OpenShift Container Platform 3.11

スケーリングおよびパフォーマンスガイド

OpenShift Container Platform 3.11 スケーリングおよびパフォーマンスガイド

Last Updated: 2023-03-25

OpenShift Container Platform 3.11 スケーリングおよびパフォーマンスガイド

OpenShift Container Platform 3.11 スケーリングおよびパフォーマンスガイド

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律上の通知

Copyright © 2022 | You need to change the HOLDER entity in the en-US/Scaling_and_Performance_Guide.ent file |.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

概要

実稼働環境でのクラスターのスケールアップおよびパフォーマンスのチューニング

目次

第1章 概要 **4**

第2章 推奨のインストール作業 **5**
2.1. 依存関係の事前インストール 5
2.2. ANSIBLE のインストールの最適化 5
2.3. ネットワークの留意事項 6

第3章 ホストの推奨プラクティス **7**
3.1. OPENSHIFT CONTAINER PLATFORM マスターホストの推奨プラクティス 7
3.2. OPENSHIFT CONTAINER PLATFORM ノードホストの推奨プラクティス 8
3.3. OPENSHIFT CONTAINER PLATFORM ETCD ホストの推奨プラクティス 9
3.3.1. OpenStack の PCI パススルーを使用した etcd ノードへのストレージ提供 14
3.4. TUNED プロファイルを使用したホストのスケーリング 15

第4章 コンピュートリソースの最適化 **17**
4.1. オーバーコミット 17
4.2. イメージの留意事項 17
4.2.1. 事前デプロイ済みのイメージを使用した効率の強化 17
4.2.2. プルする前のイメージ 18
4.3. RHEL ツールのコンテナーイメージを使用したデバッグ 18
4.4. ANSIBLE ベースのヘルスチェックを使用したデバッグ 18

第5章 永続ストレージの最適化 **20**
5.1. 概要 20
5.2. 一般的なストレージガイドライン 20
5.3. ストレージの推奨事項 21
5.3.1. 特定アプリケーションのストレージの推奨事項 22
5.3.1.1. レジストリー 22
5.3.1.2. スケーリングされたレジストリー 22
5.3.1.3. モニターリング 23
5.3.1.4. ロギング 23
5.3.1.5. アプリケーション 23
5.3.2. 特定のアプリケーションおよびストレージの他の推奨事項 24
5.4. グラフドライバーの選択 24
5.4.1. SELinux で OverlayFS または DeviceMapper を使用する利点 28
5.4.2. Overlay と Overlay2 のグラフドライバーの比較 28

第6章 一時ストレージの最適化 **30**
6.1. 概要 30
6.2. 一般的なストレージガイドライン 30

第7章 ネットワークの最適化 **32**
7.1. ネットワークパフォーマンスの最適化 32
7.1.1. ネットワークでの MTU の最適化 32
7.2. ネットワークサブネットの設定 33
7.3. IPSEC の最適化 34

第8章 ルーティングの最適化 **35**
8.1. OPENSHIFT CONTAINER PLATFORM HAPROXY ルーターのスケーリング 35
8.1.1. ベースラインのパフォーマンス 35
8.1.2. パフォーマンスの最適化 36
8.1.2.1. 最大接続数の設定 36
8.1.2.2. CPU および割り込みアフィニティー 36

8.1.2.3. スレッド数の増加 37
8.1.2.4. バッファー増加の影響 37
8.1.2.5. HAProxy 再読み込みの最適化 37
第9章 クラスターメトリクスのスケーリング 38
9.1. 概要 38
9.2. OPENSHIFT CONTAINER PLATFORM についての推奨事項 38
9.3. クラスターメトリクスのキャパシティーピニング 38
9.4. OPENSHIFT CONTAINER PLATFORM メトリクス POD のスケーリング 39
9.4.1. 前提条件 39
9.4.2. Cassandra コンポーネントのスケーリング 39
第10章 CLUSTER MONITORING OPERATOR のスケーリング 41
10.1. 概要 41
10.2. OPENSHIFT CONTAINER PLATFORM についての推奨事項 41
10.3. CLUSTER MONITORING OPERATOR の容量計画 41
10.3.1. ラボ環境 42
10.3.2. 前提条件 42
第11章 クラスターごとのテスト済み最大数 43
11.1. メジャーリリースについての OPENSHIFT CONTAINER PLATFORM のテスト済みクラスターの最大値 43
11.2. OPENSHIFT CONTAINER PLATFORM のテスト済みのクラスターの最大値 44
11.2.1. ルート最大値 45
11.3. OPENSHIFT CONTAINER PLATFORM クラスターの最大値をテストする環境および設定 45
11.4. クラスターの最大値に合わせた環境計画 46
11.5. アプリケーション要件に合わせた環境計画 46
第12章 クラスターローダーの使用 48
12.1. クラスターローダーの機能 48
12.2. クラスターローダーのインストール 48
12.3. クラスターローダーの実行 48
12.4. クラスターローダーの設定 48
12.4.1. 設定フィールド 48
12.4.2. クラスターローダー設定ファイルの例 52
12.5. 既知の問題 53
第13章 CPU マネージャーの使用 54
13.1. CPU マネージャーの機能 54
13.2. CPU マネージャーの設定 54
第14章 HUGE PAGE の管理 58
14.1. HUGE PAGE の機能 58
14.2. 前提条件 58
14.3. HUGE PAGE の消費 58
第15章 GLUSTERFS ストレージでの最適化 60
15.1. データベースのコンバージョンモードに関するガイド 60
15.2. テスト済みのアプリケーション 60
15.3. サポートマトリックス 60
15.4. テスト結果 61

第1章 概要

本ガイドは、OpenShift Container Platform クラスターのパフォーマンスを向上し、OpenShift Container Platform プロダクションスタックの異なるレベルでスケーリングを行う方法についてその手順や例を提供しています。また、本書には、OpenShift Container Platform クラスターのビルド、スケーリング、チューニングの推奨プラクティスが説明されています。

チューニングの留意点は、クラスターの設定により異なり、本書に記載のパフォーマンスに関する推奨事項を実行することで他の部分に影響が及ぶ可能性があるので注意してください。

第2章 推奨のインストール作業

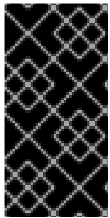
2.1. 依存関係の事前インストール

ノードホストは、ネットワークにアクセスして、**atomic-openshift-***, **iptables**、および **CRI-O** または **Docker** などの RPM 依存関係をインストールします。これらの依存関係を事前にインストールすると、インストール中にホストごとに何度もアクセスするのではなく、必要な時にしか RPM にアクセスしないので、より効率的にインストールができます。

また、セキュリティー上の理由でレジストリーにアクセスできないマシンにも有用です。

2.2. ANSIBLE のインストールの最適化

OpenShift Container Platform のインストール手法では Ansible を使用します。Ansible は並行して実行する操作に役立ち、迅速かつ効率的なインストールを促進します。ただし、これらの操作はチューニングオプションを追加してさらに強化することができます。利用可能な Ansible 設定オプションの一覧については、Ansible の設定セクション を参照してください。



重要

並行動作は、イメージレジストリーや Red Hat Satellite サーバーなどのコンテンツソースに負荷をかける可能性があります。サーバーのインフラストラクチャー Pod やオペレーティングシステムのパッチを用準備することで、この問題の回避できる可能性があります。

レイテンシーを最小限に抑えたコントロールノード (LAN 速度) からインストーラーを実行します。ワイドエリアネットワーク (WAN) での実行や、ネットワーク接続が途切れる可能性のある環境でのインストールの実行は推奨しません。

Ansible では、RHEL 6.6 以降を使用して OpenSSH のバージョンが ControlPersist をサポートすることを確認することや、クラスター内のマシンから **実行せずに**、クラスターと同じ LAN からインストーラーが実行されるようにするなどの、独自の**パフォーマンスやスケーリングに関する指針**が提供されます。

以下は、Ansible がまとめた推奨事項を組み込んだ、大規模なクラスターのインストールや管理を行うための Ansible の設定例です。

```
# cat /etc/ansible/ansible.cfg
```

出力例

```
# config file for ansible -- http://ansible.com/
# ==============================================
[defaults]
forks = 20 (1)
host_key_checking = False
remote_user = root
roles_path = roles/
gathering = smart
fact_caching = jsonfile
fact_caching_connection = $HOME/ansible/facts
fact_caching_timeout = 600
log_path = $HOME/ansible.log
```

```
nocows = 1
callback_whitelist = profile_tasks

[privilege_escalation]
become = False

[ssh_connection]
ssh_args = -o ControlMaster=auto -o ControlPersist=600s -o ServerAliveInterval=60
control_path = %(directory)s/%%h-%%r
pipelining = True 2
timeout = 10
```

1 フォークは 20 に設定することが理想です (フォークが多くなるとインストールに失敗する可能性があるため)。

2 パイプラインは、コントロールノードとターゲットノードの間の接続数を減らし、インストーラーのパフォーマンスを向上させます。

2.3. ネットワークの留意事項

ネットワークサブネットの変更はインストール後に実行できますが、これを容易に実行することはできません。サイズを少なく見積もってしまうとクラスターを拡張する際に問題が発生する可能性があるので、ネットワークサブネットのサイズをインストール前に検討することにより作業を大幅に容易にすることができます。

ネットワークサブネットに関する推奨プラクティスは、ネットワークの最適化 のトピックを参照してください。

第3章 ホストの推奨プラクティス

3.1. OPENSHIFT CONTAINER PLATFORM マスターホストの推奨プラクティス

OpenShift Container Platform インフラストラクチャーで、Pod トラフィックの他に最も使用されるデータパスは OpenShift Container Platform マスターホストと etcd 間のデータパスです。OpenShift Container Platform API サーバー (マスターバイナリーの一部) は、ノードのステータス、ネットワーク設定、シークレットなどについて etcd に確認します。

以下を実行してこのトラフィックパスを最適化します。

- etcd をマスターホストで実行する。デフォルトで、etcd はすべてのマスターホストの静的 Pod で実行されます。
- マスターホスト間でレイテンシーが低く、混雑していない通信リンクを確保する

OpenShift Container Platform マスターは、CPU 負荷を軽減するためにデシリアライズされたバージョンのリソースを積極的にキャッシュします。ただし、1000 Pod 未満の小規模なクラスターでは、このキャッシュにより、無視できる程度の CPU 負荷を削減するために大量のメモリーが浪費される可能性があります。デフォルトのキャッシュサイズは 50,000 エントリーですが、リソースのサイズによっては 1 から 2 GB メモリーを占有する程度まで拡大する可能性があります。キャッシュのサイズは、**/etc/origin/master/master-config.yaml** で以下の設定を使用して縮小できます。

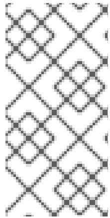
```
kubernetesMasterConfig:
  apiServerArguments:
    deserialization-cache-size:
    - "1000"
```

API サーバーに送信されるクライアント要求または API 呼び出しの数は、1 秒あたりクエリー (QPS: Queries Per Second) の値によって決まり、API サーバーで処理できる同時要求の数は、maxRequestsInFlight 設定によって決まります。クライアントが QPS レートを超えることのできる要求の数は、バースト値によって異なります。これは、バースト性があり、不規則な数の要求を実行できるアプリケーションに役立ちます。とくに大容量または高密度のクラスターなど、API サーバーが処理するる同時要求が多数ある場合は、要求の応答時間のレイテンシーが長くなる可能性があります。Prometheus の **apiserver_request_count** レートメトリクスを監視し、**maxRequestsInFlight** と **QPS** を適宜調整することが推奨されます。

API サーバーの CPU およびメモリーの消費としてデフォルト値を変更すると、適切なバランスが必要となります。また、追加の要求を並行して処理する際に etcd IOPS が増えます。また、監視されていない要求が多くなると、60 秒の固定タイムアウトの後にキャンセルされ、クライアントが再試行を開始するため、API サーバーに過負荷がかかる可能性がある点に注意してください。

API サーバーシステムで十分な CPU およびメモリーリソースが利用できることを前提とすると、API サーバー要求のオーバーロードする問題は、上記の要素を考慮し、 *_/etc/origin/master/master-config.yaml で maxRequestsInFlight、API qps およびバースト値を引き上げることによって安全に軽減することができます。

```
masterClients:
  openshiftLoopbackClientConnectionOverrides:
    burst: 600
    qps: 300
servingInfo:
  maxRequestsInFlight: 500
```

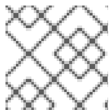


注記

上記の maxRequestsInFlight、qps および burst の値は OpenShift Container Platform のデフォルト値です。要求が 1 秒未満の場合、qps は maxRequestsInFlight 値よりも大きくすることができます。maxRequestsInFlight をゼロに設定されている場合には、サーバーが処理できる同時要求の数に制限はありません。

3.2. OPENSHIFT CONTAINER PLATFORM ノードホストの推奨プラクティス

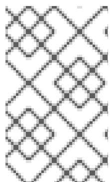
OpenShift Container Platform ノード設定ファイルには、iptables 同期期間、SDN ネットワークの Maximum Transmission Unit (MTU)、プロキシーモードなどの重要なオプションが含まれます。ノードを設定するには、適切な ノード設定マップ を変更します。



注記

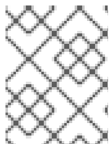
node-config.yaml ファイルを直接編集しないでください。

ノード設定ファイルでは、kubelet (node) プロセスに引数を渡すことができます。**kubelet --help** を実行すると、利用可能なオプション一覧を表示できます。



注記

kubelet オプションは、OpenShift Container Platform ですべてサポートされておらず、アップストリームの Kubernetes ですべてが使用されている訳ではありません。そのため、オプションによってはサポートに制限があります。

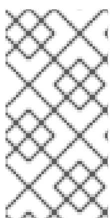


注記

OpenShift Container Platform の各バージョンでサポートされている最大制限については、Cluster maximums ページを参照してください。

/etc/origin/node/node-config.yaml ファイルでは、1 つのパラメーターでノードにスケジュールできる Pod の最大数 **max-pods** を制御します。**max-pods** オプションを使用すると、ノード上の Pod の数が制限されます。この値を超えると、次の結果になる可能性があります。

- OpenShift Container Platform と Docker の両方で CPU 使用率が増加する。
- Pod のスケジューリングの速度が遅くなる。
- メモリー不足のシナリオが生じる可能性がある (ノードのメモリー量によって異なる)。
- IP アドレスのプールを消費する。
- リソースのオーバーコミット、およびこれによるアプリケーションのパフォーマンスの低下。



