エンタープライズシステムにおける安全なリフト＆シフトを見据え

Javaコンテナアプリケーショントラブルへの備え
～おなじみの情報を採取するために～

2023年12月15日

NECソリューションイノベータ　南 貴之

自己紹介

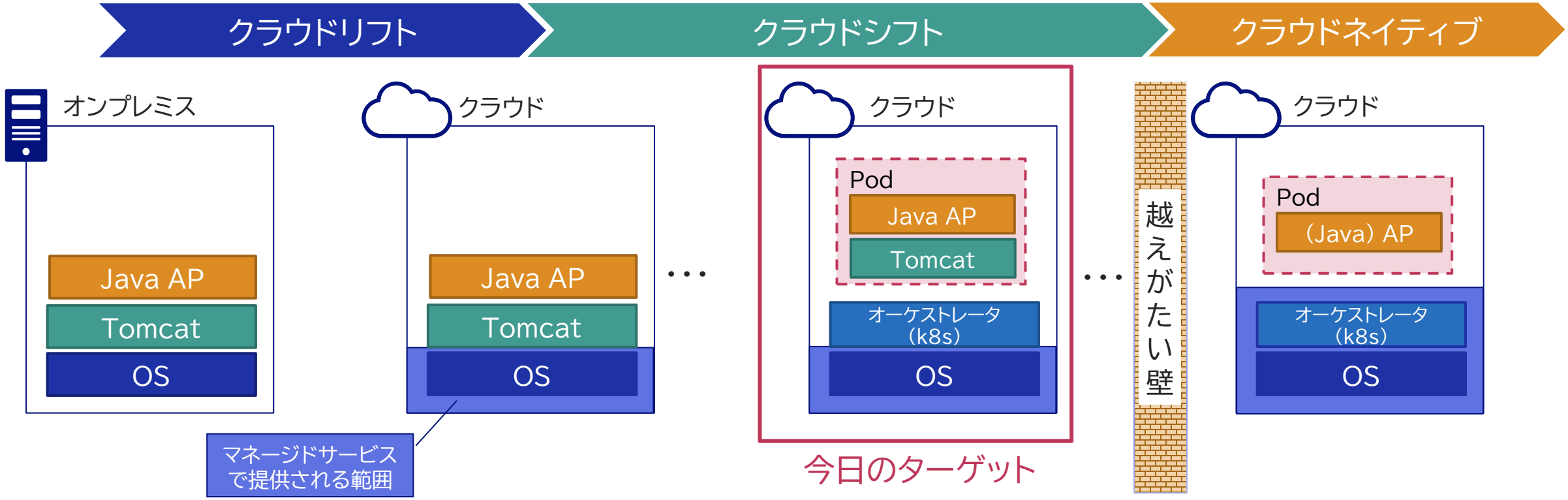
- 名前： 南　貴之(みなみ　たかゆき)

- 現在の役割
  - OSSを主とする先進的技術を活用したシステムの課題解決
  - 専門領域：クラウド/コンテナ、IT自動化

- 担当業務概要
  - クラウド/コンテナ関連OSS（主にCNCF傘下PJ）の適用領域見極め ←★今日はこの立場でのお話
  - Ansible/3scale API Managementの技術支援
  - 心は今でもJavaプログラマー

「安全なリフト＆シフトを見据え」とは？

- クラウドネイティブに至るまでの道のりにおいて、アプリケーションの大規模改修を伴うフェーズが大きな壁になるため、この段階を超えない時点での長期的運用を視野に入れる場合も多いかと思います



過渡期においても極力安全に、従来の構成や採取情報を活かしつつ運用するには？を考えてみます

1-1. 本セッションの主旨

◆ 今日持ち帰って頂きたい話

元々、Webアプリケーションサーバ上に構築されたJavaアプリケーションを、
コンテナ化して動作させる場合に、アプリケーションが出力する標準的な
ログ/ダンプ(以後単に「ログ」とも表記)を、トラブル発生時に解析が可能になるような
形式で出力させるにはどうしたらいいのか？

◆ 目指すゴール

- オンプレミスのJavaアプリケーションでは、アプリケーション自身のログ以外に、GCログ、スレッドダンプ、ヒープダンプ等、種類ごとにファイルに出力しているケースが多い

