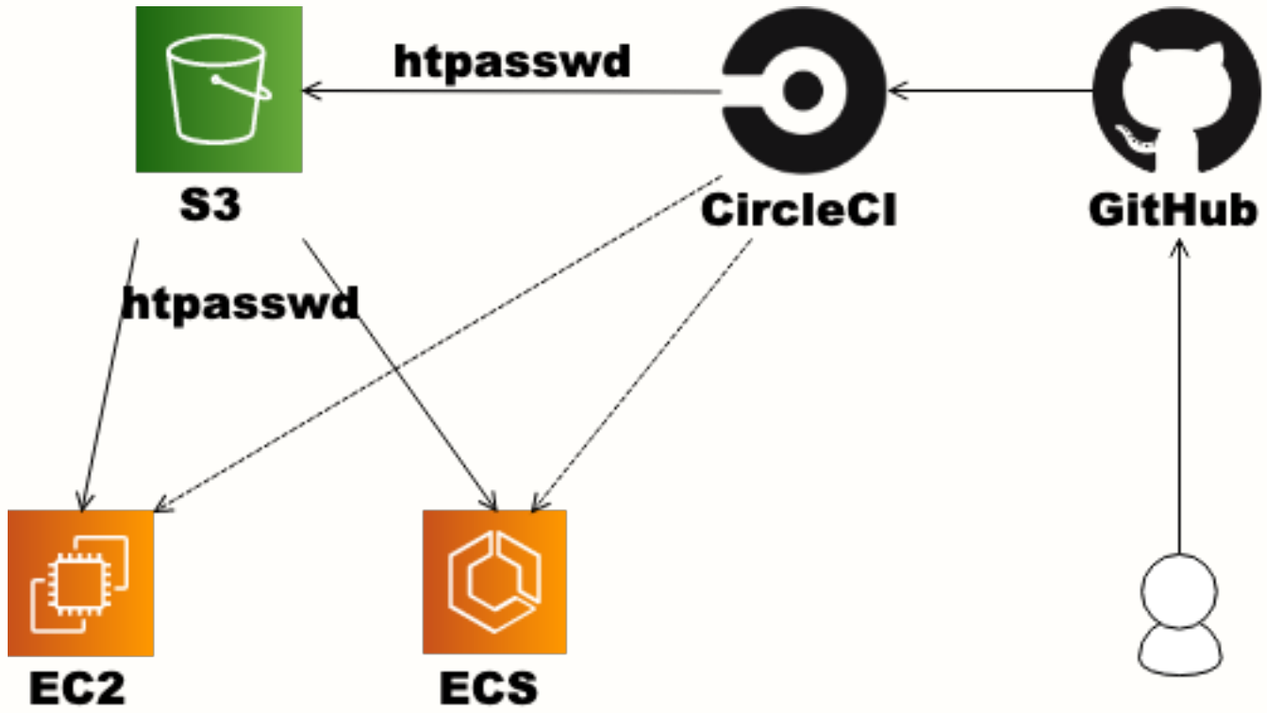
令和にこれはちょっと

せめてリポジトリで履歴管理

1. GitHubのプライベートリポジトリで htpasswd ファイルを管理
2. 追加削除はブランチを切って編集後にコミット
3. 問題なければ master に merge する
4. **EC2 / ECS になんとかして同期する**

