

LANGFUSE

Langfuse v3 使ってますか？

2025

Langfuse
Night #1

YU OTSUBO

.....

ABOUT ME

Yu Otsubo

ゆるふわエンジニア



KDDIアジャイル開発センター所属

生成AIを活用したプロジェクトでゆるふわエンジニア

妻と犬と細々暮らしている

ちなみに、志田未来さんと神木隆之介さん
それぞれが生まれた誕生日の間に生まれている



と ころ で

.....

Langfuse v3

使っている？

.....

Langfuse Cloud版

使 っ て い る 人

挙手お願いします🙋🙋

.....

Langfuseセルフホステッド版

使っている人

挙手お願いします🙋🙋

.....

Langfuseセルフホステッド版でV3

使っている人

挙手お願いします🙋🙋

.....

December 9, 2024

Langfuse v3 stable release



Max Deichmann

Releases 273



v3.0.0

Latest

today

Langfuse v3

STABLE RELEASE ON 12/9/2024.

.....

大幅にアーキテクチャが変わったことで話題でした。
マイグレーションにチャレンジした方も多いのではないのでしょうか？

V3 どんなイメージですか...?

やたらコンポーネント多いな...

コンテナ増えた...

Clickhouseってなに...?

開発者にメリットあるの...?



今日のLTを見て

す ぐ に ア ッ プ デ ー ト

し ま し ょ う ~

.....

その前にLangfuse V3の特徴

アップデートでどんなことが
変わったのでしょうか？

Async Worker

Webサーバーの
重たい処理を
非同期で処理



OLAP(Clickhouse)

