IDCフロンティア
ネットワーク運用システムの課題と挑戦

株式会社IDCフロンティア
見崎 徳仁
2019年1月25日

アジェンダ

- トラフィックの可視化
  ～スケールアウト重視のモニタリングシステムの構築～
- ネットワーク作業の自動化
  ～Network CI/CDへの挑戦～

IDCFrontier

トラフィックの可視化

～ スケーラビリティ重視のモニタリングシステムの構築 ～

IDCFrontier

IDCFのモニタリングシステムで発生した問題

- SNMPのポーリングが回りきらない
  - 複数台Stack構成の機器や、多数の論理インターフェースを持つ機器で詰まった
  - 秘伝のタレ満載の独自システムだったためデバッグが困難
- RRDToolベースのシステムでストレージのI/O性能不足
  - RRDCachedを使いたくても使えないシステムだったり
- シングル構成のシステムが乱立しすぎて管理しきれない
  - 複数のシステムに重複して登録していたり
  - どこにも登録されていなかったり
  - ハードウェア故障で一網打尽になったり

あまりに無残な状態だったので一からやり直し

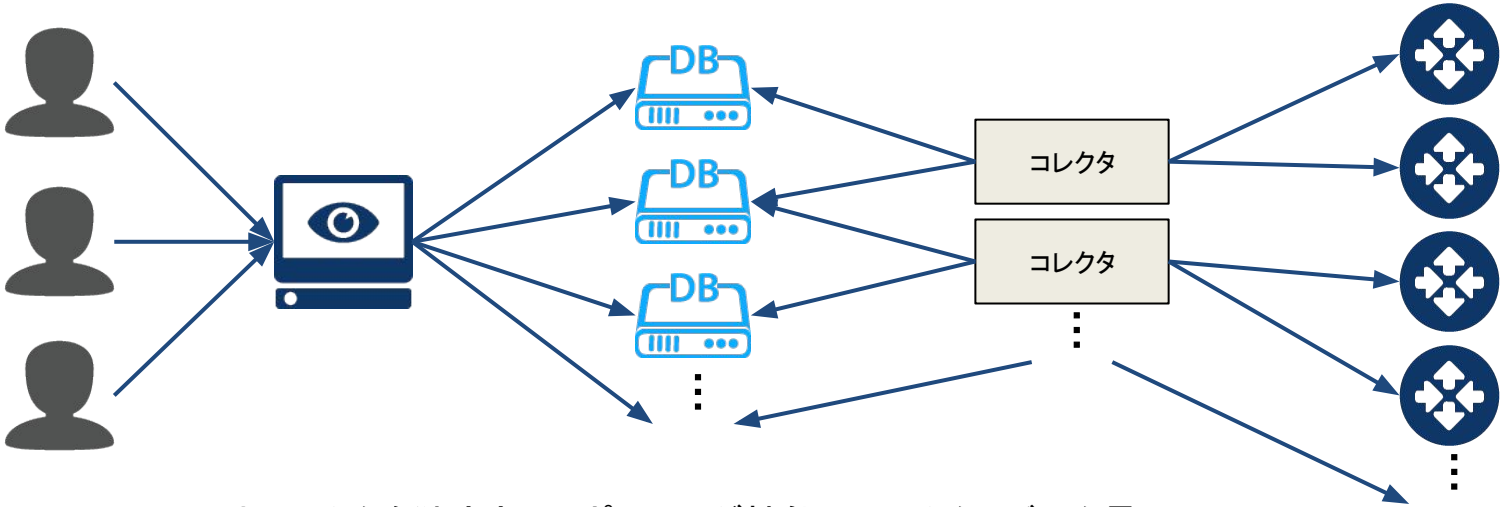
IDCFrontier

スケーラビリティと可用性の向上が求められた

- サービス提供用NW機器全てのトラフィックを取得する
  - ノード数：1,200台～
  - インターフェース数：100,000～
  - 取得間隔：5分以内
  - 利用プロトコル：SNMP
- 目的のトラフィックグラフを探しやすくする
  - 多数のシステムから目的の機器を探すというのはNG
- システムの冗長化を行うこと
  - DRの観点から東西で冗長できていることが望ましい

IDCFrontier

DBとコレクタは規模の透過を
可視化はデータの位置の透過を目指す

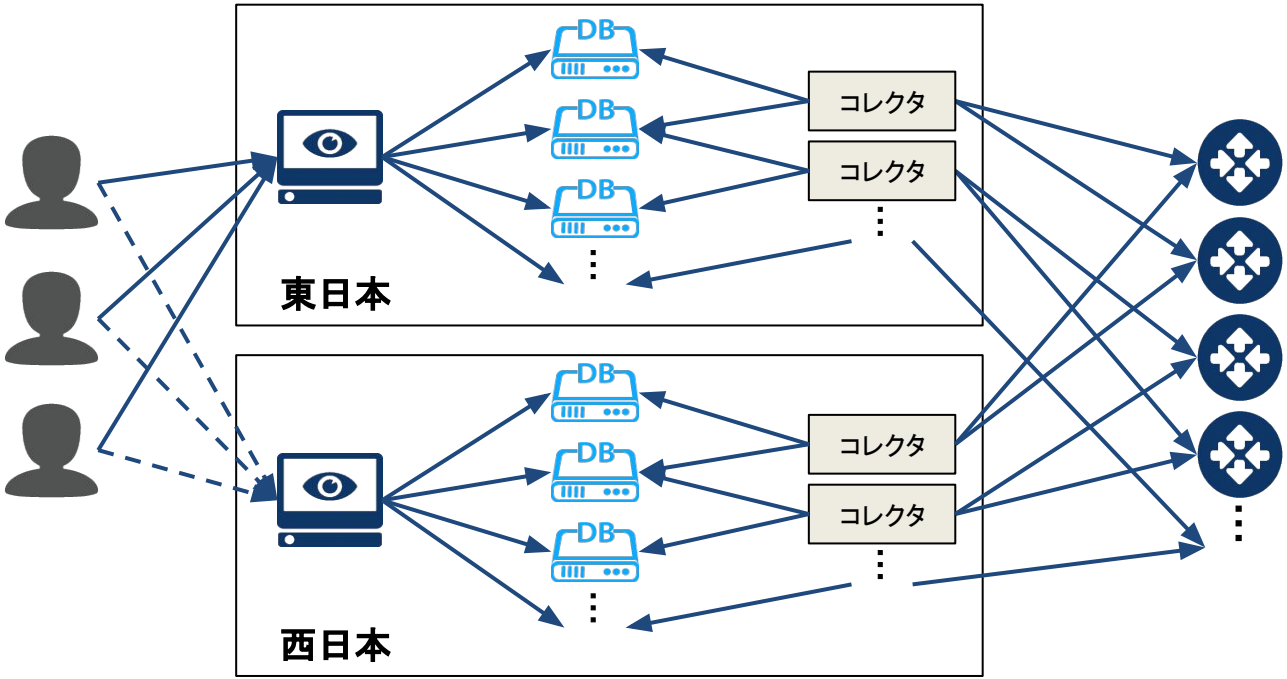


DBとコレクタを独立させ、ポーリング性能はコレクタ、データ量は DBのスケールアウトで対応できるようにする

IDCFrontier

可用性の向上を行うために

東西で独立して同じ構成で動作させる(Act-Act)

