Zabbixをクラスター化して大規模監視に対応した話

自己紹介

- 津野哲平
  - フリーランスインフラエンジニア
  - Zabbix6.0 認定プロフェッショナル
  - tsuno.teppei@gmail.com

事のあらまし

Zabbixがそこそこ触れたことで友人に誘われた現場に入ったら、
データベースのサイズがテラバイト単位でまともにメンテナンスできない
状態になっていたZabbixが5台ほど動いていた

それぞれのZabbixは別運用されていたものが集められてきたため、
設定に統一性のあるルールはなく、
中身がカオスな状態だった……

問題その１

多すぎたデータに対して適切でない設計

- アイテム１つの１週間分のグラフ表示に１０分程度必要な状態
- データベース設定の工夫はこれ以上無理っぽい
（ パーティショニングや設定チューニング済み ）
- そもそもデータベースサイズが初期設計説明資料の５倍くらいになってるんですけど…
- 再起動に１２時間かかるんですけど…

問題その２

設定？そのサーバーになければないですね

- テンプレートは同じものが複数あったり、同じ機器で同じ監視でもテンプレートとホスト直接になっていたり…
- インターバル、1分3分5分10分が特に理由なく混在
- トリガーアクションが３桁あるけど、使ってるのは１桁
- サーバープロセス設定、なんでこれだけ設定違うの？ ValueCacheが足りなくてlow memoryモードになるの初めて見たわ…

もう全部作り直そう

出てきた要件（ざっくりと）

- 最大対象ホスト数：
  - ２０，０００
- 最大監視アイテム数：
  - ４，０００，０００
- データ保持期間：
  - ７年くらい過去にさかのぼれるように
- 視覚化：
  - グラフが社員全員だれでもすぐに見れて、
    それ見れば問題の有無がすぐわかるといいよね

規模が大きくなりました

おおよそ現行の４～５倍くらい

いままで監視できてなかった対象も入れたいんだって…

渡されたリソース

- ＣＰＵ：２２４コア
- メモリ：２ＴＢ
- ＳＡＮ（フラッシュ）：５０ＴＢ
- ＳＡＮ（ニアライン）：１５０ＴＢ

メンバー

- 設計：自分
- 構築：自分
- テスト：自分
- 基盤運用：自分

以上！

解決しよう

解決すべき課題

- 監視設定の管理を一元化する
- ユーザーの管理を一元化する
- データベースの負荷を下げる
- 閲覧速度を上げる
- 通知の管理を一元化する
- 全部を少ない労力でできるようにする
- リソース計算できる監視設計にする

まずは方針

- 複数台の環境設定に差異が生じるのは絶対に避ける
- サーバー：プロキシ＝１：ｎ構成でも可能だけれど
  ディスク負荷がサーバー１台に集中するので避ける
- 台数分だけ管理が増えるのは意味がない、
  全部を簡単に管理できる仕組みにする
- 監視設定は１００％テンプレートのみにして
  複数台でも全部同じ監視設定を持つ
- ユーザー管理を複数台でやるのは論外

ユーザー管理はもうやめよう

- 閲覧専用アカウント管理は１台でもやりたくない
- 閲覧のためだけにゲストを有効にしたくない
- 必要なのは
  - 「閲覧するユーザーをコントロールできる」
  - 「登録の手間はなくす」
  - 「監視対象の情報がみれる」
- ユーザーはグラフが見れるならZabbixでなくてもよくない？
- それならGrafanaを使おう

ツール解説（１）

- Grafana
  - Javascriptを用いてデータソースから「ブラウザがグラフを描画する」Webアプリケーション、OSS
  - LDAPからユーザー情報を参照し、自動登録する機能がある
  - Zabbixを含め様々なデータソースに対応している

【解決された課題（１）】

- ユーザーの管理を一元化する[90%]
  - 閲覧だけのユーザー作成はすべてGrafanaと
    既存の全社認証システムの連携に丸投げした
- 閲覧速度を上げる[1%]
  - グラフがブラウザが描画するのでサーバー側の
    負荷が少なくできた（ただし焼け石に水）

【以後の方針（１）】

- Zabbixの管理以外のユーザーは、全社認証システムとツールが自動的に連携するものを利用しよう
- Zabbixの機能で実現が難しいところは他の適切なツールで補っていこう

【これどうするの？（１）】

- ユーザー管理をしない＝宛先管理もしないことになるので通知の設定はどうする？
  ⇒ ひとまず保留
- 巨大なＤＢではそもそもすべてが重い
  ⇒ まずこれを解決する

データストアを
速くしたい

4.0の新機能を利用する（当時）

- DBの代わりにヒストリをElasticsearchに入れる機能が試験的に実装
- しかしElasticsearch機能の内容はItemID／時間／値だけでZabbixフロントエンドからは使うための内容
- またヒストリ／トレンド／イベントのリアルタイムエクスポートが実装され、こちらはZabbixに依存しない内容
- だったらElasticsearchにリアルタイムエクスポート出力を全部入れてそちらを見よう
- GrafanaはElasticsearchはデフォルトで対応しているしね

ツール解説（２）

- Elasticsearch
  - NoSQLデータストアの分散型検索/分析エンジン
  - 最初からクラスターで設計されていて、とても速い
  - JSON形式のデータととても相性がいい

データの入れ込み

- Elasticsearchへのデータ送信は別にツールが必要
- 同時にほかのストレージサービスに同じ内容を送ることでバックアップ取得が不要になるんじゃない？
- 複数の対象に送れるfluentdにしておこう
- 不要データを削除したり、追加情報を付与したりもできる
- リアルタイムエクスポートのヒストリは数値系とテキスト系が混在しているので、それの分離もできる

ツール解説（３）

- fluentd（旧td-agent）
  - データ収集ソフトウェアの一つ、OSS
  - 様々なデータをsyslog、S3などのオブジェクトストレージ、解析用の各データウェアハウス、SaaSなどに送信できる
  - その際データを加工が可能で、タグをつけるなど情報を追加したり、必要な情報だけ切り出し、件数カウント、不要な情報の削除などもできる