分散トレーシングという
重たい分析
ワークロードを解決



Queuing

Async Workerと
Webサーバーの間を
安全に受け持つ



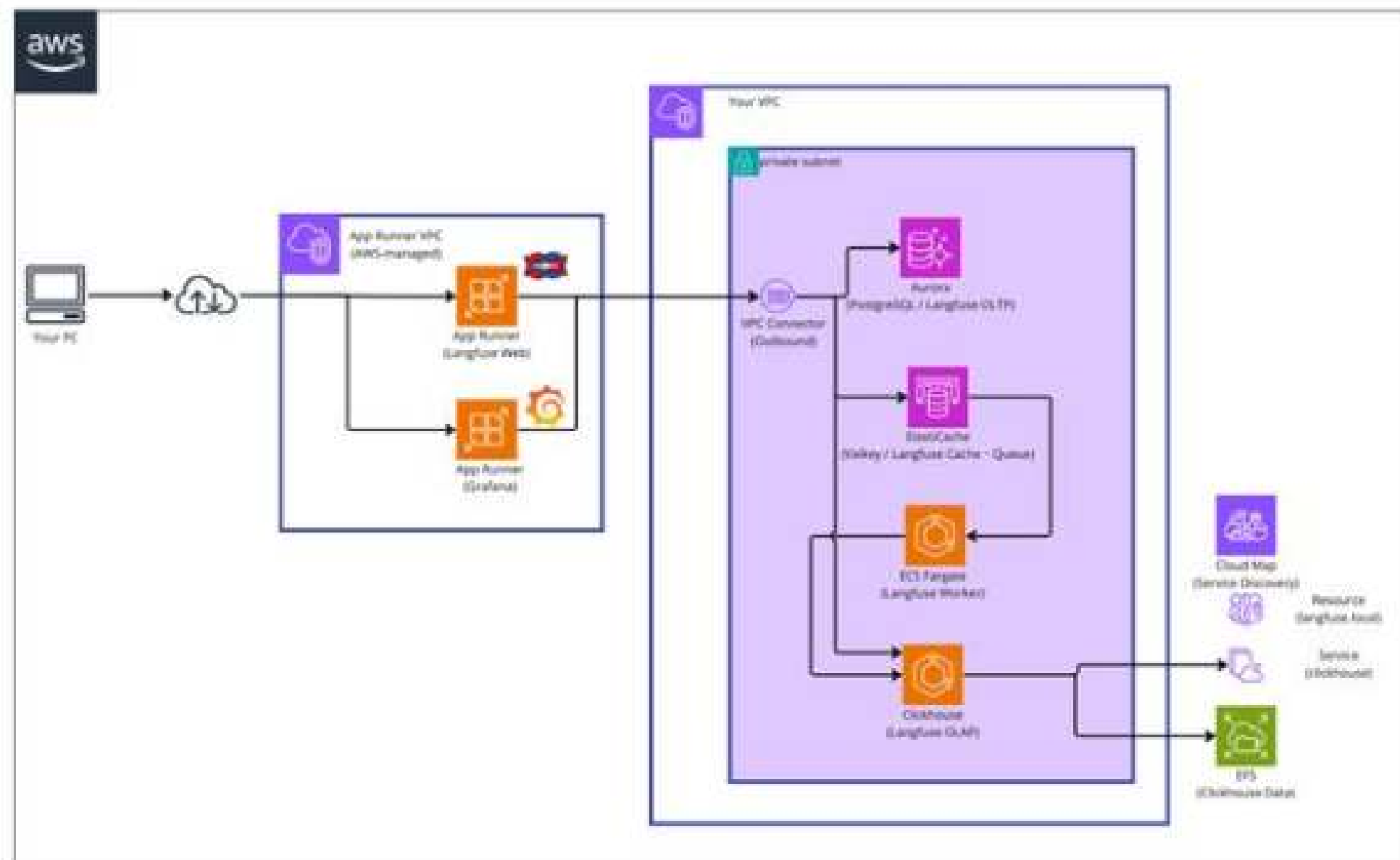
その前にLangfuse V3の特徴

アップデートでどんなことが
変わったのでしょうか？

詳しくはBlogを
読んでね！

Webサ
全に受け





Langfuse v3はv2からどのように変わったのかを噛み締めながらAWSマネージドサービスでLangfuse v3を作りきる

AWSはマネージドサービスがやっぱりいいですね。 Table of Contents Langfuse v3がついにGAしました だいぶ長い記事なので先に結論 Langfuseとは 我々の Langfuse v2アーキテクチャ Langfuse v...

今日はどのように
V2→V3マイグレーション
を実施するか実践的な話
をします

.....

まずはじめに

我々のV2構成はこれだ！

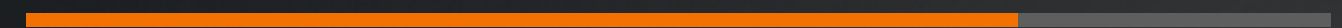
.....

AWS App RunnerとAurora構成
LangfuseはNext.jsで動いているので
App Runnerでリクエストベースでの
課金可以实现できます。