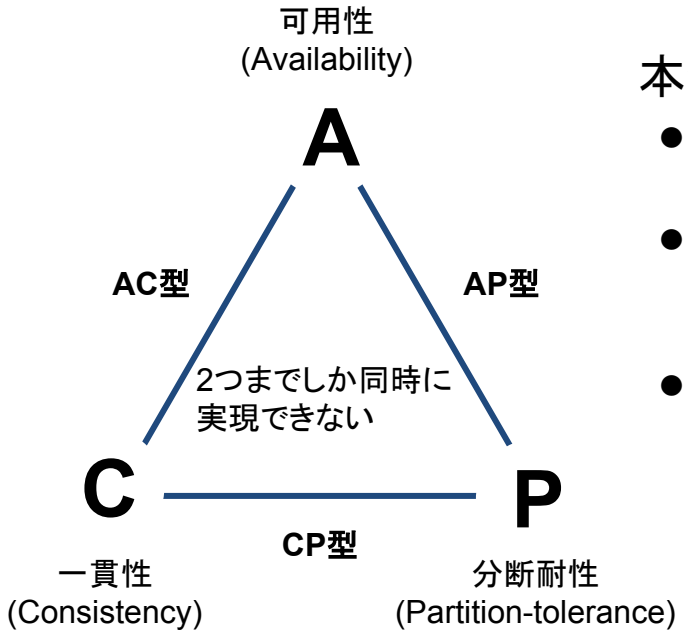


東西で大量のデータのレプリケーションは大変なので諦める

IDCFrontier

(参考)CAPの定理

分散システムにおいて可用性・一貫性・分断耐性の全てを同時に満たすことはできない



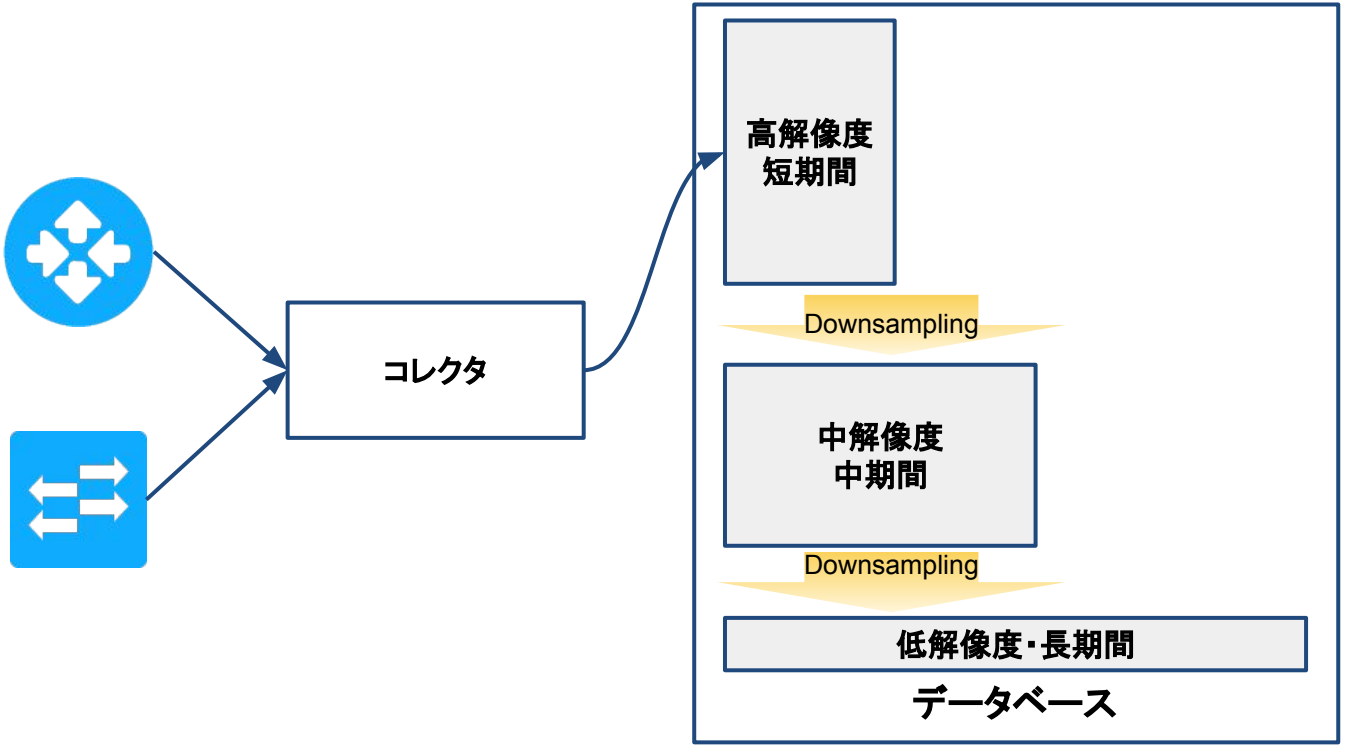
本システムの考えをここに当てはめるとAP型と言える

- 可用性
  - 単一障害点を作らず、どこかでデータが取れていることを重視
- 分断耐性
  - ネットワーク障害時でも極力影響を受けないよう、東西で分断されても動き続けることを重視
- 一貫性
  - その時点でのデータが見えればよく、全部の系で全く同じタイムスタンプのデータである必要はない

