CTC Engineers Voyage #1

CTC金融エンジニアのAWS LTチャレンジ

“脱AWS CodeCommit“
セルフマネージドGitLabへの考察

伊藤忠テクノソリューションズ株式会社

金融ＮＥＸＴ企画部

川島　耕二郎

無限の未来と、幾千のテクノロジーをつなぐ。

CTC Financial Services Group

自己紹介

伊藤忠テクノソリューションズ株式会社

金融ＮＥＸＴ営業本部／金融ＮＥＸＴ企画部

リードスペシャリスト

川島　耕二郎

最近愛兎を失い、ペットロスを実践

PagerDutyのPageyくんに話しかけている

ペーちゃん

無限の未来と、幾千のテクノロジーをつなぐ。

CTC Financial Services Group

今日お話しすること

- AWSにおいてセルフマネージドGitLab Community Editionを使ったGit構築方法
- マルチアカウントでパイプラインを構築する場合の接続方法

※全般的にダッシュでご説明するので詳細は後日展開される資料をご確認ください

今日お話ししないこと

- 各AWSサービスの詳細な構築手順（10分でお話しできず）

Code兄弟、グループ解散へ

CodeCommitの新規提供終了へ

Amazon Web Services ブログ

AWS CodeCommit リポジトリを他の Git プロバイダーに移行する方法

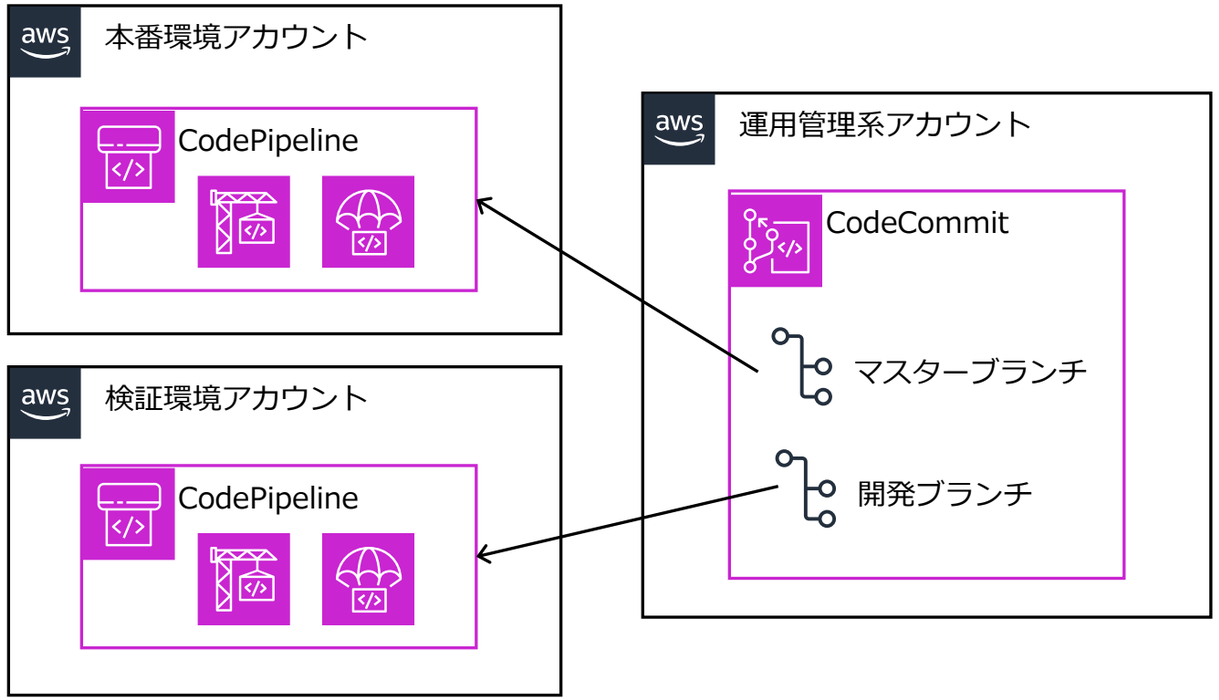
by Masahiro Matsumoto | on 26 7月 2024 | in Advanced (300), AWS CodeCommit, Developer Tools, General, Technical How-to | Permalink | Share

本記事は 2024 年 7 月 31 日時点のブログ How to migrate your AWS CodeCommit repository to another Git provider を翻訳したものです。

