GoTanKen!

簡単な高性能ウェブ開発の試し

2024.5.3 QI Zhecheng

何を作りたい

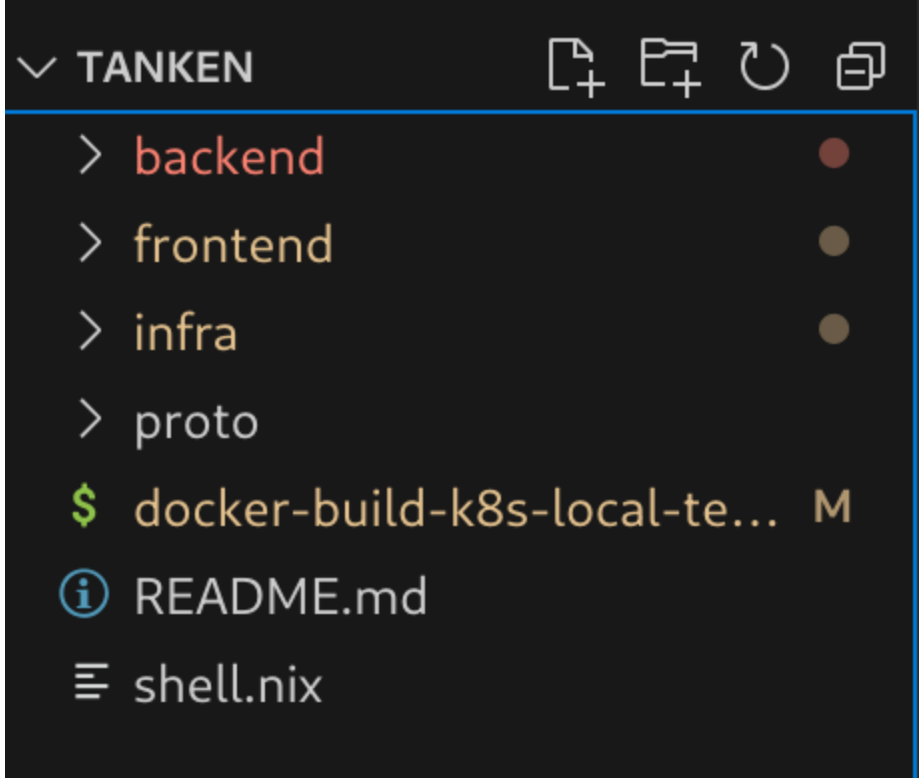
- Instagramみたいのプラットフォーム
- 地理位置情報を含まれている写真をアップロードできる
- 近くの面白い場所が見つける

目次

- システムデザイン
  - フロントエンド
  - バックエンド1
  - インフラ
  - バックエンド2
- テスト
- レポジトリブランチ策略

まず、システムデザイン

- 一人で作るので、モノレポを使う
  - フロントエンド、バックエンド、インフラ全部一つのgit repositoryの中に管理する



システムデザイン in フロントエンド

- フロントエンドは、今はReact.jsとVue.jsの二分天下
- 今回はVue.js（Nuxt.js 3フレームワーク）を使う
- 個人としての感じは
  - React.jsの抽象程度が高い（伝統Javascript開発と全然違う）、独特なBest Practiceが多い
  - Vue.jsの生態系はReact.jsより未熟
  - まあ、どれが優れは言えない、争いの元になれる

システムデザイン in バックエンド

- 簡単な処理なら、特にフロントとバックを区別しなくてもいい、フロントのフレームワーク（Next.jsやNuxt.js）にバックエンドの処理ができる。でもこのプロジェクトにたいしては少々足りないと思う

- Go言語を使う、マイクロサービスの理念で実現

- サービスの例
  - data-fetcher: データベース・キャッシュと他のサービス（フロントなど含まれている）の間の橋
  - monitor: k8sクラスターの資源を監視、キャッシュのバックアップなどの操作を含まれている

バックエンドは一体？

- 重要点は:
  - データモデル: データはどんな感じで保存しています
  - サービス間データ通信ルール: データはどんな感じでサービス間転移しています、サービスはどんなデータが提供していますか