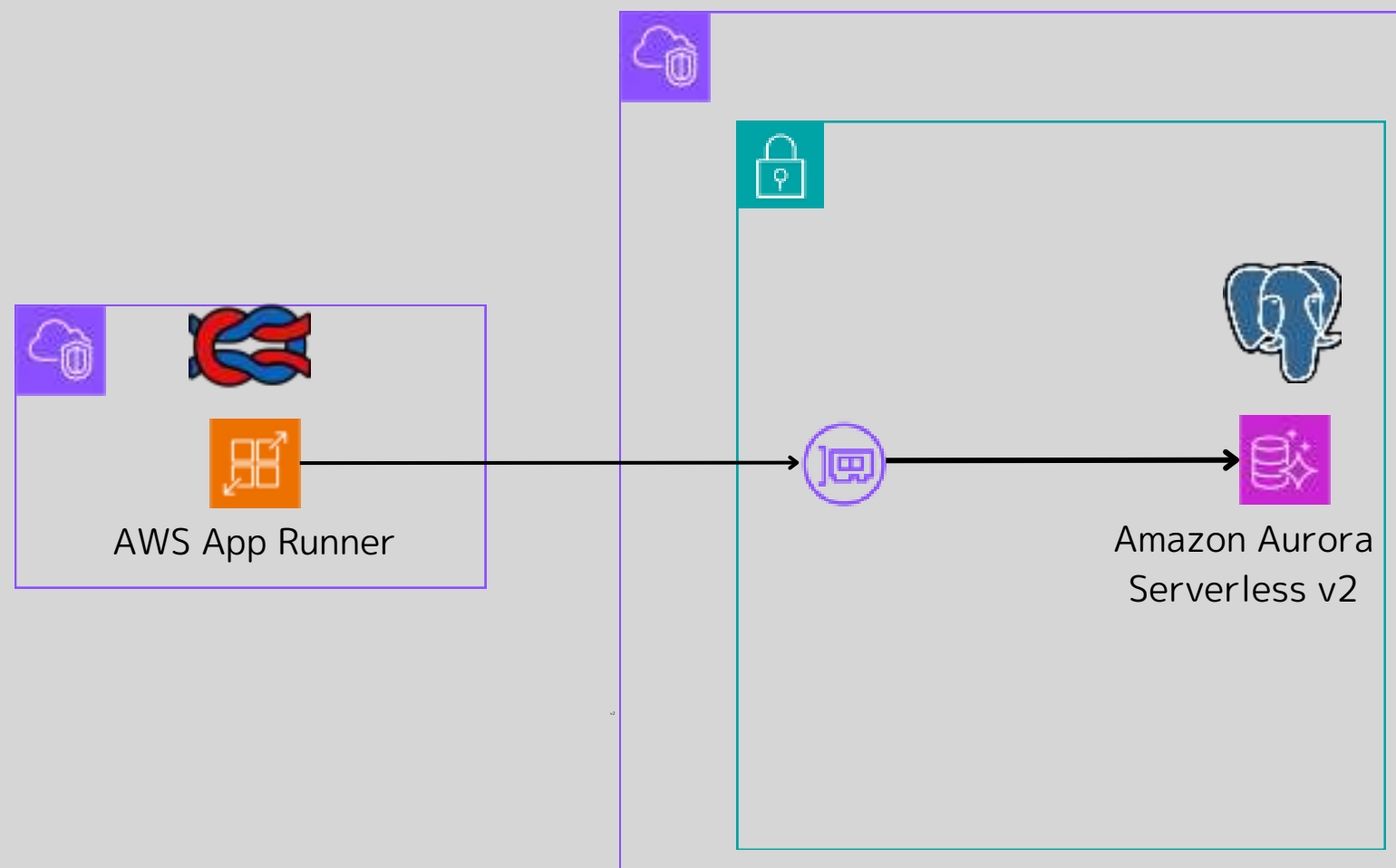
シンプルさ



可用性



インフラコスト



v3に移行するに あたって

右の4つを重要項目として
V3切り替えを目指します。

会社のMission/Vision/Value
みたい...