極力現実的な手段でこのGAPを埋めよう！

- アプリケーションがコンテナ化前提となる環境では、ログを標準出力経由でログ収集基盤で収集/解析する方式が一般的

1-2. 前提事項

◆ 対象とするコンテナ基盤

コンテナオーケストレータとして代表的な以下の環境で共通的に利用できる方法を検討した

- Kubernetes（オンプレミス）
- OpenShift（非マネージド）
- マネージドKubernetes（Amazon EKS）

◆ 対象とするログ

- オンプレミスで動作していたJavaAPをコンテナ環境に移行した際に、トラブル発生時に備えて出力・採取するログ

※ コンテナランタイムやKubernetesのコントロールプレーンが出力するログ（コンテナ基盤の管理者やサービス事業者が参照すべきログ）は本セッションでは扱いません

その他、細かな前提事項はAppendix1を参照

1-3. ログの種別

◆ 一般的に障害発生に備えて採取するログは以下の3種類があります

ログ種別名称	一般的な用途	例
イベント記録形式	状態の変化などイベント発生毎に取得する情報。問題の発生タイミングや状態などを詳細に把握可能	• アプリケーションログ • GCログ
スナップショット形式	その瞬間の情報を取得する形式。定期的な間隔で取得することでどのタイミングで問題が発生していたのかを確認することが可能	• スレッドダンプ • JMX • SNMP MIB
サマリ形式	一定時間の合計や平均を表示する。平均などを簡単に確認できるため「CPU平均5%」など、ある取得期間での状況把握に利用可能	• sar • vmstat

■ 本セッションで対象とした、ログ/ダンプは「イベント記録形式」と「スナップショット形式」の2種類に分別できます。分類結果は次ページ参照

1-4. JavaAPコンテナが出力するログと出力方式

◆ コンテナJavaAPにおいて出力されるログとその出力の標準方式は以下の通りです

名称	概要	ログ種別	出力の標準方式
APログ	アプリケーションの作りこみで出力するログ	イベント記録形式	サイドカー経由
サーバログ/ APサーバドメインログ	APサーバを使用する構成において、それぞれ、サーバの挙動について、および管理ドメインについて出力されるログ		
標準出力/エラー出力	標準出力/標準エラー出力をリダイレクトしたログ		
GCログ	Java VMが行うGC（garbage collection）を行った際のヒープ状況の変化を出力したログ		
スレッドダンプ	ある時点の Java VM 全てのスレッドのスタックトレースの情報をログとして記録したもの	スナップショット形式	jstackコマンドによりファイル出力
ヒープダンプ	ある時点の Java VM ヒープ内に含まれていたすべてのオブジェクトのスナップショット		ボリュームへファイル出力
JMX	Java アプリケーションをモニタおよび管理するための仕様で、APサーバの内部動作状況などを取得してログとして記録したもの		Prometheus等により採取
Coreダンプ	Java VMプロセスのある時点の使用中のメモリの内容をそのまま記録したもの		ボリュームへファイル出力

