コンテナ技術による
小規模タスクの継続的運用手法

DataCenter とソフトウェア開発ワークショップ
2016/2/18 松江テルサ
株式会社クルウィット 清原智和

自己紹介

- 清原智和
  - 株式会社クルウィット 所属
  - JAIST 博士前期 修了

もくじ

- データセンター制御システムの要件
- 要件を実現するための実装例
- 実装時に発生した課題と解決方法

もくじ

- データセンター制御システムの要件
- 要件を実現するための実装例
- 実装時に発生した課題と解決方法

データセンター制御の特徴

- 制御対象が多様である
- 制御対象が多量である
- 制御対象が変動する
- 制御期間が長期間にわたる

データセンター制御システムに求められる特徴

- スケーラビリティ
  - 異種透過性(制御対象の種類を容易に拡張できること)
  - 規模透過性(制御対象の量を容易に拡張できること)
- 安定性・保守性(長期にわたって制御できること)

もくじ

- データセンター制御システムの要件
- 要件を実現するための実装例
- 実装時に発生した課題と解決方法

要件を実現するための実装例

- PrairieStack Infrastructure
  - 採用したアーキテクチャ
  - 採用したフレームワーク

要件を実現するための実装例

- PrairieStack Infrastructure
  - 採用したアーキテクチャ
  - 採用したフレームワーク

PrarieStack の詳細については
別資料を参照の程

要件を実現するための実装例

- PrairieStack Infrastructure
  - 採用したアーキテクチャ
  - 採用したフレームワーク

PrairieStack Infrastructure で採用したアーキテクチャ

- タスク＆ワーカーモデル
  - タスクとワーカーからなる処理フローを採用（詳細後述）
- ワーカー新陳代謝モデル
  - ワーカーの新陳代謝を実現するワーカー管理法を採用（詳細後述）

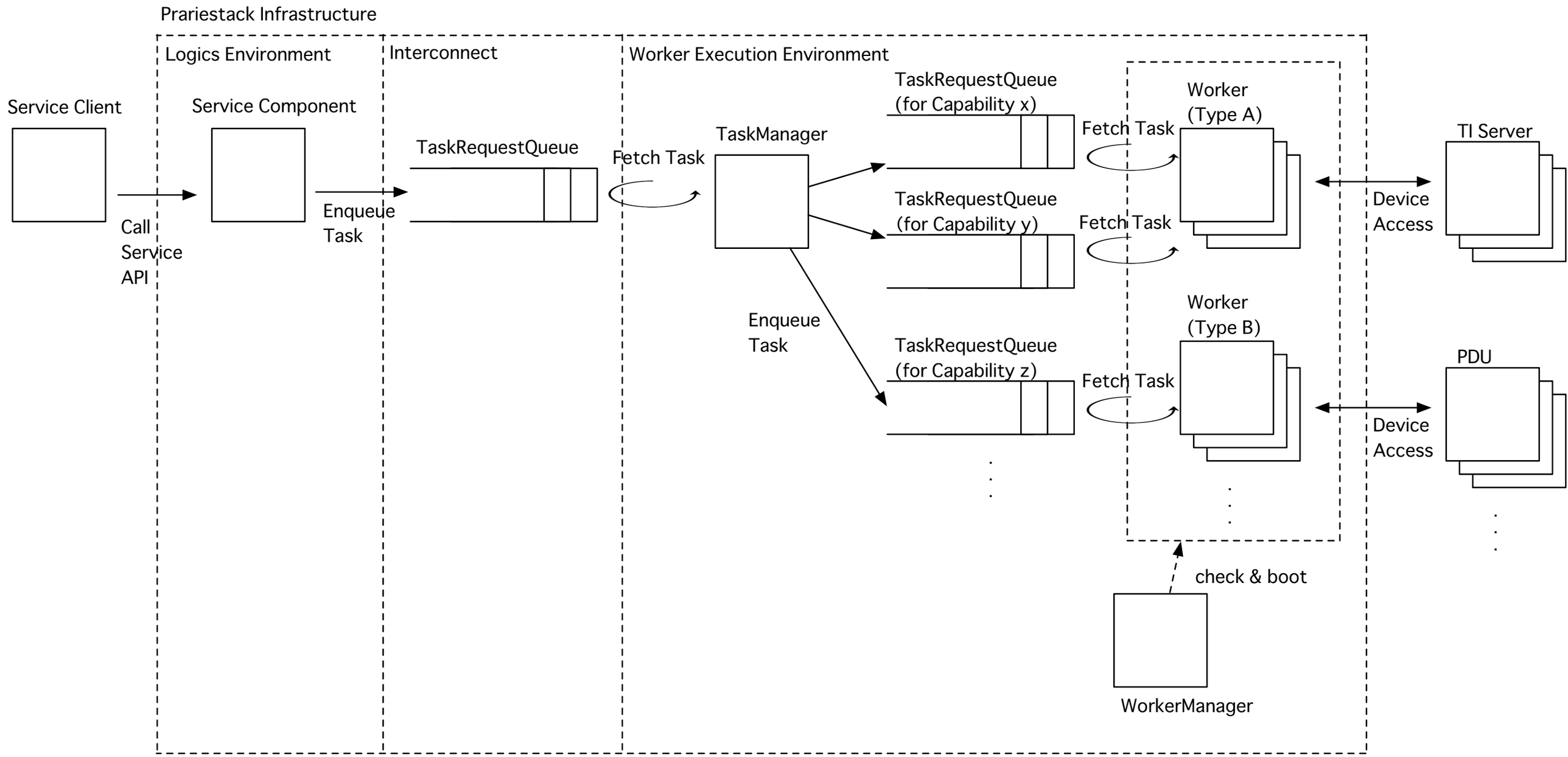
PrairieStack Infrastructure で採用したアーキテクチャ