Keep Data

データは今運用しているものを
消さずに



Minimum Downtime

できるだけダウンタイムは少なくしたい



Reduce the Cost

V3になったからと言って大幅にコスト
アップは許されない



Easy Deploy

面倒な手順書は作成したくない
IaCでパッと華麗に

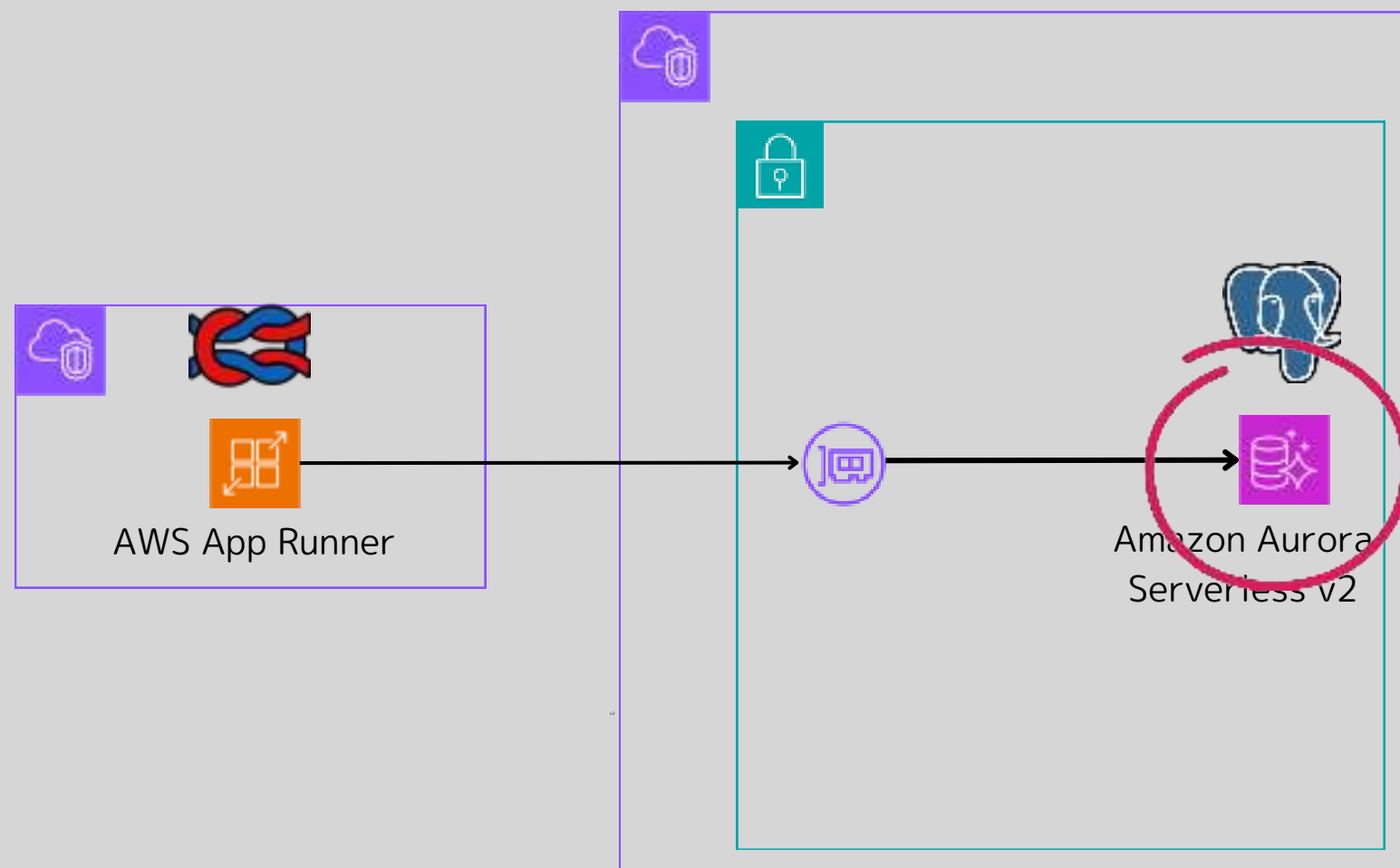
Keep Data

PostgreSQLのインスタンスは
そのままに

.....

現行動いている構成のAuroraは消したり、
移行したりせずそのまま放置する。

念の為作業前にスナップショットを取って
おく。(保険は重要)



Comparing changes

Choose two branches to see what's changed or to star



base: v2.93.4 ▾



compare: v3.0.0 ▾

```
1 + -- AlterTable
2 + ALTER TABLE "projects" ADD COLUMN "deleted_at" TIMESTAMPT(3);
```

```
1 + -- DropForeignKey
2 + ALTER TABLE "job_executions" DROP CONSTRAINT "job_executions_job_output_score_id_fkey";
3 +
4 + -- DropForeignKey
5 + ALTER TABLE "traces" DROP CONSTRAINT "traces_session_id_project_id_fkey";
```

Keep Data

DBマイグレーションに注意

.....