各ログの出力指針と方式選定理由については、Appendix2を参照

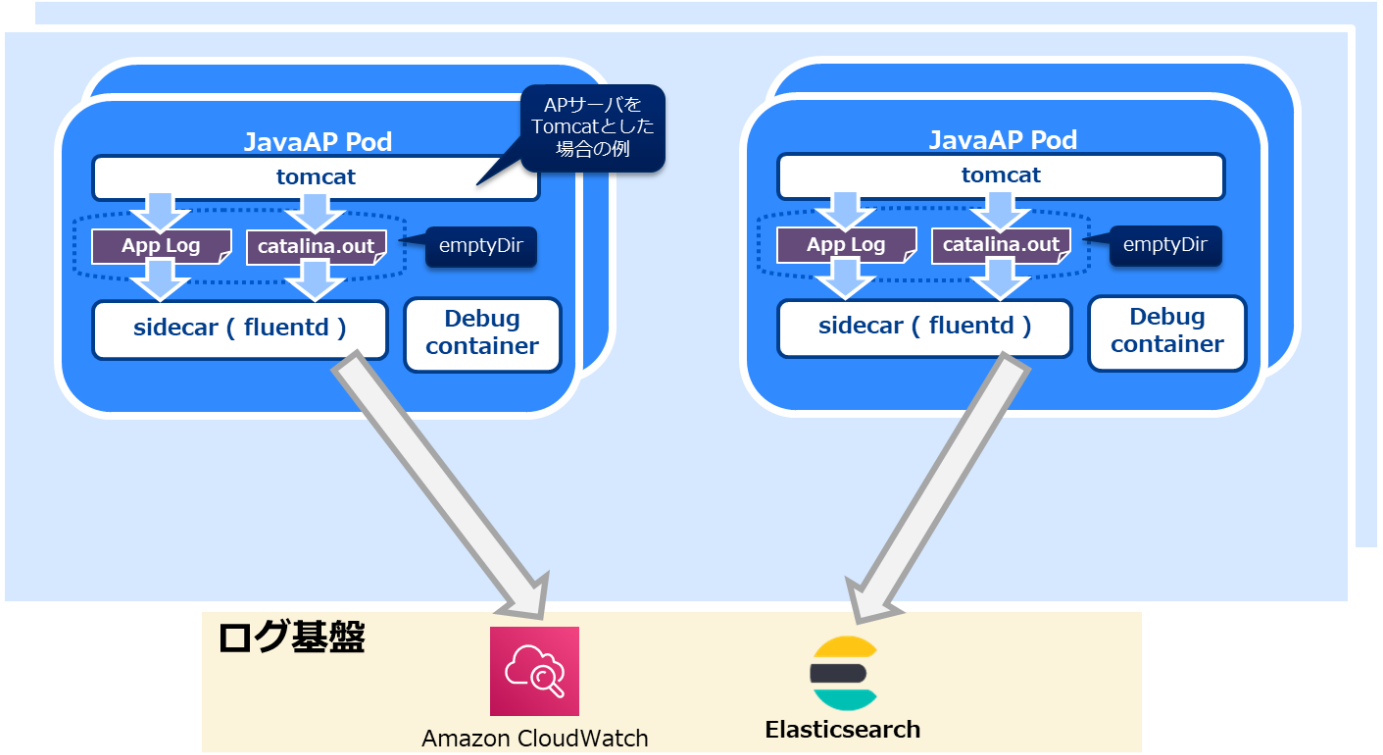
2-1. イベント記録形式ログの出力方式実装例

◆ 前頁で4種類ご紹介した、イベント記録形式ログの出力方式の例は下図の通りです

Point

1. emptyDirを用いてvolumeを作成する
2. volumeを各コンテナにマウントして、コンテナ間共有ディレクトリとして使用する
3. ログ収集基盤に応じたoutputプラグインを導入したFluentdコンテナを予め用意する
4. ログ毎にタグを定義し、収集基盤側で区別できるようにする

