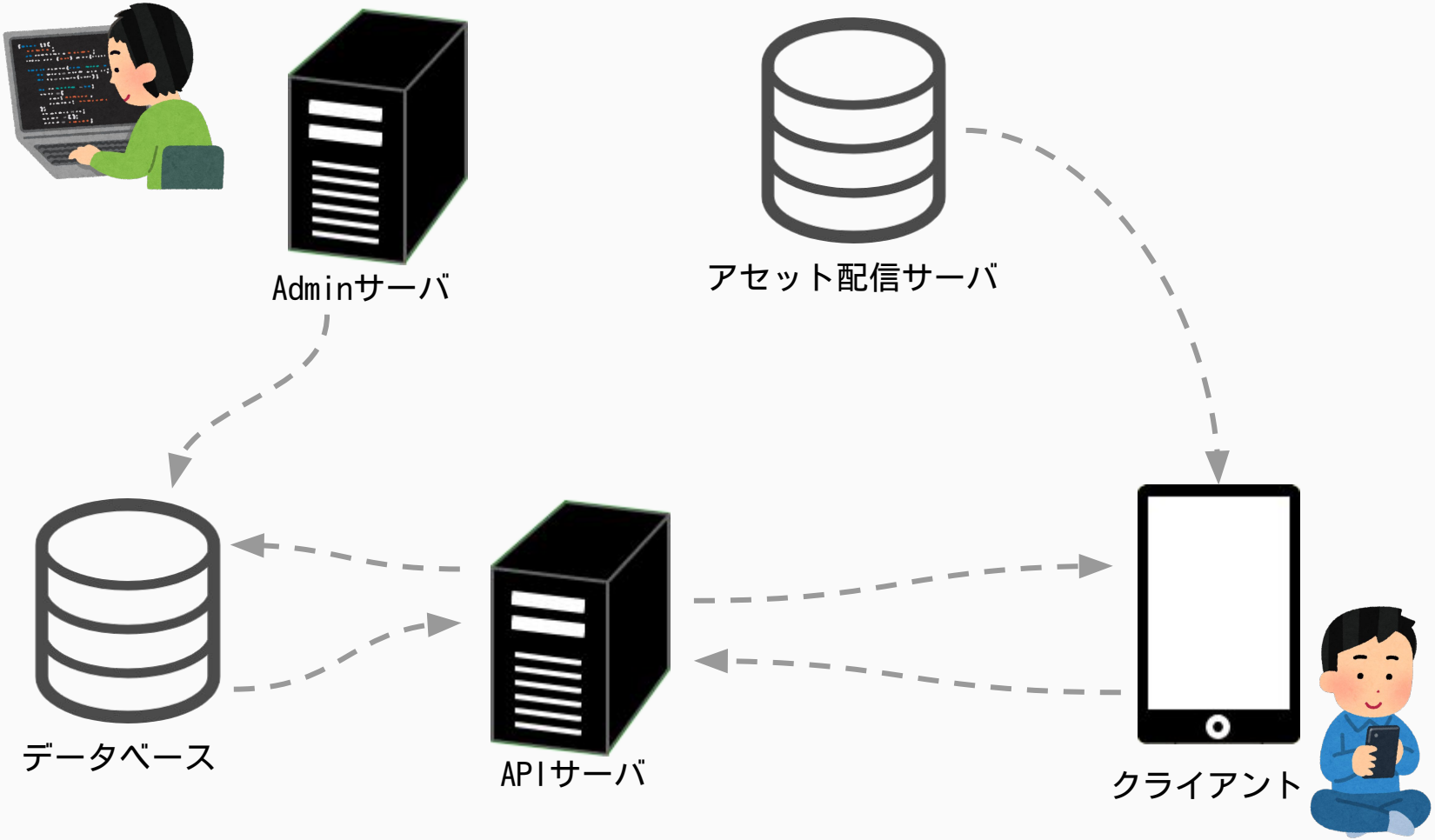
モバイルゲーム開発概観

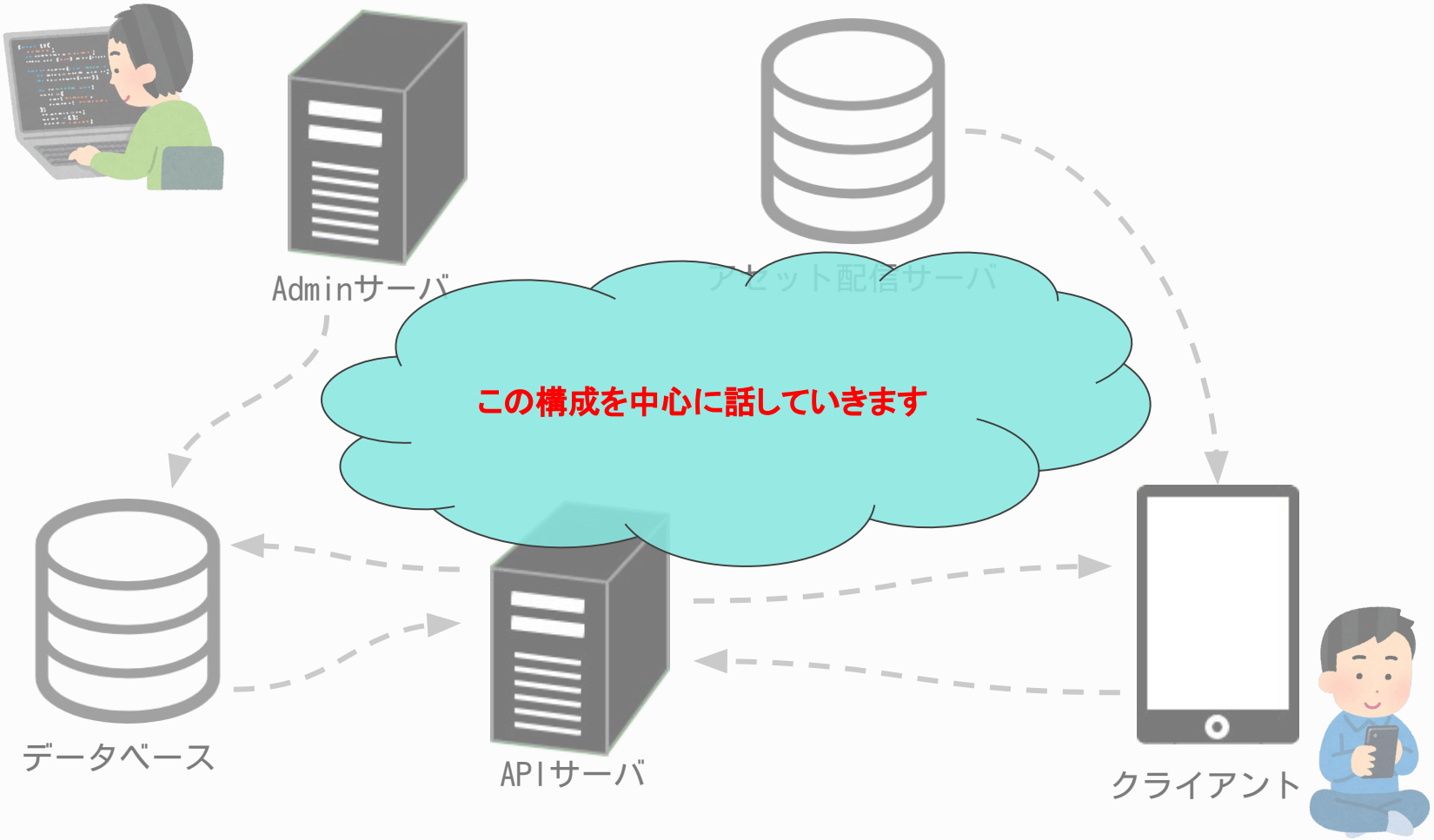
今日の話の要旨

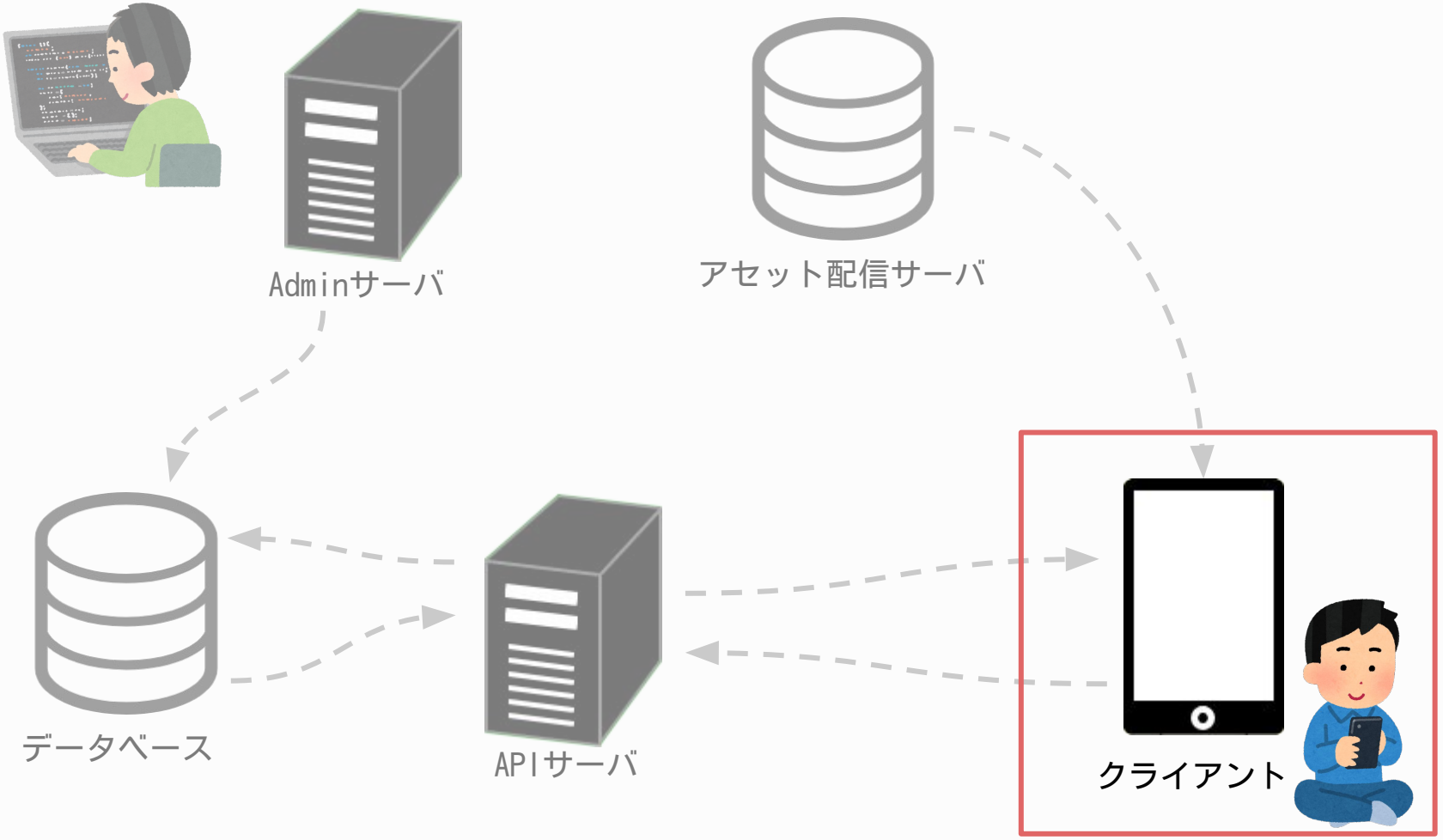
- ゲーム開発/ゲームサービスについてざっくり話をします
- 第2回以降の講義がどんなふうに効いてくるか理解する
  - 少しでも興味を持って次回以降聞いていただけると今回のゴール
- 人によって見え方が違うものですが、一例として
  - 役割、職種、経験値、信念、etc…
- 皆さんで一緒に考えたり議論したりできると良いと思います

ゲームサービスのアーキテクチャ

- クライアントサーバモデル
  - クライアント
  - サーバ(Adminサーバ、APIサーバ、アセット配信サーバ)
  - DB