【解決された課題（２）】

- 閲覧速度を上げる[100%]
  - データの閲覧が非常に速くできた
- データベースの負荷は下げる[50%]
  - Zabbixで閲覧のための長期データを持つ必要をなくした
  - データを最初からバックアップストレージにも送ることでデータベース・データストアから取るのを不要にした

【以後の方針（２）】

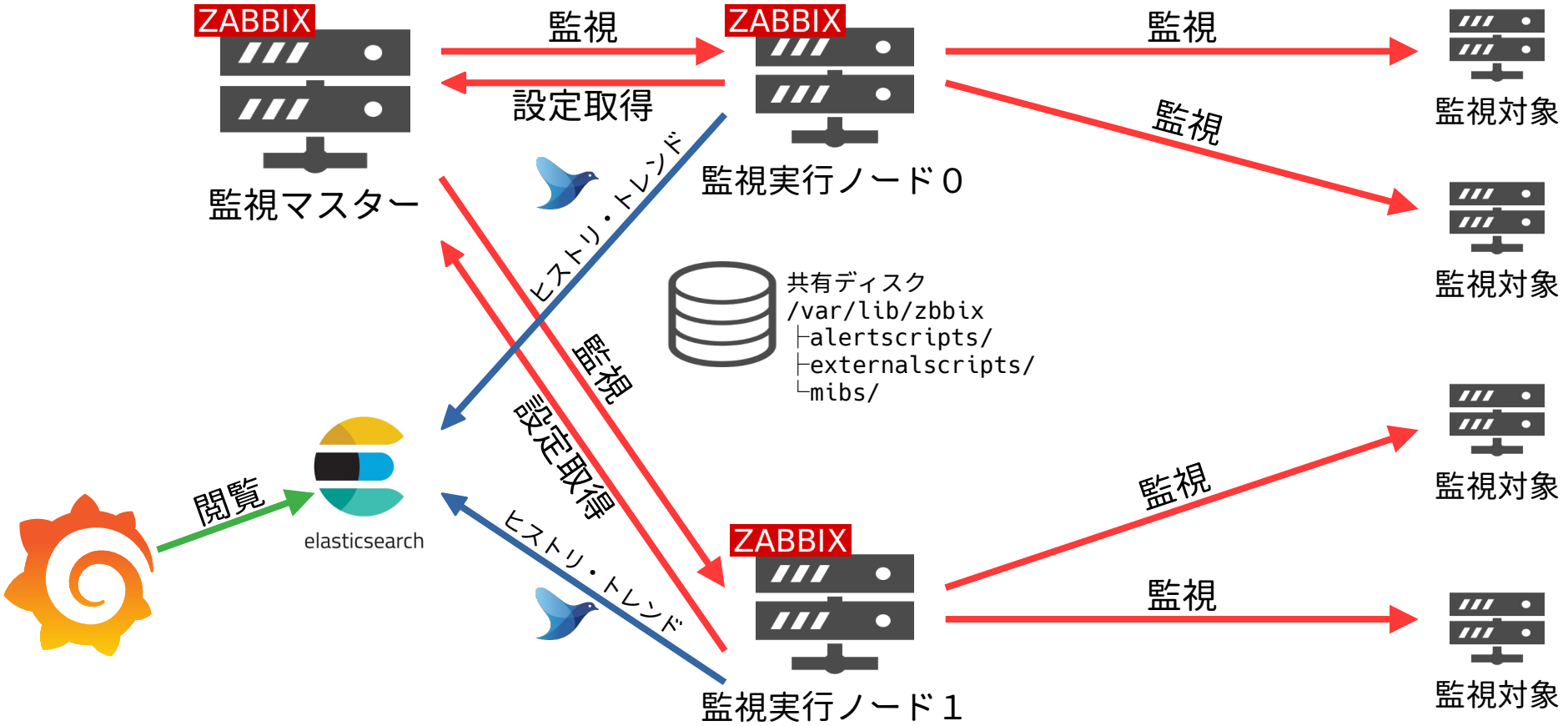
- 取得出力／送信／保持、それぞれのツールを分割し疎結合（ツールの状態が他の動作に影響を及ぼさない）にしたので、他の機能も疎結合構成にしよう
- 全てのツールで負荷分散をスケールインアウトで行う仕組みで統一しよう