emptyDir利用時の注意・補足事項については、Appendix3を参照



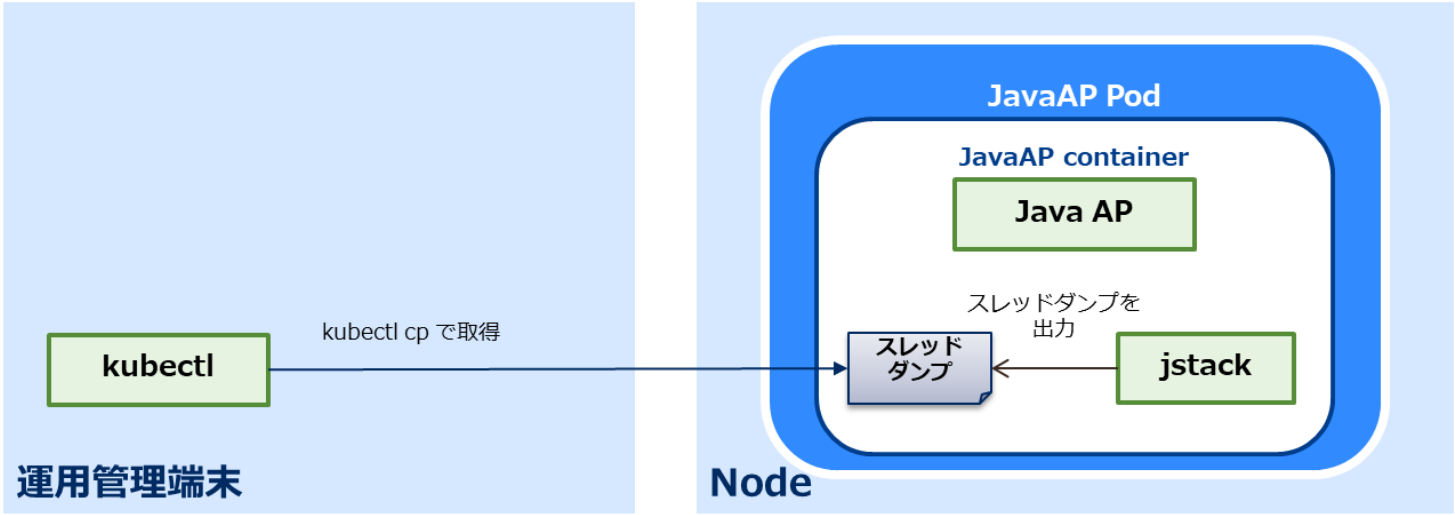
※ Kubernetesのマニュアル「Logging Architecture」の章で述べられている代表的なログ出力方式の5パターンのうち「Sidecar container with a logging agent」に相当
https://kubernetes.io/docs/concepts/cluster-administration/logging/

2-2. スレッドダンプの出力方式実装例

- JavaAPコンテナへログインし、jstackコマンドによりスレッドダンプを出力します
  - 取得したスレッドダンプはkubectl cpなどでPod外部へ持ち出します

注意・補足事項

- コンテナにログインしてスレッドダンプを採取する前提のため、ログインシェルが用意されたイメージを用いてコンテナを作成する必要があります
  - JDKを使用している場合はそのままjstackを使用可能。JREやカスタムJDKを使用している場合は別途用意が必要です
- JavaAPコンテナを作成する際に、予めjstackコマンドを使える状態にする必要があります



2-3. ヒープダンプの出力方式実装例

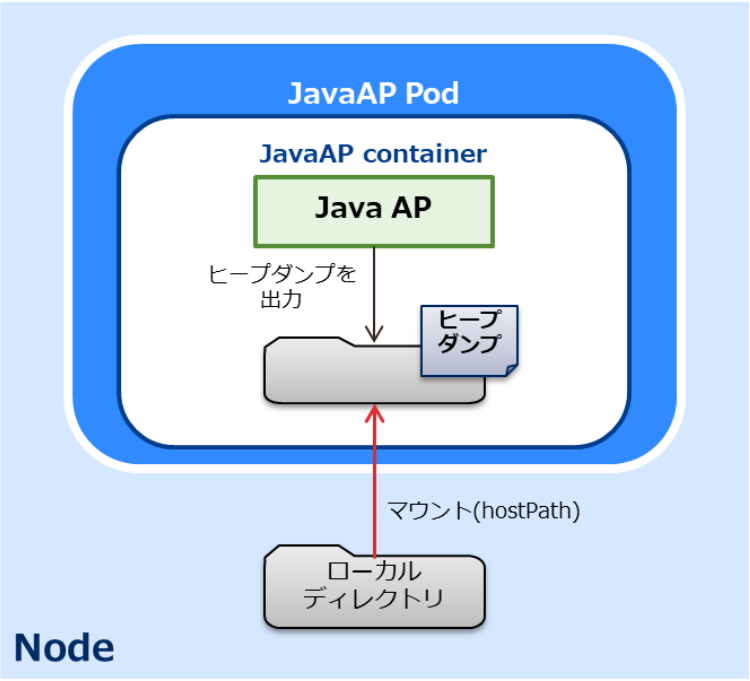
- JavaAP Podに対してPVをマウントし、そこにヒープダンプを出力するように設定します
  - OutOfMemoryError発生でPodが停止した際に、ヒープダンプがロストするのを防ぐためPVを利用しています