- タスク＆ワーカーモデル
  - タスクとワーカーからなる処理フローを採用（詳細後述）
- ワーカー新陳代謝モデル
  - ワーカーの新陳代謝を実現するワーカー管理法を採用（詳細後述）

タスク＆ワーカーモデル
用語

- タスク
  - データセンター制御にかかる処理のうち連続性を要する処理をひとまとめにしたもの
    - 制御内容に応じて複数の種類が存在する
- ワーカー
  - タスクを実行する主体
  - 制御対象の種類・量に応じて種類・量を用意する

タスク＆ワーカーモデル
処理フロー図



タスク＆ワーカーモデル
採用理由

- 異種透過性を高めるため
  - 制御対象種の追加（≒ワーカー種の追加）が容易になる
- 規模透過性を高めるため
  - 制御対象量の変更（≒ワーカー量の変更）が容易になる
- 保守性を高めるため
  - 部分的・段階的モジュール更新が容易になる

PrairieStack Infrastructure で採用したアーキテクチャ

- タスク＆ワーカーモデル
  - タスクとワーカーからなる処理フローを採用（詳細後述）
- ワーカー新陳代謝モデル
  - ワーカーの新陳代謝を実現するワーカー管理法を採用（詳細後述）

ワーカー新陳代謝モデル
実現方法

- ワーカーを定期的に停止する
  - ワーカー自身がタスク実行後に自主的に停止する
- ワーカーを定期的に起動する
  - ワーカーを起動するプログラムを個別に動作させる
    - 指定したワーカーイメージで起動する
    - 必要な量を維持するように起動する

ワーカー新陳代謝モデル
採用理由

- 保守性を高めるため
  - ワーカーの差し替え（バージョンアップ）が容易になる
- 安定性を高めるため
  - 想定外の動作をするワーカーに対するシステム全体の耐障害性を高められる

要件を実現するための実装例

- PrairieStack Infrastructure
  - 採用したアーキテクチャ
  - 採用したフレームワーク

PrairieStack Infrastructure で採用したフレームワーク（1/2）

- Docker
  - アプリケーション実行環境をイメージとして作成・配付・実行するためのフレームワーク
  - 異種透過性に優れている

PrairieStack Infrastructure で採用したフレームワーク（2/2）

- Kubernetes (k8s)
  - Docker コンテナのオーケストレーション技術を提供するフレームワーク
  - 規模透過性に優れている

もくじ

- データセンター制御システムの要件
- 要件を実現するための実装例
- 実装時に発生した課題と解決方法

ワーカー新陳代謝機能を実装するにあたって発生した課題

- k8s 上で泡沫的な Docker コンテナを扱う場合に制限が発生した
  - k8s は長時間動作する Docker コンテナを前提としている
  - 短時間で終了する Docker コンテナを扱う場合には制限が発生する（詳細後述）