IDCFrontier

無理なくデータを保持するために

高解像度データは短く保持、低解像度データで長く保持



※解像度: 単位時間あたりのサンプル数

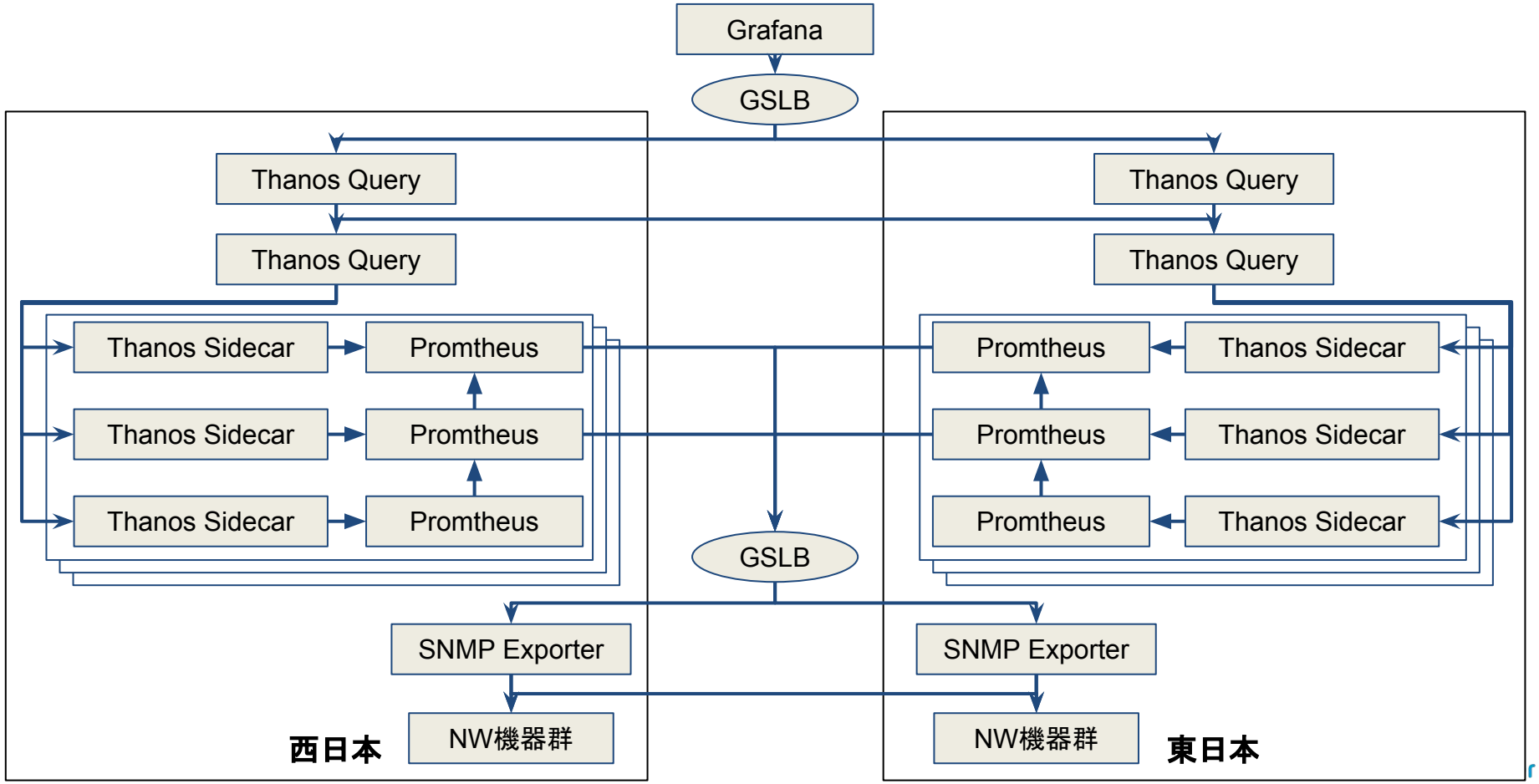
IDCFrontier

利用したツール

- Prometheus
  - 時系列DBを中心としたモニタリングツール
  - DBとコレクタが独立して動作し、コレクタが収集したデータをDB側からHTTPでスクレイピングするというユニークな仕様
  - クラスタリング機能を持たないが、他のPrometheusからデータを収集する仕組みを持っている
- Thanos
  - Prometheusと連携して動作するツール
  - 複数のPrometheusに対してクエリを投げて結果を集約することができる
  - オブジェクトストレージにPrometheusのデータをエクスポートすることも可能
- Grafana
  - おなじみのダッシュボード

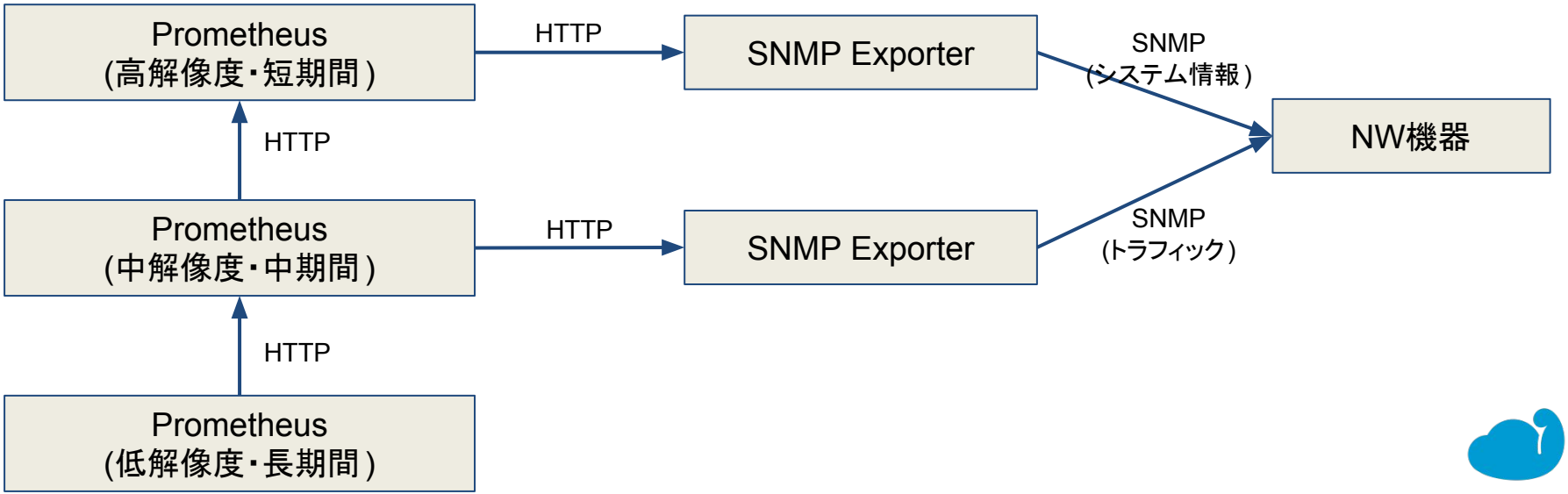
IDCFrontier

モニタリングシステムの構成(全体)



データの収集と蓄積

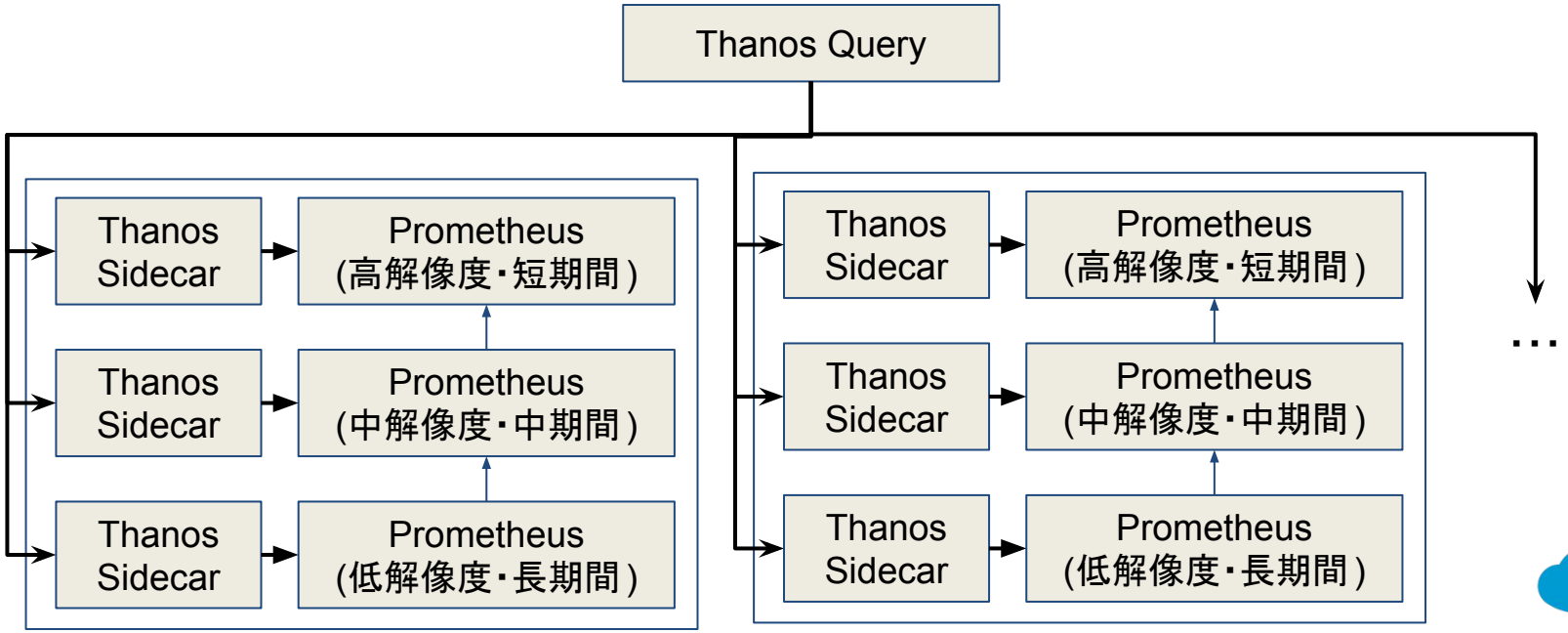
- 取得したい解像度のPrometheusでメトリクス収集を行う
- 長期間保存用にPrometheus間でダウンサンプリングを行う
- 解像度×保存期間の値を揃えてストレージのキャパシティ設計を容易にする