Point

1. JavaAP PodにPVをマウントし、コンテナから参照できるようにする
2. Javaアプリケーションの起動オプションでヒープダンプの出力を有効化し、ヒープダンプ出力先としてPVをマウントした領域を指定する

注意・補足事項

•コンテナ上でファイル出力をする場合、ファイルキャッシュの量もコンテナのメモリ制限(Limits)にカウントされるため、ヒープダンプのように容量の大きいファイルを出力する場合にはメモリのLimitsを余裕を持たせた設定値とすること



2-4. JMXの出力方式について

- JavaアプリケーションのJMXを取得する場合、以下の2つの方法が考えられます
  1. Prometheusなど、JMXを監視情報として取得できるツールを用いる
  2. jconsoleなど、JMXの値をそのまま取得する従来型ツールを用いる

- 一般に本番運用では、一定間隔でJMXのメトリクスを取得し続けて監視を行うことになるため、上記の項番1の方法で採取することを推奨します
  - 補足：Prometheusの場合、JMX exporter (https://github.com/prometheus/jmx_exporter) を用いて採取する方式が考えられます

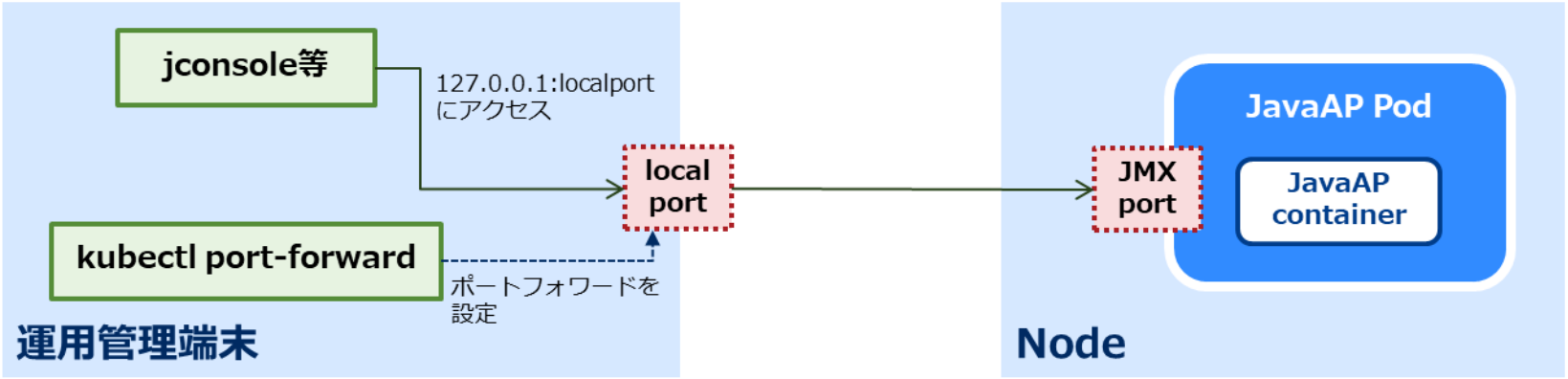
- 一時的な情報取得が必要等の理由により2で採取したい場合、推奨する方法ではありませんが採取することは可能です。**参考情報として**、次ページに設定方法を紹介します

(参考)jconsoleを用いたJMXの出力方式実装例

- 運用管理端末で“kubectl port-forward”を実行して、JavaAP PodのJMX用ポートのポートフォワードを設定し、127.0.0.1:【ローカルポート】でJMX接続を行います
  - JavaAPでは “java.rmi.server.hostname=127.0.0.1” を設定し、JMX取得ツールが参照するアドレスである”127.0.0.1”を受け付けるようにします

Point

1. “java.rmi.server.hostname”は、JMXを取得するツールがアクセス可能なホスト名/IPアドレスを指定する
2. “kubectl port-forward”を用いてポートフォワードを設定し、ツールから127.0.0.1を用いてアクセスできるようにする