- クライアントとサーバが協調して動作するシステム


Adminサーバ
アセット配信サーバ
データベース
APIサーバ
クライアント


Adminサーバ
アセット配信サーバ
この構成を中心に話していきます
データベース
APIサーバ
クライアント


Adminサーバ
アセット配信サーバ
データベース
APIサーバ
クライアント

クライアント

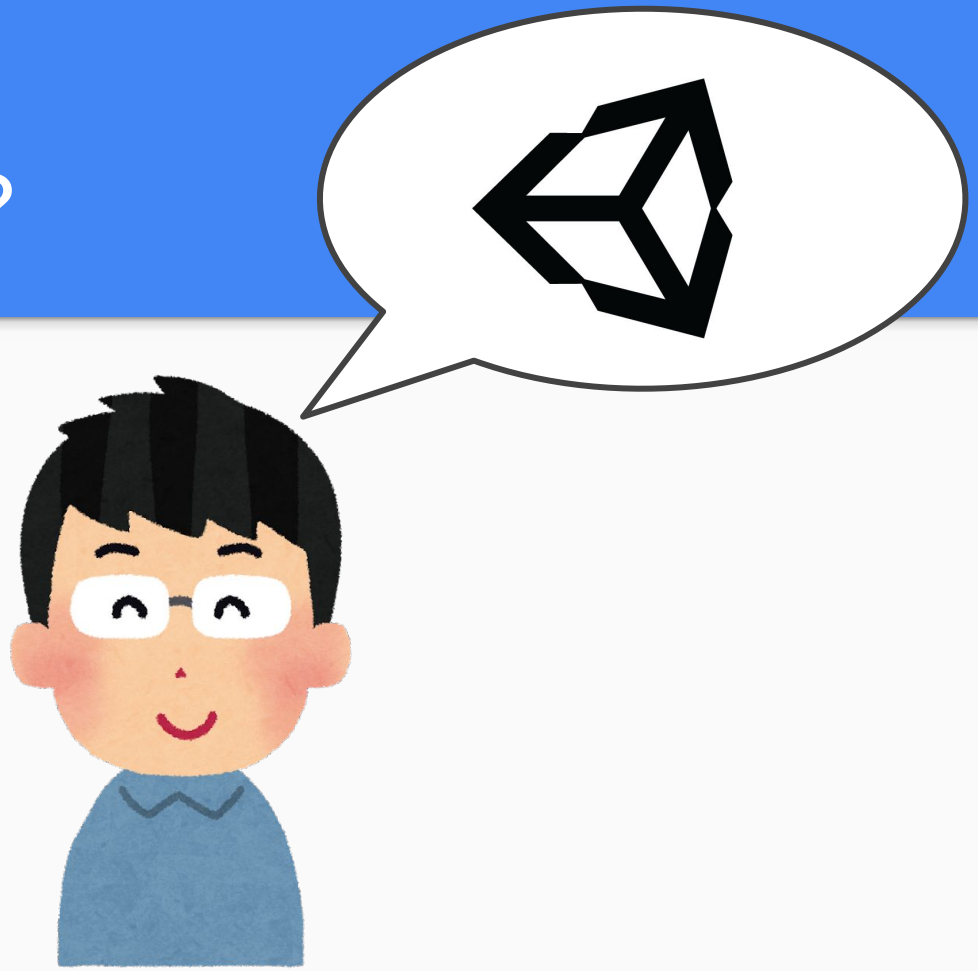
- ゲームの本体
  - プレイヤーとサービスの接点
  - サービスをゲームとして成り立たせる要素
- ユーザデータを操作するためのインタフェース
  - アプリケーションそのものに重要なユーザ情報はない（基本サーバに全て永続化される）
  - プレイヤーの操作は最終的には全てユーザデータの更新に繋がる
- 主にゲームエンジンを使って実装される