IDCFrontier

データの規模と位置の透過

- 監視対象の増加に合わせてクラスタ(Prometheusのセット)を増設
- Thanos(Query/Sidecar)を利用し、複数のクラスタ・Prometheusを透過してクエリを実行する

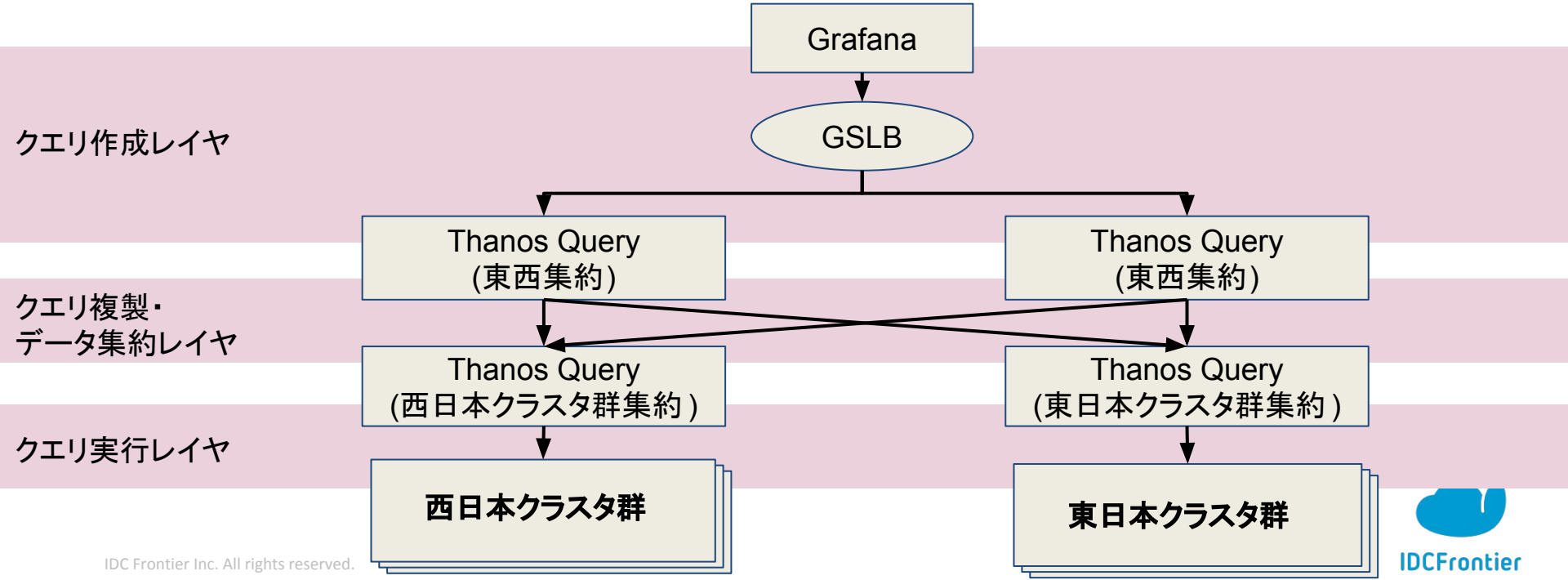


IDCFrontier

可用性の向上

Thanos Queryを2層配置し、東西全てのPrometheusからデータを集約

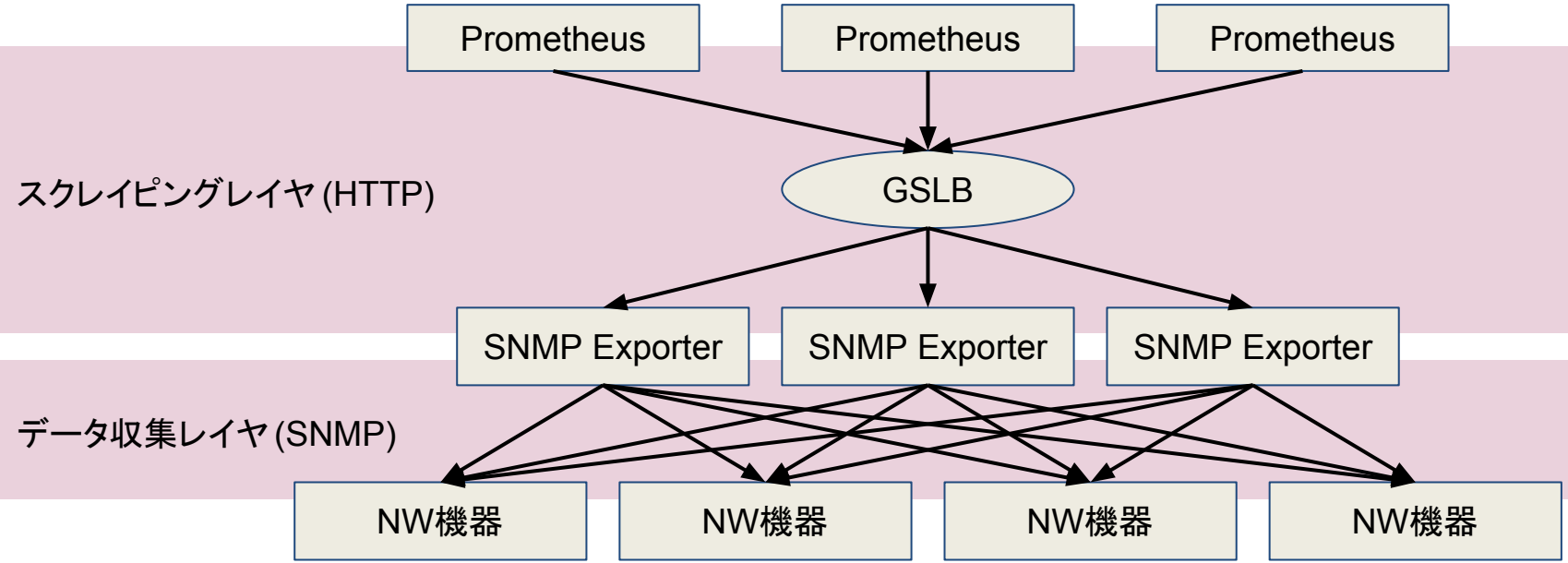
- GrafanaからのクエリはGSLBで東西のThanos Queryに分散
- 東西を集約するThanos Queryを設置することで東西クラスタ群を透過



IDCFrontier

コレクタの規模の透過

- NW機器へSNMPのリーチャビリティがある場所にSNMP Exporterを複数設置
- GSLBを用いることで負荷分散とSNMP Exporter群の規模を透過
- NW機器の増加と共にSNMP Exporterを増設していく



IDCFrontier

うまくいったこと

- 全てのコンポーネントで水平方向のスケールアウトが可能な構成が作れた
- 全てのPrometheusをThanosで透過させているため、トラフィックグラフを利用する人は機器がどこに収容されているかを気にする必要がない
- トラフィックグラフの可用性が向上した
    - 東西で同じPrometheusが同時に落ちない限りグラフの見た目への影響がない
    - 同一の解像度のPrometheus群が全て落ちても、低解像度のデータによってグラフが補完される