万が一のv3→v2切り戻しに備えて、バージョンアップで走るPostgreSQLに対してのPrisma Migrationについて注意。

down.sqlはLangfuseプロジェクトで未作成なので、万が一のときは手動でダウンマイグレーションを実行する必要がある。

RDS Data API 情報

RDS Data API を有効にする

Data API のステータス



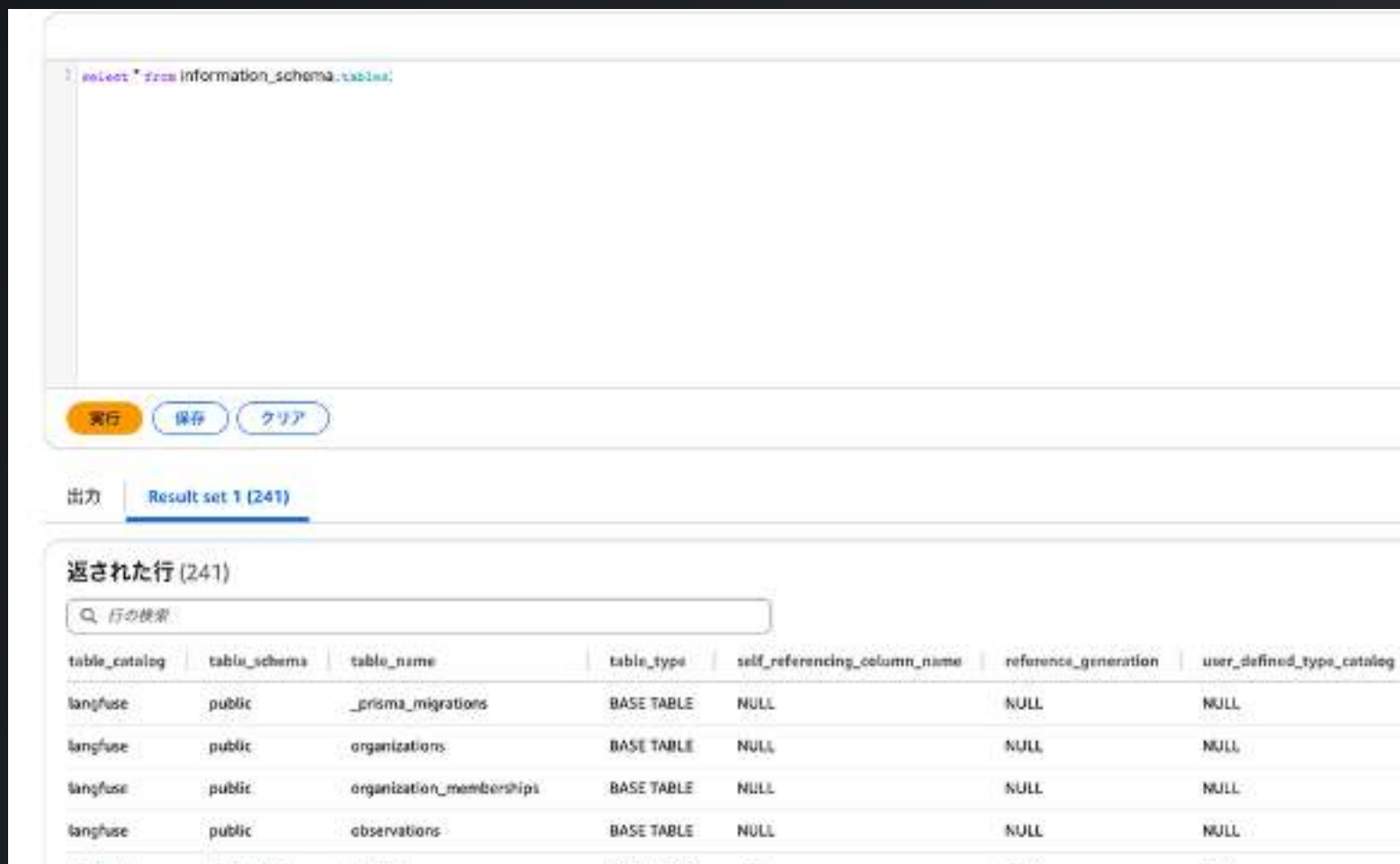
Keep Data

Auroraのクエリエディタを有効化しておく

.....

あまりおすすめされないが、万が一のときはPostgreSQLのテーブルに対して操作が必要なため

- ダウンマイグレーションの際
- Clickhouseへのバックグラウンドマイグレーションを強制実行したい場合



The screenshot shows the AWS Aurora Query Editor interface. At the top, there is a SQL query editor with the text: `select * from information_schema.tables;`. Below the editor are buttons for '実行' (Execute), '保存' (Save), and 'クリア' (Clear). The '出力' (Output) section shows 'Result set 1 (241)'. Below this, a table titled '返された行 (241)' (Returned rows (241)) displays the results of the query. The table has columns: table_catalog, table_schema, table_name, table_type, self_referencing_column_name, reference_generation, and user_defined_type_catalog. The first four rows are visible, all from the 'langfuse' catalog and 'public' schema.

table_catalog	table_schema	table_name	table_type	self_referencing_column_name	reference_generation	user_defined_type_catalog
langfuse	public	_prisma_migrations	BASE TABLE	NULL	NULL	NULL
langfuse	public	organizations	BASE TABLE	NULL	NULL	NULL
langfuse	public	organization_memberships	BASE TABLE	NULL	NULL	NULL
langfuse	public	observations	BASE TABLE	NULL	NULL	NULL