ゲームエンジンとは？

- なんですかね？

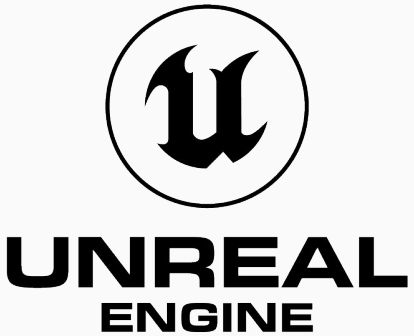
ゲームエンジンとは？

- なんですかね？



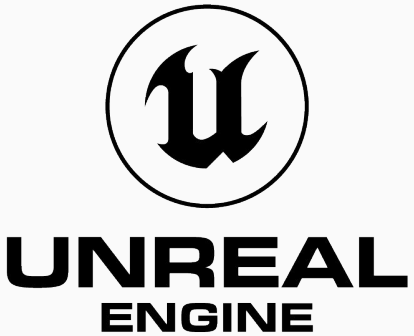
ゲームエンジン

- ゲーム開発のための共通機能を提供するソフトウェアの総称
  - 物理演算
  - 描画処理
  - UI制御
  - など・・・



ゲームエンジン

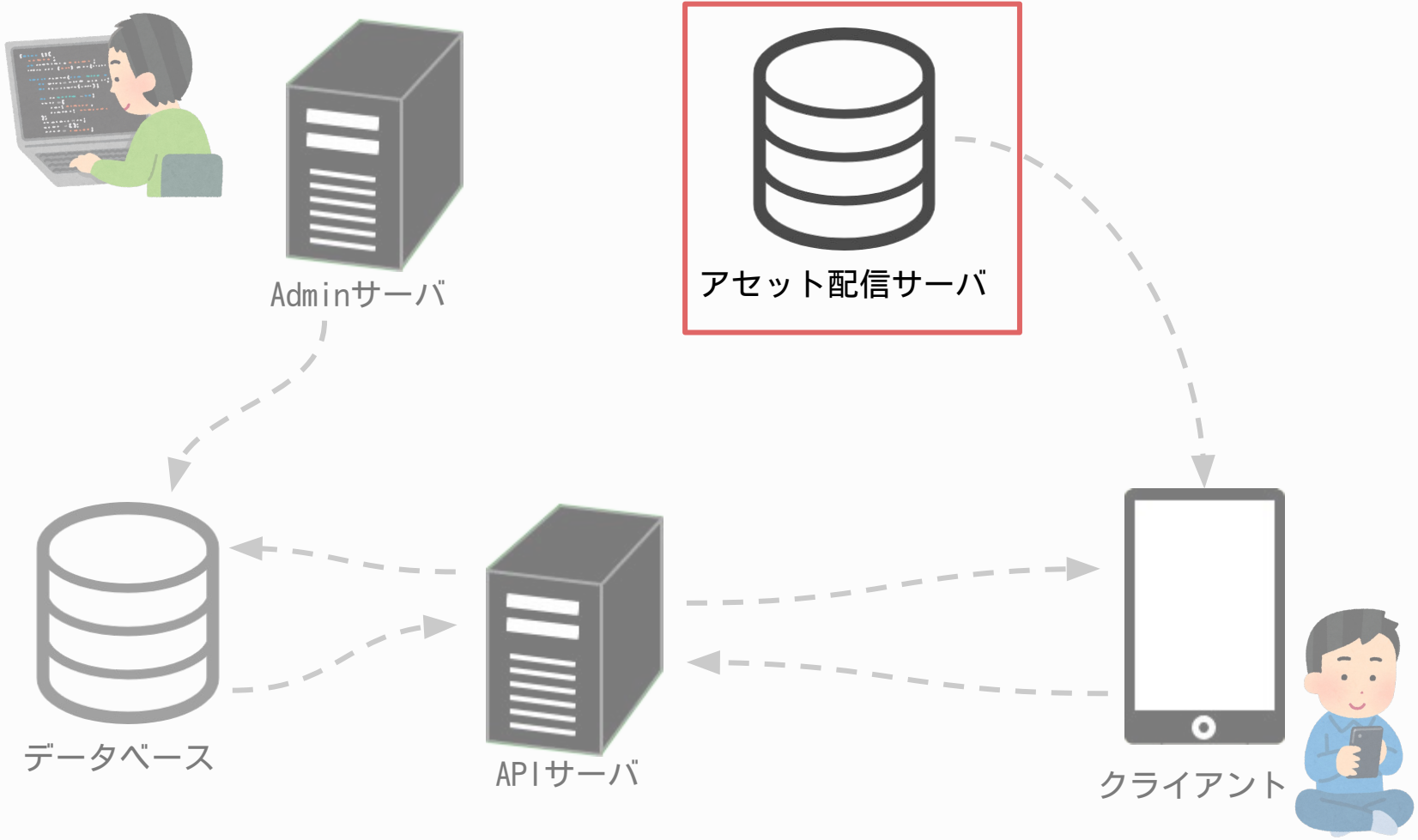
- ゲーム開発のための共通機能を提供するソフトウェアの総称
  - 物理演算
  - 描画処理
  - UI制御
  - など・・・



Unityの話

- ゲームエンジンとエディタをまとめたパッケージ的なツール
- 開発言語はC#
- モバイルゲーム開発で便利な機能がいろいろ
  - クロスプラットフォーム
  - 2D/3Dどちらでも
  - モックアップが速い
- 個人開発でもよく使われている
  - ~~コンシューマとかではあまり使われてなさそうなイメージ~~
  - 最近コンシューマでUnityを使う事例もちらほら
    - ポケモンBDSP, Fall guys, etc…

質問


Adminサーバ
アセット配信サーバ
データベース
APIサーバ
クライアント

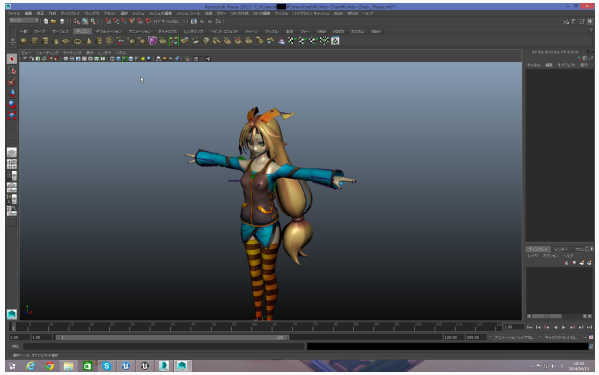
アセット配信サーバ

- ゲームを動作させるためのアセットをクライアントに配信する
  - 3Dモデル
  - テクスチャ
  - 音源などいろいろ
- Unityではアセットバンドル
  - Unityが↑のいろいろなものを取り扱うためのファイルフォーマット
- アセットバンドル管理基盤