IDCFrontier

要注意なポイント

- どこかが落ちてもトラフィックグラフ上は正常に動いているように見えるため、システムの監視をちゃんとしないと落ちていることに気づかない
- クラスタの増設に伴い無駄なクエリが増加する
  - 現状問題にはなっていないが、キャッシュの機能があると嬉しいかも
- （余談）グラフの利用者はGrafanaしか意識しなくなるので「Grafanaに〇〇の機器を追加して欲しい」という依頼に辟易する

IDCFrontier

現状の悩み

- ダッシュボードをどう作っていくのかのノウハウがない
  - 「必要な情報をわかりやすい形で表示する」というのが意外と難しかった
- クエリのコントロール
  - 大量のデータを参照するクエリを投げてPrometheusが落ちたことは数知れず
  - 「自分が見やすいようにGrafanaのダッシュボードを作って」と言い切れない…
- コストのコントロール
  - 機器の追加に伴いコレクタやDBが増えていくだろう
  - 利用者や利用用途の増加に伴い必要なメモリが増えていくだろう
  - 必要なメトリクスの追加に伴い必要なストレージが増えていくだろう

IDCFrontier

モニタリングシステムの今後

- 収集したデータの活用
  - 大量にデータを収集してもグラフとして見られるのはほんの一握りなのはもったいない
  - 様々なデータを複合的に用いることで、効率のよいアラートの作成ができそう
- 脱SNMP(API/Telemetryの利用)
  - より細かいデータを収集できるようにし、高解像度化を進めたい
  - SNMPでは取得できないデータを収集し、実機でなければ確認できないことを減らしたい

IDCFrontier

ネットワーク作業の自動化

～ Network CI/CDへの挑戦 ～

IDCFrontier

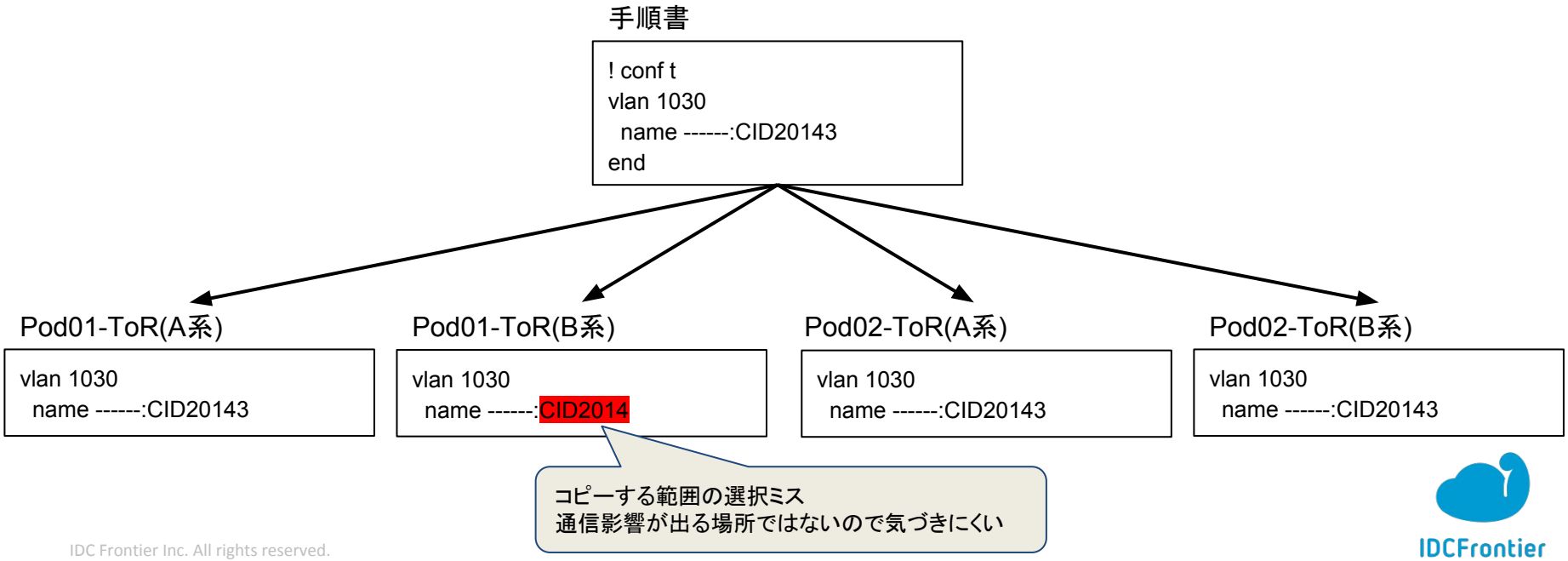
IDCFにおけるネットワーク自動化の実情

- 会社・部署としてのノウハウがない
  - スクリプトをかける人はちらほらいるが、およそソフトウェア開発とは程遠い
  - 単発の情報収集であればそれでも良いが、作業の自動化にはなかなか繋がらなかった
- 自動化に集中して取り組める人がいなかった
  - (当たり前だけど)設備の新設・増設、保守対応などに注力
  - 自動化はオプション的な扱いのため、優先度が低く後回しになってしまう
  - 作業単位でみると手作業でなんとかなってしまう

IDCFrontier

手作業を続けることのリスク

「手順書をコピペするだけの作業」も
ミスなくやり続けるのは難しいと認識しましょう



IDCFrontier

プログラムを書くネットワーク作業の自動化は避ける

- 「プログラムが書ける≠ソフトウェア開発ができる」
  - 自動化もれっきとしたソフトウェア開発であるため、そこには様々なデザインパターンやアンチパターンが存在するはず
  - ただしそれを習得・実践するためのハードルは高い…
- 他分野のノウハウを活用して、確実に・楽に・早く自動化を進めたい
  - 自分で稚拙なプログラムを書いて失敗するより、公開されているノウハウ・ツールを応用した方が良いと割り切った