バックエンド about データベース

- 永続性データベースが必要
  - SQL or NoSQL?　とりあえずSQLを選んだ。NoSQLは料金高い(AWS)
  - 画像データを除く（Amazon S3）、全てのデータを保存
  - Amazon RDSなら、データベースへのアクセス回数で課金します、なるべくアクセス回数を減らすとは重要（キャッシュの重要性）

- キャッシュデータベースも必要
  - Redisを使用、geospatial機能を活用
  - 永続性データベースへのアクセス回数を減って、バジェットに優しい
  - 永続性データベースよりだいぶ快速

データモデルデザイン

- 永続性データベース
- キャッシュ
- フロントエンドに受けるもの

- 全部違うものなので、type定義するが重要
  - type定義＝データモデルデザイン　と思う
  - type定義がいいなら、エラー発生が減る

データモデルデザイン　例

```
message Post {
  string postId = 1;
  int64 timestamp = 2;
  User author = 3;
  string content = 4;
  bytes pictureChunk = 5;
  Location location = 6;
  int64 likes = 7;
  int64 bookmarks = 8;
  repeated Comment comments = 9;
  repeated string tags = 10;
}
```

database > migrations > V1__initial_schema.sql

```
-- V1__initial_schema.sql

CREATE TABLE users (
    user_id UUID PRIMARY KEY,
    username TEXT NOT NULL,
    email TEXT NOT NULL UNIQUE,
    profile_picture_link TEXT,
    bio TEXT,
    oauth_provider TEXT NOT NULL,
    last_login TIMESTAMP,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    subscribed INT DEFAULT 0
);
```

キャッシュ

- 特徴
  - アクセススピードが早いですが、容量は限度がある（メモリですから）
  - なにをキャッシュするが大事
  - キャッシュと永続性データベースとの一致性が大事
  - キャッシュ策略が大事
- Redis-Clusterを使う
  - 全てのkey-value pairsを、いくつのredisサーバーで分散
  - 負荷を平均する

キャッシュ　一致性問題

- 例えば setPost(postId, postContent) 関数
- キャッシュにいますか？すぐにDBのデータ更新するが必要ですか？
  - いるならキャッシュ直接set
  - いないなら、DBでset、これをキャッシュに入れますか
- だからSignatureは
  setPost(postId, postContent, needWriteBackNow, needCache)
  - needXはboolean値

システムデザイン in インフラ

- まず、すべてのコード（フロントとバック）をDockerize、kubernetesで起動
- kindを使って、kubernetesクラスターをローカル（パソコン）で起動できる