アセットについて

- Asset = 資産
- ゲームで取り扱う全てのもの
- Unityではスクリプトもアセットとして取り扱う
- 「アセットバンドル」のことを指すことも多いよ

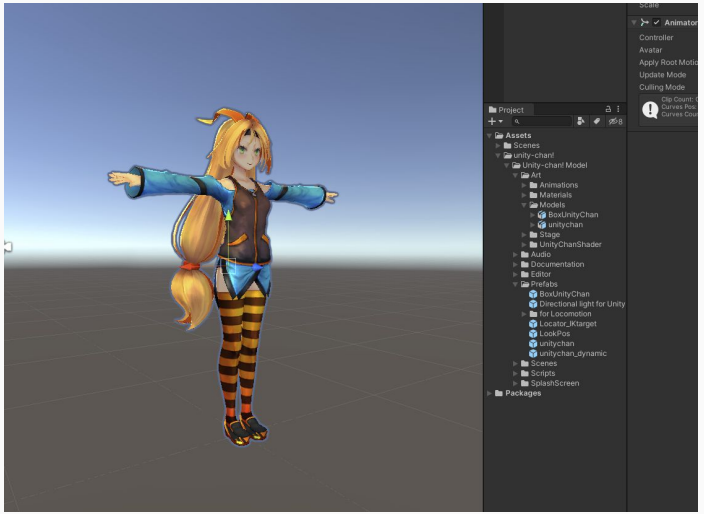
Unityのアセットバンドル



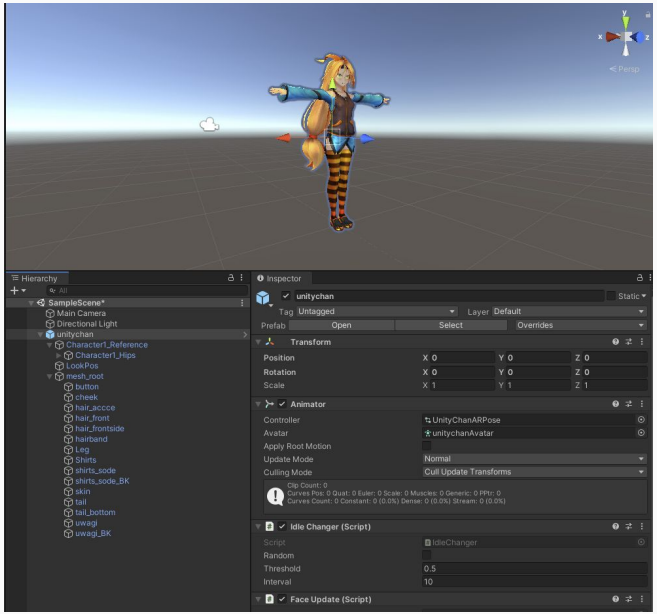
DCCツール
mayaとか

import

Unity



Unityのアセットバンドル



Unityで動作させるための設定

- コンポーネント追加
- アニメーション設定
- 各種パラメータ設定
- など・・・

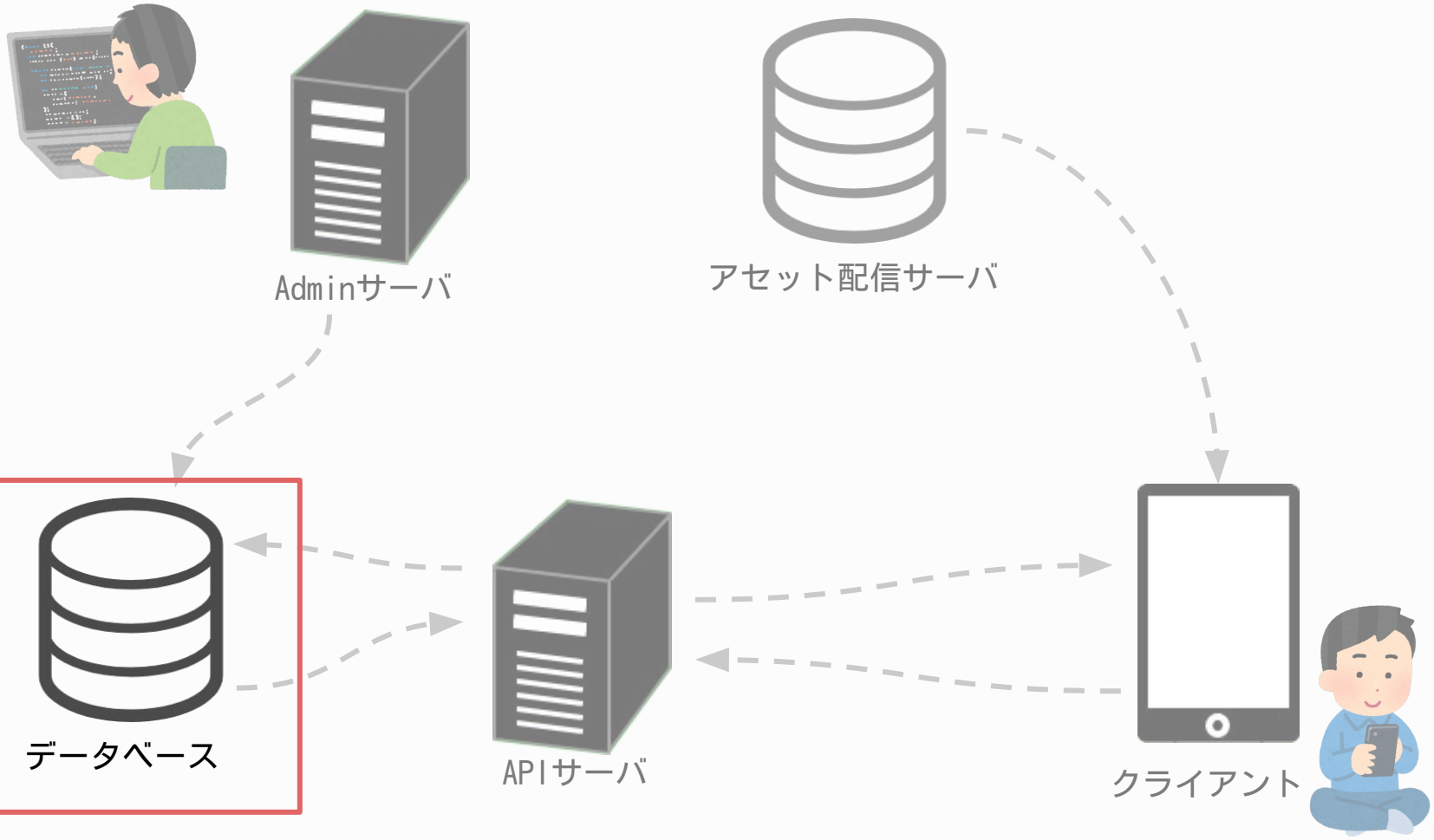
アセットバンドルビルドして完成！

Unityのアセットバンドル

- アセットをUnityアプリケーションで扱えるフォーマットに整えたもの
  - 3Dモデル、テクスチャ(画像)、オーディオファイルなど
  - バイナリ形式
- アプリサイズを小さくするためにアセットを事前にビルドしてアセットバンドルとしてサーバ経由で配信する
- 受け取ったバイナリをリソースとしてアプリケーションが取り扱う