【これどうするの？（２）】

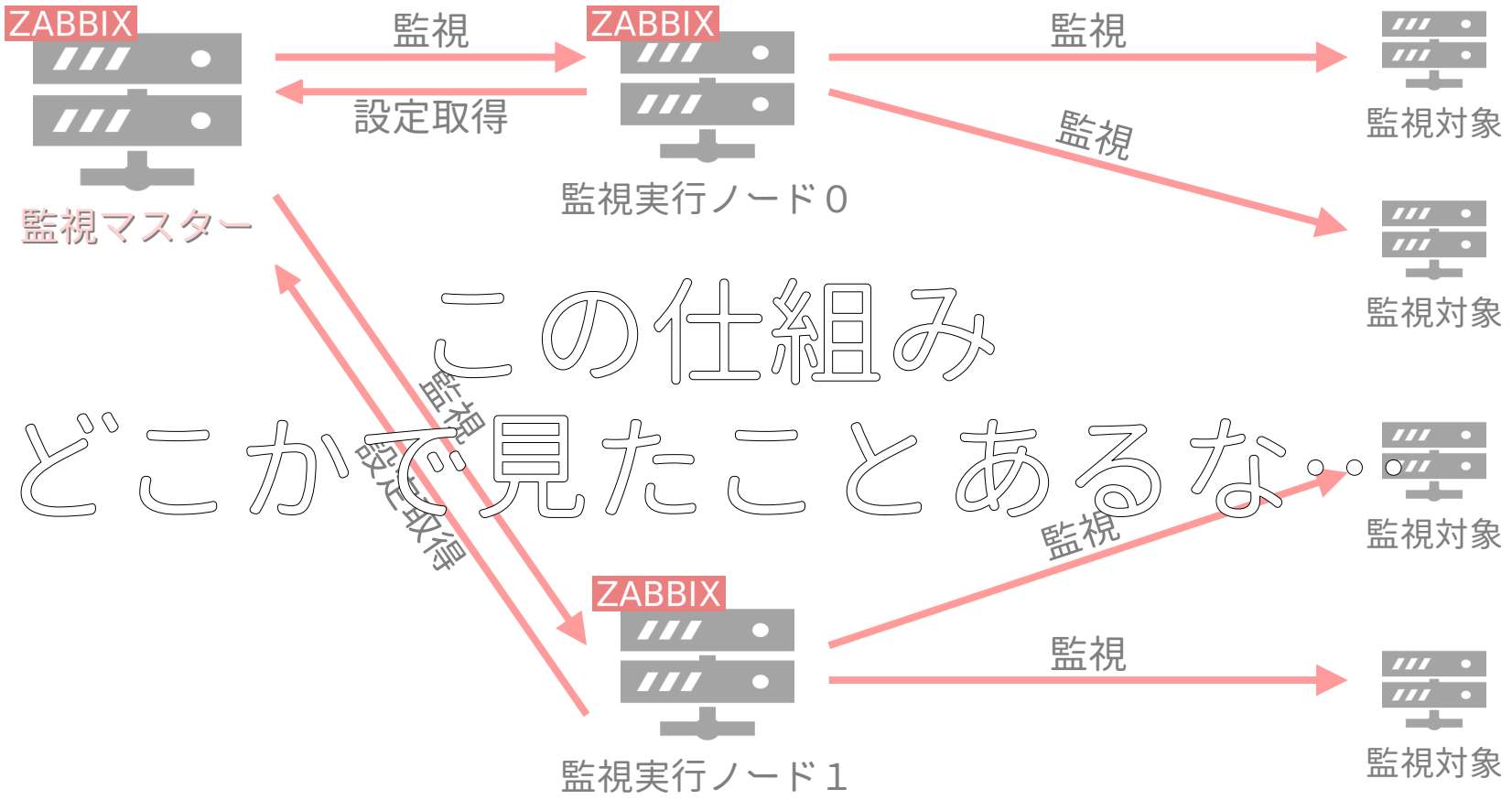
- Zabbixでスケールインアウトは
  どうやればいい？
- ユーザー管理をしない＝宛先管理もしないことになるので
  通知の設定はどうする？
  ⇒ 保留中

Zabbixをクラスター化

構想編


ZABBIX
監視
設定取得
ZABBIX
監視
監視
監視対象
監視対象
監視マスター
監視実行ノード０
ヒストリ・トレンド
共有ディスク
/var/lib/zbbix
├alertscripts/
├externalscripts/
└mibs/
監視
設定取得
閲覧
elasticsearch
ヒストリ・トレンド
ZABBIX
監視
監視
監視対象
監視対象
監視実行ノード１

構想編


ZABBIX
監視
設定取得
ZABBIX
監視
監視
監視対象
監視マスター
監視実行ノード０
監視対象
この仕組み
どこかで見たことあるな…
監視
設定取得
監視
監視対象
ZABBIX
監視
監視対象
監視実行ノード１

構想編

あ、これ2.4で
オミットされた
Zabbix Nodeだ！

構想編

このシステムにおけるZabbixのクラスター動作を定義

1．監視に関する設定は１台のZabbixが全てもち、それを絶対の正にする。
　以後このZabbixを「監視マスター」と定義する。

2．監視動作は監視マスターから監視設定を複製し、担当する対象を監視する。
　以後このZabbixを「監視実行ノード」と定義する。

3．監視マスターは設定管理と、監視実行ノードの監視だけを行う。

4．監視実行ノード上の設定およびデータは一時的なものとする。

5．監視実行ノード上の設定およびデータは失われることを許容する。

6．監視実行ノードはスケールアウト／スケールインできることを前提とする。

7．監視対象の設定は、どの監視実行ノードでも必ず同じ動作ができる。

構想編

- 設定は一元管理、ディスクＩＯも処理も分散、これをすべて満たすことは可能か？

  ⇒ Zabbix APIを利用して複数台の設定を同期すれば、だいたい可能ではないか

【以後の方針（３）】

- Zabbixを監視マスターと監視実行ノードに
  役割を分担した１：ｎ構成のクラスターにする
- 監視実行ノードは設定もデータも保持しないので
  「使い捨てる」運用をする

【これどうするの？（３）】

- Zabbixをこんな風に動作させる機能無くなってるけどどうする？
  ⇒ 自分で作るしかない
- ユーザー管理をしない＝宛先管理もしないことになるので通知の設定はどうする？
  ⇒ 保留中

作成編

- Zabbix APIで監視マスターから監視実行ノードに設定を移植するスクリプトはこれを使う
  - Python 3.x
  - PyZabbix
  - データベース接続のモジュール（PostgreSQL/MySQL）

作成編

- 移植しなければならない設定
  - 監視本体：ホストグループ／ホスト／テンプレート／ユーザーマクロ
  - 通知動作：メディアタイプ／アクション
  - 規定　　：グローバル設定

作成編

- Zabbix APIでだいたいできる
  - 監視は全てconfigurationで対応
  - 規定の内グローバルマクロはusermacroで対応

作成編

- Zabbix APIでできないこと
  - 通知動作はどちらもZabbix内部IDで指定されているので監視実行ノードにそのまま持ってくことができない
    ⇒ configurationと同じように内部IDはnameに変換して出力、移植先で必要なデータが入った後にnameを内部IDに変換
  - 必ず存在する設定はホストグループとテンプレート、トリガーの優先度だけなので、その３つでトリガーアクションを設定

作成編

- トラブルシュート
  - configurationで大量のホスト設定を送り込むとタイムアウトしてしまう（５００くらい）
  - 理由はAPIがシングルスレッド動作で時間がかかりすぎた

    ⇒ エクスポートされたJSONからホスト部分を抜き出し、host.create()で処理を小さくし、またスクリプト側をマルチスレッドで並列発行して全体の時間も短縮