注記

Kubernetes では、単一コンテナーを保持する Pod は実際には 2 つのコンテナーを使用します。2 つ目のコンテナーは実際のコンテナーの起動前にネットワークを設定するために使用されます。そのため、10 の Pod を使用するシステムでは、実際には 20 のコンテナーが実行されていることになります。

max-pods は、ノードのプロパティーにかかわらず、ノードが実行できる Pod 数を固定値に設定します。クラスターの制限 では、**max-pods** のサポートされる最大値について説明しています。

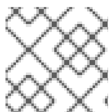
```
kubeletArguments:
  max-pods:
    - "250"
```

上記の例を使用すると、**max-pods** のデフォルト値は **250** です。

OpenShift Container Platform クラスターの推奨制限については、インストールドキュメントの サイジングに関する考慮事項 セクションを参照してください。推奨のサイズは、コンテナーのステータス更新時の OpenShift Container Platform とコンテナーエンジンの連携に対応します。この連携により、大量のログデータの書き込みなどを含め、マスターやコンテナーエンジンプロセスへの CPU の負荷がかかります。

kubelet が API サーバーと通信する速度は、qps およびバースト値によって異なります。各ノードで実行中の Pod に限りがある場合には、デフォルト値で十分です。ノードに CPU およびメモリーリソースが十分にある場合、qps および burst の値は **/etc/origin/node/node-config.yaml** ファイルで調整できます。

```
kubeletArguments:
  kube-api-qps:
  - "20"
  kube-api-burst:
  - "40"
```



注記

上記の qps およびバースト値は OpenShift Container Platform のデフォルトです。

3.3. OPENSHIFT CONTAINER PLATFORM ETCD ホストの推奨プラクティス

etcd は、OpenShift Container Platform が設定に使用するキーと値の分散ストアです。

OpenShift Container Platform のバージョン	etcd のバージョン	ストレージスキーマのバージョン
3.3 以前	2.x	v2
3.4 および 3.5	3.x	v2
3.6	3.x	v2 (アップグレード)
3.6	3.x	v3 (新規インストール)
3.7 以降	3.x	v3

etcd 3.x では、クラスターのサイズに拘わらず、CPU、メモリー、ネットワーク、ディスク要件を軽減する、スケーラビリティーおよびパフォーマンスの重要な強化機能が導入されました。また、etcd 3.x は、オンディスク etcd データベースの 2 段階移行をスムーズに行うため、後方互換のあるストレージ

API を実装します。移行の目的で、OpenShift Container Platform 3.5 の etcd 3.x で使用されるストレージモードは v2 モードのままとなっています。OpenShift Container Platform 3.6 の時点で、新規インストールで v3 のストレージモードが使用されます。OpenShift Container Platform の以前のバージョンからアップグレードしても、v2 から v3 に自動で **移行されません**。提供されている Playbook を使用して、ドキュメントに記載のプロセスに従い、データを移行する必要があります。

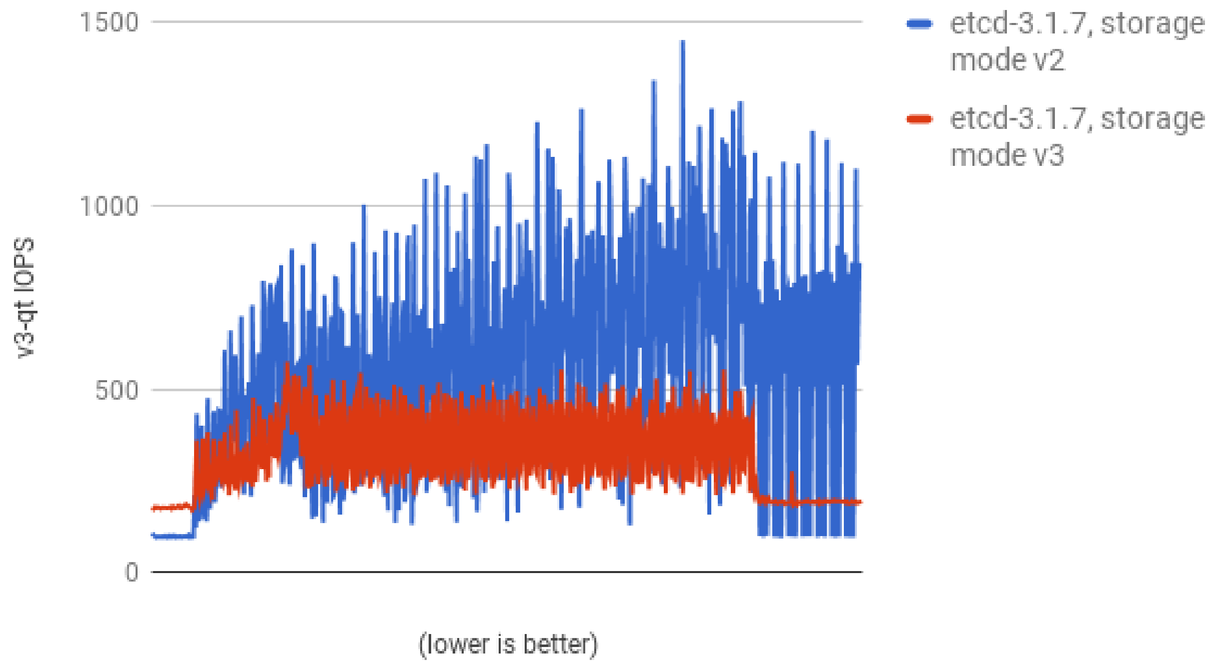
etcd のバージョン 3 では、on-disk etcd データベースの 2 ステップ移行を簡素化する後方互換性対応のストレージ API を実装します。移行の目的で、OpenShift Container Platform 3.5 の etcd 3.x で使用されるストレージモードは v2 モードのままとなっています。OpenShift Container Platform 3.6 の時点で、新規インストールで v3 のストレージモードが使用されます。OpenShift Container Platform 3.7 へのアップグレードプロセスの一環として、必要な場合は etcd ストレージ API を v3 にアップグレードします。バージョン 3.7 以降では、v3 API を使用する必要があります。

新規インストールでストレージモードを v3 に変更するのに加え、OpenShift Container Platform 3.6 は、全 OpenShift Container Platform タイプに対して強制的に **quorum reads** を実行します。これは、etcd に対するクエリーが古くなったデータを返さないようにするために実行されます。単一ノードの etcd クラスターでは、古くなったデータが入っていても懸念はありません。実稼働クラスターで一般的に使用される高可用性の etcd デプロイメントでは、quorum read はクエリーの結果が有効になるようにします。quorum read は、データベース用語で **線形化可能性** のことで、すべてのクライアントにクラスターが最新の状態に更新されたものが表示され、同じ順番の読み取りおよび書き込みが表示されます。パフォーマンスの向上に関する情報は、etcd 3.1 の announcement を参照してください。

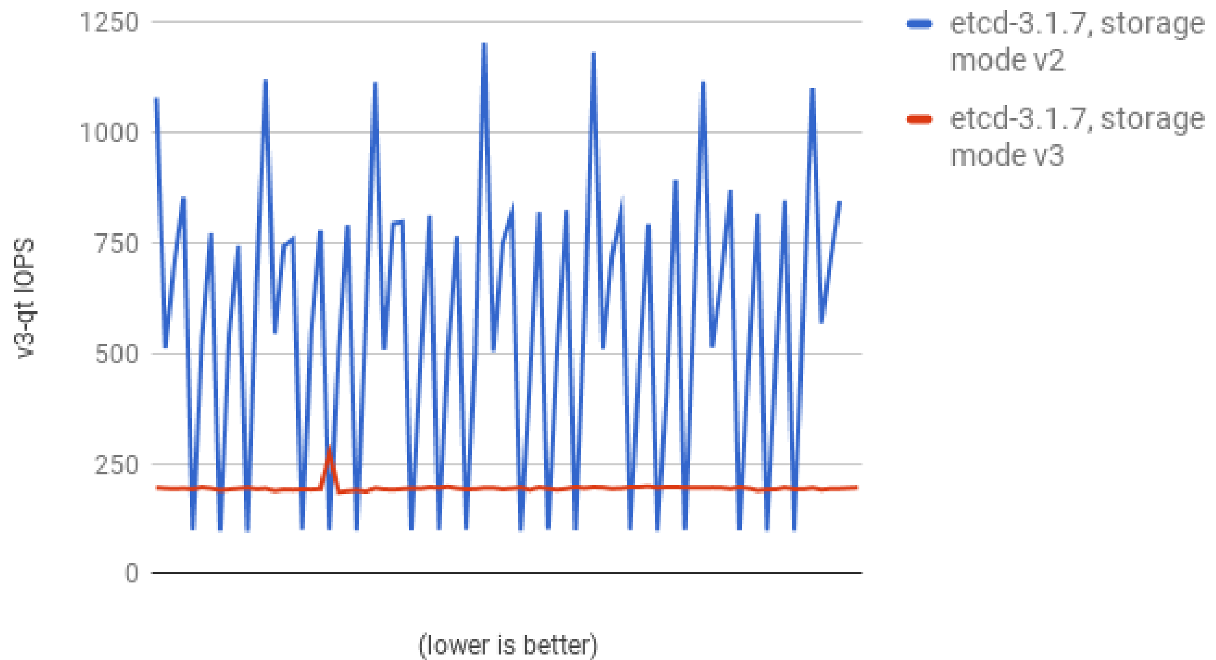
OpenShift Container Platform は、etcd を使用して Kubernetes 自体が必要な情報以外の追加情報を保存する点を留意することが重要です。たとえば、Kubernetes 以外に OpenShift Container Platform が追加する機能で必要になるので、OpenShift Container Platform は、etcd にあるイメージ、ビルド、他のコンポーネントの情報を保存します。最終的に、etcd ホストのパフォーマンスやサイジングに関する指針やその他の推奨事項は、Kubernetes とは顕著に異なります。Red Hat は最も正確な推奨事項を提案できるように、OpenShift Container Platform のユースケースやパラメーターを念頭において、etcd のスケーラビリティーやパフォーマンスをテストしています。

パフォーマンスの向上は、cluster-loader ユーティリティーで 300 ノードの OpenShift Container Platform 3.6 クラスターを使用して、定量化されています。etcd 3.x (ストレージモード v2) と etcd 3.x (ストレージモード v3) を比較すると、以下の図に示されるようにパフォーマンスの向上が明確に確認できます。

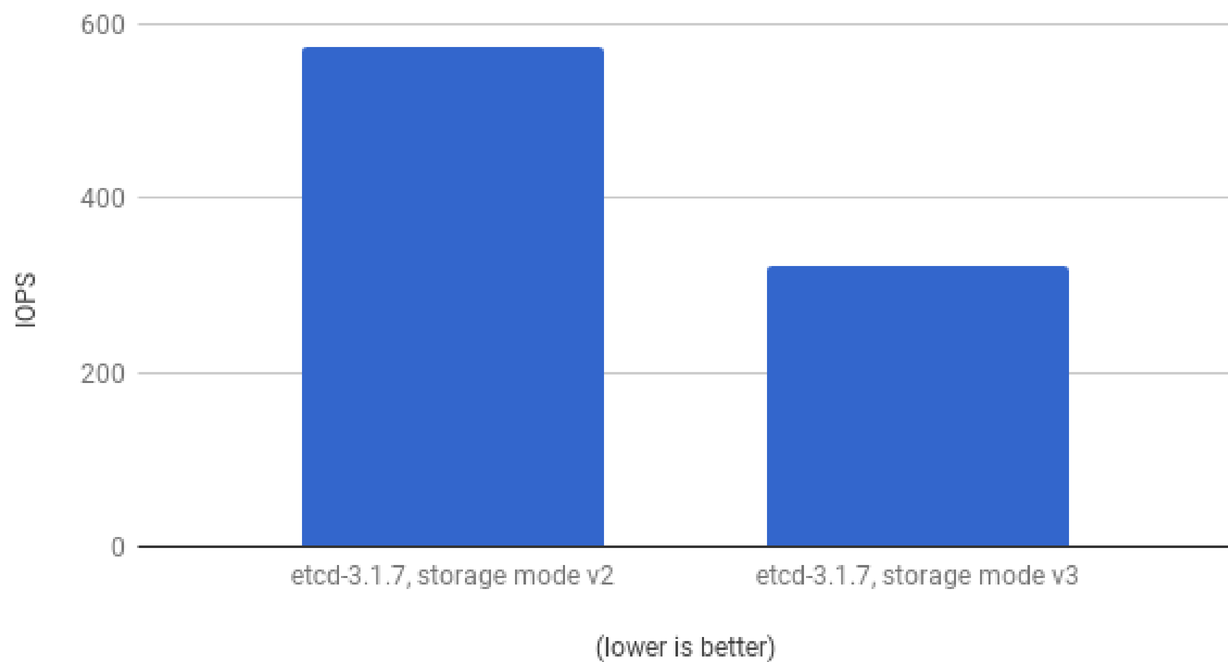
負荷のある状態でのストレージ IOPS が大幅に減少している:



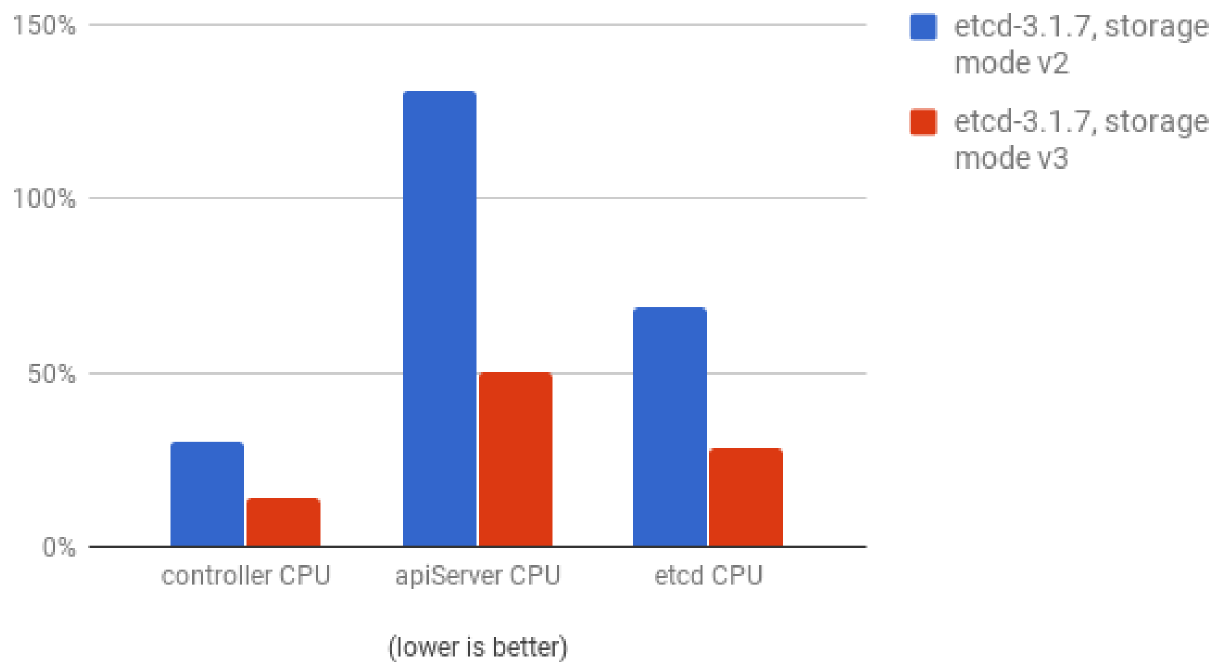
安定した状態でのストレージ IOPS が大幅に減少している:



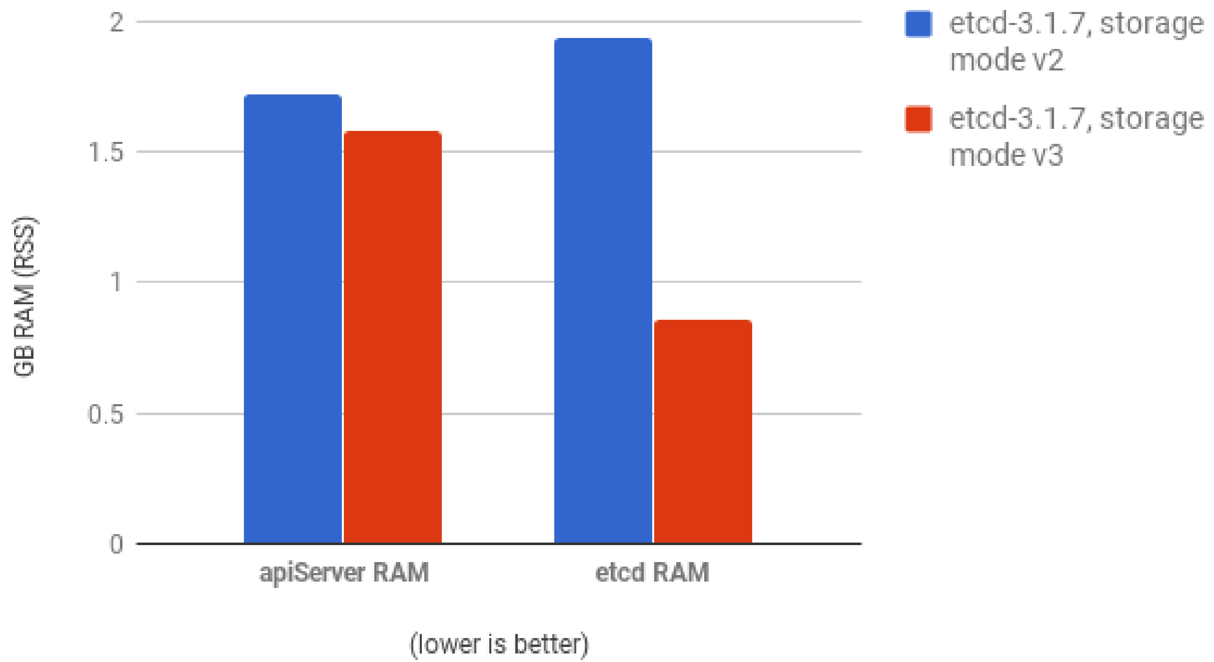
同じ I/O データの表示。 両モードでの平均 IOPS

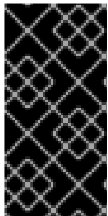


API サーバー (マスター) と etcd プロセス両方の CPU 使用率が減少している:



API サーバー (マスター) と etcd プロセスの両方のメモリー使用率も減少している:

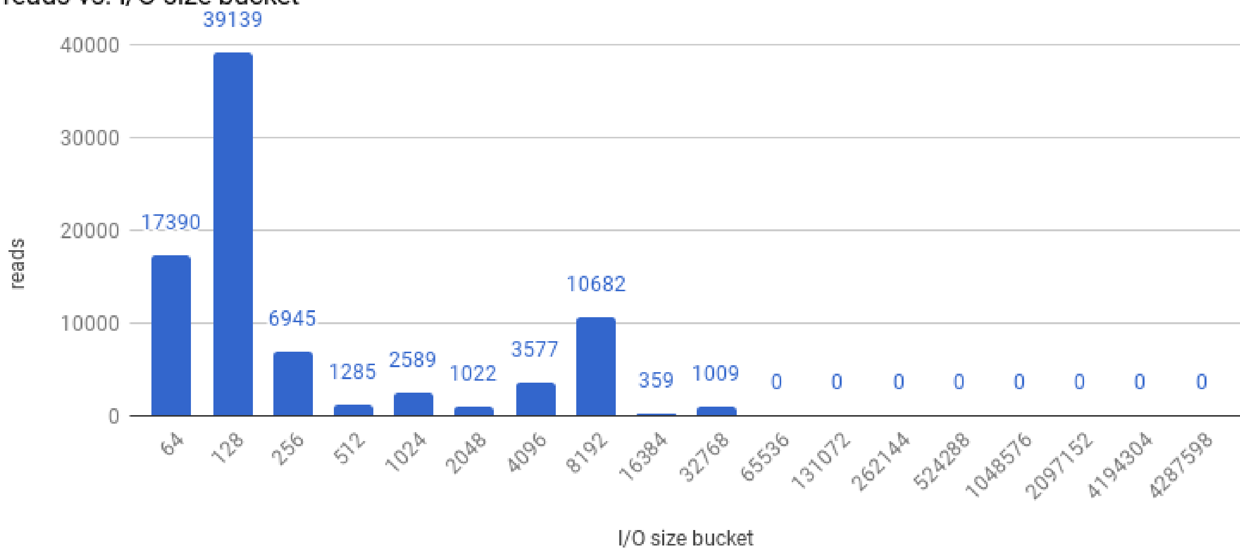




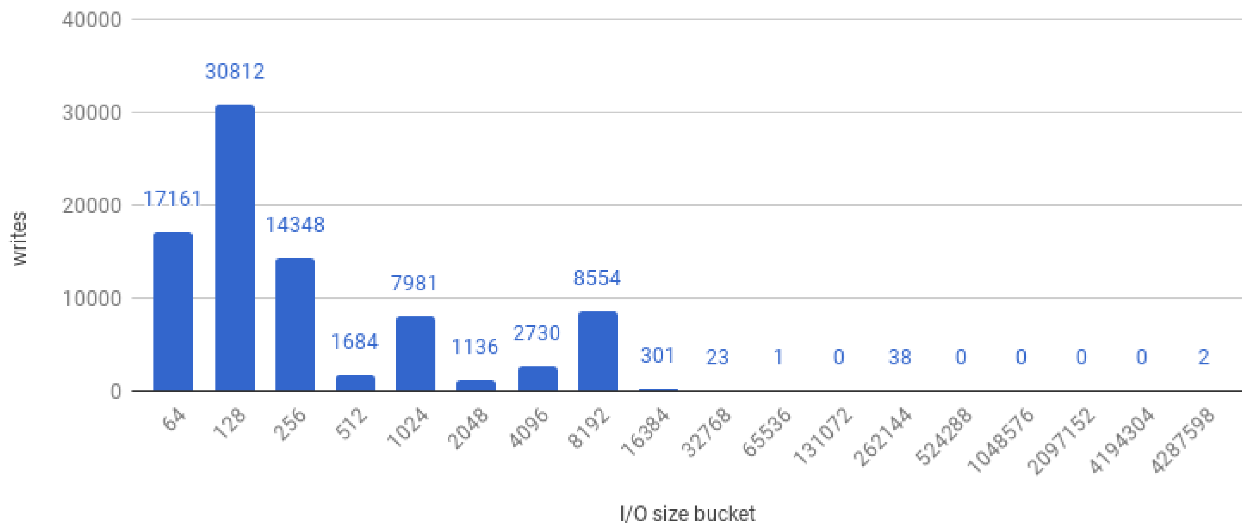
重要

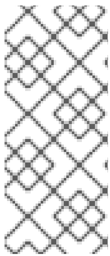
OpenShift Container Platform で etcd をプロファイリングした後に、etcd は少量のストレージインプットおよびアウトプットを頻繁に実行しています。SSD など、少量の読み取り/書き込み操作をすばやく処理するストレージで etcd を使用することを強く推奨します。

etcd 3.1 の 3 ノードクラスター (quorum reads を強制実行してストレージ v3 モードを使用) で実行したサイズ I/O 操作を確認してみると、読み取りサイズは以下のようになります。



また、書き込みサイズは以下のようになります。





注記

etcd プロセスは通常はメモリー集約型であり、マスター/ API サーバープロセスは CPU 集約型です。これらは、単一マシンや仮想マシン内で共同設置する上で有効なペアになります。これにより、1 台のマシンまたは仮想マシン内の適切なコロケーションペアが作成されます。etcd とマスターホスト間の通信を最適化するには、同じホストに共存させるか、専用のネットワークを指定します。

3.3.1. OpenStack の PCI パススルーを使用した etcd ノードへのストレージ提供

大規模な環境で etcd を安定させるために etcd ノードにストレージをすばやく提供するには、NVMe (Non-Volatile Memory express) デバイスを直接 etcd ノードに渡す PCI パススルーを使用します。これを Red Hat OpenStack 11 以降で設定するには、PCI デバイスが存在する OpenStack コンピュートノードで以下を実行してください。

1. Intel Vt-x が BIOS で有効化されているようにします。
2. IOMMU (Input-Output Memory Management Unit) を有効化します。**/etc/sysconfig/grub** ファイルで、**GRUB_CMDLINX_LINUX** の行末に、引用符で囲って **intel_iommu=on iommu=pt** を追加します。
3. 以下を実行して **/etc/grub2.cfg** を再生成します。

```
$ grub2-mkconfig -o /etc/grub2.cfg
```

4. システムを再起動します。
5. コントローラーの **/etc/nova.conf** を以下のように設定します。

```
[filter_scheduler]

enabled_filters=RetryFilter,AvailabilityZoneFilter,RamFilter,DiskFilter,ComputeFilter,ComputeCapabilitiesFilter,ImagePropertiesFilter,ServerGroupAntiAffinityFilter,ServerGroupAffinityFilter,PciPassthroughFilter

available_filters=nova.scheduler.filters.all_filters
```

```
[pci]

alias = { "vendor_id":"144d", "product_id":"a820",
"device_type":"type-PCI", "name":"nvme" }
```

6. コントローラーで **nova-api** と **nova-scheduler** を再起動します。

7. コンピュートノードの **/etc/nova/nova.conf** で以下のように設定します。

```
[pci]

passthrough_whitelist = { "address": "0000:06:00.0" }

alias = { "vendor_id":"144d", "product_id":"a820",
"device_type":"type-PCI", "name":"nvme" }
```

パススルーする NVMe デバイスの **address**、**vendor_id** および **product_id** の必須値を取得するには、以下を実行します。

```
# lspci -nn | grep devicename
```

8. コンピュートノードで **nova-compute** を再起動します。

9. 実行する OpenStack バージョンで NVMe を使用するように設定し、etcd ノードで起動します。

3.4. TUNED プロファイルを使用したホストのスケーリング

Tuned は、Red Hat Enterprise Linux (RHEL) および他の Red Hat 製品で有効な Tuning プロファイルの配信メカニズムです。**Tuned** は、sysctls、電源管理、カーネルコマンドラインオプションなどの Linux の設定をカスタマイズして、異なるワークロードのパフォーマンスやスケーラビリティーの要件に対応するために、オペレーティングシステムを最適化します。

OpenShift Container Platform は **tuned** デーモンを活用して、**openshift**, **openshift-node** and **openshift-control-plane** と呼ばれる **Tuned** プロファイルを追加します。これらのプロファイルは、カーネルで一般的に発生する垂直スケーリングの上限を安全に増やし、インストール時に自動的にシステムに適用します。

Tuned プロファイルは、プロファイル間の継承をサポートします。また、プロファイルが仮想環境で使用されるかどうかにより、親プロファイルを選択する親の自動割り当て機能もサポートします。**openshift** プロファイルは **openshift-node** および **openshift-control-plane** プロファイルの親で、両機能を使用します。これには、それぞれ OpenShift Container Platform アプリケーションノードとコントロールプレーンノードの両方に関連するチューニングが含まれます。**openshift-node** および **openshift-control-plane** プロファイルは、アプリケーションおよびコントロールプレーンノードにそれぞれ設定されます。

openshift プロファイルがプロファイル階層の親である場合に、OpenShift Container Platform システムに配信されるチューニングは、ベアメタルホスト向けの **throughput-performance** (RHEL のデフォルト) と、RHEL 向けの **virtual-guest** または RHEL Atomic Host ノード向けの **atomic-guest** を組み合わせて作成されます。

お使いのシステムでどの **Tuned** プロファイルが有効になっているかを確認するには以下を実行します。

```
# tuned-adm active
```

出力例

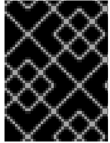
```
Current active profile: openshift-node
```

Tuned に関する詳細情報は、**Red Hat Enterprise Linux パフォーマンスチューニングガイド**を参照してください。

第4章 コンピュートリソースの最適化

4.1. オーバーコミット

CPU およびメモリーなどのリソースを必要とするクラスターの部分から、このようなリソースにアクセスしやすくなるように、オーバーコミットの手順を使用します。



重要

ハイパーバイザーと Kubernetes 間のスケジュールの衝突による不安定なクラスター動作を防ぐために、ハイパーバイザーレベルではオーバーコミットしないでください。