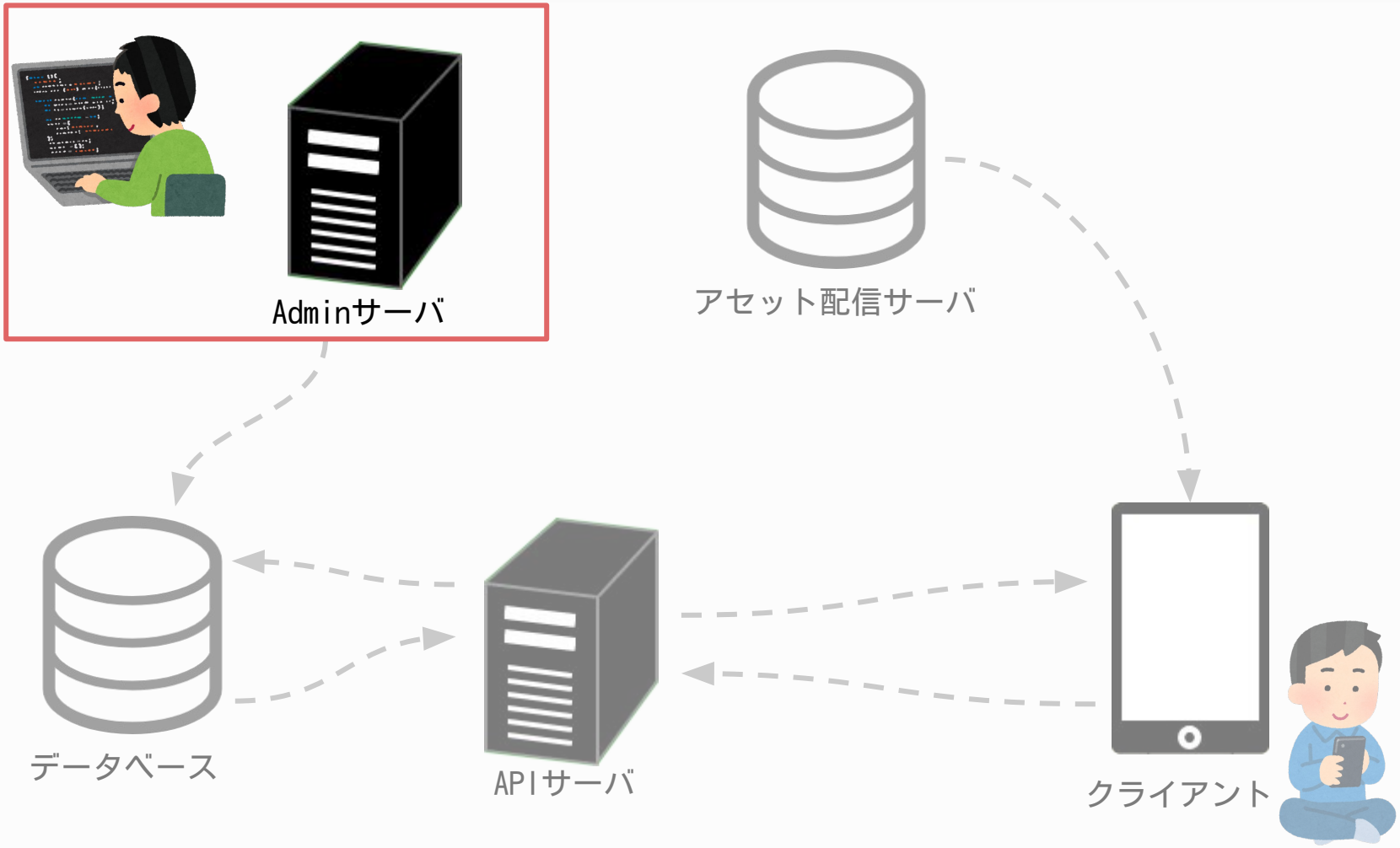
詳しくはゲームエンジンの回で!!

質問タイム


Adminサーバ
アセット配信サーバ
データベース
APIサーバ
クライアント

データベース

- ゲームで取り扱ういろいろなデータを置いておく
  - マスタデータ
  - ユーザデータ
- RDB
- NoSQL


Adminサーバ
アセット配信サーバ
データベース
APIサーバ
クライアント

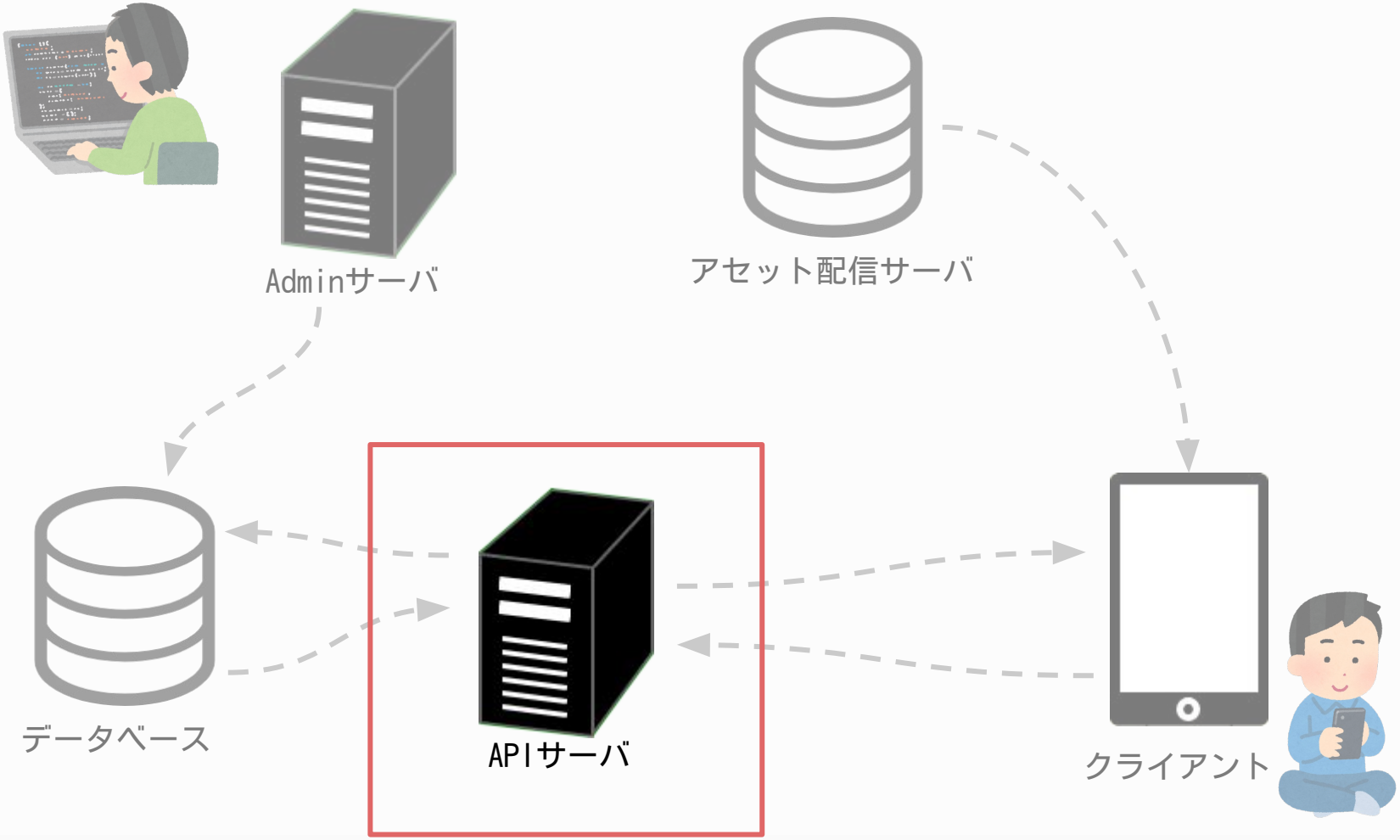
マスタデータ

- DBに存在するシステム設定情報
- 端末のストレージやメモリに置かれたゲーム情報
- csvに入力されたものをAdminサーバ経由でリリースされる
- マスタデータに従ってシステムは動作する
- 次のようなデータが格納される
  - ゲームデザイン
  - レベルデザイン
  - システムの挙動のためのデータ
  - サービス運用のためのデータ
  - キャラクターの名前、パラメータなど