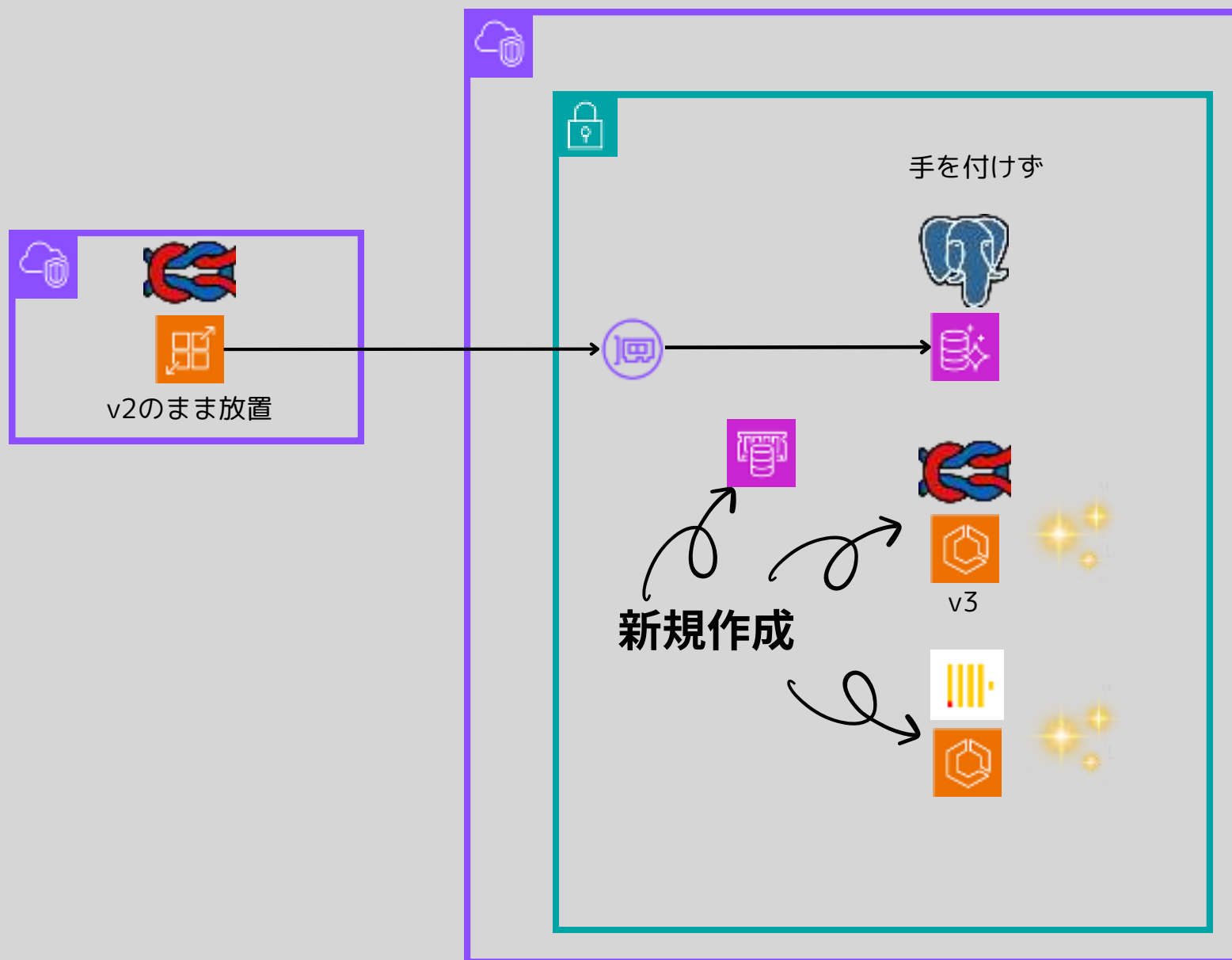
Minimum Downtime

Webサーバー以外を
すべて新規作成し、
最後にv3に上げる

.....

Async worker単体で動くことはないので
(Queueにジョブ登録されるまで動かない)
一気に裏側を作成し、
最後にWeb Serverのv3バージョンアップ
を実施しv3に変更する。

App Runnerの自動デプロイを賢く使って
公式のMigration Guideを少しでも簡易化
させた方法です。





Minimum Downtime

それでも不安なら
夜間メンテにしてしまう

.....