IDCFrontier

自動化のスローガン

あらゆること(everything)を自動化しよう

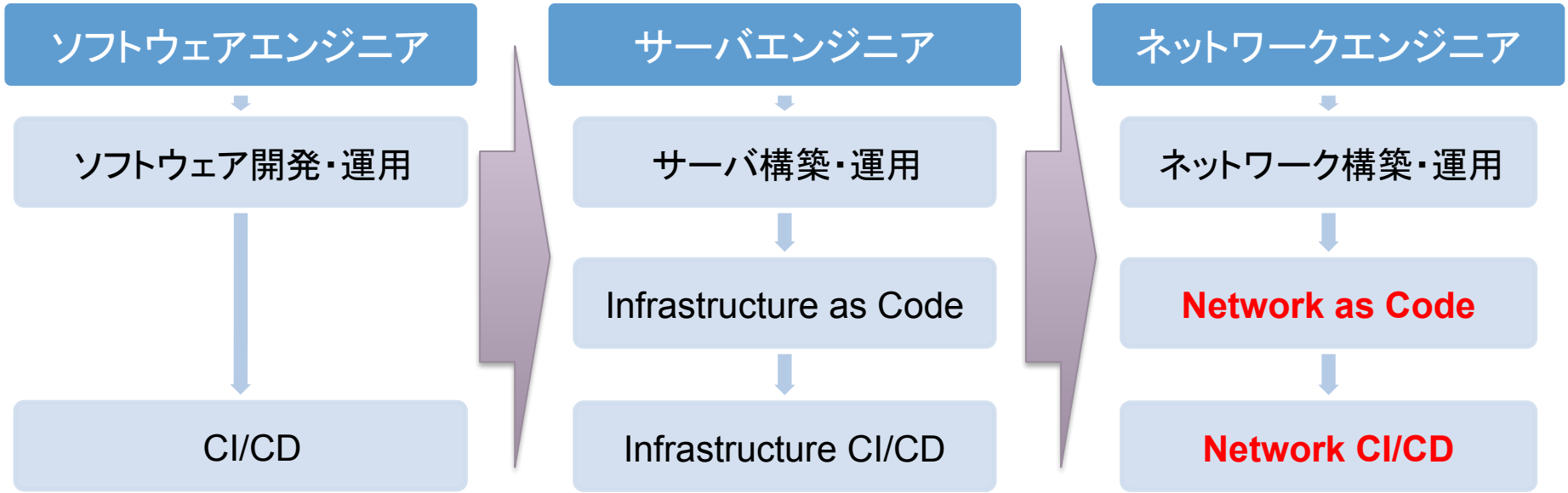
でも

「あらゆる方法(everyhow)で」はやめよう

IDCFrontier

目指したネットワーク自動化の姿

ネットワークをコード化し他分野のノウハウを活用



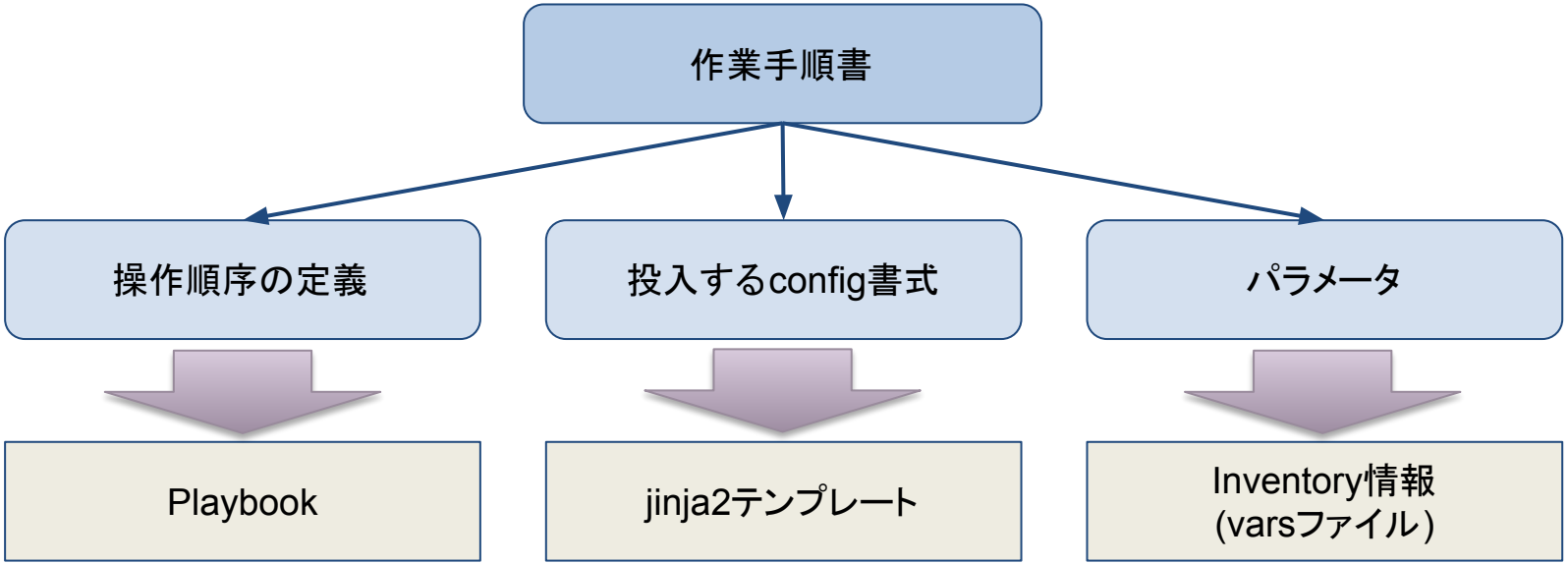
IDCFrontier

利用したツール

- GitLab
  - オープンソースの Gitリポジトリマネージャ
  - 機能が豊富
- GitLab CI
  - GitLab標準で利用可能なCIツール
  - 標準機能だけあってGitLabの機能をCIの中で活用できる
- Ansible
  - おなじみ構成管理ツール
  - ネットワークモジュールがどんどん充実してきている
  - ネットワーク自動化のデファクトスタンダードになってきている感がある
- Docker
  - おなじみのコンテナ環境
  - Ansibleの実行やテストの実行環境として利用

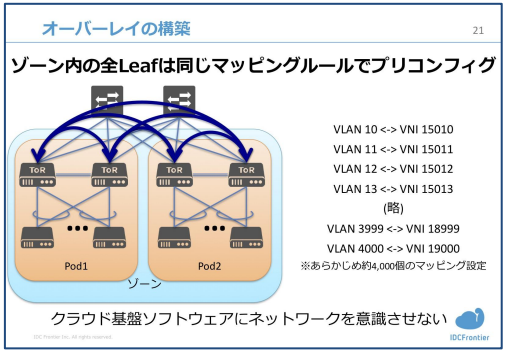
IDCFrontier

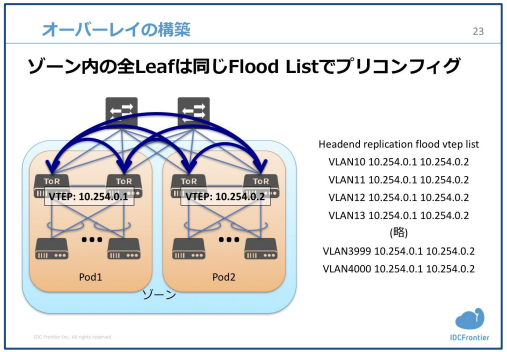
Ansibleを利用してネットワークの作業をCode化

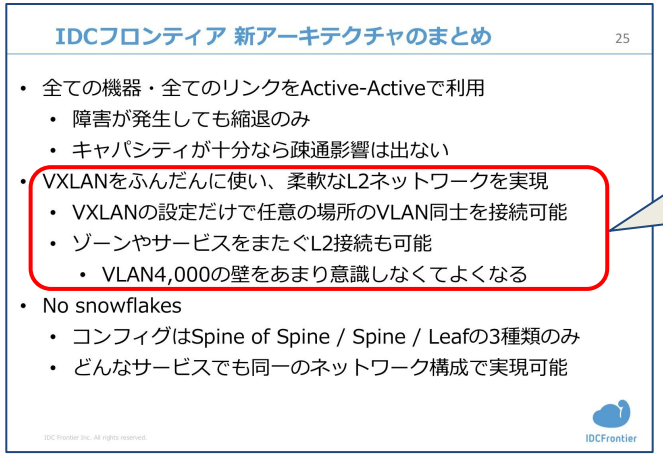


IDCFrontier

Static VXLANの開通から作業の自動化に着手







これをサービス化したものを
CI/CDの対象とした

JANOG39「IDCフロンティア"データ集積地構想/Data Centric Cloud"を支える ネットワークのアーキテクチャ」より

IDCFrontier

開通作業をCI/CD化するメリット

- 作業が頻繁に行われるため自動化の恩恵が大きい
  - 作業者がネットワーク機器の操作に熟練しているとは限らない
  - 同じような機器名、同じようなパラメータが頻発するために作業ミスのリスクを低減できる
- 作業手順が定型化されているためPlaybookで表現しやすい
- 開通作業は「本番環境へ継続的に変更を適用する」というCI/CDの考え方と似ているので、インフラCIのノウハウを応用しやすい