k8s 用語

- Pod
  - k8s で Docker コンテナを管理する際の最小単位
  - 以下の情報を含む
    - Docker コンテナのイメージ名
    - Docker コンテナの取り扱い方法を定める各種パラメータ
- Docker コンテナが終了しても Pod は残る
  - 終了コードやエラー情報などが保存されている

k8s が提供する再起動の仕組みを利用する場合（1/2）

- Pod の設定として "Restart Policy = Always" と指定する
  - Docker コンテナが終了すると k8s が自動的に再起動してくれる
    - 最新の Docker コンテナで起動するよう指定することも可能
- ただし、あくまで異常発生時のリカバリ機能という位置づけ（後述参照）

k8s が提供する再起動の仕組みを利用する場合（2/2）

- 制限
  - 条件を満たさないコンテナは再起動までに遅延をいれられる
    - 条件を満たさない＝10分未満で終了する
    - 初回は即時に再起動だが再起動を繰り返す毎に遅延時間が増大していく（最大で10分）
    - 条件や遅延時間は変更不可

独自に再起動の仕組みを実装する場合（1/2）

- Pod の設定として "Restart Policy = Never" と指定する
  - k8s の再起動の仕組みを無効化
- Pod 監視プログラムを追加で動作させる
  - Docker コンテナが動作中の Pod 数をチェック
  - 動作中の Docker コンテナが意図した数となるように Pod を追加する

独自に再起動の仕組みを実装する場合（2/2）

- 制限
  - 運用時間に伴って Pod 追加パフォーマンスが悪くなる
  - k8s の再起動の仕組みを利用する場合と異なり Pod が増加し続ける
  - Pod が増えれば増えるほどPod 追加パフォーマンスが悪化
    - Pod 10 件で新規 Pod 追加 → 1秒未満
    - Pod 100,000 件で新規 Pod 追加 → 12 秒程

さらに追加で独自に再起動の仕組みを実装する場合（1/2）

- （前項の仕組みに加えて）
  Docker コンテナが停止している Pod を定期的に削除する仕組みを追加する

さらに追加で独自に再起動の仕組みを実装する場合（2/2）

- 制限
  - Pod の削除中に新規 Pod 追加ができない（削除処理が完了するまで遅延させられる）
  - Pod 数が増大すると Pod の削除処理時間が増大する
    - Pod 10 件で 1 Pod を削除する → 1秒未満
    - Pod 100,000 件で 1 Pod を削除する → 2秒程
      （N Pod 消すときはこれを繰り返し）

さらにさらに追加で検討したこと

- Pod を残さないようにする（≒ 内部の Docker コンテナが停止したら Pod も自動的に消える）オプションあるのでは？
  - ない（将来的に増えそうな気配はある）
- （遅延しないよう）k8s の再起動機能ではなく直接 Docker コンテナを再度起動させられるのでは？
  - 停止した Docker コンテナを再起動する方法は提供されていない

    （将来的にサポートされそうな気配はある）

採用した方法（1/2）

- k8s が提供する再起動の仕組みを利用した
- 加えてワーカーの最低寿命を10分とした
- その間は複数のタスクを処理するようにした

↓

パフォーマンス劣化を最小化できた

採用した方法（2/2）

- 制限
  - ワーカーの実装が複雑化（複数回タスクを処理）
  - 新陳代謝の速度に制限（最大で10分待つ必要）

まとめ

- データセンター制御システムの要件をまとめた
- 要件を実現するための実装例を示した
  - タスク＆ワーカーモデル
  - ワーカー新陳代謝モデル
- 実装時に発生した課題と解決方法を紹介した

補足1 Replication Controller について

- 起動している Docker コンテナの量を「動的に変動させる」＋「維持させる」仕組み
    - 「維持させる」仕組みとしては前述の再起動の仕組み（Pod の設定で "Restart Policy = Always" を指定する方法）をそのまま使っている
        - 制限も同様に適用される（＝再起動の遅延）
        - Docker コンテナを意図的に再起動させるための仕組みではない
- PrairieStack Infrastructure のワーカーは Replication Controller を通じてコントロールしている

補足2 Job について

- 複数の Docker コンテナに「処理」をさせて結果を集計する仕組み
  - トータル実行数と並列実行数を指定することが出来る
- Exit Code が 0 以外で終了した Docker コンテナをリトライさせる機能がある
  - 外部要因による異常発生に対応するための機能
  - Docker コンテナを意図的に再起動させるための仕組みではない