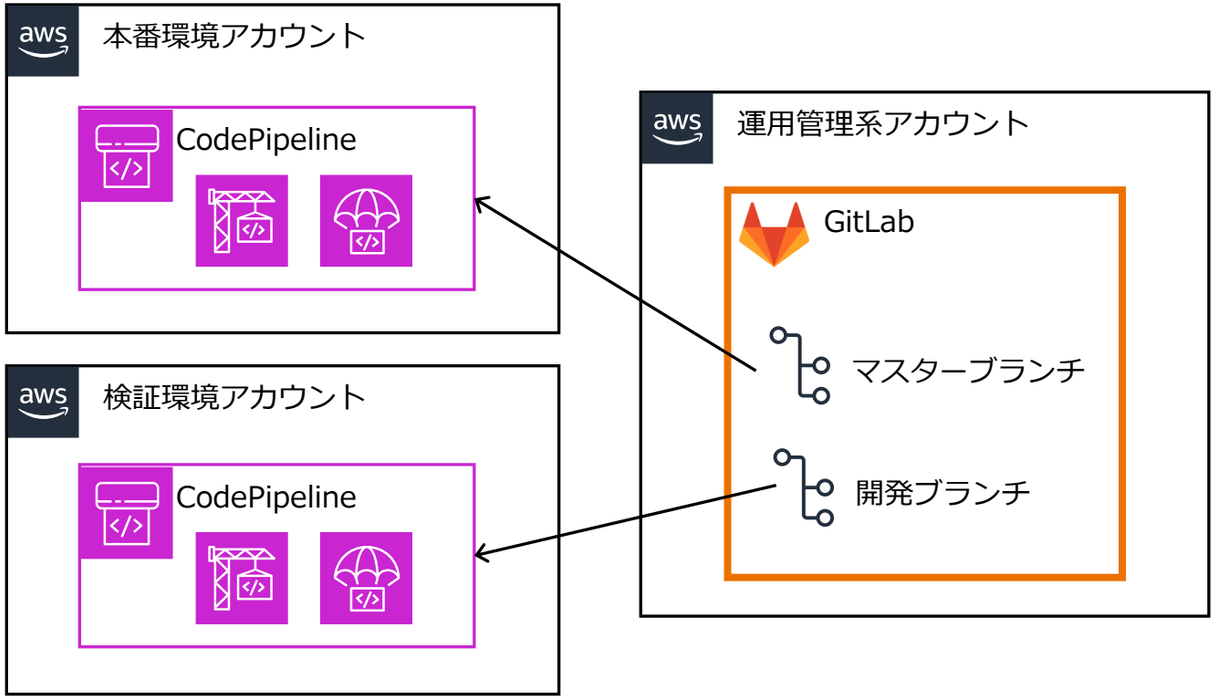
慎重に検討を重ねた結果、 *2024* 年 *7* 月 *25* 日をもちまして、 *AWS CodeCommit* について、新規のお客様向けのアクセスを閉じることを決定いたしました。 *AWS CodeCommit* を既にお使いのお客様は、これまで通りサービスをご利用いただくことが可能です。 *AWS* は *AWS CodeCommit* のセキュリティ、可用性、パフォーマンスの改善に引き続き投資を行ってまいりますが、新機能の導入は予定しておりません。

出典：https://aws.amazon.com/jp/blogs/news/how-to-migrate-your-aws-codecommit-repository-to-another-git-provider/

Code兄弟を活用されている方の構成イメージ



低コストで利用できるCode兄弟を活用して、各ブランチ戦略にのっとりパイプラインを構築されている方は多いかと思います。ただ、このように疎結合になっていればCodeCommitを他のGitに切り替えることは現行運用を維持したまま行えるのではないかと考えます。



先ほどのブログにもGitHubやGitLabへの移行について記載もあり、SaaSへの切替も選択肢としてありますが、金融系のお客さまを考慮して「セルフマネージド（オンプレ型）」が存在するGitLabを選択してみました。今回は、無償のCommunity Editionで構築します。