不測のことはつきものなのでアプリケーション全体をメンテナンスにしてしまう。

ビジネス部門との調整に難航する場合は、v3化によるスケーラビリティを説明するとよい。

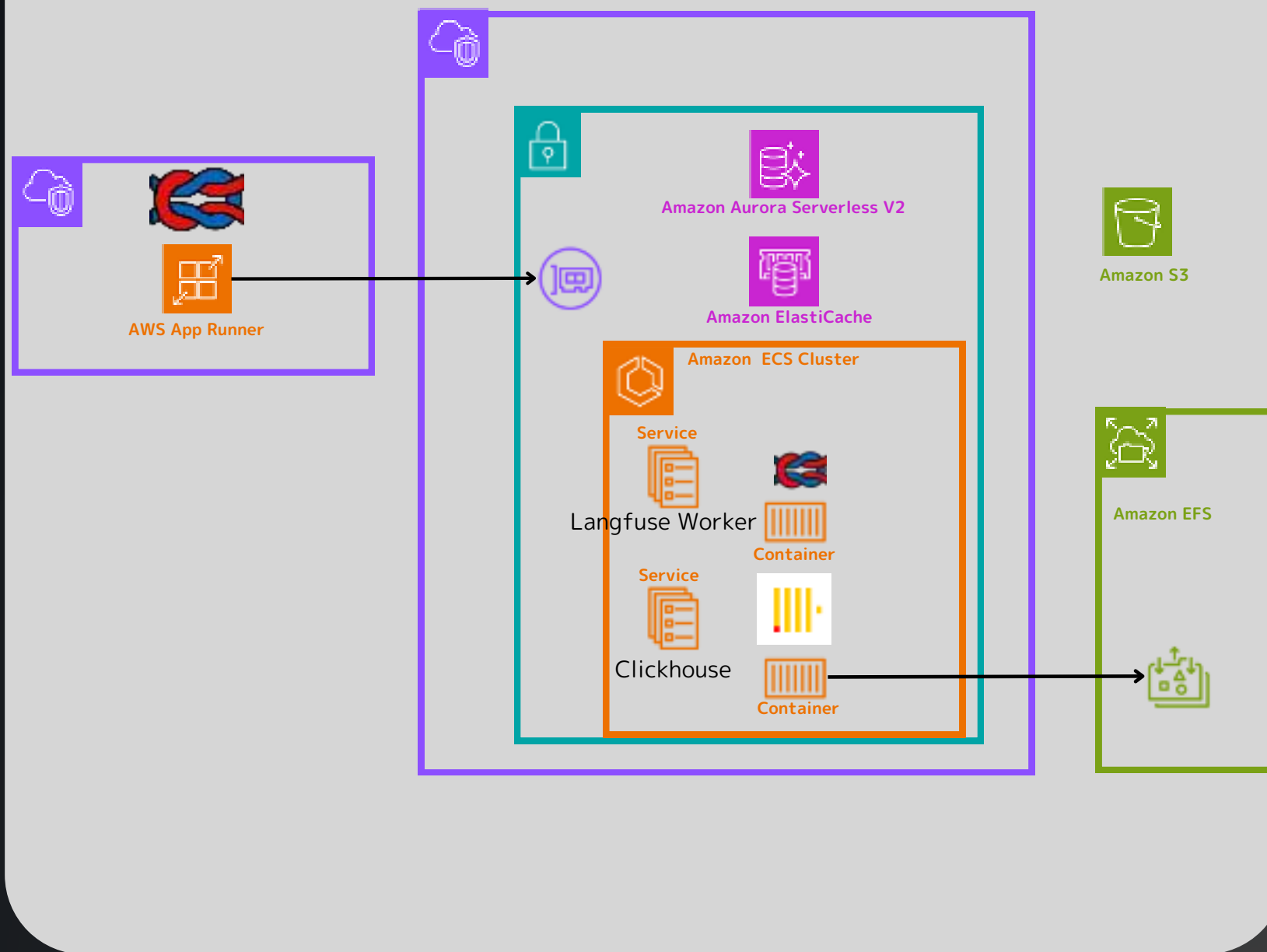
Reduce the Cost

可能な限りマネージドに

.....