作成編

- 設定のキャッシュ＆バージョン管理
  - APIを毎度監視マスターに実行するより、データはJSONだしkvsでキャッシュしてそちらからもらう方が速い
  - Redisならキャッシュの永続化可能なので、日付とIDをつけた設定セットを保存すれば、バージョン管理ができる
  - これを利用して監視マスターのDBの復旧も容易にできる

【解決された課題（４）】

- 監視設定の管理を一元化する[100%]
  - 設定を管理するのは監視マスターのみにした
- ユーザーの管理を一元化する[100%]
  - 監視実行ノードは基本APIで操作されているので特別なアカウントは不要、作業用のアカウントを管理するのも監視マスターのみにした
- データベースの負荷は下げる[100%]
  - 監視実行ノードは自身が監視に使うデータだけあればよく、また分散するので１台当たりのiopsを小さくできるようにした

【以後の方針（ ４ ）】

- 監視実行ノードは起動して設定を入れたあとは
  基本的にフロントエンドに入って操作しない

  ⇒ ホスト追加削除など運用にどうしても必要なものだけ
  スクリプト化
- ルールで無理なく回避できるものは無理に技術的に解決しない

【これどうするの？（４）】

- トリガーアクションのメンテナンスによる停止はどうするか？
- あと保留にしていた通知の一元管理は？

通知編

- 通知先の設定をZabbixでもってない
  ⇒ トリガーアクションを細かく作れない
- メンテナンスを全てのノードに全部設定する
  ⇒ 無駄が多い
- 監視実行しているノードにのみ設定する
  ⇒ 全部のノードが同じ設定ではない

通知編

- チケットシステムとしてServiceNOWを利用
- 全社認証システムと連携して通知先をもっているので、起票するアクションだけ設定した
- 通知はServiceNOWに~~丸投げ~~集約
  監視以外の運用でもつかっているので

通知編

- ラッパーをAWSのサーバレスで作成
- メッセージはすべて一度ここに集約されS3にもバックアップされる
- コントロールを「タグ」のみで指定することにし、「１回限り」と同じ様なメンテナンス機能を実装した
- ラッパー内部ではホスト、ホストグループもタグとして扱っている

【解決された課題（５）】

- 通知の管理を一元化する[100%]

【これどうするの？（５）】

- 最終的にこれらをどうやって動作させる？

基盤編

- クラスターはある程度ノード数がないとメリットが薄い
- 分散数を多くすると管理が大変になる
- VMはVMイメージを作成管理する周辺システムが必要
- 今はコンテナがこういうのを楽にできそう
- 公式コンテナならミドルウェア管理も不要になるし
- どうせなら全部できそうなkubernetes使おう

ツール解説（４）

- kubernetes
  - 「コンテナ化したアプリケーションのデプロイ、スケーリング、および管理を行うための、オープンソースのコンテナオーケストレーションシステムである」
  - 「ホストのクラスターを横断してアプリケーションコンテナを自動デプロイ、スケーリング、操作するためのプラットフォームを提供する」
  - コンテナを利用したクラスターコンピューティング環境を作るソフトウェア

    コンテナ実行やネットワーク設定、そしてそれらの状態管理を冗長構成、高可用性設定含め一手に引き受けてくれる

    そのすべてをYAMLのマニフェストファイルで設定できる

【解決された課題（６）】

- 全部を少ない労力でできるようにする[100%]
  - まさにIaC、YAMLファイルでツールの動作、プロセス設定の全てを管理できるようになった
  - ノード数の調整は設定を変える程度になった
  - 基盤故障や不正処理で落ちても数秒～十数秒で自動的に移動・復旧し、冗長化の切り替えと遜色ないため冗長化をする必要がなくなった

【以後の方針（６）】

- 「落ちないシステム」ではなく、
「短時間で復旧させるシステム」を前提とした運用にする

【これどうするの？（６）】

- このシステムでリソースは足りてる？

渡されたリソース（おさらい）

- ＣＰＵ：２２４コア
- メモリ：２ＴＢ
- ＳＡＮ（フラッシュ）：５０ＴＢ
- ＳＡＮ（ニアライン）：１５０ＴＢ

必要リソースを計算できる設計

- Zabbixのリソース計算はインターバル設定に依存するので、
  システム仕様として決めてしまう
  - 死活監視：１分
  - 通常の数値取得：５分
  - 変化の少ない文字列の取得など（LLDも含む）：１時間

- この設定で全テンプレートを整理＆作り直し
  （モジュール化含めて２００くらい）

必要リソースを計算できる設計

- １アイテムの必要リソース量（通常の数値取得）
  - レコード数計算
    - ヒストリ：５分インターバル＝１時間に１２回＝３０日で８６４０レコード
    - トレンド：１日に２４回＝３０日で７２０回＝１年で約８６４０レコード

    トレンドで１年分のレコード数 ≒ ヒストリで１か月分のレコード数
    １アイテムはヒストリ換算約２０ヵ月分のレコード数
    （ヒストリ４００日＋トレンド７年） ＝ １７２，８００

１レコード１００Ｂと見積り、１アイテムはおよそ１７ＭＢで計算する

必要リソースを計算できる設計

- ４００万アイテムだとおよそ７０ＴＢ
  - フラッシュストレージだけだとオーバーしている
  - しかしElasticsearch内でデータは生の1/3くらいになるので問題なし

必要リソースを計算できる設計

- 生データはGZIP圧縮したら1/9くらい
- ニアラインストレージは
  生データ受け取り領域：１００ＴＢ
  コールド保存領域：３０ＴＢ
  に分けてバックアップを二重化

必要リソースを計算できる設計

- 想定される最大nVPSの単純計算（小数点繰り上げ）
  - 死活監視
    ２万 ÷ ６０秒 ＝ ３３４
  - 数値取得
    ４００万 ÷ ３００秒 ＝ １３，３３４
  - トレンド生成
    ４００万 ÷ ３６００秒 ＝ １，１１２