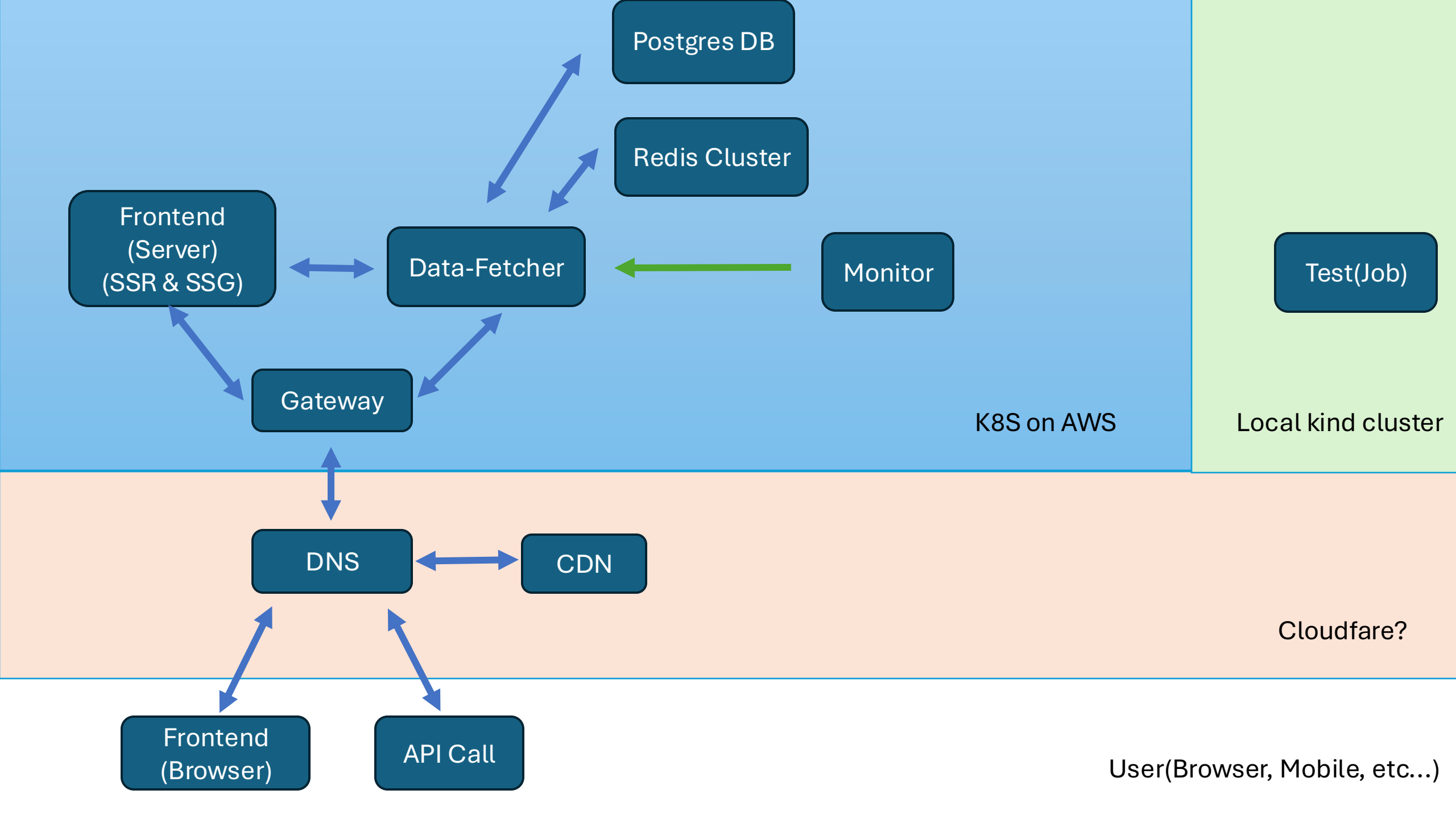
```
 (>∀<)  kga
NAME                                READY   STATUS      RESTARTS         AGE
pod/data-fetcher-9968f88fb-zspvq    1/1     Running     3 (8m31s ago)    3h42m
pod/database-7785777476-brthc       1/1     Running     1 (9m ago)       3d
pod/geocache-76bc787d87-nqc86       1/1     Running     3 (9m ago)       8d
pod/migrate-job-s2vmw               0/1     Completed   0                3h40m
pod/postcache-77cfbb7fdb-74rtl      1/1     Running     3 (9m ago)       8d
pod/test-job-p7x2z                  0/1     Completed   0                3h40m
pod/usercache-687d98d6f7-nw2m8      1/1     Running     1 (9m ago)       3d4h

NAME                       TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)     AGE
service/data-fetcher       ClusterIP   10.96.239.220   <none>        50051/TCP   3d4h
service/database-service   ClusterIP   None            <none>        5432/TCP    3d
service/geocache           ClusterIP   10.96.56.250    <none>        6379/TCP    8d
service/kubernetes         ClusterIP   10.96.0.1       <none>        443/TCP     8d
service/postcache          ClusterIP   10.96.243.125   <none>        6379/TCP    8d
service/usercache          ClusterIP   10.96.251.83    <none>        6379/TCP    3d4h

NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/data-fetcher      1/1     1            1           3d4h
deployment.apps/database          1/1     1            1           3d
deployment.apps/geocache          1/1     1            1           8d
deployment.apps/postcache         1/1     1            1           8d
deployment.apps/usercache         1/1     1            1           3d4h

NAME                                         DESIRED   CURRENT   READY   AGE
replicaset.apps/data-fetcher-9968f88fb       1         1         1       3d
replicaset.apps/database-7785777476          1         1         1       3d
replicaset.apps/geocache-76bc787d87          1         1         1       8d
replicaset.apps/postcache-77cfbb7fdb         1         1         1       8d
replicaset.apps/usercache-687d98d6f7         1         1         1       3d4h

NAME                    STATUS     COMPLETIONS   DURATION   AGE
job.batch/migrate-job   Complete   1/1           4s         3h40m
job.batch/test-job      Complete   1/1           3s         3h40m
```


Postgres DB
Redis Cluster
Frontend
(Server)
(SSR & SSG)
Data-Fetcher
Monitor
Test(Job)
Gateway
K8S on AWS
Local kind cluster
DNS
CDN
Cloudfare?
Frontend
(Browser)
API Call
User(Browser, Mobile, etc...)