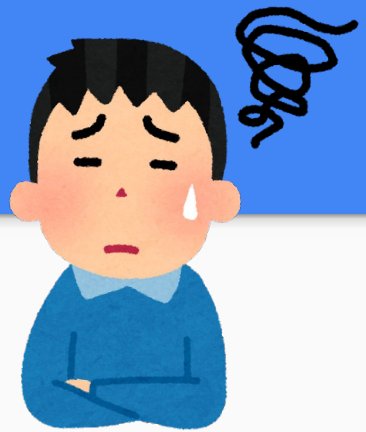
Adminサーバ

- 管理画面サーバ
- マスタデータの書き換え
- 運営チームがシステムを書き換えるための窓口
- 開発用の機能を提供したり
- バッチ処理が動いてたりもする
  - 最近はあまり見ない

質問タイム


Adminサーバ
アセット配信サーバ
データベース
APIサーバ
クライアント

ユーザデータ



- ユーザの状態を表すデータ
- サービス運営上、最も大事な資産
- APIサーバを通じてクライアントに配信される
- クライアントローカルの暗号化SQLに保存される場合もある(設計次第)
- 次のようなデータが格納される
  - ユーザに紐づく情報
  - ユーザが取得したキャラクタ
  - 取得した武器
  - 課金情報など

APIサーバ

- クライアントからの**要求**に**応答**する
  - リクエスト/レスポンス
- データベースの内容を参照したり書き換えたりする
  - マスタデータ
  - ユーザデータ

詳しくはデータ設計の回!!

質問タイム

プロダクトに**バグ**が発生しました

サーバ

アセット配信サーバ

ここのバグ、
サーバじゃない?

いや、クライアントでしょ

データベース

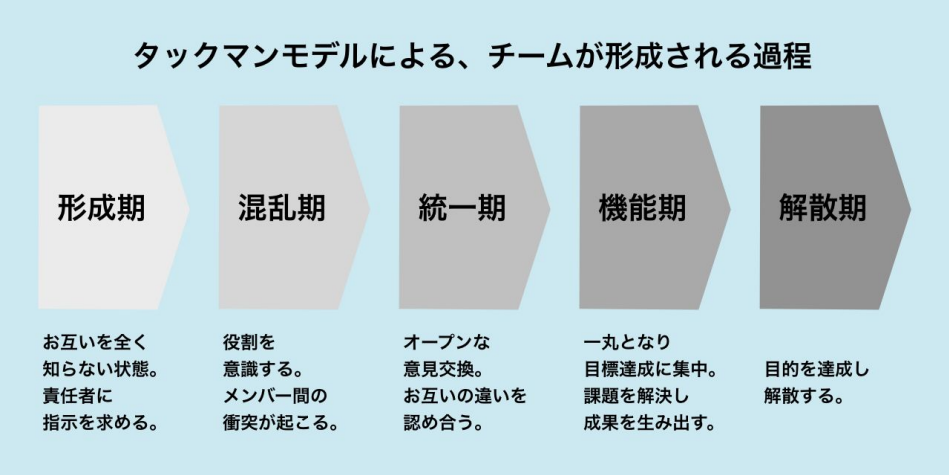
APIサーバ

クライア


Admin
データベース
APIサーバ
クライアント

チームビルディング

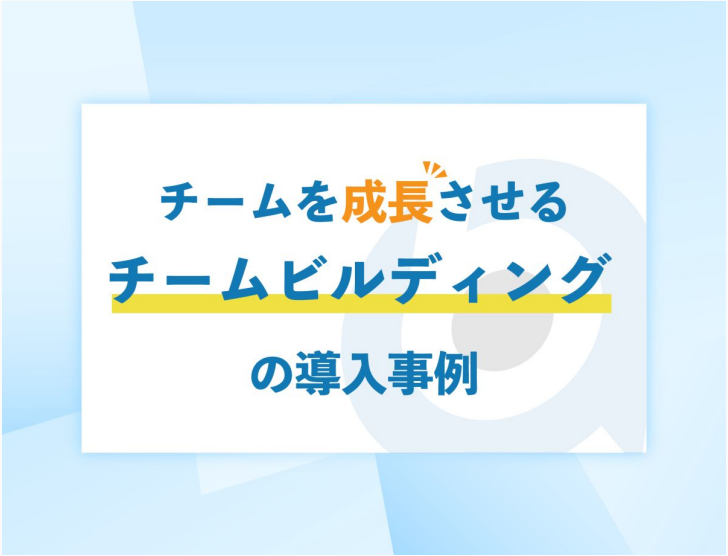
チームビルディングとは、強い組織を構築していくための手法。チームメンバーのスキルや能力、経験などを最大限に引き出し、目標を達成できるチームづくりに取り組む。



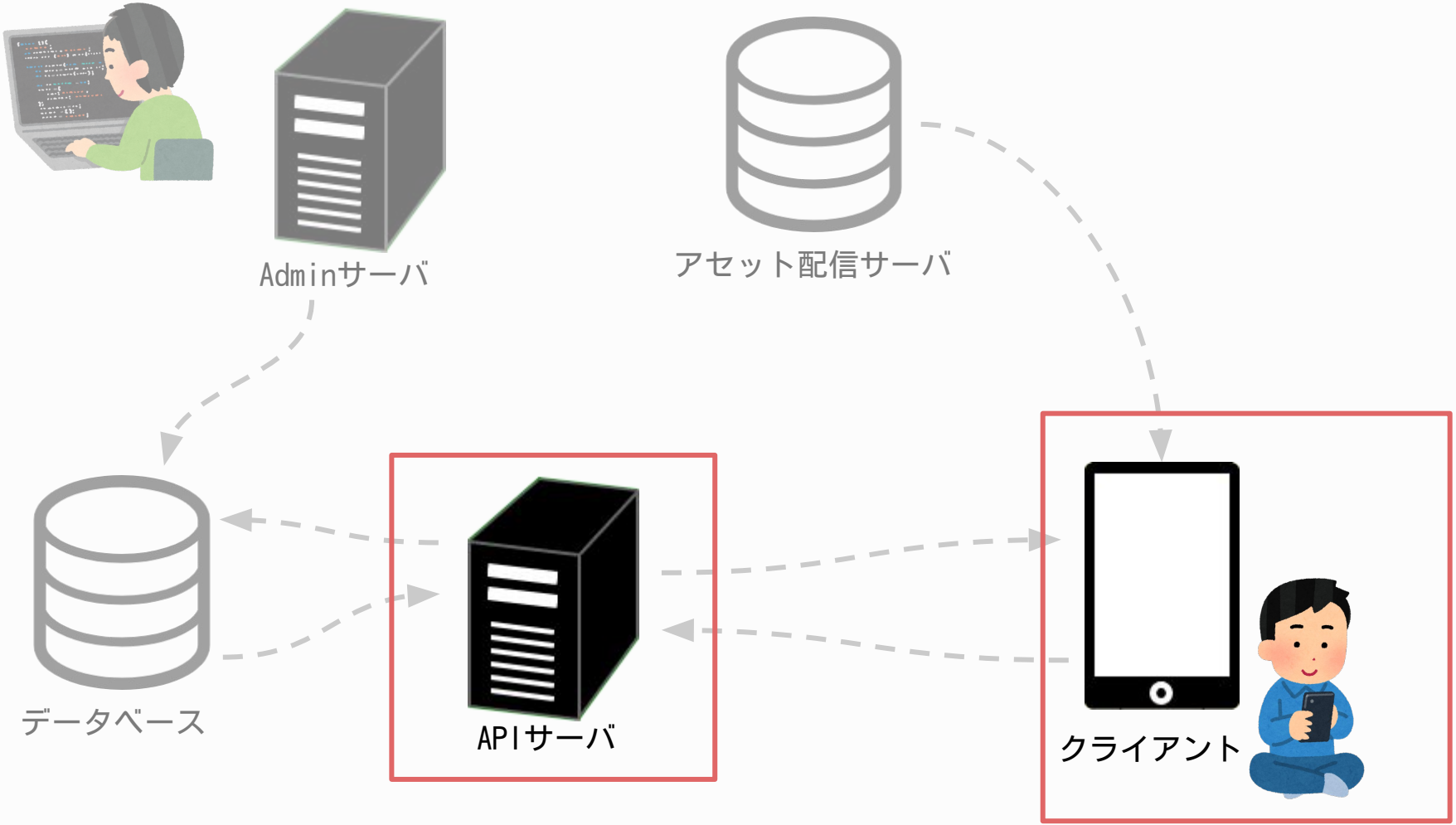
なぜチームビルディング?