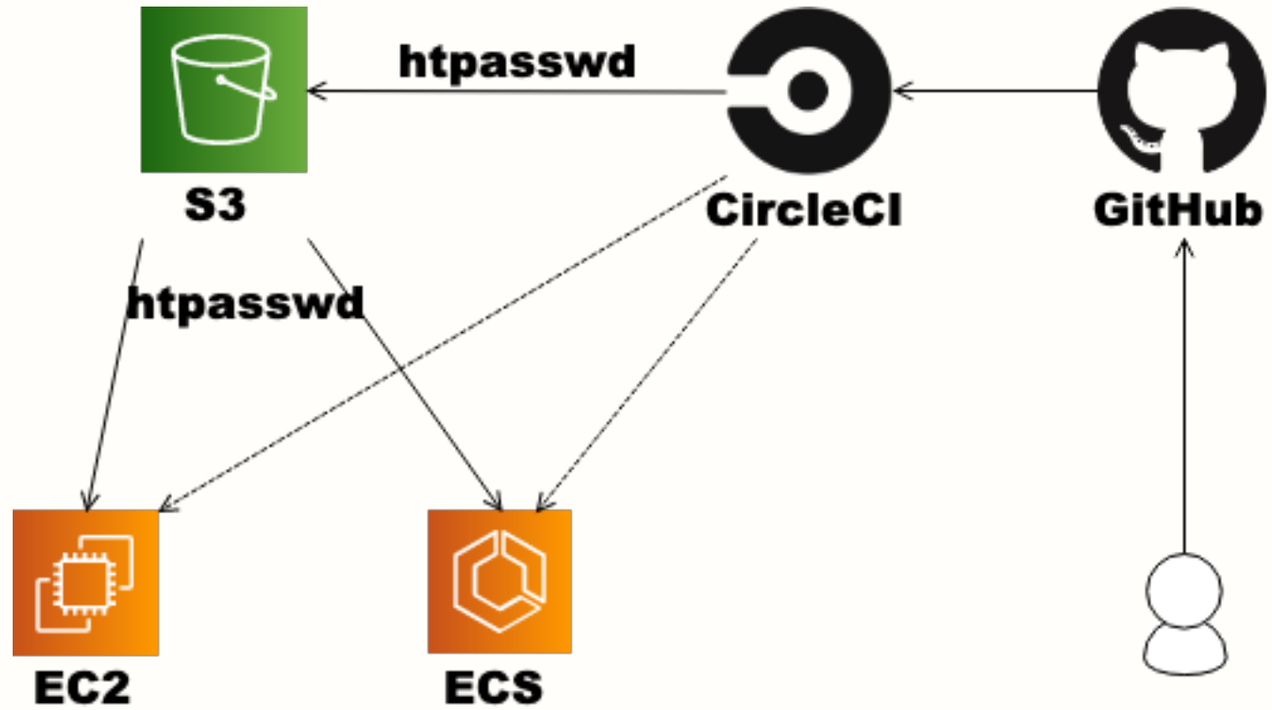
なんとか……?

CircleCI でデプロイ



htpasswd ファイルを保存する S3 bucket を用意

CircleCI で workflow を実行する



1. S3 にファイルをアップロード

2. EC2 に SSM(Systems Manager) run command を発行し
   EC2 上で S3 から取得するコマンドを実行

3. ECS サービスを更新してタスクを入れ換え
   コンテナは起動時に S3 からファイルを取得後、プロセスを起動するように作っておく

[redacted].com APP 10:47

htpasswdのデプロイを開始します(by [redacted])

Project	Job Number
[redacted].com	86

Visit Job

🎉 A deploy_htpasswd job has succeeded!

Project	Job Number
[redacted].com	86

Visit Job

デプロイフロー整備で考えること

実際の作業者に抵抗はないか？

リポジトリで管理して CircleCI からデプロイは普通にやっている

Route53 の DNS管理(Roadworker), IAM ユーザ管理(miam)