オーバーコミットすると、別のアプリケーションが必要としているリソースを必要な時にアクセスできなくなってしまうリスクがあり、結果的にパフォーマンスが低下しますが、パフォーマンスが低下する代わりに、密度が高まり、コストが削減されるので、代償として妥当な範囲である場合もあります。ただし、パフォーマンスが低下する代わりに、密度が高まり、コストが削減されるので、許容できるトレードオフになる場合もあります。たとえば、開発、品質保証 (QA) またはテスト環境でオーバーコミットできても、実稼働環境ではできない場合があります。

OpenShift Container Platform は、コンピュートリソースモデルやクォータシステムでリソース管理を実装します。詳細は、OpenShift リソースモデル を参照してください。

オーバーコミットに関する詳細情報およびストラテジーは、クラスター管理ガイドのオーバーコミットを参照してください。

4.2. イメージの留意事項

4.2.1. 事前デプロイ済みのイメージを使用した効率の強化

効率性の向上、全ノードホストでの設定の一貫性の維持、反復タスクの削減を図るため、複数のタスクを組み込んだ、ベースの OpenShift Container Platform イメージを作成できます。これは、事前デプロイ済みのイメージとして知られています。

たとえば、Pod を実行するために、すべてのノードには **ose-pod** イメージが必要なので、各ノードは定期的にコンテナーイメージレジストリーに接続して最新のイメージをプルする必要があります。100 のノードが同時にレジストリーに接続し、最新のイメージをプルしようとすると問題が発生する場合があり、イメージレジストリーでのリソースの競合や、ネットワーク帯域幅の無駄遣い、Pod の起動にかかる時間の増加などが発生する可能性があります。

事前にデプロイ済みのイメージをビルドするには、以下を実行します。

- 必要とされるタイプおよびサイズのインスタンスを作成します。
- コンテナー用の永続ボリュームとは別に、専用のストレージデバイスが CRI-O または Docker のローカルイメージやコンテナーストレージで利用できるようにします。
- システムを完全に更新すると共に、CRI-O または Docker がインストールされていることを確認します。
- ホストがすべての yum リポジトリーにアクセスできるようにします。
- シンプロビジョニングされた LVM ストレージを設定します。

- 一般的に使用するイメージ (**rhel7** ベースイメージ) および OpenShift Container Platform インフラストラクチャーコンテナーイメージ (**ose-pod**、**ose-deployer** など) を事前にデプロイ済みのイメージに事前に設定します。

OpenStack または AWS で実行できるなど、事前デプロイ済みのイメージに対して、適切なクラスター設定やその他のクラスター設定が済んでいるようにします。

4.2.2. プルする前のイメージ

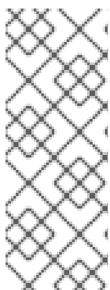
イメージを効率的に生成するには、全ノードホストに、必要とされるコンテナーイメージをすべてのノードホストに事前にプルしておきます。最初にイメージをプルする必要がないので、サイズが大きくなる可能性のある S2I、メトリクス、ロギングなどのイメージは特に、接続速度が遅いことが原因でパフォーマンスが低下することなく、時間を節約できます。

また、セキュリティー上の理由でレジストリーにアクセスできないマシンにも有用です。

または、指定したデフォルトレジストリーではなく、ローカルイメージを使用できます。これを実行するには、以下を行います。

1. Pod 設定の **imagePullPolicy** パラメーターを **IfNotPresent** または **Never** に設定して、ローカルイメージからプルします。

2. クラスターのすべてのノードで、同じイメージがローカルに保存されていることを確認します。



注記

ノードの設定を制御できる場合は、ローカルレジストリーからプルすることが適切ですが、GCE など、自動的にノードを交換しないクラウドプロバイダーでは確実に機能しない場合があります。ただし、GCE などのノードを自動的に置き換えるクラウドプロバイダーでは確実に動作しません。Google Container Engine (GKE) で実行している場合、Google Container Registry 認証情報が設定された各ノードに **a.dockercfg** ファイルがすでに存在します。

4.3. RHEL ツールのコンテナーイメージを使用したデバッグ

Red Hat は **rhel-tools** コンテナーイメージを配信します。これは、スケーリングがパフォーマンスの問題のデバッグをサポートするパッケージツールです。 このコンテナーイメージを使用すると、以下が可能です。

- ベースのディストリビューションからこのサポートコンテナーにパッケージを移動して、フットプリントが最小のコンテナーホストをデプロイできます。

- 不変のパッケージツリーが含まれる Red Hat Enterprise Linux 7 Atomic Host のデバッグ機能を提供します。**rhel-tools** には、tcpdump、sosreport、git、gdb、perf など、より一般的なシステム管理ユーティリティーが多数含まれています。

以下を実行して **rhel-tools** コンテナーを使用します。

```
# atomic run rhel7/rhel-tools
```

詳しい情報は、RHEL ツールコンテナーのドキュメント を参照してください。

4.4. ANSIBLE ベースのヘルスチェックを使用したデバッグ

追加のヘルスチェックは、OpenShift Container Platform クラスターのインストールおよび管理に使用する Ansible ベースのツール で利用できます。この正常性チェックでは、現行の OpenShift Container Platform インストールによくあるデプロイメントの問題を報告できます。

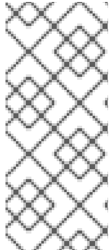
これらのチェックは、**ansible-playbook** コマンドの使用 (クラスターインストール で使用されるのと同じ方式) によるか、または **openshift-ansible** の コンテナー化されたバージョン として実行できます。**ansible-playbook** 方式については、チェックは **openshift-ansible** RPM パッケージを使って行われます。コンテナー化方式の場合は、**openshift3/ose-ansible** コンテナーイメージが Red Hat Container レジストリー 経由で配布されます。

利用可能な正常性チェックや使用例については、クラスター管理ガイドの Ansible ベースの正常性チェック を参照してください。

第5章 永続ストレージの最適化

5.1. 概要

ストレージを最適化すると、すべてのリソースでストレージの使用を最小限に抑えることができます。管理者は、ストレージを最適化することで、既存のストレージリソースが効率的に機能できるようにすることができます。



注記

本ガイドは主に、永続ストレージの最適化にフォーカスしています。Pod のライフタイムで使用するデータ向けのローカルの一時ストレージにはオプションが少なくなっています。一時ストレージは、一時ストレージのテクノロジープレビューを有効化した場合のみ利用できます。この機能はデフォルトで無効にされています。詳細情報は、一時ストレージの設定 を参照してください。

5.2. 一般的なストレージガイドライン

以下の表では、OpenShift Container Platform で利用可能な永続ストレージ技術を紹介します。

表5.1 利用可能なストレージオプション

ストレージタイプ	説明	例
ブロック	• ブロックデバイスとしてオペレーティングシステムに公開されます。 • ストレージを完全に制御し、ファイルシステムを通過してファイルの低いレベルで操作する必要のあるアプリケーションに適しています。 • ストレージエリアネットワーク (SAN) とも呼ばれます。 • 共有できません。 一度に 1 つのクライアントだけがこのタイプのエンドポイントをマウントできるという意味です。	converged mode/independent mode GlusterFS [1]、iSCSI、Fibre Channel、Ceph RBD、OpenStack Cinder、AWS EBS [1]、Dell/EMC Scale.IO、VMware vSphere Volume、GCE Persistent Disk [1]、Azure Disk
ファイル	• マウントされるファイルシステムのエクスポートとして、OS に公開されます。 • ネットワークアタッチストレージ (NAS) とも呼ばれます。 • 同時実行、レイテンシー、ファイルロックのメカニズムその他の各種機能は、プロトコルおよび実装、ベンダー、スケールによって大きく異なります。	コンバージドモード/インデペンデントモード GlusterFS [1]、RHEL NFS、NetApp NFS [2]、Azure File、Vendor NFS、Vendor GlusterFS [3]、Azure File、AWS EFS

ストレージタイプ	説明	例
オブジェクト	● REST API エンドポイント経由でアクセスできます。 ● OpenShift Container Platform レジストリーで使用するために設定できます。 ● アプリケーションは、ドライバーをアプリケーションやコンテナーに組み込む必要があります。	コンバージドモード/インデペンデントモード GlusterFS [1]、Ceph Object Storage(RADOS Gateway)、OpenStack Swift、Aliyun OSS、AWS S3、Google Cloud Storage、Azure Blob Storage、Vendor S3 [3]、Vendor Swift [3]

1. コンバージドモード/インデペンデントモード GlusterFS、Ceph RBD、OpenStack Cinder、AWS EBS、Azure Disk、GCE 永続ディスク、および VMware vSphere は、OpenShift Container Platform で永続ボリューム (PV) のネイティブプロビジョニングをサポートします。
2. NetApp NFS は Trident プラグインを使用する場合に動的 PV のプロビジョニングをサポートします。
3. Vendor GlusterFS、Vendor S3 および Vendor Swift のサポート機能および設定機能は異なる場合があります。

コンバージドモード GlusterFS (ハイパーコンバージドまたはクラスターホストのストレージソリューション) またはインデペンデントモード GlusterFS (外部ホストのストレージソリューション) を、OpenShift Container Platform レジストリー、ロギング、メトリクス用のブロック、ファイルおよびオブジェクトストレージに使用できます。

5.3. ストレージの推奨事項

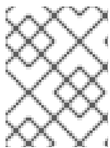
以下の表では、特定の OpenShift Container Platform クラスターアプリケーション向けに設定可能な推奨のストレージ技術についてまとめています。

表5.2 設定可能な推奨ストレージ技術

ストレージタイプ	RWO [1]	ROX [2]	RWX [3]	レジストリー	スケーリングされたレジストリー	モニターリング	ロギング	アプリ
ブロック	○	○ [4]	×	設定可能	設定不可	推奨	推奨	推奨
ファイル	○	○ [4]	○	設定可能	設定可能	設定可能 [5]	設定可能 [6]	推奨
オブジェクト	○	はい	○	推奨	推奨	設定不可	設定不可	設定不可 [7]

1. ReadWriteOnce

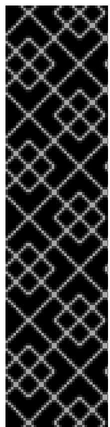
2. ReadOnlyMany
3. ReadWriteMany
4. これは、物理ディスク、VM 物理ディスク、VMDK、NFS 経由のループバック、AWS EBS、Azure Disk および Cinder (Block の場合は後者) には該当しません。
5. モニターリングコンポーネントの場合、ReadWriteMany (RWX) アクセスモードのファイルストレージを信頼できる方法で使用することはできません。ファイルストレージを使用する場合、監視機能と共に使用されるように設定される永続ボリューム要求 (PVC) で RWX アクセスモードを設定しないでください。
6. ロギングの場合、共有ストレージを使用することはアンチパターンとなります。logging-es ごとに 1 つのボリュームが必要です。
7. オブジェクトストレージは、OpenShift Container Platform の PV/PVC で消費されません。アプリは、オブジェクトストレージの REST API と統合する必要があります。



注記

スケーリングされたレジストリーとは、3 つ以上の Pod レプリカが稼働する OpenShift Container Platform レジストリーのことです。

5.3.1. 特定アプリケーションのストレージの推奨事項



重要

テストにより、RHEL NFS サーバーを RHEL でコンテナーイメージレジストリーのストレージバックエンドとして使用することに関する問題が検出されています。これには、OpenShift Container レジストリーおよび Quay、メトリクスストレージの Prometheus、およびロギングストレージの ElasticSearch が含まれます。そのため、コアサービスで使用される PV をサポートするために RHEL NFS サーバーを使用することは推奨されていません。

他の NFS の実装ではこれらの問題が検出されない可能性があります。OpenShift コアコンポーネントに対して実施された可能性のあるテストに関する詳細情報は、個別の NFS 実装ベンダーにお問い合わせください。

5.3.1.1. レジストリー

スケーリングなし/高可用性 (HA) ではない OpenShift Container Platform レジストリークラスターのデプロイメント:

- 推奨されるストレージ技術はオブジェクトストレージであり、次はブロックストレージです。ストレージ技術は、RWX アクセスモードをサポートする必要はありません。
- ストレージ技術は、リードアフターライト (Read-After-Write) の一貫性を確保する必要があります。NAS ストレージ (オブジェクトストレージインターフェイスを使用するのでコンバージドモード/インデペンデントモード GlusterFS 以外) は、実稼働環境のワークロードがある OpenShift Container Platform レジストリークラスターデプロイメントには推奨しません。
- **hostPath** ボリュームは、スケーリングなし/非 HA の OpenShift Container Platform レジストリー用に設定可能ですが、クラスターデプロイメントには推奨しません。

5.3.1.2. スケーリングされたレジストリー

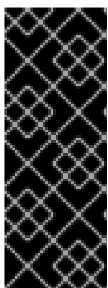
スケーリングされた/高可用性 (HA) の OpenShift Container Platform レジストリーのクラスターデプロイメント:

- 推奨されるストレージ技術はオブジェクトストレージです。ストレージ技術は、RWX アクセスモードをサポートし、リードアフターライトの一貫性を確保する必要があります。
- 実稼働環境のワークロードを処理するスケーリングされた/HA の OpenShift Container Platform レジストリークラスターのデプロイメントには、ファイルストレージやブロックストレージは推奨しません。
- NAS ストレージ (オブジェクトストレージインターフェイスを使用するのでコンバージドモード/インデペンデントモード GlusterFS 以外) は、実稼働環境のワークロードがある OpenShift Container Platform レジストリークラスターデプロイメントには推奨しません。

5.3.1.3. モニターリング

OpenShift Container がホストするモニターリングのクラスターデプロイメント:

- 推奨されるストレージ技術はブロックストレージです。
- ファイルストレージを設定する場合は、POSIX 標準に準拠していることを確認してください。



重要

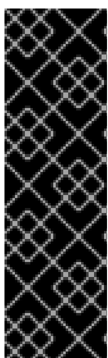
テストの結果、NFS を使用すると修復不能な大規模な破損が発生することが確認されたため、この使用は推奨されません。

他の NFS の実装ではこれらの問題が検出されない可能性があります。OpenShift コアコンポーネントに対して実施された可能性のあるテストに関する詳細情報は、個別の NFS 実装ベンダーにお問い合わせください。

5.3.1.4. ロギング

OpenShift Container Platform がホストするロギングのクラスターデプロイメント:

- 推奨されるストレージ技術はブロックストレージです。
- NAS ストレージ (iSCSI からのオブジェクトストレージインターフェイスを使用するのでコンバージドモード/独立モード GlusterFS 以外) は、実稼働環境のワークロードがある、ホスト型のメトリクスクラスターデプロイメントには推奨しません。



重要

テストにより、NFS サーバーを RHEL でコンテナーイメージレジストリーのストレージバックエンドとして使用することに関する問題が検出されています。これには、ロギングストレージの ElasticSearch が含まれます。そのため、コアサービスで使用される PV をサポートするために NFS を使用することは推奨されていません。

他の NFS の実装ではこれらの問題が検出されない可能性があります。OpenShift コアコンポーネントに対して実施された可能性のあるテストに関する詳細情報は、個別の NFS 実装ベンダーにお問い合わせください。

5.3.1.5. アプリケーション

以下の例で説明されているように、アプリケーションのユースケースはアプリケーションごとに異なります。

- 動的な PV プロビジョニングをサポートするストレージ技術は、マウント時のレイテンシーが低く、ノードに関連付けられておらず、正常なクラスターをサポートします。
- アプリケーション開発者はアプリケーションのストレージ要件や、それがどのように提供されているストレージと共に機能するかを理解し、アプリケーションのスケーリング時やストレージレイヤーと対話する際に問題が発生しないようにしておく必要があります。

5.3.2. 特定のアプリケーションおよびストレージの他の推奨事項

- OpenShift Container Platform Internal **etcd**: etcd の信頼性を最も高く保つには、一貫してレイテンシーが最も低くなるストレージ技術が推奨されます。
- データベース: データベース (RDBMS、NoSQL DB など) は、専用のブロックストレージで最適に機能することが予想されます。

5.4. グラフドライバーの選択

コンテナーのランタイムは、イメージとコンテナーを DeviceMapper および OverlayFS などのグラフドライバー (プラグ可能なストレージ技術) に保存します。グラフドライバーは DeviceMapper、OverlayFS、Btrfs などのプラグ可能なストレージ技術です。

サポート内容や使用方法の注意点など、OverlayFS に関する詳しい情報は、Red Hat Enterprise Linux (RHEL) 7 リリースノート を参照してください。

表5.3 グラフドライバーの比較

名前	説明	利点	制限
OverlayFS • overlay • overlay2	下層 (親) および上層 (子) のファイルシステムと作業ディレクトリー (子と同じファイルシステム) を組み合わせます。下層のファイルシステムはベースイメージで、新規コンテナーを作成すると、差分が含まれる新しい上層ファイルシステムが作成されます。	• コンテナーの起動、終了時間はデバイスマッパーよりも短いです。デバイスマッパーと Overlay の起動時間の違いは、1 秒未満です。 • ページキャッシュの共有が可能です。	POSIX に準拠しません。

名前	説明	利点	制限
デバイスマッパーのシンプロビジョニング	LVM、デバイスマッパー、dm-thinp カーネルモジュールを使用します。ループバックデバイスを削除して、ローパーティション (ファイルシステムなし) に直接移動する点が異なります。	• 中程度の負荷および高密度で、測定可能なパフォーマンスの利点があります。 • 容量においてコンテナー別の制限があります (デフォルトは 10GB)。	• 専用のパーティションが必要です。 • Red Hat Enterprise Linux (RHEL) ではデフォルト設定されていません。 • コンテナーおよびイメージはすべて、同じ容量のプールを共有します。プールを破棄または再作成せずに、リサイズはできません。
デバイスマッパー loop-lvm	デバイスマッパーのシンプロビジョニングモジュール (dm-thin-pool) を使用して、copy-on-write (CoW) スナップショットを実装します。デバイスマッパーのグラフの場所ごとに、ブロックデバイス 2 つをベースに (1 つはデータ、1 はメタデータ)、シンプールが作成されます。デフォルトでは、これらのブロックデバイスは、自動作成されたスパースファイルのループバックマウントを使用して、自動的に作成されます。	カスタマイズなしですぐに使用できるので、プロトタイプ化や開発の目的で役立ちます。	• Portable Operating System Interface for Unix (POSIX) がすべて機能する訳ではありません (例: **O_DIRECT**)。最も重要な点として、このモードは実稼働環境のワークロードには対応していません。 • コンテナーおよびイメージはすべて、同じ容量のプールを共有します。プールを破棄または再作成せずに、リサイズはできません。

パフォーマンスを強化するために、デバイスマッパーよりも overlayFS ストレージドライバー を使用することが推奨されます。実稼働環境でデバイスマッパーをすでに使用している場合、コンテナーイメージおよびコンテナールートファイルシステムにシンプロビジョニングを使用することを推奨します。そうでない場合、Docker エンジンに overlayfs2 を使用するか、または CRI-O に overlayFS を使用します。

ループデバイスを使用すると、パフォーマンスに影響する可能性があります。そのまま使用を継続できますが、以下の警告メッセージがログに記録されます。

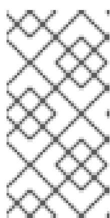
```
devmapper: Usage of loopback devices is strongly discouraged for production use.
Please use `--storage-opt dm.thinpooldev` or use `man docker` to refer to
dm.thinpooldev section.
```

ストレージの設定を容易にするには、**docker-storage-setup** ユーティリティーを使用して、設定の詳細の多くを自動化します。

Overlay の場合

1. **/etc/sysconfig/docker-storage-setup** ファイルを編集し、デバイスドライバーを指定します。

   ```
   STORAGE_DRIVER=overlay2
   ```

   注記

   CRI-O を使用している場合は、**STORAGE_DRIVER=overlay** を指定します。

   CRI-O の場合、デフォルトの **overlay** ストレージドライバーは **overlay2** の最適化を使用します。

   OverlayFS で、別の論理ボリュームに **imagefs** が必要な場合は、**CONTAINER_ROOT_LV_NAME** と **CONTAINER _ROOT_LV_MOUNT_PATH** を設定する必要があります。**CONTAINER_ROOT_LV_MOUNT_PATH** を設定するには **CONTAINER_ROOT_LV_NAME** を設定する必要があります。例: **CONTAINER_ROOT_LV_NAME="container-root-lv"**詳細は、Overlay Graph Driver の使用 を参照してください。

2. Docker ストレージ専用に別のディスクドライブがある場合 (例: **/dev/xvdb**) には、以下を **/etc/sysconfig/docker-storage-setup** ファイルに追加します。

   ```
   DEVS=/dev/xvdb
   VG=docker_vg
   ```

3. **docker-storage-setup** サービスを再起動します。

   ```
   # systemctl restart docker-storage-setup
   ```

4. docker が overlay2 を使用していることを確認して、ディスク領域の使用状況を監視するには、**docker info** コマンドを実行します。

   ```
   # docker info | egrep -i 'storage|pool|space|filesystem'
   ```

   出力例

   ```
   Storage Driver: overlay2 (1)
    Backing Filesystem: extfs
   ```

   (1) **overlay2** を使用する場合の **docker info** 出力。

Overlay は、Red Hat Enterprise Linux 7.2 以降で、コンテナーランタイムのユースケースもサポートし、起動時間の加速化、ページキャッシュ共有が可能になり、全体的なメモリー使用率を下げて高密度化できる可能性があります。

Thinpool の場合

1. **/etc/sysconfig/docker-storage-setup** ファイルを編集し、デバイスドライバーを指定します。

```
STORAGE_DRIVER=devicemapper
```

2. Docker ストレージ専用に別のディスクドライブがある場合 (例: **/dev/xvdb**) には、以下を **/etc/sysconfig/docker-storage-setup** ファイルに追加します。

```
DEVS=/dev/xvdb
VG=docker_vg
```

3. **docker-storage-setup** サービスを再起動します。

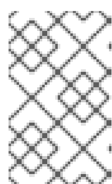
```
# systemctl restart docker-storage-setup
```

再起動後に、**docker-storage-setup** で、**docker_vg** という名前のボリュームを設定して、シンプールの論理ボリュームを作成します。RHEL でのシンプロビジョニングに関するドキュメントは、LVM 管理ガイド で確認できます。新規作成したボリュームは、**lsblk** コマンドで表示します。

```
# lsblk /dev/xvdb
```

出力例

```
NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
xvdb 202:16 0 20G 0 disk
└─xvdb1 202:17 0 10G 0 part
  ├─docker_vg-docker--pool_tmeta 253:0 0 12M 0 lvm
  │ └─docker_vg-docker--pool 253:2 0 6.9G 0 lvm
  └─docker_vg-docker--pool_tdata 253:1 0 6.9G 0 lvm
  └─docker_vg-docker--pool 253:2 0 6.9G 0 lvm
```



注記

シンプロビジョニングされたボリュームはマウントされず、ファイルシステムもないので (個別のコンテナーには XFS ファイルシステムがない)、**df** の出力には表示されません。

4. Docker が LVM シンプールを使用していることを確認して、ディスク領域の使用状況をモニターリングするには、**docker info** コマンドを使用します。

```
# docker info | egrep -i 'storage|pool|space|filesystem'
```

出力例

```
Storage Driver: devicemapper (1)
```

```
Pool Name: docker_vg-docker--pool (2)
Pool Blocksize: 524.3 kB
Backing Filesystem: xfs
Data Space Used: 62.39 MB
Data Space Total: 6.434 GB
Data Space Available: 6.372 GB
Metadata Space Used: 40.96 kB
Metadata Space Total: 16.78 MB
Metadata Space Available: 16.74 MB
```

1. **devicemapper** を使用する場合の **docker info** 出力。
2. **/etc/sysconfig/docker-storage-setup** で指定した **VG** に対応します。

デフォルトでは、シンプールは下層のブロックデバイスの 40% を使用するように設定されています。ストレージを使用していくにつれ、LVM は自動的にプールを最大 100% まで拡張します。**Data Space Total** の値が下層の LVM デバイスの古サイズに一致しないのは、この理由によります。

開発では、Red Hat ディストリビューションの Docker では ループバックをマウントしたスパースファイルにデフォルト設定されています。お使いのシステムで、ループバックモードを使用しているかどうかを確認するには、以下を実行します。

```
# docker info|grep loop0
```

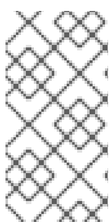
出力例

```
Data file: /dev/loop0
```

5.4.1. SELinux で OverlayFS または DeviceMapper を使用する利点

OverlayFS ファイルシステムの主な利点は、同じノードでイメージを共有するコンテナー間で、Linux ページキャッシュが共有される点です。OverlayFS のこの特性により、コンテナーの起動時の出入力 (I/O) が減り (数百ミリ秒単位でコンテナーの起動時間が短縮)、同様のイメージがノードで実行されている場合にメモリーの使用率が減少します。これらはいずれも、(ビルドファームなど) コンテナーのチャーンレートを高め、密度の最適化を目指す場合や、イメージの内容に重複が多い場合など、多くの環境で利点があります。

シンプロビジョニングのデバイスがコンテナーごとに割り当てられるので、DeviceMapper ではページキャッシュの共有はできません。



注記

OverlayFS は、Red Hat Enterprise Linux (RHEL) 7.5 のデフォルトの Docker ストレージドライバーであり、7.3 以降でサポートされています。OverlayFS を RHEL のデフォルトの Docker ストレージ設定に設定して、パフォーマンスを向上させます。Docker コンテナーランタイムで使用するために OverlayFS を設定する 手順 を参照してください。

5.4.2. Overlay と Overlay2 のグラフドライバーの比較

OverlayFS は、ユニオンファイルシステムのタイプです。これにより、あるファイルシステムを別のファイルシステムに重ねる (オーバーレイする) ことができます。上位のファイルシステムで変更が記録されても、下位のファイルシステムは変更されません。これにより、ベースイメージが読み取り専用メ

ディアにあるコンテナーや DVD-ROM などのファイルシステムイメージを、複数のユーザーが共有できるようになります。

OverlayFS は、単一の Linux ホストで 2 つのディレクトリーに階層化し、それらを 1 つのディレクトリーとして表示します。これらのディレクトリーは階層と呼ばれ、統合プロセスはユニオンプロセスと呼ばれます。

OverlayFS は、2 つのグラフドライバー **overlay** または **overlay2** のいずれかを使用します。Red Hat Enterprise Linux 7.2 の時点では、**overlay** グラフドライバーがサポートされるようになりました。Red Hat Enterprise Linux 7.4 時点で、**overlay2** がサポートされるようになりました。Docker デーモン上の SELinux は、Red Hat Enterprise Linux 7.4 でサポートされるようになりました。サポート内容やご利用のヒントなど、お使いの RHEL バージョンでの OverlayFS の使用に関する情報は、Red Hat Enterprise Linux リリースノート を参照してください。

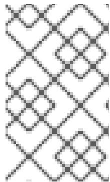
overlay2 ドライバーは、最大 128 個の 下層にある OverlayFS 階層をネイティブでサポートしますが、**overlay** ドライバーは下層の OverlayFS 階層 1 つでしか機能しません。この機能が原因で、**overlay2** ドライバーの方が、**docker build** などの階層関連の Docker コマンドのパフォーマンスが優れており、サポートするファイルシステムで使用する inode が少なくなります。

overlay ドライバーは、下層にある単一の OverlayFS 階層で機能するので、複数の OverlayFS 階層として複数階層のイメージを実装できません。代わりに、各イメージ階層は、**/var/lib/docker/overlay** の配下に独自のディレクトリーとして実装されます。下層にある階層と共有されるデータを参照する場合には、スペース効率が配慮された方法としてハードリンクが使用されます。

Docker は inode の使用において効率が良いので、overlay ドライバーではなく、OverlayFS のある **overlay2** ドライバーを使用することが**推奨されています**。

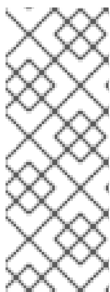
第6章 一時ストレージの最適化

6.1. 概要



注記

このトピックは、一時ストレージのテクノロジープレビューを有効にした場合にのみ適用されます。この機能はデフォルトで無効にされています。この機能を有効にするには、configuring for ephemeral storage を参照してください。