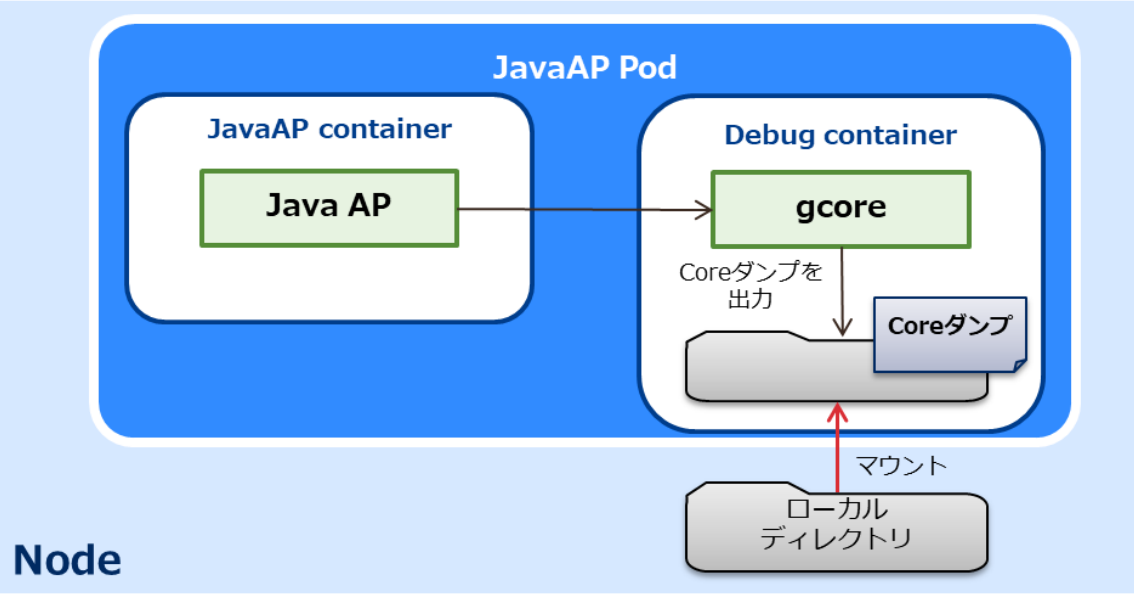
2-5. Coreダンプの出力方式実装例

- JavaAP Pod内にシェルログイン可能なデバッグ用コンテナを配置し、そこからJavaAPのCoreダンプを採取する
  - デバッグ用コンテナにはgdbおよびJavaAPコンテナに合わせたDebugInfoをインストールし、gcoreコマンドを使用できるようにしておきます

Point

1. gdbをインストールしたデバッグ用コンテナを事前にJavaアプリのPod内に配置しておく
2. デバッグ用コンテナにはCoreダンプ出力用のPVをマウントして、そこにCoreダンプを出力する
3. shareProcessNamespaceを有効にして、デバッグ用コンテナからJavaプロセスが見えるようにする

Coreダンプ出力の注意・補足事項については、Appendix4を参照



3. まとめ

“おなじみの情報”はコンテナライズド環境でも採取可能です。
採取方法に工夫が必要なケースが多くあることに注意して、安全なリフト＆シフトを実現しましょう！

◆ コンテナ環境におけるログの取得時の注意点まとめ

- ボリュームをコンテナにマウントしたりログ収集基盤を用意したり等、今までとは異なった方法が必要となるケースが多くあります
- 採取するメッセージが混在すると、既存の解析ツールとの連携が困難になる場合があるため、採取ルール等の作りこみで対応が必要です（スレッドダンプなど）
- 採取するためには、コンテナに適切な設定がされている必要があるケースもあるため注意してください
  - スレッドダンプ、ヒープダンプをコンテナに直接アクセスして取得する
    →コンテナにログインする権限があるユーザか？コンテナ自体がログインできる作りになっているか？
  - Coreダンプを取得する
    →コンテナ起動時に適切な権限を与えて起動しているか？

性能トピック:コンテナでJavaを使用する際の注意事項

注意事項

オンプレミス時のJVM設定をそのままコンテナ環境に適用しようとすると、デフォルト値が自動計算された結果として**GCアルゴリズムや、スレッド数/プール数等が意図せず非コンテナ環境と異なる値となる場合がある**

対策

- cpu.shareを指定する(2000などの値を指定 ※1)
- JVMオプションで、-Xmx、-XX:ActiveProcessorCount、-XX:MaxRAMPercentage、GCアルゴリズム、スレッド数/プール数等を明示的に指定する