単純なAWS費用だけでなく、
運用コスト含めて考えるとできるだけマネージドな
サービスを活用するとよい。

- Queue → ElastiCache(Redis)
- Async Worker / Clickhouse → ECS (Fargate)
- Clickhouseの永続化 → EFS
- Blob Storage → S3



Reduce the Cost

Fargate SpotやValkey、EFS one-zoneのことも考えてみる

.....

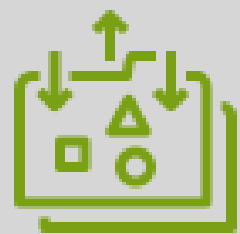
さらにコストを抑えたい場合、

- Fargate SpotでAsync Workerを実行する(*1)
- Redisの代わりにValkeyを使う
- EFSのone-zoneにする(*2)
- (開発環境)Scheduleで落とす

などが検討できる

*1: 最悪RedisにQueueが貯まるので、Langfuse画面上に反映が遅延することもある

*2: ClickhouseのコンテナのAZと揃える必要がある。揃えないで立ち上げるとProvisioningで落ちる



V3 お得構成

V2

約
\$60

App Runner
(1 vCPU & 2 GB)
で日勤帯のみアプリ利用想定

Aurora Serveless v2
(0.5ACUで稼働想定)

約
\$100

v2構成に加えて
ElastiCache
(Valkey cache.t2.micro)
EFS One Zone
Fargate Spot2台

V3 ちゃんとした構成

約
\$196

Fargate Spot→Fargate

EFSをStandardに

ちゃんと作るならこっち

どれくらいコスト変わるのか

v2と比べ+\$40は痛いですが、
抑えようと思えば\$100くらい
は圧縮できる

Easy Deploy

すべてはTerraform applyで

.....

ここまで説明してきたことを、
Terraformで一発で適用できたら嬉しい。

そんな願いを叶えるモジュールがある。

tubone24/langfuse-v3- terraform



This Terraform module provides infrastructure components for deploying Langfuse v3 self-hosted on Amazon Web Service(AWS).

3 Contributors 1 Issue 12 Stars 3 Forks



tubone24/langfuse-v3-terraform: This Terraform module provides infrastructure components for deploying...

This Terraform module provides infrastructure components for deploying Langfuse v3 self-hosted on Amazon Web Service(AWS). -

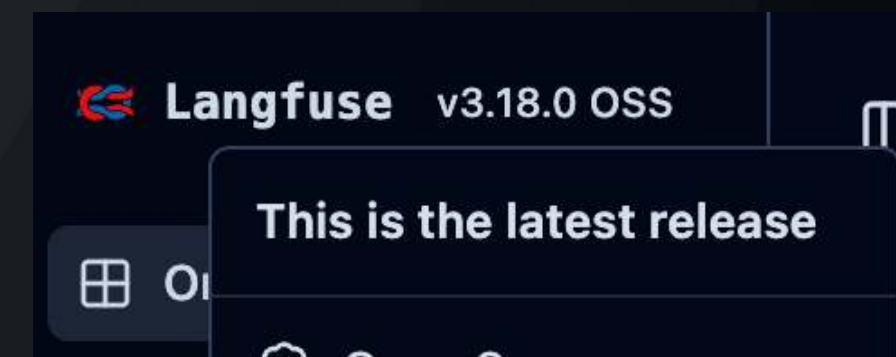
tubone24/langfuse-v3-terraform

GitHub

無事に 移行できました

なのでご安心を

.....



どんなイメージですか...?

やたらコンポーネント多いな...

コンテナ増えた...

Clickhouseってなに...?

開発者にメリットあるの...?



どんなイメージですか...?

AWSで簡単に作れるなあ..!

マネージドで安心...!

コストも工夫次第でなんとかなりそう

移行も簡単だぞ!



最後に

Langfuseセルフホステッド版 と向き合うには

OSSという特性を活かして
Just Read it!!!

OSSのコードを読むことでトラブルシュートもできるし、
趣深い改修が入っていることもわかる。

コードを読もう！

最高のドキュメントはコードです！（持論）

でも一人で読むのはツライので
皆さん情報共有しましょう！タスケテ.....



THANK YOU

FOR WATCHING