自分以外に扱えるか？

CircleCI は他のプロジェクトでも全面的に移行中

リポジトリ内のscriptを叩く処理が順番に書いてあるだけ

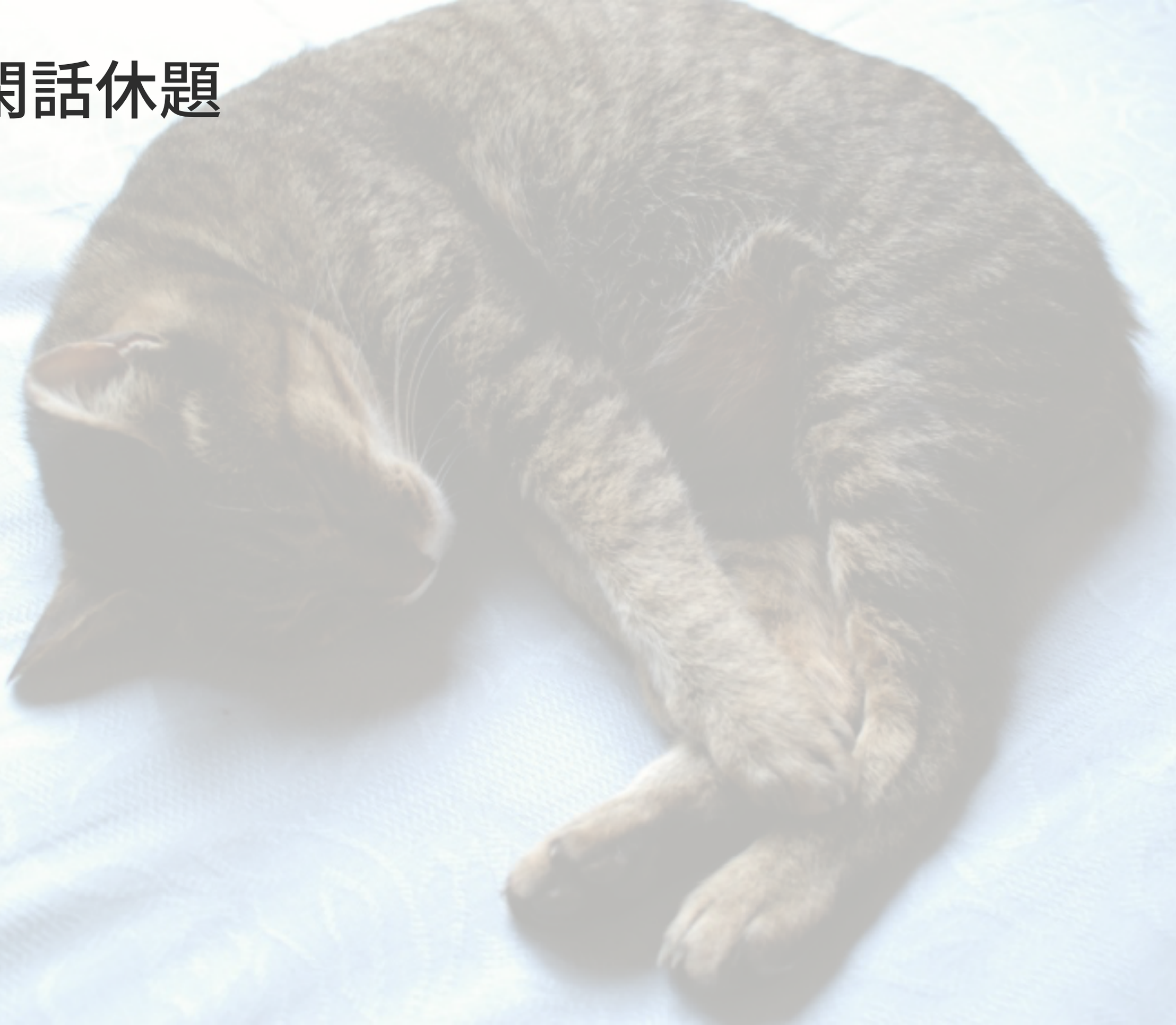
疎結合か？

デプロイ対象側は各自で S3 から取得する pull 型

デプロイ対象が増える → 通知対象が増えるだけ

並列化やスケールが用意

閑話休題



EBSをやめていく

Amazon EFS (Elastic File System)

NFSv4 でアクセスするフルマネージドなネットワークストレージ

同時に複数のEC2からマウント可能
複数 AZ でデータが保持されるので AZ 障害にも強い

どうしてもデータ保存にファイルを使わないといけない、手を入れられないアプリケーションのデータを保存する切り札

EBS → EFS データコピー

EC2 上で EFS をマウント。rsync -a でコピー
ファイル操作のレイテンシが比較的大きいため、rsyncを並列に複数走らせるほうがコピー時間を短縮できる

切り替えは多少のダウンタイムを受け入れれば簡単

1.アプリケーション稼働中にrsyncで初期同期
2.アプリケーション停止
3.rsync で差分を反映
4.EC2 のマウントポイントを EBS から EFS に切り替える
5.アプリケーション起動

EFS のコスト

ストレージタイプ	単価
EFS 標準ストレージ (GB-月)	0.36USD[4]
EFS 低頻度アクセスストレージ (GB-月)	0.054USD
EFS 低頻度アクセスリクエスト (転送 GB あたり)	0.012USD
EBS (GP2) GB-月	0.12USD