チームビルディングのメリット

- コミュニケーションの活発化
- メンバーのモチベーション向上
- 新たなアイデアが生まれやすい
- チームの生産性が向上



栗村さんのてっくぼっと記事もあるのでぜひ!!


Adminサーバ
アセット配信サーバ
データベース
APIサーバ
クライアント

システム設計（アーキテクチャ）

- MVC
- 3層（レイヤード）アーキテクチャ
- ヘキサゴナルアーキテクチャ
- オニオンアーキテクチャ
- クリーンアーキテクチャ
- ドメイン駆動設計(DDD)

なぜアーキテクチャを取り入れる必要があるのか?

個人的な意見ですが、

- ソースコードに秩序が生まれる
  - 保守性、拡張性が格段に上がる
- レビューコストが下がる
  - 統一した思想で実装できるのでレビューの指標を統一できる

取り入れないと..（極論ですが...）

- プロジェクト独自のルールが出来上がったり、
- 好きなところに好きなコード書いて保守性がなくなる

詳しくはシステム設計の回で!!

質問タイム

データ構造

C#標準

- Dictionary<TKey,TValue>
- Hashtable
- Stack<T>
- Queue<T>
- List<T>
- Array
- ArrayList
- ImmutableList<T>

まだまだいっぱいある...

なぜ学ぶ必要があるのか?

- 適切にデータ構造を選択できる
- 特定要素へのアクセス速度、メモリ確保量を把握できる

要素数	List<int>	Dictionary<int,int>
1000	4	21.8
10000	39.1	197.6
100000	390.7	424.4

単位(KB)

詳しくはデータ構造の回で!!

質問タイム

アルゴリズム

アルゴリズム：ソートアルゴリズム、探索アルゴリズム etc…

アルゴリズム評価の観点

- 計算量（オーダー）
- メモリ使用量
- コーディングのしやすさ（可読性）

なぜ学ぶ必要があるのか?

- オーダーを正しく理解すれば、計算量の小さいアルゴリズムが選択できるようになる
- ソートや探索は言語側で提供されている場合もあるが、なぜそれを選んだか言葉で説明できる必要がある
- コードの良し悪しを判断し、効率の良いプログラムが書ける

詳しくはアルゴリズムの回で!!

質問タイム

パフォーマンスチューニング

パフォーマンスチューニングとは、システムの処理性能や信頼性を高めるために、システムの動作環境を最適化すること。

パフォーマンスチューニングを実施することで、システムの性能を最大限に生かし、安定稼働させられるようになる。

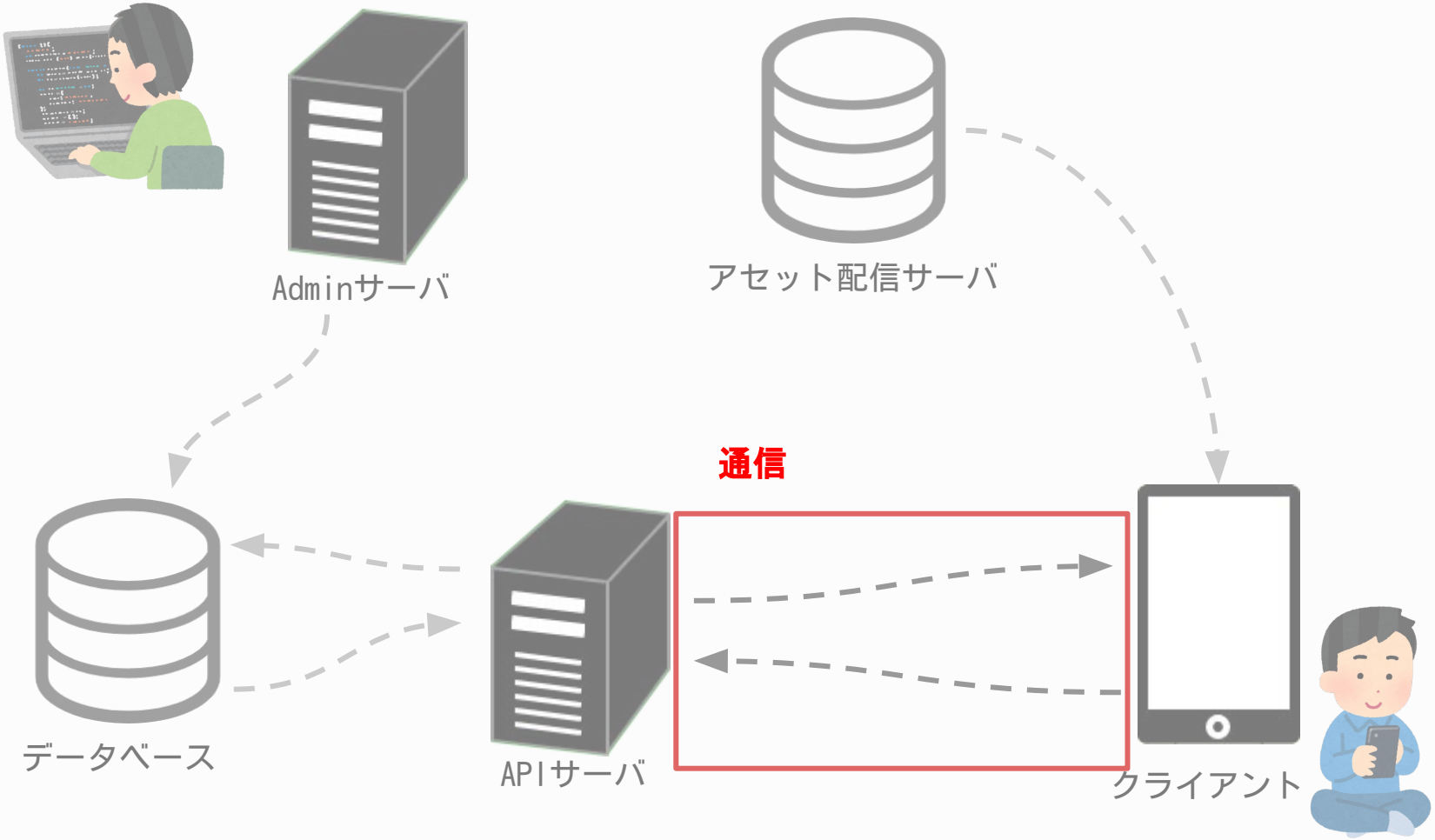
負荷の高い処理を見つけ、軽減する。

サーバ・クライアントのパフォーマンスチューニング

- サーバ
  - 負荷検証（パフォーマンスのテスト）
  - DBへのアクセス
  - etc…
- クライアント
  - CPU・GPUの負荷軽減
  - 描画処理系
  - メモリ・GCのアロケーション
    - 文字列結合
  - etc…