注記

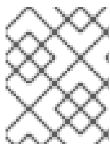
テクノロジープレビューリリースは、Red Hat 製品のサービスレベルアグリーメント (SLA) ではサポートされておらず、機能的に完全でない可能性があり、Red Hat では実稼働環境での使用を推奨しません。これらの機能は、近々発表予定の製品機能をリリースに先駆けてご提供することにより、お客様は機能性をテストし、開発プロセス中にフィードバックをお寄せいただくことができます。詳しい情報は、テクノロジープレビュー機能のサポート範囲 を参照してください。

Pod は、一時ファイルの保存などの内部操作に一時ストレージを使用します。この一時ストレージは、個別の Pod の寿命より長くなることはなく、一時ストレージは Pod 間で共有することはできません。

OpenShift Container Platform 3.10 よりも前のバージョンでは、一時ローカルストレージは、コンテナーの書き込み可能な階層、logs ディレクトリー、EmptyDir ボリュームを使用して Pod に公開されていました。ローカルストレージのアカウントや分離がないことに関連する問題には、以下が含まれます。

- Pod は利用可能なローカルストレージのサイズを認識しない。
- Pod がローカルストレージを要求しても確実に割り当てられない可能性がある。
- ローカルストレージはベストエフォートのリソースである。
- Pod は、他の Pod でローカルストレージがいっぱいになるとエビクトされる可能性があり、十分なストレージが回収されるまで、新しい Pod は入れない。

一時ストレージは同様に Pod に公開されていますが、Pod の一時ストレージの使用に対する要求や制限を実装する新たな方法が追加されています。



注記

CRI-O をコンテナーランタイムとして使用し、ファイルベースロギングをロギングに使用している場合にのみ、コンテナーログの管理は該当します。

一時ストレージはシステム内の Pod すべてで共有され、OpenShift Container Platform には管理者およびユーザーが確立した要求や制限を超えるサービスを保証する仕組みはない点を理解することが重要です。たとえば、一時ストレージでは、スループット、秒ごとの I/O 操作またはストレージパフォーマンスについての保証は一切ありません。

6.2. 一般的なストレージガイドライン

ノードのローカルストレージは、プライマリーパーティションとセカンダリーパーティションに分割できます。一時ローカルストレージには、プライマリーパーティションのみが使用できます。プライマリーパーティションでは、root とランタイムの 2 つがサポートされています。

- Root
  Root パーティションは、デフォルトで kubelet の root ディレクトリー /**var**/**lib**/**kubelet**/ と /**var**/**log**/ ディレクトリーを保持します。このパーティションを Pod、オペレーティングシステム、OpenShift Container Platform システムデーモンの間で共有できます。Pod は、EmptyDir ボリューム、コンテナーログ、イメージ階層、コンテナーの書き込み可能階層を使用してこのパーティションにアクセスできます。OpenShift Container Platform は、このパーティションの共有アクセスと分離を管理します。

- ランタイム
  ランタイムパーティションは、オーバーレイファイルシステムに使用可能なオプションのパーティションです。OpenShift Container Platform は、このパーティションの分離および共有アクセスを特定して提供します。このパーティションには、コンテナーイメージ階層と書き込み可能な階層が含まれます。ランタイムパーティションが存在する場合は、**root** パーティションにはイメージ階層も書き込み可能階層も含まれません。

第7章 ネットワークの最適化

7.1. ネットワークパフォーマンスの最適化

OpenShift SDN は OpenvSwitch、VXLAN (Virtual extensible LAN) トンネル、OpenFlow ルール、iptables を使用します。このネットワークは、ジャンボフレーム、ネットワークインターフェイスカード (NIC) のオフロード、マルチキュー、ethtool の設定を使用してチューニングが可能です。

VXLAN は、4096 から 1600 万以上にネットワーク数が増え、物理ネットワーク全体で階層 2 の接続が追加されるなど、VLAN での利点が提供されます。これにより、異なるシステム上で実行されている場合でも、サービスの背後にある Pod すべてが相互に通信できるようになります。

VXLAN は、User Datagram Protocol (UDP) パケットにトンネル化されたトラフィックをすべてカプセル化しますが、CPU 使用率が上昇してしまいます。これらの外部および内部パケットは、移動中にデータが破損しないようにするために通常のチェックサムルールの対象になります。これらの外部および内部パケットはどちらも、移動中にデータが破損しないように通常のチェックサムルールの対象になります。CPU のパフォーマンスによっては、この追加の処理オーバーヘッドによってスループットが減り、従来の非オーバーレイネットワークと比較してレイテンシーが高くなります。

クラウド、仮想マシン、ベアメタルの CPU パフォーマンスでは、1 Gbps をはるかに超えるネットワークスループットを処理できます。10 または 40 Gbps などの高い帯域幅のリンクを使用する場合には、パフォーマンスが低減する場合があります。これは、VXLAN ベースの環境では既知の問題で、コンテナーや OpenShift Container Platform 固有の問題ではありません。VXLAN トンネルに依存するネットワークも、VXLAN 実装により同様のパフォーマンスになります。

1 Gbps 以上にするには、以下を実行してください。

- ネイティブのコンテナールーティングの設定 を使用する。このオプションには、ルーターでのルーティングテーブルの更新など、OpenShift SDN の使用時には存在しない重要な操作上の注意事項があります。
- Border Gateway Protocol (BGP) など、異なるルーティング技術を実装するネットワークプラグインを評価する。
- VXLAN オフロード対応のネットワークアダプターを使用します。VXLAN オフロードは、システムの CPU から、パケットのチェックサム計算と関連の CPU オーバーヘッドを、ネットワークアダプター上の専用のハードウェアに移動します。これにより、CPU サイクルを Pod やアプリケーションで使用できるように開放し、ネットワークインフラストラクチャーの帯域幅すべてをユーザーは活用できるようになります。

VXLAN オフロードはレイテンシーを短縮しません。ただし、CPU の使用率はレイテンシーテストでも削減されます。

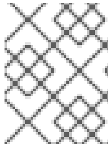
7.1.1. ネットワークでの MTU の最適化

重要な Maximum Transmission Unit (MTU) が 2 つあります (ネットワークインターフェイスカード (NIC) MTU と、SDN オーバーレイ MTU です)。

NIC MTU は、お使いのネットワークでサポートされる最大値以下でなければなりません。スループットを最適化する場合は、最大の値を選択するようにしてください。最も低いレイテンシーにおいて最適化するには、小さい値を選択します。

SDN オーバーレイの MTU は、最低でも NIC MTU より 50 バイト少なくなければなりません。これは、SDN オーバーレイのヘッダーに相当します。そのため、通常の Ethernet ネットワークでは、この値を 1450 に設定してください。ジャンボフレームの Ethernet ネットワークの場合は、これを 8950 に

設定してください。



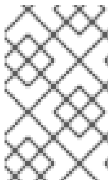
注記

50 バイトのオーバーレイヘッダーは OpenShift SDN に関連します。他の SDN ソリューションの場合はこの値を若干変動させる必要があります。

MTU を設定するには、適切な ノード設定マップ を編集して、以下のセクションを変更します。

```
networkConfig:
  mtu: 1450 (1)
  networkPluginName: "redhat/openshift-ovs-subnet" (2)
```

1. Pod オーバーレイネットワークの Maximum transmission unit (MTU)
2. ovs-subnet プラグインの場合は **redhat/openshift-ovs-subnet** に、ovs-multitenant プラグインの場合は **redhat/openshift-ovs-multitenant** に、ovs-networkpolicy プラグインの場合は **redhat/openshift-ovs-networkpolicy** に設定します。これは、他の CNI 互換のプラグインにも設定できます。



注記

OpenShift Container Platform SDN を設定するすべてのマスターおよび ノードで MTU サイズを変更する必要があります。また、tun0 インターフェイスの MTU サイズはクラスターを設定するすべてのノードで同一である必要があります。

7.2. ネットワークサブネットの設定

OpenShift Container Platform は、Pod とサービスに対して IP アドレス管理を提供します。デフォルト値の許容範囲は以下のとおりです。

- 最大のクラスターサイズは 1024 ノード
- 1024 ノードごとに、/23 を割り当てる (Pod で利用可能な IP は 510 個)
- サービス用の IP アドレスは約 65,536 個

多くの状況下で、デプロイメント後にはこれらのネットワークの変更はできません。そのため、拡張関連のプランニングをすることが重要です。

ネットワークのリサイズに関する制限は、SDN の設定ドキュメント に記載されています。

より大規模な環境を計画する場合は、Ansible インベントリーファイルの **[OSE3:vars]** セクションに以下の推奨値を設定するようにしてください。

```
[OSE3:vars]
osm_cluster_network_cidr=10.128.0.0/10
```

これにより、使用可能な IP アドレス 510 個が割り当てられる 8192 ノードが許可されます。

インストールしているソフトウェアバージョンのノード/Pod の上限については、OpenShift Container Platform ドキュメントにあるサポートの制限について参照してください。

7.3. IPSEC の最適化

ノードホストの暗号化、復号化に CPU 機能が使用されるので、使用する IP セキュリティーシステムにかかわらず、ノードのスループットおよび CPU 使用率の両方でのパフォーマンスに影響があります。

IPSec は、NIC に到達する前に IP ペイロードレベルでトラフィックを暗号化して、NIC オフロードに使用されてしまう可能性のあるフィールドを保護します。つまり、IPSec が有効な場合には、NIC アクセラレーション機能を使用できない場合があり、スループットの減少、CPU 使用率の上昇につながります。

第8章 ルーティングの最適化

8.1. OPENSHIFT CONTAINER PLATFORM HAPROXY ルーターのスケーリング

8.1.1. ベースラインのパフォーマンス

OpenShift Container Platform ルーター は、宛先が OpenShift Container Platform サービスのすべての外部トラフィックに対する Ingress ポイントです。

1 秒に処理される HTTP 要求について、単一の HAProxy ルーターを評価する場合に、パフォーマンスは多くの要因により左右されます。特に以下が含まれます。

- HTTP keep-alive/close モード
- ルートタイプ
- TLS セッション再開のクライアントサポート
- ターゲットルートごとの同時接続数
- ターゲットルート数
- バックエンドサーバーのページサイズ
- 基礎となるインフラストラクチャー (ネットワーク/SDN ソリューション、CPU など)

個別の環境でのパフォーマンスは異なりますが、ラボは、サイズが 4 vCPU/16GB RAM のパブリッククラウドインスタンスでテストします。 ルート 100 個を処理し、1kB 静的ページに対応するバックエンドで終端される 100 ルートを処理する単一の HAProxy ルーターは、1 秒ごとに以下の数のトランザクションを処理できます。

HTTP keep-alive モードのシナリオの場合:

暗号化	ROUTER_THREADS 設定なし	ROUTER_THREADS=4
なし	23681	24327
edge	14981	22768
passthrough	34358	34331
re-encrypt	13288	24605

HTTP close (keep-alive なし) のシナリオの場合:

暗号化	ROUTER_THREADS 設定なし	ROUTER_THREADS=4
なし	3245	4527

暗号化	ROUTER_THREADS 設定なし	ROUTER_THREADS=4
edge	1910	3043
passthrough	3408	3922
re-encrypt	1333	2239

TLS セッション再開は暗号化ルートについて使用されています。HTTP keep-alive の場合は、単一の HAProxy ルーターがページサイズが 8kB でも、1 Gbit の NIC を飽和させることができます。

最新のプロセッサーが搭載されたベアメタルで実行する場合は、上記のパブリッククラウドインスタンスのパフォーマンスの約 2 倍のパフォーマンスになることを予想できます。このオーバーヘッドは、パブリッククラウドにある仮想化層により発生し、プライベートクラウドベースの仮想化にも多くの場合、該当します。以下の表は、ルーターの背後で使用するアプリケーション数についてのガイドです。

アプリケーション数	アプリケーションタイプ
5-10	静的なファイル/Web サーバーまたはキャッシュプロキシー
100-1000	動的なコンテンツを生成するアプリケーション

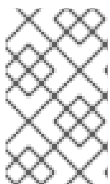
通常、HAProxy は、使用されるて技術に応じて 5 から 1000 のアプリケーションのルーターをサポートします。ルーターのパフォーマンスは、言語や静的コンテンツと動的コンテンツの違いを含め、その背後にあるアプリケーションの機能およびパフォーマンスによって制限される可能性があります。

アプリケーションに対してより多くのルートを提供し、ルーティング層の水平スケーリングを図る場合には、ルーターのシャード を使用する必要があります。

8.1.2. パフォーマンスの最適化

8.1.2.1. 最大接続数の設定

HAProxy のスケーラビリティーで最も重要でチューニング可能なパラメーターの 1 つに、**maxconn** パラメーターがあります。このパラメーターは、プロセス別の最大同時接続数を特定の値に設定します。このパラメーターを調節するには、OpenShift Container Platform HAProxy ルーターのデプロイメント設定ファイルにある **ROUTER_MAX_CONNECTIONS** 環境変数を編集してください。



注記

接続にはフロントエンドおよび内部バックエンドが含まれます。これは 2 つの接続としてカウントされます。必ず **ROUTER_MAX_CONNECTIONS** の値を作成しようとしている接続数の 2 倍以上になるように設定してください。

8.1.2.2. CPU および割り込みアフィニティー

OpenShift Container Platform では、HAProxy ルーターは単一のプロセスのとして実行されます。OpenShift Container Platform HAProxy ルーターは通常、周波数が低く、数の多いコアを持つ対称型マルチプロセッシング (SMP) よりも、周波数が高く、数の少ないコアが搭載されたシステムでより優れたパフォーマンスを実現します。

HAProxy プロセスを 1 つの CPU コアに、また別の CPU コアにネットワーク割り込みをピニングすると、ネットワークパフォーマンスが向上する傾向にあります。同じ Non-Uniform Memory Access (NUMA) ノードにプロセスと割り込みを配置すると、共有 L3 キャッシュを確保してメモリーアクセスを回避しやすくなります。ただし、このレベルの制御は、パブリッククラウド環境では一般的に不可能です。ベアメタルホストでは、**irqbalance** は、割り込み要求線 (IRQ) があれば、自動的に PCI (peripheral component interconnect) ローカリティーと NUMA アフィニティーを処理します。クラウド環境では、このレベルの情報は一般的にオペレーティングシステムには提供されません。

CPU ピニングは **taskset** または HAProxy の **cpu-map** パラメーターを使用して実行されます。このディレクティブは、プロセス ID と CPU コア ID の 2 つの引数を取ります。たとえば、HAProxy プロセス **1** を CPU コア **0** にピニングするには、以下の行を HAProxy の設定ファイルの Global セクションに追加します。

```
cpu-map 1 0
```