    最大で１５，０００程度

必要リソースを計算できる設計

- ZabbixはＤＢとリアルタイムエクスポートで２倍の３０，０００iopsを見込む
- Elasticsearch側はレプリカ１で構築し３０，０００opsのIndexing/secを見込む
- これは同じハードウェア／フラッシュストレージ利用の別Elasticsearchが、２００，０００ops以上出せているので問題なし

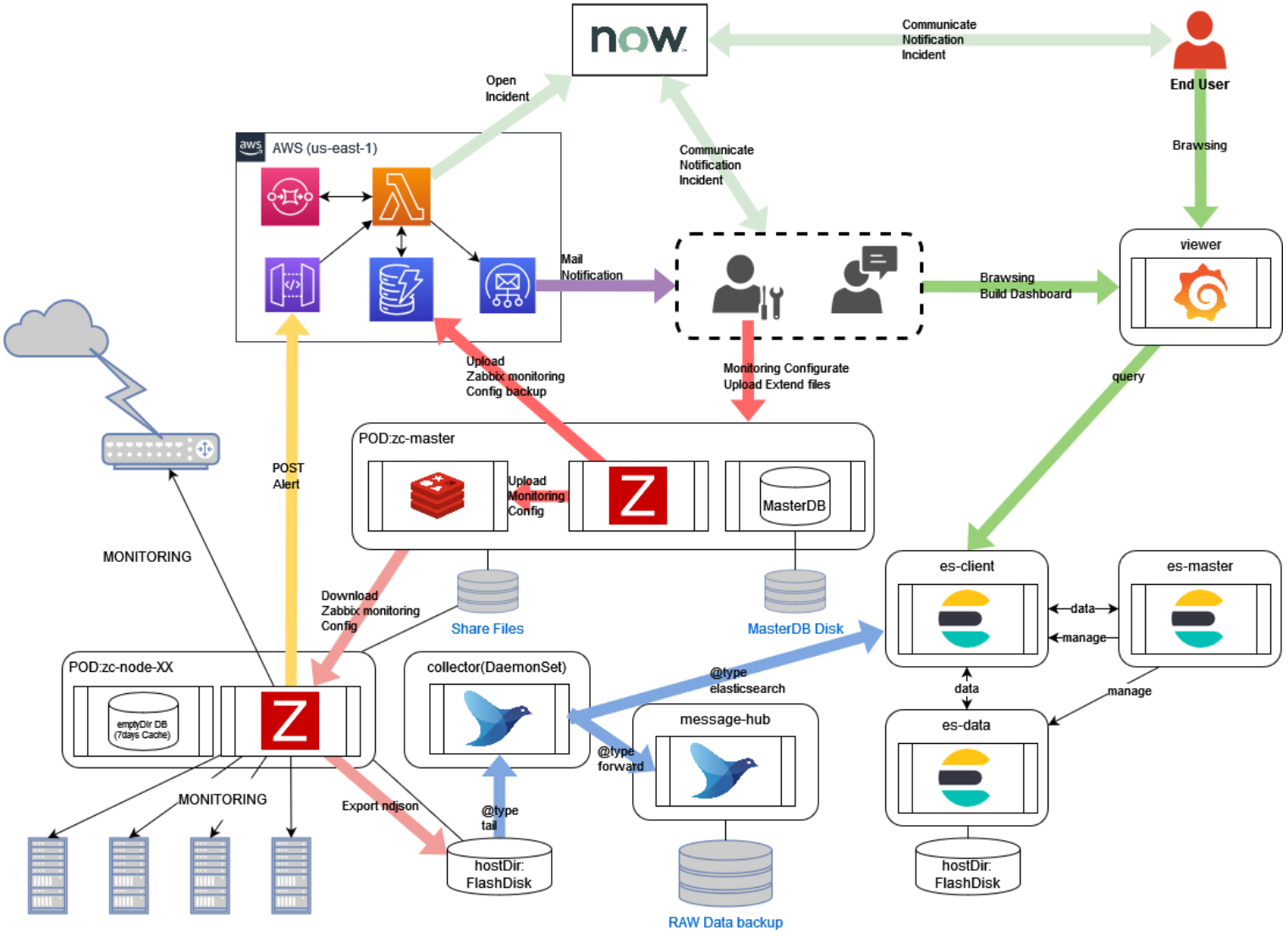
【解決された課題（７）】

- リソース計算できる監視設計にする[100%]

解決すべき課題Complete！

- 監視設定の管理を一元化する[100%]
- ユーザーの管理を一元化する[100%]
- データベースの負荷を下げる[100%]
- 閲覧速度を上げる[100%]
- 通知の管理を一元化する[100%]
- 全部を少ない労力でできるようにする[100%]
- リソース計算できる監視設計にする[100%]

出来上がったもの


now
Communicate
Notification
Incident
End User
Open
Incident
Communicate
Notification
Incident
Brawsing
AWS (us-east-1)
Mail
Notification
viewer
Brawsing
Build Dashboard
Upload
Zabbix monitoring
Config backup
Monitoring Configurate
Upload Extend files
query
POST
Alert
POD:zc-master
Upload
Monitoring
Config
MasterDB
MONITORING
Download
Zabbix monitoring
Config
Share Files
MasterDB Disk
es-client
es-master
data
manage
POD:zc-node-XX
emptyDir DB
(7days Cache)
collector(DaemonSet)
@type
elasticsearch
data
manage
message-hub
es-data
@type
forward
MONITORING
Export ndjson
@type
tail
hostDir:
FlashDisk
hostDir:
FlashDisk
RAW Data backup