バックエンド サービス間通信

- Connect-RPC（多くのRPCプラットフォームの中の一種）を使う
  - HTTP Restful-APIより効率がいい（HTTP/2の特性を利用して）
  - 定義明確、メンテナンスが優しい（でもHTTPより複雑性も問題）

- 元々gRPC（また多くのRPCプラットフォームの中の一種）を使いたいですが、gRPCを生成したJavascriptは古いCommonJS style、現代ES6 styleを生成できない
  - Googleさん、何で今までes6のサポートしてくれないの？

通信ルール定義

proto > post.proto > ...

```
58  message UploadNewPostRequest {
59    string userId = 1;
60    bytes pictureChunk = 2;
61    Location location = 3;
62    string content = 4;
63    repeated string tags = 5;
```

```
service PostsService {
  rpc GetNewPostId(GetNewPostIdRequest) returns (GetNewPostIdResponse) {}
  rpc GetPostsByLocation(GetPostsByLocationRequest) returns (GetPostsByLocationResponse) {}
  rpc GetPostsByPostIds(GetPostsByPostIdsRequest) returns (GetPostsByPostIdsResponse) {}
  rpc GetPostsByUser(GetPostsByUserIdRequest) returns (GetPostsByUserIdResponse) {}
  rpc UploadNewPost(UploadNewPostRequest) returns (UploadNewPostResponse) {}
  rpc AddLike(AddLikeRequest) returns (AddLikeResponse) {}
  rpc RemoveLike(RemoveLikeRequest) returns (RemoveLikeResponse) {}
  rpc AddBookmark(AddBookmarkRequest) returns (AddBookmarkResponse) {}
  rpc RemoveBookmark(RemoveBookmarkRequest) returns (RemoveBookmarkResponse) {}
  rpc AddComment(AddCommentRequest) returns (AddCommentResponse) {}
  rpc RemoveComment(RemoveCommentRequest) returns (RemoveCommentResponse) {}
}
```

= 3;

frontend > server > api > TS rpc.ts > default > defineEventHandler() callback

```ts
import { createPromiseClient } from "@connectrpc/connect";
import { createConnectTransport } from "@connectrpc/connect-node";
import { PostsService } from '~/rpc/post_connect';

const transport = createConnectTransport({
  baseUrl: "http://data-fetcher:50051",
  httpVersion: "2"
});

const client = createPromiseClient(PostsService, transport);

export default defineEventHandler(async (event) => {
  const {
    userId,
    pictureChunk,
    location,
    content,
    tags
  } = await readBody(event);

  try {
    const response = await client.uploadNewPost({
      userId,
      pictureChunk: new Uint8Array(pictureChunk),
      location,
      content,
      tags
    });
```

フロントエンド

クライアントのRequest

backend > data-fetcher > rpc > grpc.go > (*server).UploadNewPost

```
22  type server struct {
23      geo_postid_rdb *redis.Client
24      post_cache_rdb *redis.Client
25      db             *sql.DB
26      // uploader       *s3manager.Uploader
27  }
28
29  func (s *server) GetNewPostId(ctx context.Context, req *connect.Request[pb.GetNewPostIdRequest]) (*
53  }
54
55  func (s *server) GetPostsByLocation(ctx context.Context, req *connect.Request[pb.GetPostsByLocation
56      return connect.NewResponse(&pb.GetPostsByLocationResponse{}), nil
57  }
58
59  func (s *server) GetPostsByUser(ctx context.Context, req *connect.Request[pb.GetPostsByUserIdReques
60      return connect.NewResponse(&pb.GetPostsByUserIdResponse{}), nil
61  }
62
63  func (s *server) GetPostsByPostIds(ctx context.Context, req *connect.Request[pb.GetPostsByPostIdsRe
64      return connect.NewResponse(&pb.GetPostsByPostIdsResponse{}), nil
65  }
66
67  func (s *server) UploadNewPost(ctx context.Context, req *connect.Request[pb.UploadNewPostRequest])
120 }
```