セルフマネージドGitLabの構築手段

セルフマネージドのGitLabにはおもに以下の構築手段があります。

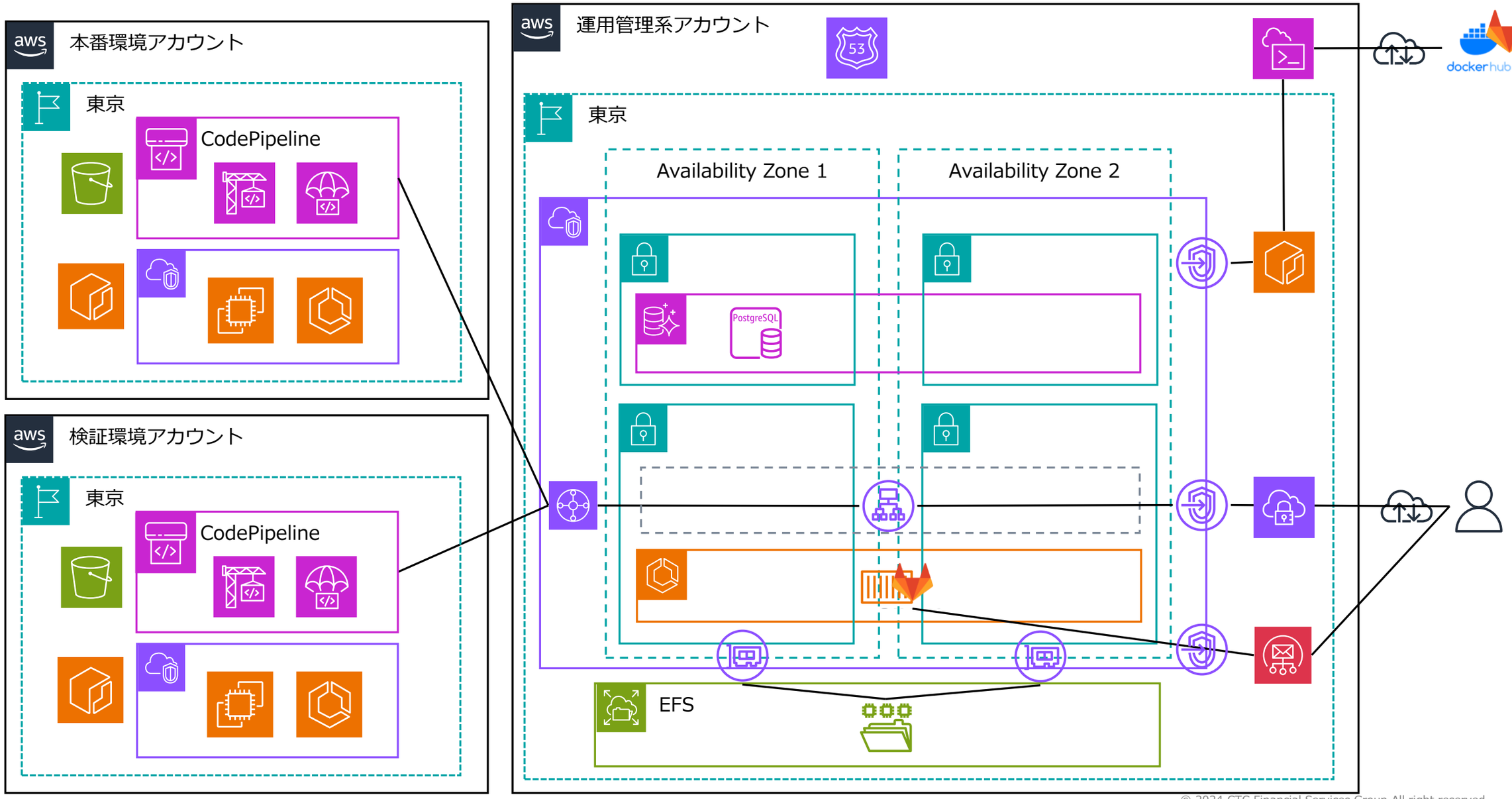
- Linux package
- Helm chart（Kubernetes）
- GitLab Operator（Kubernetes）
- Docker
- Self-complied

出典：https://docs.gitlab.com/ee/install/install_methods.html

EC2のパッチ管理はしたくないし、EKSの管理もこれだけのためにやりたくない。

ということで、今回は「**Docker**」を選択し、**ECS(Fargate)**で構築します。

ゴールイメージ



ゴールイメージのポイント

- GitLabのECSはプライベートサブネットに配置する
- そのため、ECR・S3・SES等VPC外にあるサービスとエンドポイントで接続する
- GitLabのストレージはEFS、DBはAurora Postgresに外出しする
- GitLabからデプロイする先は検証・本番等の異なるアカウントであることを想定し、当アカウントも独立させ、Transit Gatewayで各アカウントと接続する
- パイプラインはAWSのメリットを最大限享受するためCodePipelineを採用し、GitLabに対してCodeConnectionsで接続する
- CodeConnectionsはHTTPS接続が必要なため、ALBをTLSの終端とする