- コンテナの起動オプションの例

```
docker run -ti --rm --memory 300m –cpus 2 java
```

- 上記例において、自動計算の結果CPU/メモリ関連パラメータは以下となる

リソース	Java内部パラメータ	値	計算
CPU	availableProcessors	1	cpu.shares ÷ 1024とcpu_countの小さい方の値が使用される。 例では cpu.shares = 2(デフォルト値)、cpu_count=2(--cpusオプションの指定値)となり、cpu.share(2) ÷ 1000 = 1(最小値)が使用される
メモリ	physicalMemory	314572800(300MB)	physicalMemory=300m(--memoryの指定値)となる (※2)

※1 cpu.shareはAmazon ECS/Fargateの場合、追加で以下の固有の制約がある。
タスクのCPU設定は、cpu_count(CPU÷1024)に反映され、コンテナのCPU設定はcpu.shareに反映される。
また、タスク内で複数のコンテナを起動する場合、コンテナ単位のCPUの合計をタスクのCPU以下とする必要がある。

例:タスクのCPUを2048とし、コンテナを2つ起動する場合は、コンテナ1を1792、コンテナ2を256など、
合計2048以下に設定する。(2000を指定することで、availableProcessors=2とすることはできない)

※2 最大ヒープサイズは、
physicalMemory×MaxRAMPercentageで計算されるため、
デフォルトでコンテナのメモリ上限の25%になる。
Javaプロセスでコンテナのメモリ上限を専有できる前提であれば、
25%では一般的にリソース効率が低くなるため注意。

Appendix

Appendix1：前提事項詳細

◆ 対象とするJavaAP/ログ（1）

■ 対象とするJavaAP/ログの想定は以下の表の通り

項番	項目	内容	理由/備考
1	JDKのバージョン	• JDK 8（1.8）を対象とする	• オンプレミスで動作しているJavaAPはアプリケーションサーバ上での動作するものがほとんどであり、基本的にJREではなくJDK必須となる。また、現行はバージョン8（1.8）が大多数であると考えられる。そのため、JDKかつJVMのバージョンを変更せずにコンテナ化するケースを対象とした
2	コンテナ化における考え方	• 本セッションでは、既存のオンプレで動作していたAPをコンテナ化することでポータビリティを得ることを主目的と考える • JavaAPコンテナのSW構成には極力手を入れないことを前提とする	• APの設計変更を含めて同時に進めると、SW構成や既存のAPコードに大きな変更が発生し、ログ出力方式の考え方も変わるため
3	対象JavaAPの特性	• log4jやlogbackなどの標準的なログ出力ライブラリを利用して、ファイル出力しているようなバッチ、WebAP • ただし、一般的にWebAPサーバ上で動作するAPのログが一番種類が多く、他のタイプのAPのログはその種類に内包されるため本セッションではWebAPにおけるログを基準として記載した	• Javaを利用したAPで、新規に設計してクラウドネイティブな仕組みになっているものは、クラウドネイティブに適した方式でログ採取すべきであるため対象外とする

Appendix1：前提事項詳細

◆ 対象とするJavaAP/ログ（2）

項番	項目	内容	理由/備考
4	ベースコンテナイメージ	• OSベースイメージ（RHEL、等） • JDKベースイメージ • APサーバベースイメージ	• AlpineLinux、BusyBoxのようにOSの基本機能を絞って軽量化したOSイメージでは、セッション本編に記載の手法でのログ採取ができない可能性もあるため
5	出力するログ/ダンプ	• ログ： アプリケーションログ/サーバログ/APサーバドメインログ、GCログ、標準出力/標準エラー出力 • ダンプ： ヒープダンプ、スレッドダンプ、Coreダンプ • JMX経由で取得するログ	• いずれも一般的なJavaAPでは採取/解析対象となるログ/ダンプであるため

補足

JDKの提供元やエディションによっては標準機能ではない「Java Mission Control(JMC)」「Java Flight Recorder(JFR)」が出力するログは個別には言及していません。JMC/JFRのログは、本編に記載の「イベント形式」と「スナップショット形式」の両方のログを取得可能なツールであるため、オンプレ環境で取得していた形式に合わせて、コンテナ環境での出力方式を選択してください。