バックエンド

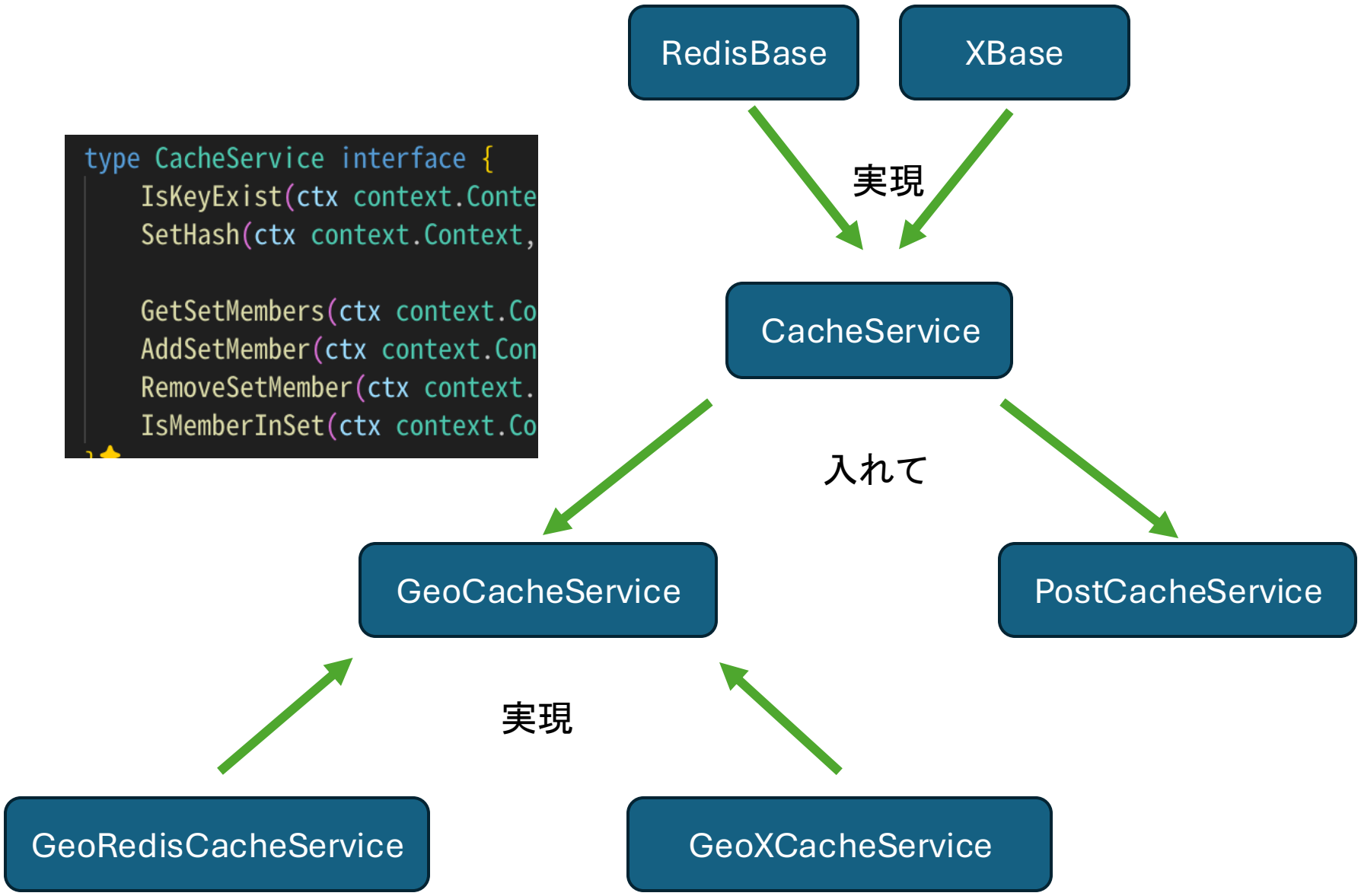
data-fetcher
の実現

認証（Authentication）について

- 自分でセッション管理は大変
  - 大手のOAuthを利用、例えばLogin with Google

Interface Design

- Interfaceからデザイン、コードの質を向上
  - RPCのprotoファイルは　通信双方の約束、どんな言語でも、protoファイル定義したmethodを実現すれば、通信ができる
  - バックエンドもそうだ、DatabaseServiceを定義して、DBに関わるmethodのInterfaceを定義して、そしてPostgreServiceをこのInterfaceを実現、そうしたら、もしいつかMySQLを使いたいなら、MySQLServiceを書けばいい、Interfaceは変わらないもの、それを実現ればOK




```
type CacheService interface {
    IsKeyExist(ctx context.Conte
    SetHash(ctx context.Context,

    GetSetMembers(ctx context.Co
    AddSetMember(ctx context.Con
    RemoveSetMember(ctx context.
    IsMemberInSet(ctx context.Co
```

```
type PostCacheService interface {
    CacheService

    GetPostDetails(ctx context.Cont
    SetPostDetails(ctx context.Cont

    GetPostLikedBy(ctx context.Cont
    AddPostLikedBy(ctx context.Cont
    RemovePostLikedBy(ctx context.C
```

```go
type PostRedisCacheService struct {
    *RedisBase
}

var _ PostCacheService = (*PostRedisCacheService)(nil)

func NewPostRedisCacheService(client *redis.Client) *PostRedisCacheService {
    return &PostRedisCacheService{
        RedisBase: NewRedisBase(client),
    }
}

func (r *PostRedisCacheService) GetPostDetails(ctx context.Context, postID s
    postDetailsMap, err := r.client.HGetAll(ctx, postID).Result()
```

```go
prcs, ok := rs.(*cache.PostRedisCacheService)
if !ok {
    return nil, fmt.Errorf("cache service does not support GetPostDetailsCmd")
}

ctx, pipe := rs.NewPipe(ctx)
postDetailsCmd, _ := prcs.GetPostDetailsCmd(ctx, postId)
```

テスト

- 二種類のテストを実装しました：
  - Unit Test
    - 特定関数が正常に動いていますか
    - 環境を依存していない、コードから直接テスト
  - Integrated Test
    - サービスの提供したメソードが正常に動いていますか