運用中の監視対象数

- 監視実行ノード数：
  ９
- 対象ホスト：
  ７，５００前後
- 監視アイテム数：
  ８３０，０００前後
- 最大nVPS
  ４，５００前後

Elasticsearch上のデータ

- データノード数：
  １２
- インデックス数：
  ９５０ 前後
- レコード数：
  ９００億 前後
- サイズ：
  ２０ＴＢ 前後

データ閲覧可能期間

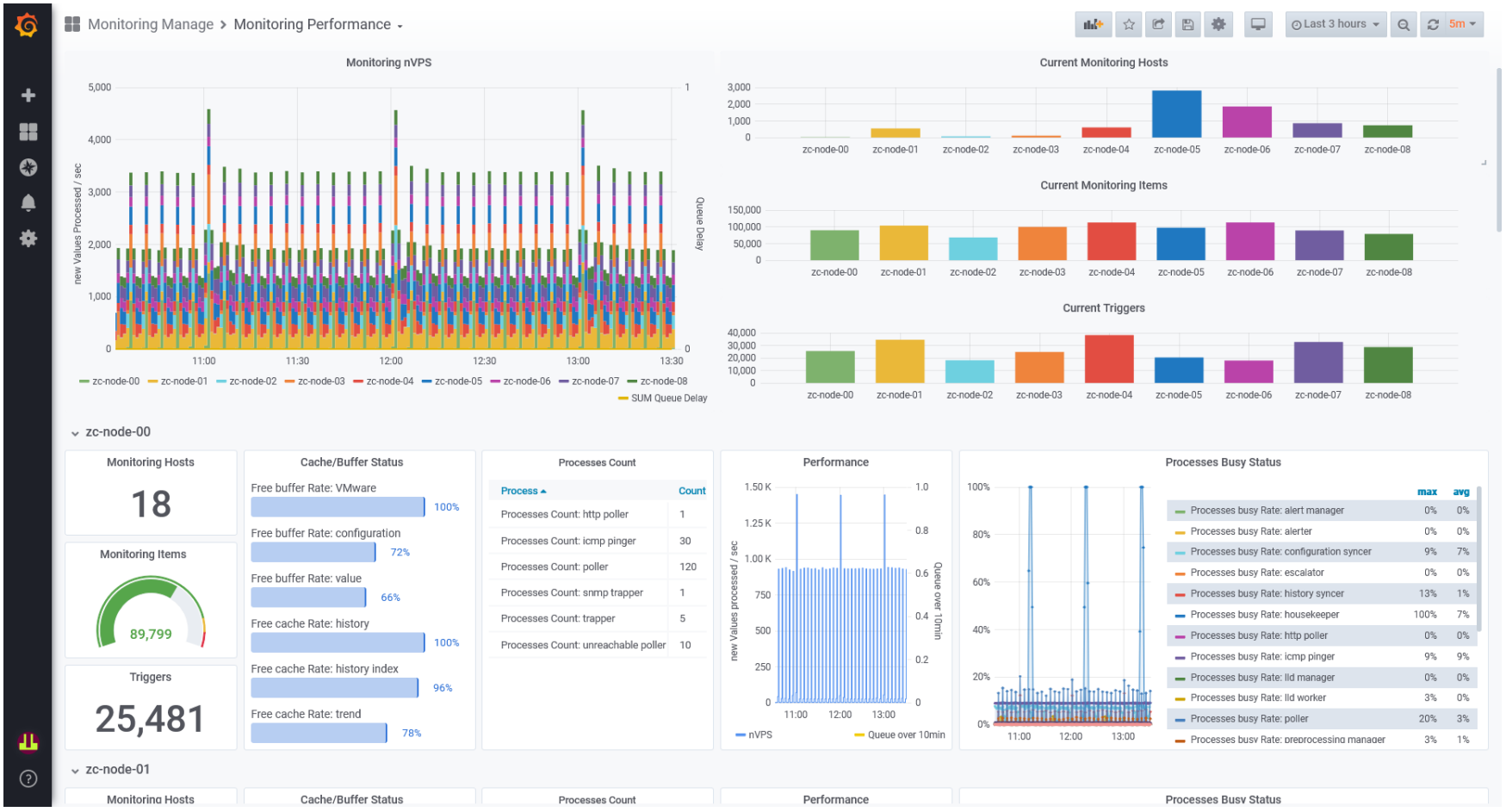
- ヒストリ：
  最新４００日
- トレンド：
  稼働から４年分すべて
  ＋前世代監視環境２年分
- データバックアップ
  ヒストリー／トレンド４年分
  ＋前世代監視環境２年分トレンド