構築要素のご紹介

GitLabイメージの取得

GitLabのイメージはDocker Hubに公開されています。

これをECRに登録します。

ECSの構築

GitLabをECS(Fargate)のサービスとして構築します。

GitLabはRedis/PosgreSQLなどをコンテナ内に内包していますがDBとストレージを外出しして永続化します。

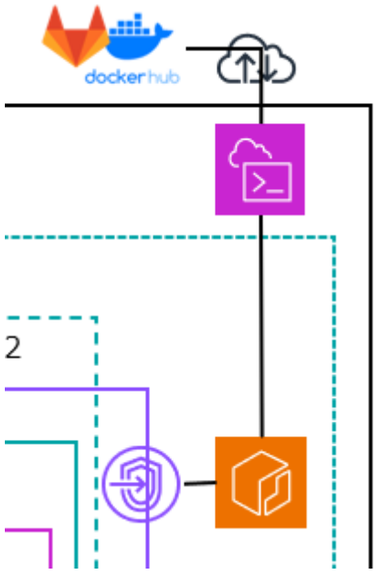
GitLabの設定

GitLabのコンフィグを編集し、Auroraなど外出しした定義を設定します。

CodePipelineとの接続

GitLabとCodePipelineを接続し、CI/CD環境を構築します。

①Dockerイメージの ECRへの格納



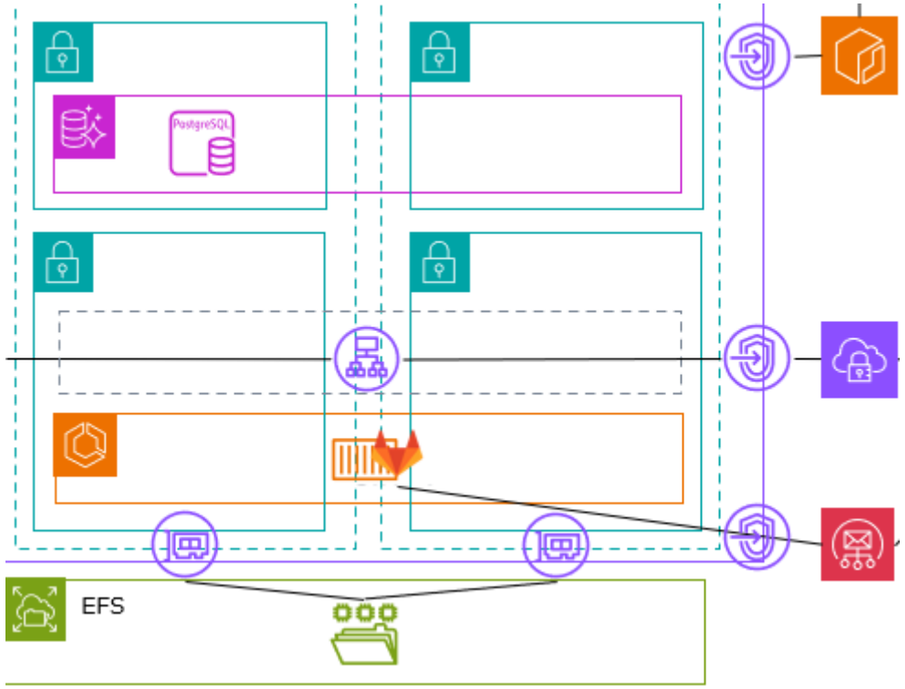
セルフマネージドのGitLabのパブリックイメージはDocker Hubにあります。
Docker HubからECRへイメージを連携するには以下のパターンがあります。

- ECRのプルスルーキャッシュを使ってDocker Hubから直接キャッシュする
- 同じ環境にあるEC2やCloud9、CloudShellにプルしてバージョン管理する

常にlatestでECSタスクを再起動したら最新を取り込むポリシーでいくのであれば前者、ECRのリポジトリをイミュータブル(タグの上書き不可)の設定にして、タグをきちんと管理して適用していくのであれば後者かと思います。
イミュータブルにしないとSecurity Hubに指摘はされます(※)。
リポジトリのライフサイクルポリシーもちゃんと設定して何世代前まで管理するかも決めておく必要があります。

※Security Hubのコントロール：https://docs.aws.amazon.com/ja_jp/securityhub/latest/userguide/ecr-controls.html#ecr-2

②ECSの構築



ECSのタスク定義において、コンテナのサイズは以下URLに記載がありますので参考にしてください。(※1)。

GitLabのコンテナイメージ内にはRedisもPostgreSQLも入っています。ローカルのストレージにも設定ファイルを持っていますが、ご存じの通り、ECSを再起動したら消えてしまいますのでストレージをEFSに、内部のPostgreSQLをAurora等外部のDBに変更してあげる必要があります。
但し、EFSの使用はGitLabのマニュアル上は推奨されていませんので使用時は吟味してください(※2)。

※1：https://docs.gitlab.com/ee/install/requirements.html
※2：https://docs.gitlab.com/ee/administration/nfs.html#avoid-using-cloud-based-file-systems

②ECSの構築　–タスク実行ロールとタスクロール

ECSにはタスクを実行する際に必要な「タスク実行ロール」とタスク起動後にタスクが必要とする「タスクロール」の二つがあります。最低限必要なロールは以下の通りです。

タスク実行ロール
AmazonEC2ContainerRegistryReadOnly
AmazonECSTaskExecutionRolePolicy
AmazonS3ReadOnlyAccess
右記のインラインポリシー

タスク実行ロール：
https://docs.aws.amazon.com/AmazonECS/latest/developerguide/task_execution_IAM_role.html#create-task-execution-role

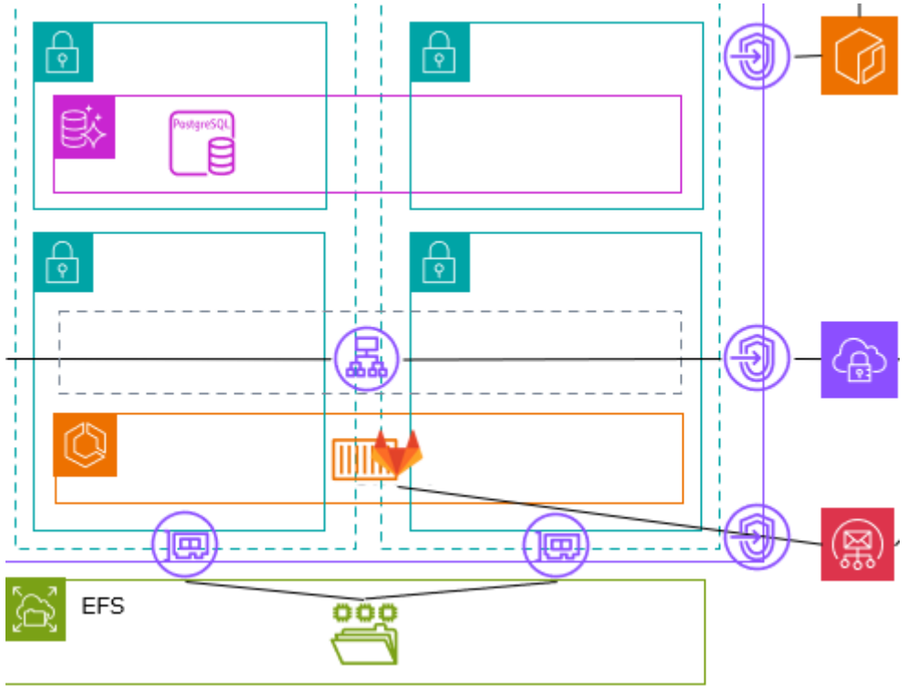
インラインポリシー①
{ "Version": "2012-10-17", "Statement": [{ "Effect": "Allow", "Action": ["ecs:ListClusters", "ecs:ListContainerInstances", "ecs:DescribeContainerInstances", "ecr:BatchImportUpstreamImage", "ecr:CreateRepository"], "Resource": "*" }] }