Appendix2：コンテナにおけるログ出力方針

① イベント記録形式

取得方法	• ファイル経由でサイドカーのFluentdでタグを付与した上でログ基盤に転送します • オンプレミス環境で標準出力に出していた情報は、リダイレクト等で一旦ファイルを経由するように変更します
理由	• JavaAPのイベント記録形式ログは、元々ファイル出力しているケースがほとんどであり、アプリケーション側の設定変更が不要 • ファイルを読み取ってログ基盤に転送する機能は、利用実績が多いFluentdを第一候補として検討

補足

- ファイルのローテート/世代管理については、従来と同様APサーバもしくはJVMの機能で行います。Fluentdのtailプラグインは「tail -F」コマンドに似た仕組みで、指定したファイルにローテートが発生した場合でも新しいファイルを自動で読み込む動作をするため対応可能です。
  （参考）https://docs.fluentd.org/input/tail
- APサーバのログ出力設定は、デフォルトでファイルと標準出力両方になっているケースが多いため、二重に採取されないよう標準出力への出力を無効にしてください。無効にできない場合や、標準出力をファイルにリダイレクトできない場合、そのまま標準出力に流し、Fluentdで標準出力経由かファイル経由かをタグ付けすることで、ログ基盤側で区別が可能です

Appendix2：コンテナにおけるログ出力方針

② スナップショット形式

取得方法	• 従来と同じくコマンドやオプションによりファイルに出力させます • 出力先はマウントしたPV（Persistent Volume）とします
理由	• JavaAPにおけるスナップショット形式のログは、障害時の解析が主目的であり、常時取り続ける必要はありません • スレッドダンプは標準出力へも出力可能ですが、常時出力されているイベント記録形式のログと混在するため、個別のファイルとして取得することとしました • 取得サイズが大きくなる（特にヒープダンプはヒープサイズによっては数十GB）ため、ディスクサイズに余裕のあるボリュームのマウントと出力設定が必要です

補足

• コマンドを実行するにあたり、環境ごとに権限設定が必要となる場合があります

• 特定のTCPポートを経由して採取するログ（JMXなど）は、あらかじめ採取ツールからポートへのアクセスを許可する必要があります

Appendix3：emptyDir利用時の注意・補足事項

- emptyDirはファイルシステムとメモリが選択可能ですが、基本的にデフォルトのままファイルシステムに出力することを第一選択肢として下さい
  - Kubernetesの設定でmediumの設定値を“Memory”とすることでtmpfs（RAM上のファイルシステム）に設定することも可能ですが、以下のような問題にぶつかる可能性があるため、推奨はいたしません
    - 書き込むファイル容量がコンテナのメモリ制限（Limits）にカウントされ、設計時にメモリ消費量＋ログサイズでの考慮が必要になります
    - “Memory”選択時に、サイズ指定をしていないと、デフォルトでNode Allocatable（つまりほぼノードのメモリサイズ）のサイズでtmpfsが作成され、それを複数作成も可能ですので意図しない大量のメモリ消費に繋がります

Appendix4：Coreダンプ出力の注意・補足事項

1. Coreダンプ出力設定について

- コンテナに障害が発生した場合のCoreダンプは、Workerノードのシステム設定（/proc/sys/kernel/core_pattern、ulimit、等）に従って出力されます。コンテナはホストとカーネルパラメータを共有するため、**設定はノード内の全コンテナで共通**となります
- **マネージドサービスの制約で、OSのCoreダンプ出力設定の変更が許可されていない場合**、コンテナ障害時のCoreダンプ出力先変更や、ダンプ取得ができないケースがあります

2. gcoreによるCoreダンプ採取時の権限設定について

- 「2-5. Coreダンプの出力方式実装例」に記載した方法でCoreダンプを採取する場合、**対象のJavaプロセスの構成によっては、取得時に「SYS_PTRACE」権限の付与が必要になる**場合があります
  - 過去の動作確認を通して確認できた範囲では、あるJavaプロセスからforkして別のJavaプロセスを呼び出すような構成をしたJavaAPにおいて、「SYS_PTRACE」権限の付与が必要なケースが確認されました

\Orchestrating a brighter world

NECは、安全・安心・公平・効率という社会価値を創造し、
誰もが人間性を十分に発揮できる持続可能な社会の実現を目指します。


Orchestrating a brighter world
NEC