HAProxy 設定ファイルの変更については、カスタマイズされた HAProxy ルーターのデプロイ を参照してください。

8.1.2.3. スレッド数の増加

HAProxy ルーターは、OpenShift Container Platform のマルチスレッドのサポートと共に提供されます。複数の CPU コアシステムでは、スレッド数が増加すると、ルーター上で SSL を終了する場合などにパフォーマンスが増大する場合があります。

HAProxy ルーターのスレッド数を指定するには、Enable HAProxy Threading および Router Environment Variables を参照してください。

8.1.2.4. バッファー増加の影響

OpenShift Container Platform HAProxy ルーター要求のバッファー設定で、アプリケーションからの受信要求や応答のヘッダーサイズを制限します。HAProxy パラメーター **tune.bufsize** を増やして、より大きいヘッダーを処理し、多くのパブリッククラウドプロバイダーが提供するロードバランサーで許可されるアプリケーションなど、非常に大きい cookie を使用するアプリケーションを機能させることができます。ただし、これにより、多数の接続が開放されている場合など、合計のメモリー使用率に影響があります。非常に多くの接続が開かれている場合には、メモリー使用率は、このチューニング可能なパラメーターの増加とほぼ比例します。

8.1.2.5. HAProxy 再読み込みの最適化

Websocket 接続などの長時間続く接続が、長いクライアント/サーバー HAProxy タイムアウトと短い HAProxy 再読み込み間隔が組み合わされると、HAProxy プロセスが多数インスタンス化されてしまう可能性があります。これらのプロセスは、HAProxy 設定が再読込みされる前に開始されていた以前の接続を処理する必要があります。これらの数多くのプロセスは、システムに不必要な負荷がかかり、メモリー不足の状態などの問題につながる可能性があるために理想的とは言えません。

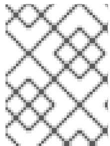
この動作に影響を与えるルーターの環境変数は、とくに **ROUTER_DEFAULT_TUNNEL_TIMEOUT**、**ROUTER_DEFAULT_CLIENT_TIMEOUT**、**ROUTER_DEFAULT_SERVER_TIMEOUT** および **RELOAD_INTERVAL** などです。

第9章 クラスターメトリクスのスケーリング

9.1. 概要

OpenShift Container Platform は、Heapster で収集してバックエンドに保存可能なメトリクスを公開します。OpenShift Container Platform の管理者は、1 つのユーザーインターフェイスでコンテナーやコンポーネントメトリクスを表示できます。これらのメトリクスは、Horizontal Pod Autoscaler によるスケーリングのタイミングと方法の決定にも使用されます。

以下のトピックでは、メトリクスコンポーネントのスケーリングに関する情報に触れます。



注記

Hawkular および Heapster などのメトリクスコンポーネントの自動スケーリングは OpenShift Container Platform ではサポートされていません。

9.2. OPENSHIFT CONTAINER PLATFORM についての推奨事項

- 専用の OpenShift Container Platform インフラストラクチャーノード でメトリクス Pod を実行する
- メトリクスの設定時は永続ストレージを使用する。**USE_PERSISTENT_STORAGE=true** を設定します。
- OpenShift Container Platform メトリクスデプロイメントで **METRICS_RESOLUTION=30** パラメーターを保持する。**METRICS_RESOLUTION** をデフォルト値の **30** よりも小さい値に設定することは推奨していません。Ansible メトリクスのインストール手順を使用する場合は、このパラメーターは **openshift_metrics_resolution** に置き換えてください。
- ホストメトリクス Pod が指定された OpenShift Container Platform ノードを詳しくモニターリングして、ホストシステムの容量不足 (CPU およびメモリー) を早い段階で検出する。このような容量不足により、メトリクス Pod で問題が発生する可能性があります。
- OpenShift Container Platform バージョン 3.7 のテストでは、最大 Pod 数 25,000 のテストケースが OpenShift Container Platform クラスターでモニターリングされました。

9.3. クラスターメトリクスのキャパシティーピニング

OpenShift Container Platform ノードが 210 台と 990 台で実行したテストでは、それぞれ Pod が 10500 台と 11000 台をモニターリングし、Cassandra データベースのサイズが、以下の表に記載の速度で増加しました。

表9.1 クラスター内のノード/Pod の数に基づく Cassandra データベースのストレージ要件

ノード数	Pod 数	Cassandra ストレージの増加速度	1 日あたりの Cassandra ストレージの増加量	1 週間あたりの Cassandra ストレージの増加量
210	10500	1 時間あたり 500 MB	15 GB	75 GB
990	11000	1 時間に 1 GB	30 GB	210 GB

上記の計算では、ストレージ要件が計算値を超過しないようにするためのオーバーヘッドとして、予期されたサイズのおよそ 20% が追加されています。

METRICS_DURATION および **METRICS_RESOLUTION** の値がデフォルト (それぞれ **7** 日と **15** 秒) のままの場合は、上記の値にあるように、安全策として週ごとの Cassandra ストレージのサイズ要件を計画することができます。

警告

OpenShift Container Platform メトリクスはメトリクスデータのデータストアとして Cassandra データベースを使用するので、メトリクス設定のプロセスで **USE_PERSISTENT_STORAGE=true** が設定される場合には、NFS がデフォルトとして設定された状態で **PV** がネットワークストレージの上部に配置されます。ただし、ネットワークストレージと Cassandra を組み合わせて使用することは推奨していません。

Cassandra データベースをメトリクスデータのデータストアとして使用する場合、推奨事項については、Cassandra ドキュメント を参照してください。

9.4. OPENSHIFT CONTAINER PLATFORM メトリクス POD のスケーリング

メトリクス Pod (Cassandra/Hawkular/Heapster) 1 セットでは、最低 25,000 の Pod をモニターリングできます。

注意

OpenShift Container Platform メトリクス Pod が実行されるノードのシステムの負荷に注意してください。この情報を使用して、OpenShift Container Platform メトリクス Pod の数をスケールアウトし、複数の OpenShift Container Platform ノードに負荷を分散する必要があるかどうかを判断します。OpenShift Container Platform メトリクス heapster Pod のスケーリングは推奨していません。

9.4.1. 前提条件

OpenShift Container Platform メトリクスのデプロイに永続ストレージを使用した場合には、OpenShift Container Platform メトリクスの Cassandra Pod 数をスケーリングする前に、新規 Cassandra Pod が使用されるように、永続ボリューム (PV) を作成する 必要があります。ただし、動的にプロビジョニングされる PV を使用して Cassandra がデプロイされた場合には、この手順は必要ありません。

9.4.2. Cassandra コンポーネントのスケーリング

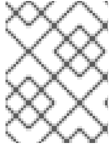
Cassandra ノードは永続ストレージを使用します。そのため、レプリケーションコントローラーでスケールアップまたはスケールダウンを実行できません。

Cassandra クラスターのスケーリングには、**openshift_metrics_cassandra_replicas** 変数を変更して、デプロイメント を再実行する必要があります。デフォルトでは Cassandra クラスターは単一ノードのクラスターとなっています。

OpenShift Container Platform メトリクスの hawkular Pod を 2 つのレプリカにスケールアップするには、以下を実行します。

```
# oc scale -n openshift-infra --replicas=2 rc hawkular-metrics
```

または、インベントリーファイルを更新して、デプロイメント を再実行します。



注記

Cassandra クラスターに対して、新規ノードを追加したり、既存のノードを削除した場合は、クラスターに保存したデータの負荷がクラスター全体で再度分散されます。

スケールダウンの方法:

1. コンテナーにリモートからアクセスする場合は、削除する Cassandra ノードに対して以下を実行します。

   ```
   $ oc exec -it <hawkular-cassandra-pod> nodetool decommission
   ```

   コンテナーにローカルでアクセスする場合には、代わりに以下を実行します。

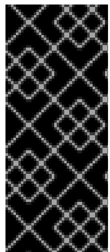
   ```
   $ oc rsh <hawkular-cassandra-pod> nodetool decommission
   ```

   このコマンドは、クラスター全体のデータをコピーするので、実行にしばらく時間がかかります。停止の進捗状況は **nodetool netstats -H** でモニターリングできます。

2. 先のコマンドに成功すると、Cassandra インスタンスの **rc** を **0** にスケールダウンします。

   ```
   # oc scale -n openshift-infra --replicas=0 rc <hawkular-cassandra-rc>
   ```

   これで Cassandra Pod が削除されます。



重要

スケールダウンプロセスが完了し、既存の Cassandra ノードが予想どおりに機能する場合には、この Cassandra インスタンスと対応する Persistent Volume Claim (PVC、永続ボリューム要求) の **rc** も削除できます。PVC を削除すると、この Cassandra インスタンスに関連付けられているデータが完全に削除されるので、スケールダウンが完全かつ正常に完了しなかった場合に、失われたデータを復元することはできません。

第10章 CLUSTER MONITORING OPERATOR のスケーリング

10.1. 概要

OpenShift Container Platform は、cluster-monitoring-operator によってバックエンドで収集し、保存できるメトリクスを公開します。OpenShift Container Platform 管理者は、Grafana という 1 つのダッシュボードインターフェイスでシステムリソース、コンテナーおよびコンポーネントのメトリクスを表示できます。

以下のトピックでは、cluster monitoring operator のスケーリングについての情報を提供します。

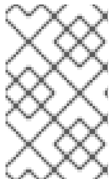
Prometheus を永続ストレージで使用する場合は、Ansible インベントリーファイルの **openshift_cluster_monitoring_operator_prometheus_storage_enabled** 変数を **true** に設定する必要があります。

10.2. OPENSHIFT CONTAINER PLATFORM についての推奨事項

- 3 つ以上の インフラストラクチャー (infra) ノードを使用します。
- NVMe (non-volatile memory express) ドライブを搭載した 3 つ以上の **openshift-container-storage** ノードを使用します。
- OpenShift Container Storage (OCS) ブロック などの永続ブロックストレージを使用します。

10.3. CLUSTER MONITORING OPERATOR の容量計画

異なるスケールサイズについて各種のテストが実行されました。以下の表が示すように、Prometheus データベースが拡張されました。



注記

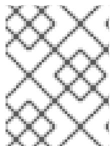
以下の Prometheus ストレージ要件は規定されていません。ワークロードのアクティビティーおよびリソースの使用に応じて、クラスターで観察されるリソースの消費量が大きくなる可能性があります。

表10.1 クラスター内のノード/Pod の数に基づく Prometheus データベースのストレージ要件

ノード数	Pod 数	1 日あたりの Prometheus ストレージの増加量	15 日ごとの Prometheus ストレージの増加量	RAM 領域 (スケールサイズに基づく)	ネットワーク (tsdb チャンクに基づく)
50	1800	6.3 GB	94 GB	6 GB	16 MB
100	3600	13 GB	195 GB	10 GB	26 MB
150	5400	19 GB	283 GB	12 GB	36 MB
200	7200	25 GB	375 GB	14 GB	46 MB

上記の計算では、ストレージ要件が計算値を超過しないようにするために、オーバーヘッドとして予期されたサイズのおよそ 20% が追加されています。

上記の計算は、デフォルトの OpenShift Container Platform **cluster-monitoring-operator** について開発されました。スケールが拡張する場合、Ansible インベントリーファイルで、デフォルトが **50Gi** に設定されている **openshift_cluster_monitoring_operator_prometheus_storage_capacity=94Gi** 変数を編集します。



注記

CPU の使用率による影響は大きくありません。比率については、およそ 50 ノードおよび 1800 Pod ごとに 1 コア (/40) になります。

10.3.1. ラボ環境

すべての実験は、OpenStack 環境の OpenShift Container Platform で実行されました。

- インフラストラクチャーノード (VM) - 40 コア、157 GB RAM。
- CNS ノード (VM) - 16 コア、62 GB RAM、NVMe ドライブ。

10.3.2. 前提条件

スケーリングに基づいて、Prometheus データストアの関連する PV サイズを計算し、設定します。デフォルトの Prometheus Pod レプリカ数は 2 であるため、3600 Pod を含む 100 ノードの場合、188 GB が必要になります。

例を以下に示します。

```
195 GB (space per 15 days ) * 2 (pods) = 390 GB free
```

この方程式に基づいて、**openshift_cluster_monitoring_operator_prometheus_storage_capacity=195Gi** を設定します。

第11章 クラスターごとのテスト済み最大数

OpenShift Container Platform クラスターの計画時に以下のテスト済みのクラスターオブジェクトの最大値を考慮します。

これらのガイドラインは、最大規模のクラスターに基づいています。クラスターの規模が小さい場合には、最大値も比例して少なくなります。指定のしきい値に影響を与える要因には、etcd バージョンやストレージデータ形式などの多数の要因があります。

ほとんど場合、これらの制限値を超えると、パフォーマンスが全体的に低下します。ただし、これによって必ずしもクラスターに障害が発生する訳ではありません。

OpenShift Container Platform 3.x のテスト済みクラウドプラットフォーム: Red Hat OpenStack、Amazon Web Services および Microsoft Azure

11.1. メジャーリリースについての OPENSHIFT CONTAINER PLATFORM のテスト済みクラスターの最大値

最大値のタイプ	3.x テスト済みの最大値
ノード数	2,000
Pod 数 [1]	150,000
ノードごとの Pod 数	250
コアごとの Pod 数	デフォルト値はありません。
namespace 数	10,000
ビルド数: パイプラインストラテジー	10,000 (デフォルトの Pod: メモリー 512Mi)
namespace ごとの Pod 数 [2]	25,000
サービス数 [3]	10,000
namespace ごとのサービス数	5,000
サービスごとのバックエンド数	5,000
namespace ごとのデプロイメント数 [2]	2,000

1. ここで表示される Pod 数はテスト Pod の数です。実際の Pod 数は、アプリケーションのメモリー、CPU、ストレージ要件により異なります。
2. システムには、状態の変更に対する対応として特定の namespace にある全オブジェクトに対して反復する多数のコントロールループがあります。単一の namespace に特定タイプのオブジェクトの数が多くなると、ループのコストが上昇し、特定の状態変更を処理する速度が低下しま

す。この最大値については、アプリケーションの各種要件を満たすのに十分な CPU、メモリー、およびディスクがシステムにあることが前提となっています。

3. 各サービスポートと各サービスのバックエンドには、iptables の対応するエントリーがあります。特定のサービスのバックエンド数は、エンドポイントのオブジェクトサイズに影響があり、その結果、システム全体に送信されるデータサイズにも影響を与えます。