インラインポリシー②
{ "Version":"2012-10-17", "Statement":[{ "Effect":"Allow", "Action":["ses:SendEmail", "ses:SendRawEmail"], "Resource":"*" }] }

タスクロール
AmazonSSMFullAccess

タスクロール：https://docs.aws.amazon.com/AmazonECS/latest/developerguide/task-iam-roles.html#create_task_iam_policy_and_role

③VPCエンドポイント



当構成ではGitLabをプライベートサブネットに配置しており、VPC外のサービスとはVPCエンドポイントを経由してアクセスします。必要なエンドポイントは以下の通りです。

※↓東京リージョン(ap-northeast-1)利用時

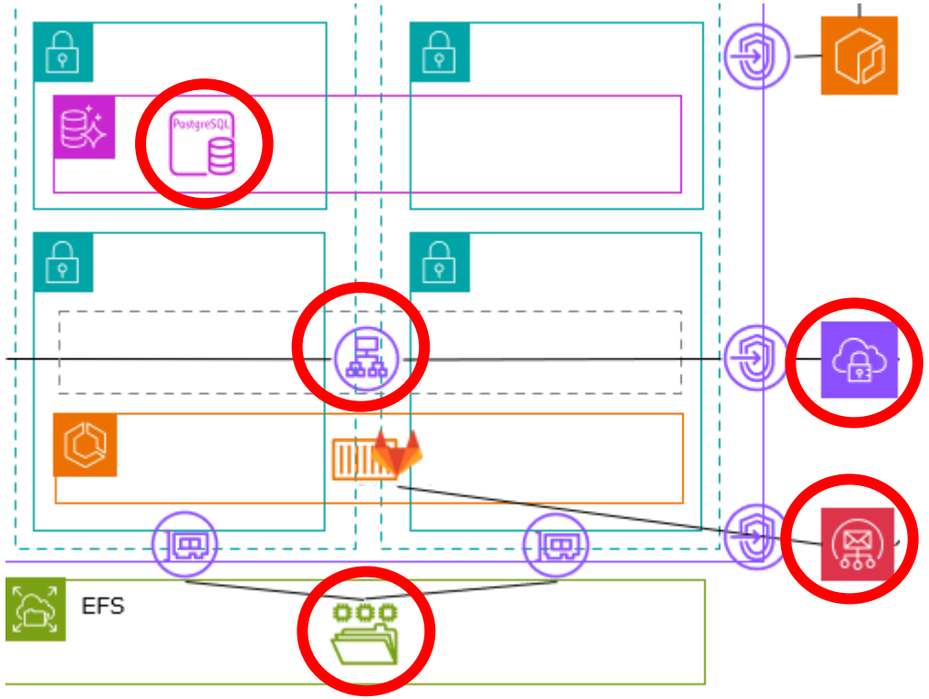
ECR	com.amazonaws.ap-northeast-1.ecr.dkr
ECR	com.amazonaws.ap-northeast-1.ecr.api
S3	com.amazonaws.ap-northeast-1.s3
SSM	com.amazonaws.ap-northeast-1.ssmmessages
SES	com.amazonaws.ap-northeast-1.email-smtp

ECRに登録されているコンテナイメージの実体はS3にあり、S3へのエンドポイント、アクセスロールが必要となるので注意

参考）ECRエンドポイント：https://docs.aws.amazon.com/ja_jp/AmazonECR/latest/userguide/vpc-endpoints.html#ecr-setting-up-vpc-create

④その他サービスの構築

今回は説明を省略しますが、先ほど述べました通り、以下のサービスの構築が必要となります。



サービス	用途
Aurora PostgreSQL	GitLabのコンテナ内に存在するPostgreSQLを外出しする
ELB（ALB）	ECS(Fargate)は固定IPを持たず再起動の都度IPが変更されるため、前面にALBが必要となるCodeConnectionsがHTTPS接続を必要とするためそれにも利用する
EFS	GitLabのローカルの設定ファイルを外出しするEFS機能の他リージョンへのリプリケーションも活用する
SES	サインアップ時のメール連携などに使用するメール連携しなくても動作には問題なし
Client VPN	GitLab＆ALBをプライベートサブネットに配置してるため、当VPCへのアクセスに必要