閲覧速度

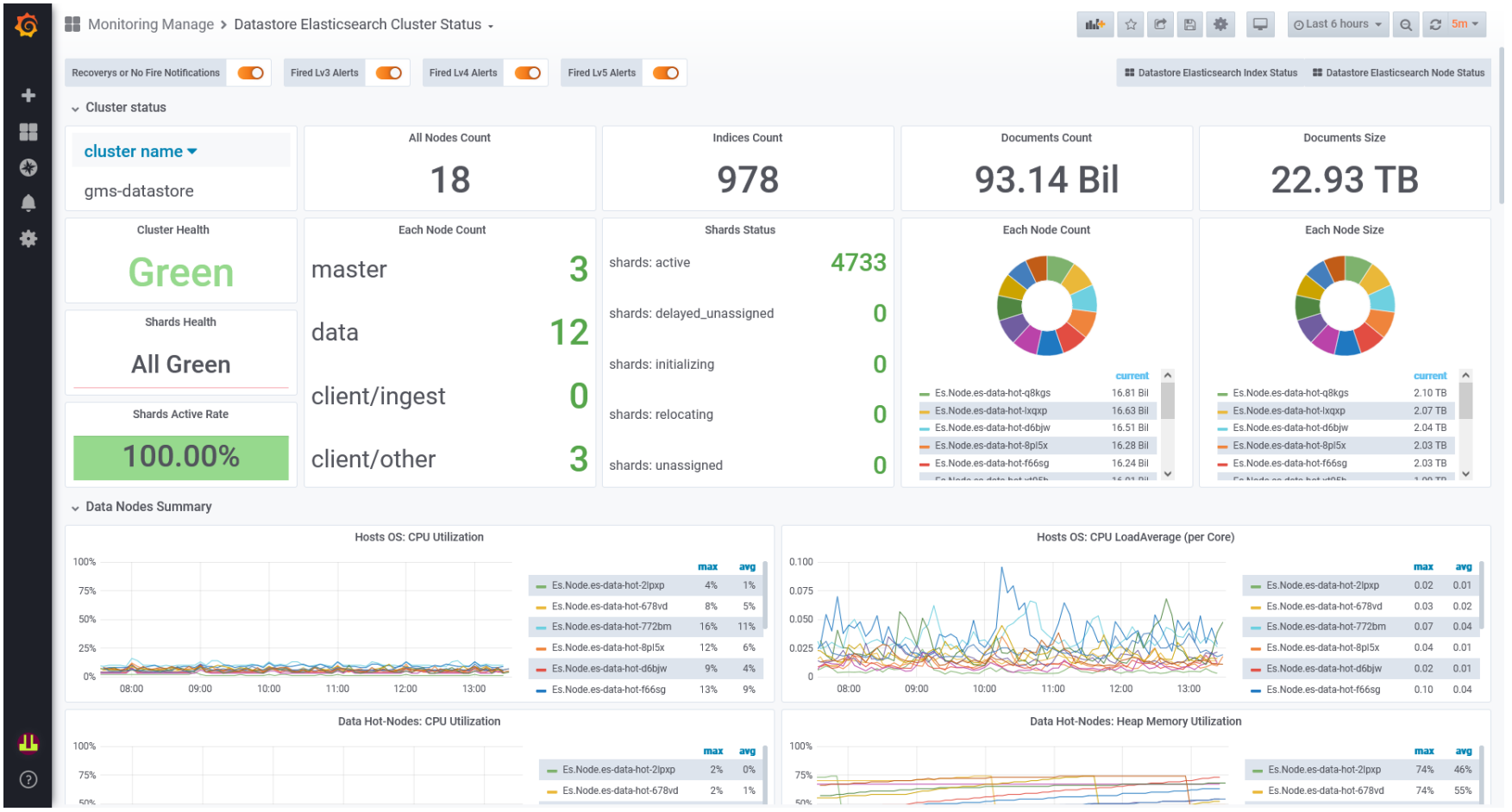
- ダッシュボード内の
  - アイテム数１００程度
  - 表示期間７日間

⇒ 約１０秒で表示

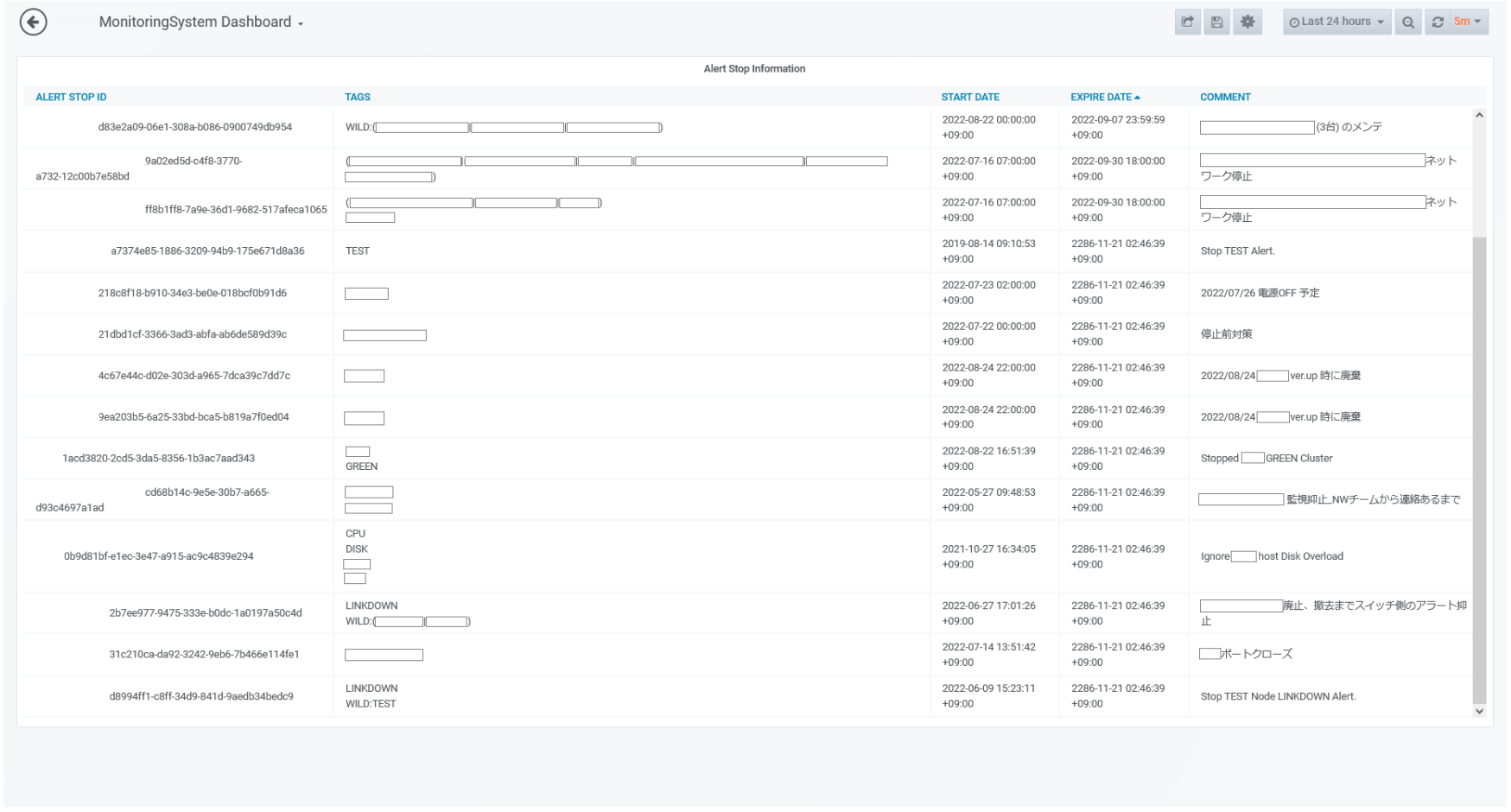
監視実行ノードの状態


Monitoring Manage › Monitoring Performance
Last 3 hours
5m
Monitoring nVPS
new Values Processed / sec
Queue Delay
zc-node-00 zc-node-01 zc-node-02 zc-node-03 zc-node-04 zc-node-05 zc-node-06 zc-node-07 zc-node-08
SUM Queue Delay
Current Monitoring Hosts
Current Monitoring Items
Current Triggers
zc-node-00
Monitoring Hosts
18
Monitoring Items
89,799
Triggers
25,481
Cache/Buffer Status
Free buffer Rate: VMware 100%
Free buffer Rate: configuration 72%
Free buffer Rate: value 66%
Free cache Rate: history 100%
Free cache Rate: history index 96%
Free cache Rate: trend 78%
Processes Count
Process
Count
Processes Count: http poller 1
Processes Count: icmp pinger 30
Processes Count: poller 120
Processes Count: snmp trapper 1
Processes Count: trapper 5
Processes Count: unreachable poller 10
Performance
new Values processed / sec
Queue over 10min
nVPS
Queue over 10min
Processes Busy Status
max
avg
Processes busy Rate: alert manager 0% 0%
Processes busy Rate: alerter 0% 0%
Processes busy Rate: configuration syncer 9% 7%
Processes busy Rate: escalator 0% 0%
Processes busy Rate: history syncer 13% 1%
Processes busy Rate: housekeeper 100% 7%
Processes busy Rate: http poller 0% 0%
Processes busy Rate: icmp pinger 9% 9%
Processes busy Rate: lld manager 0% 0%
Processes busy Rate: lld worker 3% 0%
Processes busy Rate: poller 20% 3%
Processes busy Rate: preprocessing manager 3% 1%
zc-node-01
Monitoring Hosts
Cache/Buffer Status
Processes Count
Performance
Processes Busy Status

Elasticsearchの状態



通知の停止設定



ALERT STOP ID	TAGS	START DATE	EXPIRE DATE ▲	COMMENT
d83e2a09-06e1-308a-b086-0900749db954	WILD:([redacted])	2022-08-22 00:00:00 +09:00	2022-09-07 23:59:59 +09:00	[redacted](3台) のメンテ
9a02ed5d-c4f8-3770-a732-12c00b7e58bd	([redacted])	2022-07-16 07:00:00 +09:00	2022-09-30 18:00:00 +09:00	[redacted]ネットワーク停止
ff8b1ff8-7a9e-36d1-9682-517afeca1065	([redacted])	2022-07-16 07:00:00 +09:00	2022-09-30 18:00:00 +09:00	[redacted]ネットワーク停止