仮に1TBを保存すると EBS 13,000円/月 EFS 39,000円/月

[4] すべて東京リージョンの価格

EFS はコストが高い？

標準ストレージのGB単価は EBS の3倍(!!)だが

EBSは確保した容量で課金される
1TBのEBSを用意したら中に何も入れなくても1TB分

EFSは容量を確保する必要がない
実際にファイルを保存した容量のみ

EBSは余裕を見て容量を確保することが多い
使用率 50% で確保すると実際には 1.5倍

EFS 低頻度アクセスストレージ

一定期間ファイルの内容にアクセスされなかったものを自動的に低頻度アクセスストレージに移行できる

標準ストレージより約85%安く、EBS の半額以下

データアクセスで0.012USD/GB課金、レイテンシが大きくなる

ごく一部だけアクティブなリポジトリ、古いものはめったにアクセスされないRedmineの添付ファイルを置くのには最適！

バックアップ

AWS Backup で取れます

EBS も EFS も統一的にスナップショットが取得できる

ファイル操作を誤ってもスナップショットからの復旧が可能

EC2 から EFS を利用する

ホスト上で NFSv4 としてマウントする。amazon-efs-utilsを使えばごく簡単

```
# yum install -y amazon-efs-utils
# mount -t efs fs-12345678:/ /mnt/efs
```

ECS から EFS を利用する

EC2 で EFS マウント ＋ ECS のタスク定義でホストをマウント

直接 ECS タスクから EFS マウントは現時点ではできない

```
{
  "taskDefinition": {
    "volumes": [{
  "host": {"sourcePath": "/mnt/efs"},
  "name": "efs"
}],
"containerDefinitions": [{
  "name": "httpd",
  "mountPoints": [{
    "sourceVolume": "efs",
    "readOnly": false,
    "containerPath": "/efs"
  }
]
```