⑤GitLabの設定

外出ししているAurora、EFS、SESの設定を行います。

- ECSへの接続

  踏み台となるEC2やCloud9からECS Execで接続します。

```
$ aws ecs execute-command --cluster <クラスター名> --task "<ECSタスクのARN>" --container <コンテナ―名> --interactive --command "/bin/bash"
```

※タスクの状態として”enableExecuteCommand”がtrueになっていないと接続できないので注意

```
sh-5.2$ aws ecs describe-services --cluster Gitlab-env --services Gitlab-test-service-alb |grep enableExecuteCommand
            "enableExecuteCommand": true
```

※true へ変更するコマンド

```
$ aws ecs update-service --cluster <クラスタ名> --service <サービス名> --enable-execute-command --force-new-deployment
```

⑤GitLabの設定

- /etc/gitlab/gitlab.rbの編集

  gitlab.rb というファイルにもろもろの設定があります。

EFS

タスク定義作成時にコンテナマウントポイントを”/gitlab-nfs”とした場合

```
git_data_dirs({"default" => { "path" => "/gitlab-nfs/git-data"} })
gitlab_rails['uploads_directory'] = '/gitlab-nfs/uploads'
gitlab_rails['shared_path'] = '/gitlab-nfs/shared'
gitlab_ci['builds_directory'] = '/gitlab-nfs/builds'
```

Postgres

```
postgresql['enable'] = false
gitlab_rails['db_adapter'] = 'postgresql'
gitlab_rails['db_encoding'] = 'unicode'
gitlab_rails[‘db_host’] = ‘<Auroraのクラスターエンドポイント>'
gitlab_rails[‘db_username’] = “<ユーザ名>"
gitlab_rails[‘db_password’] = ‘<パスワード>'
```

⑤GitLabの設定

- /etc/gitlab/gitlab.rbの編集

SES

```
gitlab_rails['smtp_enable'] = true
gitlab_rails['smtp_address'] = "email-smtp.ap-northeast-1.amazonaws.com"
gitlab_rails['smtp_port'] = 587
gitlab_rails['smtp_user_name'] = "<SESで作成されたユーザ名>"
gitlab_rails['smtp_password'] = "<SESで生成されたパスワード>"
gitlab_rails['smtp_authentication'] = "login"
gitlab_rails['smtp_enable_starttls_auto'] = true
gitlab_rails['gitlab_email_from'] = "<Fromに設定するメールアドレス>"
gitlab_rails['gitlab_email_reply_to'] = "<Reply toに設定するメールアドレス>"
```

※任意の受信者にメールを送るにはSESのダッシュボードから「本番申請」が必要
https://docs.aws.amazon.com/ja_jp/ses/latest/dg/request-production-access.html

- /var/opt/gitlab/redis/redis.socketの作成

Redis

```
touch /var/opt/gitlab/redis/redis.socket
chown gitlab-redis /var/opt/gitlab/redis/redis.socket
```

⑤GitLabの設定

設定を保存したら、GitLabを再起動します。(ECSの再起動ではありません)

```
$ gitlab-ctl stop
$ gitlab-ctl reconfigure
$ gitlab-ctl start
```

GitLabのrootユーザパスワードの確認をします。

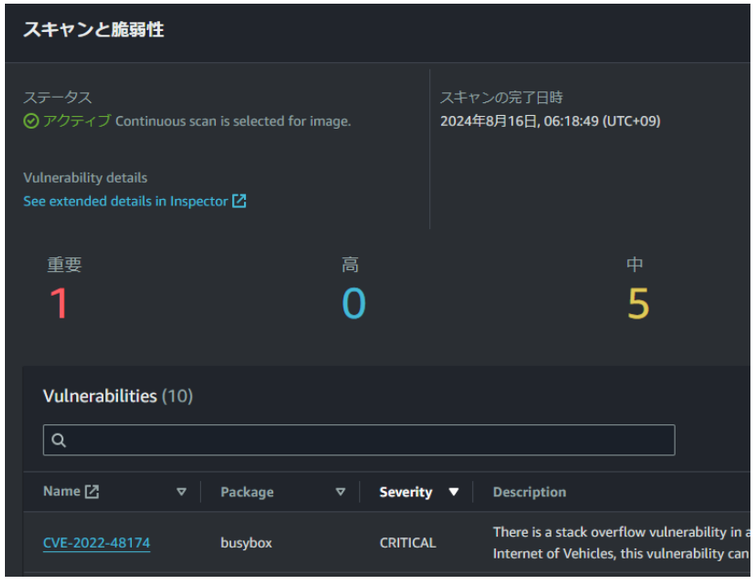
```
$ cat /etc/gitlab/initial_root_password
```

※rootパスワードのリセット方法

```
$ gitlab-rake "gitlab:password:reset[root]"
Enter password:
Confirm password:
```