a7374e85-1886-3209-94b9-175e671d8a36	TEST	2019-08-14 09:10:53 +09:00	2286-11-21 02:46:39 +09:00	Stop TEST Alert.
218c8f18-b910-34e3-be0e-018bcf0b91d6	[redacted]	2022-07-23 02:00:00 +09:00	2286-11-21 02:46:39 +09:00	2022/07/26 電源OFF 予定
21dbd1cf-3366-3ad3-abfa-ab6de589d39c	[redacted]	2022-07-22 00:00:00 +09:00	2286-11-21 02:46:39 +09:00	停止前対策
4c67e44c-d02e-303d-a965-7dca39c7dd7c	[redacted]	2022-08-24 22:00:00 +09:00	2286-11-21 02:46:39 +09:00	2022/08/24[redacted]ver.up 時に廃棄
9ea203b5-6a25-33bd-bca5-b819a7f0ed04	[redacted]	2022-08-24 22:00:00 +09:00	2286-11-21 02:46:39 +09:00	2022/08/24[redacted]ver.up 時に廃棄
1acd3820-2cd5-3da5-8356-1b3ac7aad343	[redacted] GREEN	2022-08-22 16:51:39 +09:00	2286-11-21 02:46:39 +09:00	Stopped [redacted]GREEN Cluster
cd68b14c-9e5e-30b7-a665-d93c4697a1ad	[redacted]	2022-05-27 09:48:53 +09:00	2286-11-21 02:46:39 +09:00	[redacted]監視抑止_NWチームから連絡あるまで
0b9d81bf-e1ec-3e47-a915-ac9c4839e294	CPU DISK [redacted]	2021-10-27 16:34:05 +09:00	2286-11-21 02:46:39 +09:00	Ignore[redacted]host Disk Overload
2b7ee977-9475-333e-b0dc-1a0197a50c4d	LINKDOWN WILD:([redacted])	2022-06-27 17:01:26 +09:00	2286-11-21 02:46:39 +09:00	[redacted]廃止、撤去までスイッチ側のアラート抑止
31c210ca-da92-3242-9eb6-7b466e114fe1	[redacted]	2022-07-14 13:51:42 +09:00	2286-11-21 02:46:39 +09:00	[redacted]ポートクローズ
d8994ff1-c8ff-34d9-841d-9aedb34bedc9	LINKDOWN WILD:TEST	2022-06-09 15:23:11 +09:00	2286-11-21 02:46:39 +09:00	Stop TEST Node LINKDOWN Alert.

設計外のメリットデメリット

- メリット
  - 開発環境・検証環境を本番稼働しているものと全く同じ設定のモノを用意できる
  - 監視実行ノードは設定を保持する必要がないのでバージョンアップは再作成で完了する
  - 監視実行ノードはRedisに残っている設定セットでバージョンダウンが可能
  - 監視マスターもRedis上に最終設定があるので完全作り直しが可能
    そのためDBバージョンアップが容易にできる

設計外のメリットデメリット

- デメリット
  - バージョンアップ時にAPIが結構変わるのでスクリプト修正が細かく必要
  - リアルタイムエクスポートのデータ形式が変わることがあるので fluentdで今Elasticsearchに送信している形に加工が必要
  - 監視マスターは監視実行ノード以前のバージョンにならざるを得ないので 新しい機能がすぐ利用できない
  - ミドルウェアの管理を公式コンテナ任せにしているので、ミドルウェアの バージョンにより起きる問題がちょいちょい発生する
    - 特定バージョンのOpenSSLとPythonの組み合わせで起きるSSLハンドシェイクエラー
    - 特定バージョンのlibsshに起因するsshエージェント動作の異常

ご清聴ありがとうございました