    - 具体的の環境でテスト、つまりTestと呼ぶサビース（K8Sの中ならJob）をK8Sクラスターの中で立ち上がって、ほかのサービスをテスト

Unit Test

```
run test | debug test
func TestPostgresDatabaseService_DeleteCommentById(t *testing.T) {
    db, mock, err := sqlmock.New()
    if err != nil {
        t.Fatalf("an error '%s' was not expected when opening a stub database connection", err)
    }
    defer db.Close()

    service := NewPostgresDatabaseService(db)

    commentID := "comment1"

    mock.ExpectExec("DELETE FROM comments WHERE comment_id = \\$1").
        WithArgs(commentID).
        WillReturnResult(sqlmock.NewResult(1, 1))

    ctx := context.Background()
    err = service.DeleteCommentById(ctx, commentID)

    assert.NoError(t, err)
    if err := mock.ExpectationsWereMet(); err != nil {
        t.Errorf("there were unfulfilled expectations: %s", err)
    }
}
```

Integrated Test

```go
func TestGetUserInfo(client pbconnect.DataFetcherServiceClient, userId string) error {
    req := &pb.GetUserInfoRequest{
        UserId: userId,
    }

    res, err := client.GetUserInfo(context.Background(), connect.NewRequest(req))

    if err != nil || res.Msg.Ok != 1 {
        return fmt.Errorf("error getting user info: %v, response: %v", err, res.Msg.Msg)
    }

    expectedResponseUser := &pb.User{
        UserId:             userId,
        UserName:           "alice",
        Bio:                "this bio will change",
        ProfilePictureLink: "TODO: it need to be a link",
        Subscribed:         0,
    }

    if reflect.DeepEqual(res.Msg.User, expectedResponseUser) {
        return fmt.Errorf("expected response user: %v, got: %v", expectedResponseUser, res.Msg.User)
    }

    return nil
}
```

レポジトリブランチ(git branch)策略

- Main
- |__ReleaseX.X
- |__Develop
- 　　|__Develop-k8s-local-test