ECS (Fargate) から EFS を利用する

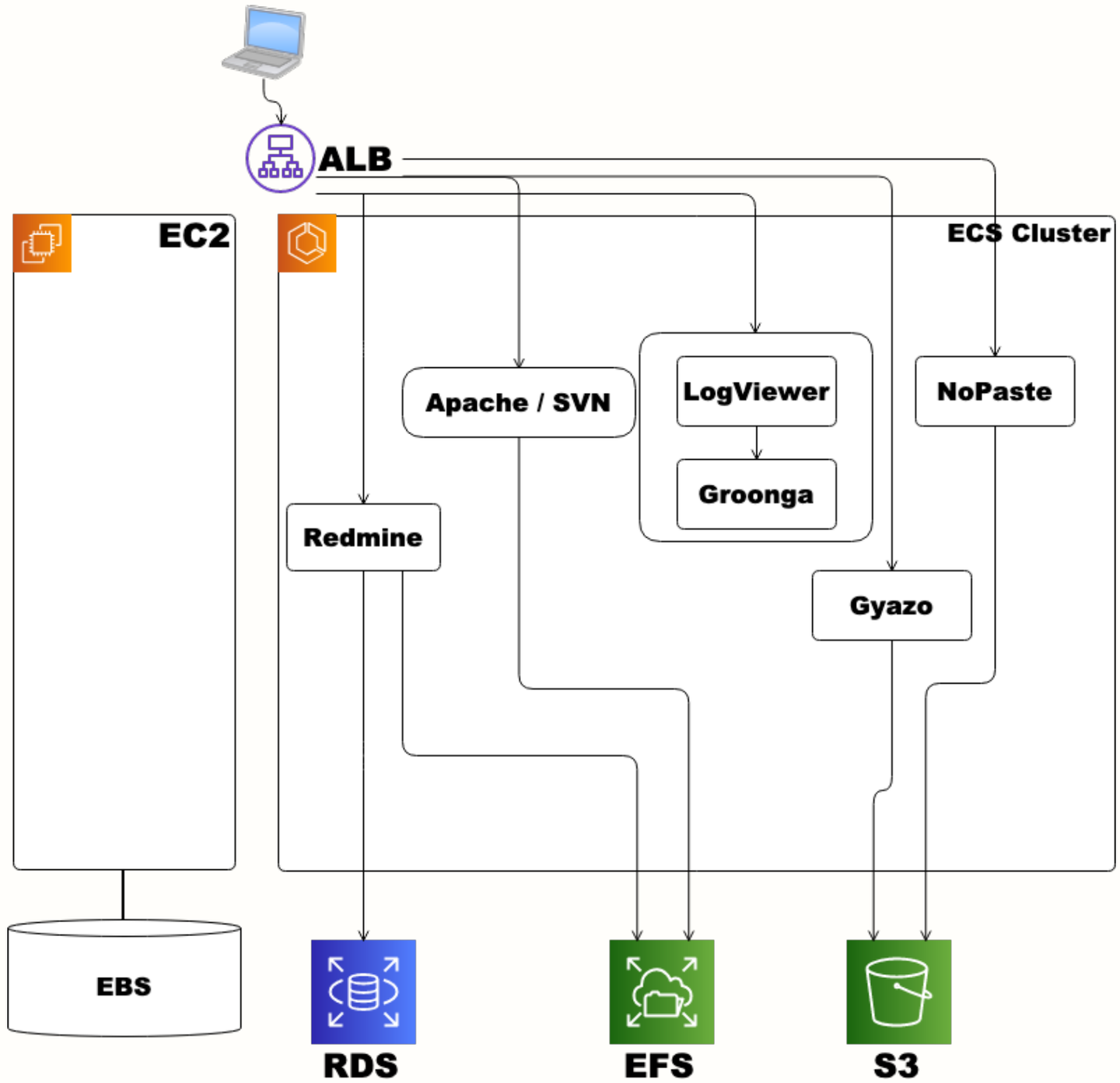
Fargate を利用する場合

EFS は使えません！(2019-08-31現在)

どうしてもEC2インスタンス上でタスクを起動する必要がある

ぜひみなさんでAWSに要望を…

ここまで終わればEC2が空っぽに！(未完)



「2.EC2 シングル構成を脱して堅牢に」達成

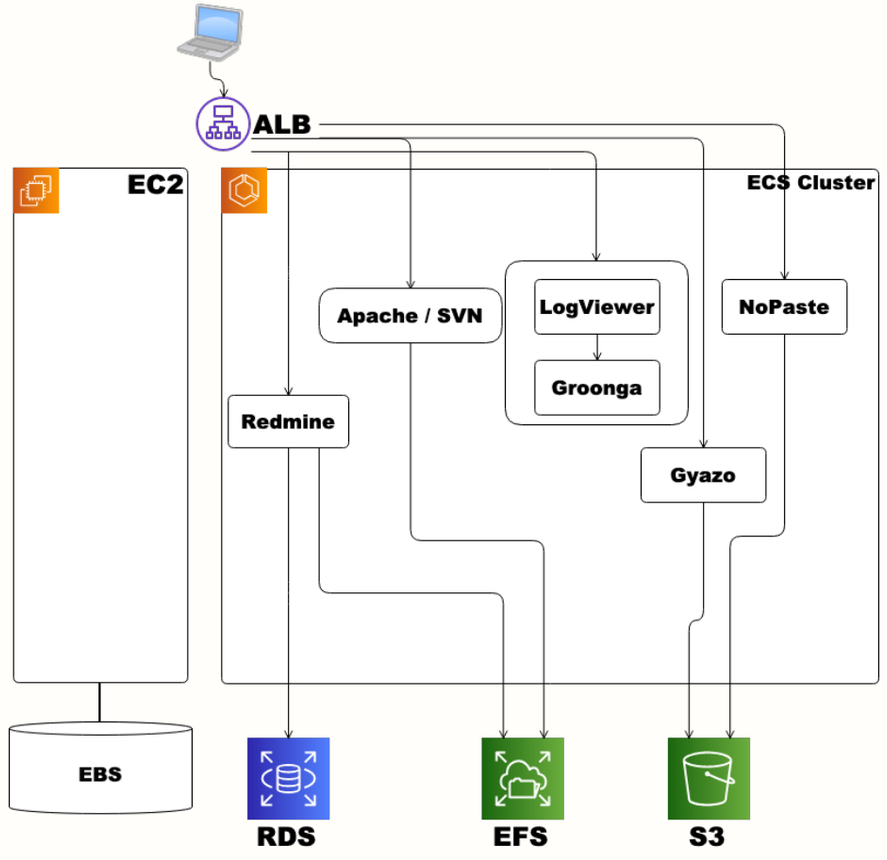
ALB, ECS, RDS, EFS, S3 が Multi-AZ で動作している 🎉

「3.リポジトリで管理されてない部分を極力なくす」達成

htpasswdリポジトリ管理 + ECS + Terraform 🎉

「4.現在のEC2をなくす」達成

EC2は空っぽなので止められる 🎉



まとめ

EC2 シングル構成、歴史が詰まったレガシーなサーバーを
マネージドサービスに分解して再構築しています

外界との接点、データ保持はマネージドサービスで
(ELB, RDS, S3, EFS...)

定期的にシステムと業務を見直し
維持するもの、切り捨てるものを決めるのは大事

メンテできる仕組みで作り直すのも大事