⑤GitLabの設定　–コンテナイメージの更新

ECRに登録したGitLabのイメージはAmazon Inspectorで検査されます。



左記のように「重要」や「高」が検出された場合には速やかにイメージの最新化を行いましょう。

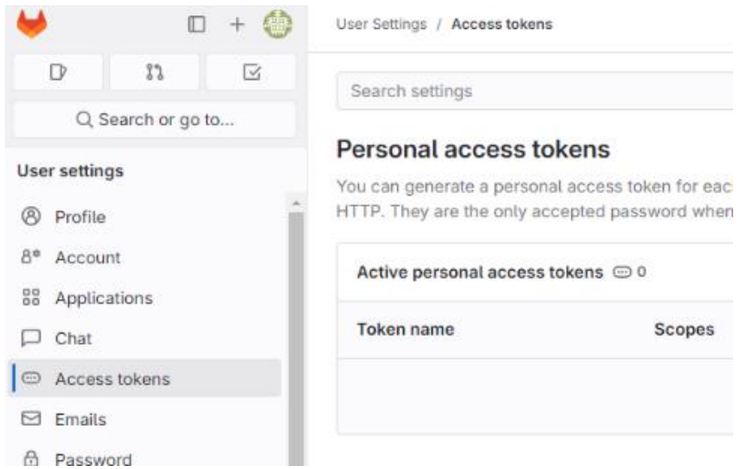
イミュータブルにしてタグを管理している場合にはECSのタスク定義が新しいイメージを指すように「新しいリビジョン」を作成する必要があります。

タスクの更新後、GitLabにrootユーザでログインして設定変更を行う際に、500エラーが発生することがあります。その場合は下記コマンドを実行してください。

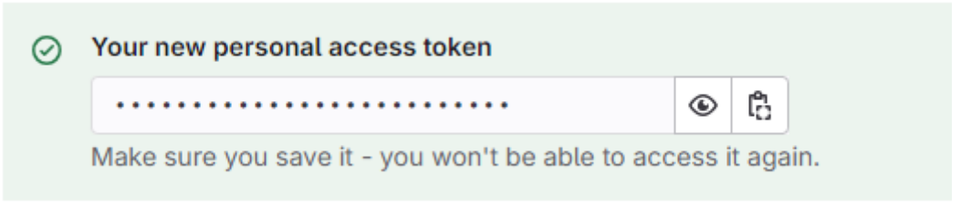
```
$ gitlab-rails c
(数十秒待つ)
irb(main):001:0> settings = ApplicationSetting.last
irb(main):001:0> settings.update_column(:runners_registration_token_encrypted, nil)
irb(main):001:0> settings.reset_error_tracking_access_token!
irb(main):001:0> settings.save
irb(main):001:0> exit
```

⑥CodePipelineとの接続

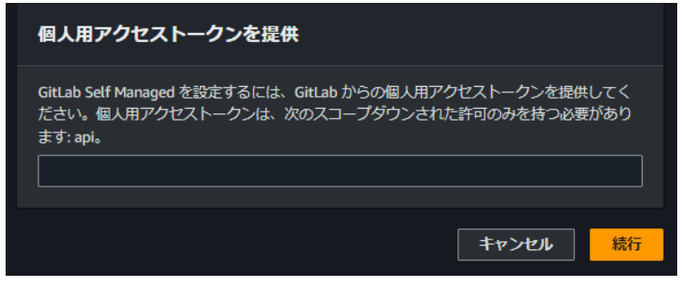
Client VPNでVPCへ接続後、ALBのURLにアクセスしてGitLabコンソールへログインします。

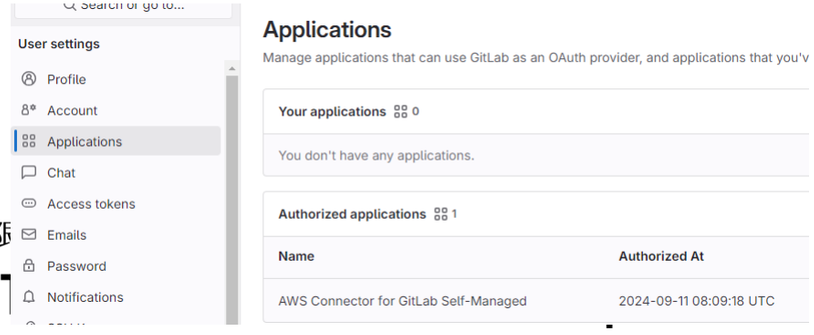


ログイン後、User Settings > Access tokensメニューよりアクセストークンを発行します。(有効期限はデフォルト365日)
「Select scopes」で「api」を選択して発行します。



⑥CodePipelineとの接続





(Transit Gateway等でGitLabのVPCへ接続できていることが前提となります。)