IDCFrontier

継続利用するPlaybookを作成する上で重要なこと

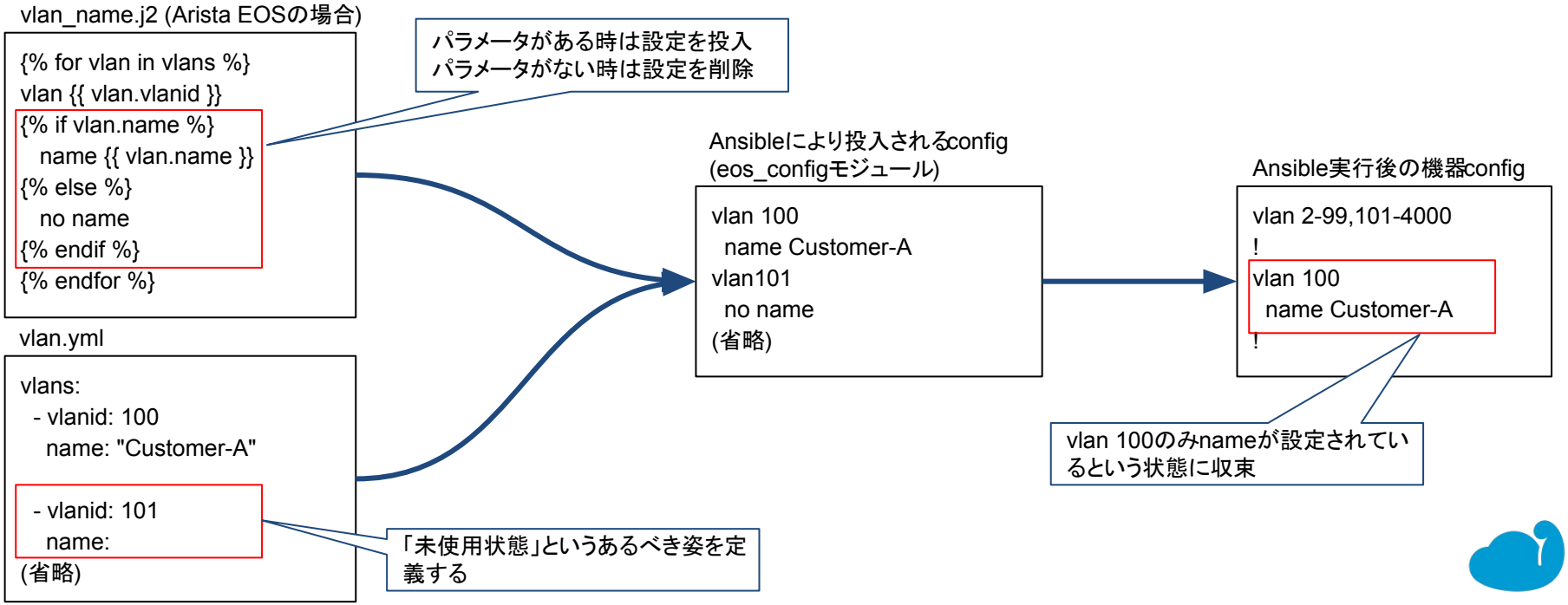
あるべき姿を定義し
その姿に収束するような形となっているか

- 何度実行しても同じ結果が得られること(冪等性)
  - 現状のパラメータに○○加算する、といった作りだと実行するたびに状態が変わってしまうので避けるべき
- デフォルトの状態も忘れずに定義しておく
  - 設定の削除=デフォルトへの変更
  - 設定は入れられるが消すことができない、というのは避ける

IDCFrontier

テンプレートとInventoryの作成例

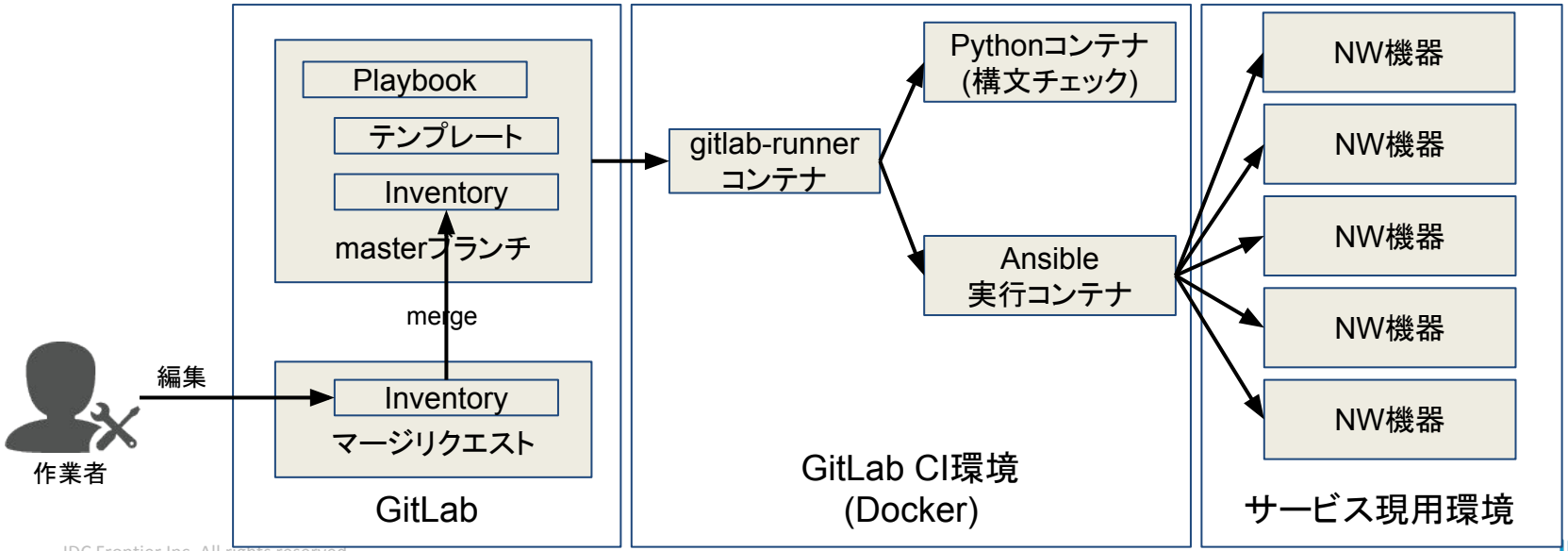
設定すべきものは設定し削除すべきものは削除する



IDCFrontier

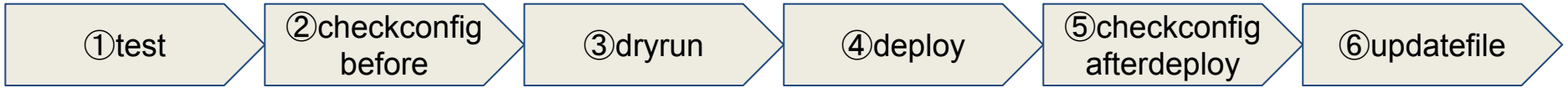
GitLabを活用しCI/CD環境を構築

- 作成したPlaybook(テンプレートやInventory含む)をリポジトリで管理
- リポジトリの変更をトリガーとして自動で本番環境への展開を行う
  - 開通作業は基本的にパラメータ(Inventory)の変更に限定される