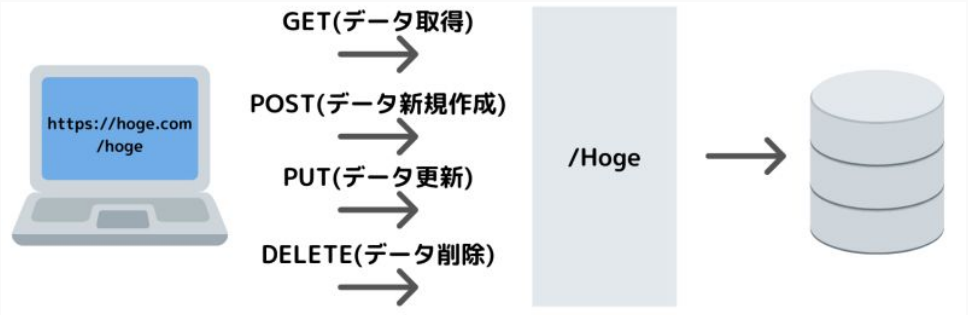
詳しくはパフォーマンスチューニングの回で!!

質問タイム


Adminサーバ
アセット配信サーバ
通信
データベース
APIサーバ
クライアント

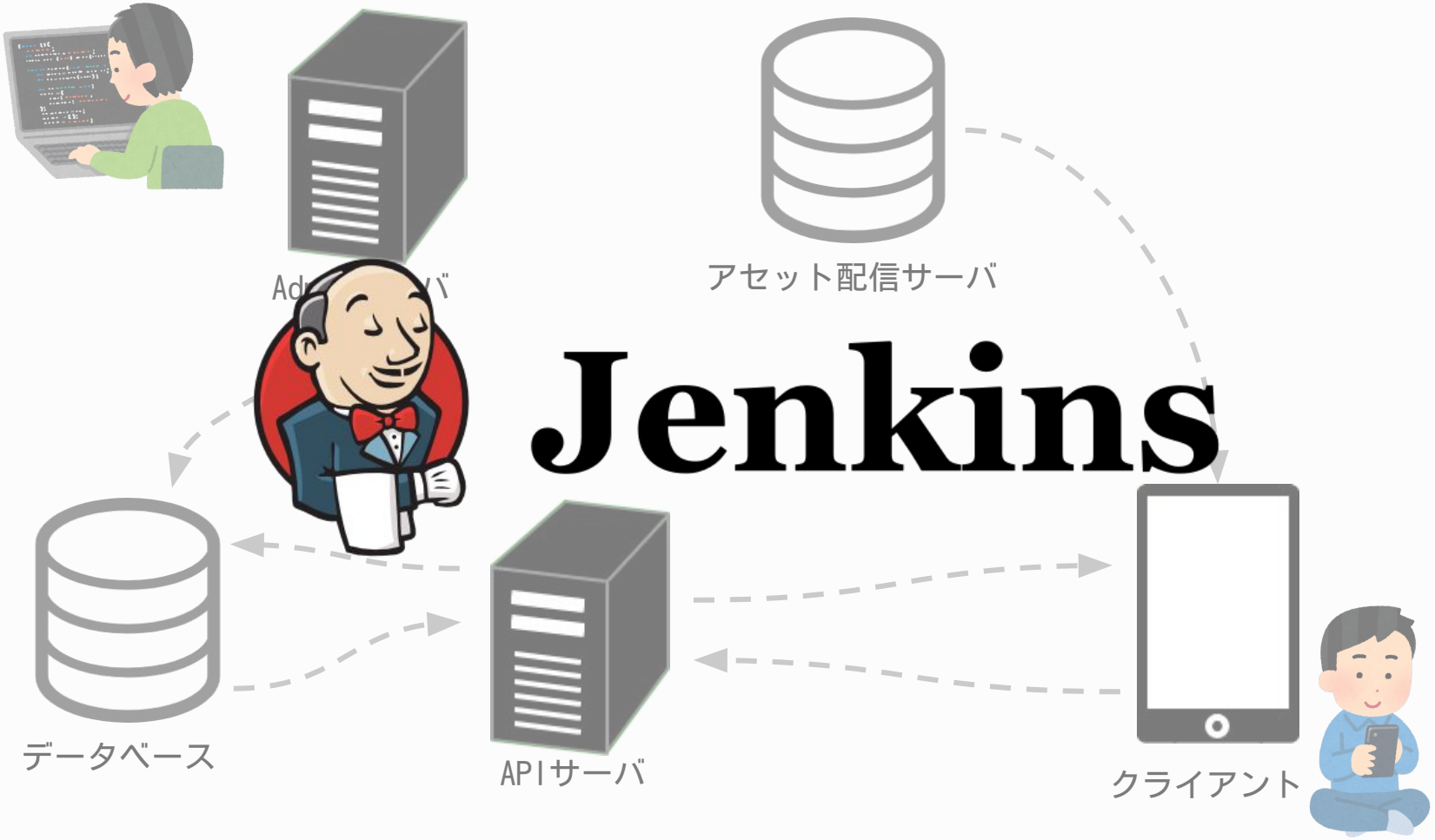
通信

- サーバ、クライアントで情報の受け渡しをするための通信
- ゲームではHTTP（REST）通信、gRPCなどで実装される
- 通信で送受信する際のバイナリ方式
  - HTTP
    - json
    - proto buffer
    - etc…
  - gRPC
    - Protobuf



詳しくは通信の回で!!

質問タイム


アセット配信サーバ
Jenkins
データベース
APIサーバ
クライアント

CI/CD

CI（継続的インテグレーション）

- コードに変更があると、ビルドからテストまで自動化する手法
- 即時に問題を発見でき、手戻りを最小限に抑え、結果的に開発にかける時間を削減できる

CD（継続的デリバリー）

- テストをパスしたソフトウェアを自動で実稼働環境にリリースできる状態にする手法
- 成果物を顧客にいち早く提供し続けることができる

Jenkins（無料）やGitHub Actions（有料）などで実現されることが多い

テスト

ソフトウェアテストとは、

- 作ったソフトウェアが正しく動作するか
- 求められている仕様を満たしているか
- 意図しない動作をしないか

を確認する作業のこと。

また、それを証明するために、仕様にない動作やバグをできる限り多く見つけることを目標とする場合があります。

どんなふうに使われているか

- テストコード
  - Unity: UnityTestRunner、NUnit、画面単体テスト
  - Golang: testingライブラリ
- Jenkinsやgithub actionsなどで継続的にテストを回す(CI/CD)
  - 理想の状態はPRの状態でエラーを討ち取りたい
  - devに入ってからあれダメだったとかは割と効率が悪い

詳しくはテストの回で!!

質問タイム

UNIXコマンド

- more（file）
  - ファイルの内容表示(ページごとに止まる)
- less（file）
  - ファイルの内容表示(スクロール操作できる)
- lv（file）
  - ファイルの内容表示(ページごとに止まる＋α)
- head（file）
  - ファイルの先頭10行を表示
- tail（file）
  - ファイルの末尾10行を表示
- grep（検索文字列）（file）
  - 指定文字列がある行だけを表示

まだまだいっぱいある...

質問タイム

まとめ

- モバイルゲーム開発で必要な知識をゆるふわ説明
- 今後のお稽古が仕事につながるイメージを持っていただけると!