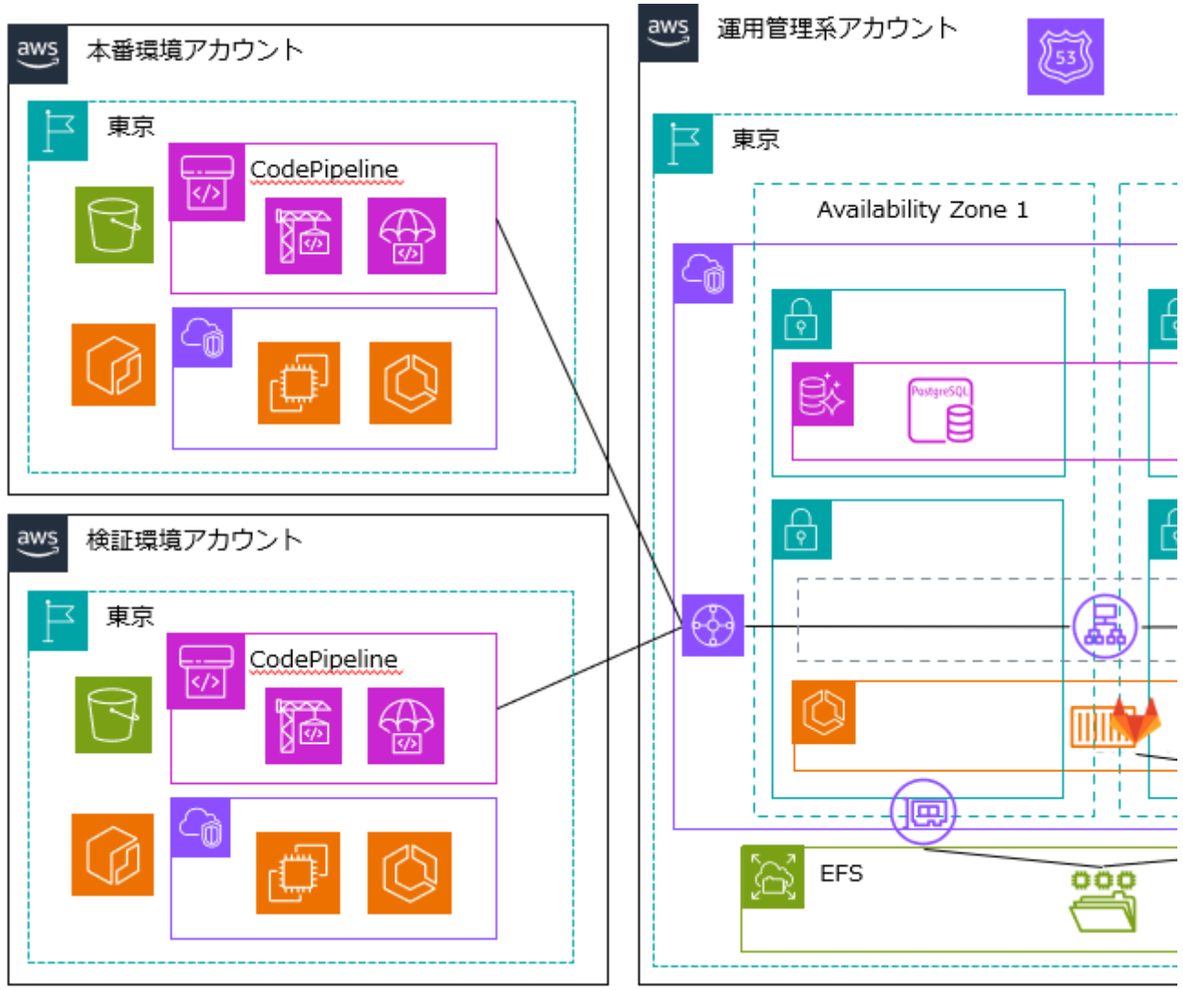
AWSコンソールのCodePipelineより、設定 > 接続からGitLabへの接続を作成します。
エンドポイントはALBのURLを指定します。

接続を作成すると接続が保留されますので当接続から「保留中の接続を更新」を押し、GitLabで作成したアクセストークンを設定します。

GitLabへリダイレクトされますのでログインし、接続を承認します。

AWSドキュメント：
https://docs.aws.amazon.com/ja_jp/dtconsole/latest/userguide/connections-create-gitlab-managed.html

まとめ



GitLabへ移行することでAuroraやALB、ECS、EFSといった管理リソースが増えるものの、アプリが稼働する既存のパイプラインには大きな影響を与えずにGitLabへの接続が実現できそうです。

まずは開発環境や検証環境向けにGitの切替を検証してみてはいかがでしょうか？

その他の考慮点：

- CodeCommitからGitLabへのリポジトリ移行
  ⇒AWSのサイトやGitLabのマニュアルを参照ください。
- GitLabのグループ機能などシステムごとのリポジトリアクセス権の制御の検討
- GitLabのECSの状態監視の検討（CloudWatchなどへのログルータの設置）
- 大阪リージョンでGitLabへのCodeConnectionsが未対応のため、マルチリージョン構成の場合の大阪リージョンへの配布方法の検討

注記

この機能は、アジアパシフィック (香港) 、アジアパシフィック (ハイデラバード) 、アジアパシフィック (ジャカルタ) 、アジアパシフィック (メルボルン) 、アジアパシフィック (大阪) 、アフリカ (ケープタウン) 、中東 (バーレーン) 、中東 ()、UAE欧州 (スペイン) 、欧州 (チューリッヒ) 、イスラエル (テルアビブ) 、または AWS GovCloud （米国西部) の各リージョンでは使用できません。利用可能なその他のアクションについては、「との製品とサービス

無限の未来と、
幾千のテクノロジーをつなぐ。

CTC Financial Services Group

CTC
Challenging Tomorrow's Changes