CI/CDパイプラインの定義

GitLab CIでジョブを定義し順番に実行
全てのジョブをPassすると成功となる

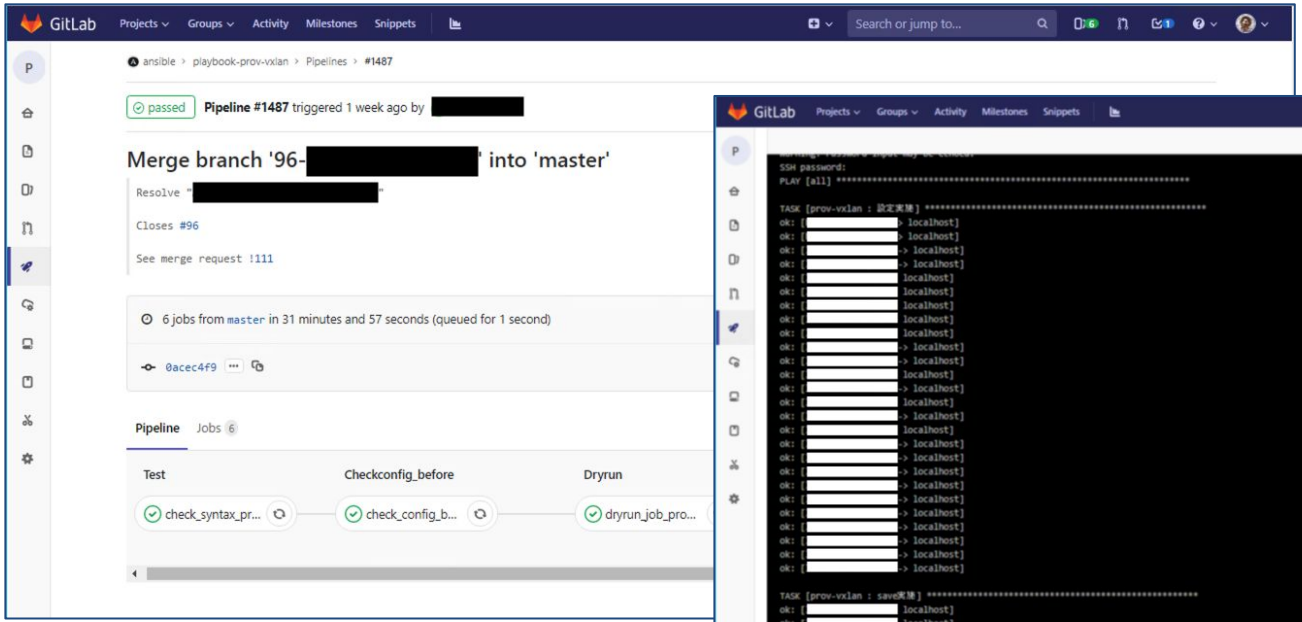


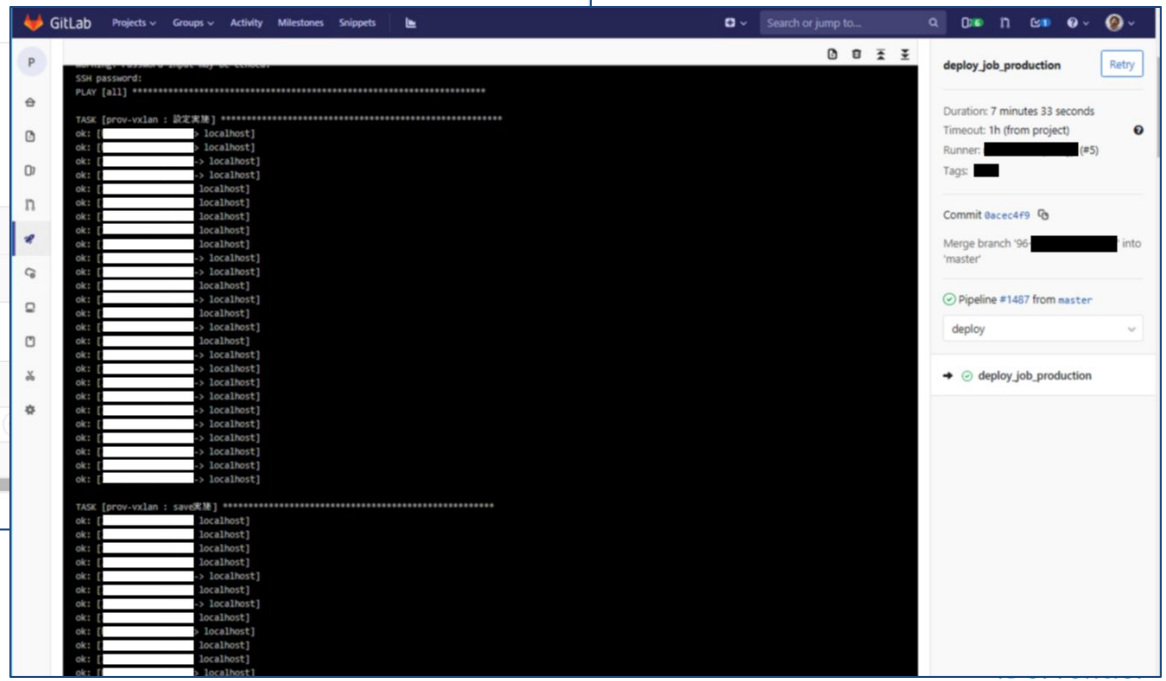
①test	②checkconfig before	③dryrun	④deploy	⑤checkconfig afterdeploy	⑥updatefile
yamlファイルの構文チェック playbookの構文チェック	現用環境の設定が、リポジトリで管理している変更前の状態と一致しているかを確認 (手動で現用機の設定を変更していた場合エラーとなる)	現用環境に対してplaybookが正常に動くか確認 ※ここでは設定変更を行わない	現用環境に対して実際にplaybookを実行し設定変更を行う	設定変更後の現用環境が意図した状態となっているかを確認	次回実施時に②で比較に用いるためのファイルを更新

IDCFrontier

できあがったもの

現用環境(対象機器70台弱)で動いてます！





うまくいったところ

- サービス提供中の現用環境に対して、障害を起こすことなく適用することができた
  - 現用機のconfigをそのまま適用した検証環境(コンテナ版OS)でAnsibleの動作を確認していたので不安はあまりなかった
- GitLabのGUI操作でやりたいことが十分できたため、学習コストが低く抑えることができた
  - GitLab上の操作手順書で十分対応できたため、GitやAnsibleの理解が十分でなくとも開通作業はできる
  - 切り戻し(Revert)もGUI上の操作だけで可能
- GitLab CIの実行ログが残るため、エラーが起きた時も調査しやすい

IDCFrontier

ハマったところ

- Ansible実行環境側の負荷が思ったより高かった
  - ネットワークモジュールはAnsible実行ホスト側で行われる処理が多いため、あまり同時実行プロセス数を増やせなかった
  - registerを利用した際に膨大な量のメモリが食われて、メモリ不足でAnsibleの実行が止まったことも…
- ネットワーク機器のAPIが受け入れ可能なリクエストサイズに悩まされた
  - 「デフォルトの状態」まで定義を行うと、Inventoryの量が多くなるため、テンプレートで展開した後のconfigの量も膨大になる
  - APIで受け入れ可能なリクエストのサイズを超過し、エラーが発生したことも…

IDCFrontier

Network CI/CDの今後

- テストの拡充
  - パラメータの整合性チェックはすぐにでもやりたい
  - 本番環境への適用前に使い捨て環境で確認を行いたい
    - コンテナ版ネットワークOSに期待
  - 設定完了後の確認をAnsible以外の方法で確認したい
    - Ansibleの実行結果をAnsibleで確かめるって正しいのか？
    - Serverspecのネットワーク機器版が欲しい
- 対象範囲の拡大
  - 手作業で行なっている他サービスへの展開
  - 開通以外の定型作業への展開

IDCFrontier