11.2. OPENSHIFT CONTAINER PLATFORM のテスト済みのクラスターの最大値

最大値のタイプ	3.7 テスト済みの最大値	3.9 テスト済みの最大値	3.10 テスト済みの最大値	3.11 テスト済みの最大値
ノード数	2,000	2,000	2,000	2,000
Pod 数 [1]	120,000	120,000	150,000	150,000
ノードごとの Pod 数	250	250	250	250
コアごとの Pod 数	デフォルト値は 10 です。	デフォルト値は 10 です。	デフォルト値はありません。	デフォルト値はありません。
namespace 数	10,000	10,000	10,000	10,000
ビルド数: パイプラインストラテジー	該当なし	10,000 (デフォルトの Pod: メモリー 512Mi)	10,000 (デフォルトの Pod: メモリー 512Mi)	10,000 (デフォルトの Pod: メモリー 512Mi)
namespace ごとの Pod 数 [2]	3,000	3,000	3,000	25,000
サービス数 [3]	10,000	10,000	10,000	10,000
namespace ごとのサービス数	該当なし	該当なし	5,000	5,000
サービスごとのバックエンド数	5,000	5,000	5,000	5,000
namespace ごとのデプロイメント数 [2]	2,000	2,000	2,000	2,000

1. ここで表示される Pod 数はテスト Pod の数です。実際の Pod 数は、アプリケーションのメモリー、CPU、ストレージ要件により異なります。

2. システムには、状態の変更に対する対応として特定の namespace にある全オブジェクトに対して反復する多数のコントロールループがあります。単一の namespace に特定タイプのオブジェ

クトの数が多くなると、ループのコストが上昇し、特定の状態変更を処理する速度が低下します。この最大値については、アプリケーションの各種要件を満たすのに十分な CPU、メモリー、およびディスクがシステムにあることが前提となっています。

3. 各サービスポートと各サービスのバックエンドには、iptables の対応するエントリーがあります。特定のサービスのバックエンド数は、エンドポイントのオブジェクトサイズに影響があり、その結果、システム全体に送信されるデータサイズにも影響を与えます。

11.2.1. ルート最大値

OpenShift Container Platform 3.11.53 では、ルーターテストは Amazon Web Services (AWS) の 3 ノード環境で実行されました。**keepalive** が **100** に設定されている 100 HTTP ルート (具体的には 100 バックエンド Nginx Pod) がありました。結果は以下のようになります。

- ターゲットルートごとに 1 接続= 1 秒あたり 24,327 要求
- ターゲットルートごとに 40 接続= 1 秒あたり 20,729 要求
- ターゲットルートごとに 200 接続= 1 秒あたり 17,253 要求

11.3. OPENSHIFT CONTAINER PLATFORM クラスターの最大値をテストする環境および設定

サービスプロバイダーとしてのインフラストラクチャー: OpenStack

ノード	vCPU	RAM (MiB)	ディスクサイズ (GiB)	パススルーディスク	カウント
マスター/Etcd [1]	16	124672	128	○、NVMe	3
Infra [2]	40	163584	256	○、NVMe	3
クラスター DNS	1	1740	71	×	1
ロードバランサー	4	16128	96	×	1
Container Native Storage [3]	16	65280	200	○、NVMe	3
bastion [4]	16	65280	200	×	1
ワーカー	2	7936	96	×	2000

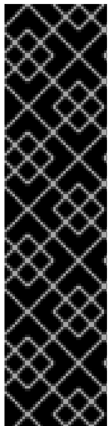
1. マスター/etcd ノードは、etcd は I/O 集約型でレイテンシーの影響を受けるため、NVMe ディスクでサポートされます。

2. インフラストラクチャーノードはルーター、レジストリー、ロギング、およびモニターリングをホストし、NVMe ディスクでサポートされます。

3. Container Native Storage または Ceph ストレージノードは NVMe ディスクでサポートされます。

4. Bastion ノードは OpenShift Container Platform ネットワークの一部であり、パフォーマンスおよびスケーリングテストのオーケストレーションに使用されます。

11.4. クラスターの最大値に合わせた環境計画



重要

ノード上で物理リソースを過剰にサブスクライブすると、Kubernetes スケジューラーが Pod の配置時に行うリソースの保証に影響が及びます。メモリースワップを防ぐ ために実行できる処置について確認してください。

テスト済みの最大値では、一方向にのみスケーリングされているので、多数のオブジェクトがクラスターで実行されている場合には、異なる可能性があります。

本書に記載されている数は、Red Hat のテスト方法、セットアップ、設定、およびチューニングに基づいています。これらの数は、独自のセットアップおよび環境に応じて異なります。

環境の計画時 に、ノード配置できる Pod 数を判断します。

```
Maximum Pods per Cluster / Expected Pods per Node = Total Number of Nodes
```

ノードで適合する Pod 数は、アプリケーション自体により異なります。アプリケーションのメモリー、CPU、ストレージ要件を検討してください。

シナリオ例

クラスターごとに 2200 の Pod のあるクラスターのスコープを設定する場合、ノードごとに最大 250 の Pod があることを前提として、最低でも 9 つのノードが必要になります。

```
2200 / 250 = 8.8
```

ノード数を 20 に増やす場合は、Pod 配分がノードごとに 110 の Pod に変わります。

```
2200 / 20 = 110
```

11.5. アプリケーション要件に合わせた環境計画

アプリケーション環境の例を考えてみましょう。

Pod タイプ	Pod 数	最大メモリー	CPU コア	永続ストレージ
apache	100	500MB	0.5	1GB
node.js	200	1GB	1	1GB

Pod タイプ	Pod 数	最大メモリー	CPU コア	永続ストレージ
postgresql	100	1GB	2	10GB
JBoss EAP	100	1GB	1	1GB

推定要件: CPU コア 550 個、メモリー 450GB およびストレージ 1.4TB

ノードのインスタンスサイズは、希望に応じて増減を調整できます。ノードのリソースはオーバーコミットされることが多く、デプロイメントシナリオでは、小さいノードで数を増やしたり、大きいノードで数を減らしたりして、同じリソース量を提供することもできます。このデプロイメントシナリオでは、小さいノードで数を増やしたり、大きいノードで数を減らしたりして、同じリソース量を提供することもできます。運用上の敏捷性やインスタンスごとのコストなどの要因を考慮する必要があります。

ノードタイプ	数量	CPU	RAM (GB)
ノード (オプション 1)	100	4	16
ノード (オプション 2)	50	8	32
ノード (オプション 3)	25	16	64

アプリケーションによっては オーバーコミット の環境に適しているものもあれば、そうでないものもあります。たとえば、Java アプリケーションや、大きいページを使用するアプリケーションの多くは、オーバーコミットに対応できません。対象のメモリーは、他のアプリケーションに使用できません。上記の例では、環境は一般的な比率として約 30 % オーバーコミットされています。

第12章 クラスターローダーの使用

12.1. クラスターローダーの機能

クラスターローダーとは、クラスターに対してさまざまなオブジェクトを多数デプロイするツールであり、ユーザー定義のクラスターオブジェクトを作成します。クラスターローダーをビルド、設定、実行して、さまざまなクラスターの状態にある OpenShift Container Platform デプロイメントのパフォーマンスメトリクスを測定します。

12.2. クラスターローダーのインストール

クラスターローダーは **atomic-openshift-tests** パッケージに含まれます。これをインストールするには、以下を実行します。

```
$ yum install atomic-openshift-tests
```

インストールが終わると、テスト用の実行ファイル **extended.test** は **/usr/libexec/atomic-openshift/extended.test** に配置されます。

12.3. クラスターローダーの実行

1. **KUBECONFIG** 変数は、管理者 **kubeconfig** の場所に設定します。

```
$ export KUBECONFIG=${KUBECONFIG-$HOME/.kube/config}
```

2. 組み込まれているテスト設定を使用してクラスターローダーを実行し、5 つのテンプレートビルドをデプロイして、デプロイメントが完了するまで待ちます。

```
$ cd /usr/libexec/atomic-openshift/
```

```
$ ./extended.test --ginkgo.focus="Load cluster"
```

または **--viper-config** のフラグを追加して、ユーザー定義の設定でクラスターローダーを実行します。

```
$ ./extended.test --ginkgo.focus="Load cluster" --viper-config=config/test
```
(1)

(1) この例では、**config/** というサブディレクトリーに **test.yml** という設定ファイルが配置されています。コマンドラインで、ファイルタイプと拡張子はツールが自動的に判断するので、設定ファイルを拡張子なしで実行します。

12.4. クラスターローダーの設定

複数のテンプレートや Pod を含む、namespaces (プロジェクト) を複数作成します。

クラスターローダーの設定ファイルを **config/** サブディレクトリーに配置します。これらの設定ファイルで参照される Pod ファイルとテンプレートファイルは、**content/** サブディレクトリーにあります。

12.4.1. 設定フィールド

表12.1 クラスターローダーの最上位のフィールド

フィールド	説明
cleanup	**true** または **false** に設定します。設定ごとに 1 つの定義を設定します。**true** に設定すると、**cleanup** は、テストの最後にクラスターローダーが作成した namespaces (プロジェクト) すべてを削除します。
プロジェクト	1 つまたは多数の定義が指定されたサブオブジェクト。**projects** の下に、作成する各 namespace が定義され、**projects** には必須のサブヘッダーが複数指定されます。
tuningsets	設定ごとに 1 つの定義が指定されたサブオブジェクト。**tuningsets** では、チューニングセットを定義して、プロジェクトやオブジェクト作成に対して設定可能なタイミングを追加することができます (Pod、テンプレートなど)。
sync	設定ごとに 1 つの定義が指定されたオプションのサブオブジェクト。オブジェクト作成時に同期できるかどうかについて追加します。

表12.2 projects の下にあるフィールド

フィールド	説明
num	整数。作成するプロジェクト数の 1 つの定義。
basename	文字列。プロジェクトのベース名の定義。競合が発生しないように、同一の namespace の数が **Basename** に追加されます。
tuning	文字列。オブジェクトに適用するチューニングセットの 1 つの定義。 これは対象の namespace にデプロイします。
ifexists	**reuse** または **delete** のいずれかが含まれる文字列。ツールが実行時に作成するプロジェクトまたは namespace の名前と同じプロジェクトまたは namespace を見つける場合のツールの機能を定義します。
configmaps	キーと値のペア一覧。キーは ConfigMap の名前で、値はこの ConfigMap の作成元のファイルへのパスです。
secrets	キーと値のペア一覧。キーはシークレットの名前で、値はこのシークレットの作成元のファイルへのパスです。

フィールド	説明
pods	デプロイする Pod の 1 つまたは多数の定義を持つサブオブジェクト
templates	デプロイするテンプレートの 1 つまたは多数の定義を持つサブオブジェクト

表12.3 pods および templates のフィールド

フィールド	説明
total	このフィールドは使用しません。
num	整数。デプロイする Pod またはテンプレート数。
image	文字列。プルが可能なリポジトリーの Docker イメージの URL。
basename	文字列。作成するテンプレート (または Pod) のベース名の 1 つの定義。
file	文字列。ローカルファイルへのパス。 作成する PodSpec またはテンプレートのいずれかです。
parameters	キーと値のペア。**parameters** の下で、Pod またはテンプレートでオーバーライドする値の一覧を指定できます。

表12.4 tuningsets の下にあるフィールド

フィールド	説明
name	文字列。チューニングセットの名前。 プロジェクトのチューニングを定義する時に指定した名前と一致します。
pods	Pod に適用される **tuningsets** を特定するサブオブジェクト
templates	テンプレートに適用される **tuningsets** を特定するサブオブジェクト

表12.5 tuningsets pods または tuningsets templates の下にあるフィールド

フィールド	説明
stepping	サブオブジェクト。ステップ作成パターンでオブジェクトを作成する場合に使用するステップ設定。
rate_limit	サブオブジェクト。オブジェクト作成速度を制限するための速度制限チューニングセットの設定。

表12.6 tuningsets pods または tuningsets templates、stepping の下にあるフィールド

フィールド	説明
stepsize	整数。オブジェクト作成を一時停止するまでに作成するオブジェクト数。
pause	整数。**stepsize** で定義したオブジェクト数を作成後に一時停止する秒数。
timeout	整数。オブジェクト作成に成功しなかった場合に失敗するまで待機する秒数。
delay	整数。次の作成要求まで待機する時間 (ミリ秒)。

表12.7 sync の下にあるフィールド

フィールド	説明
server	**enabled** および **port** フィールドを持つサブオブジェクト。ブール値 **enabled** を指定すると、Pod を同期するために HTTP サーバーを起動するかどうか定義します。**port** の整数はリッスンする HTTP サーバーポートを定義します (デフォルトでは **9090**)。
running	ブール値。**selectors** に一致するラベルが指定された Pod が **Running** の状態になるまで待機します。
succeeded	ブール値。**selectors** に一致するラベルが指定された Pod が **Completed** の状態になるまで待機します。
selectors	**Running** または **Completed** の状態の Pod に一致するセレクター一覧
timeout	文字列。**Running** または **Completed** の状態の Pod を待機してから同期をタイムアウトするまでの時間。**0** 以外の値は、単位 [ns|us|ms|s|m|h] を使用してください。

12.4.2. クラスターローダー設定ファイルの例

クラスターローダーの設定ファイルは基本的な YAML ファイルです。

```
provider: local 1
ClusterLoader:
  cleanup: true
  projects:
    - num: 1
      basename: clusterloader-cakephp-mysql
      tuning: default
      ifexists: reuse
      templates:
        - num: 1
          file: ./examples/quickstarts/cakephp-mysql.json

    - num: 1
      basename: clusterloader-dancer-mysql
      tuning: default
      ifexists: reuse
      templates:
        - num: 1
          file: ./examples/quickstarts/dancer-mysql.json

    - num: 1
      basename: clusterloader-django-postgresql
      tuning: default
      ifexists: reuse
      templates:
        - num: 1
          file: ./examples/quickstarts/django-postgresql.json

    - num: 1
      basename: clusterloader-nodejs-mongodb
      tuning: default
      ifexists: reuse
      templates:
        - num: 1
          file: ./examples/quickstarts/nodejs-mongodb.json

    - num: 1
      basename: clusterloader-rails-postgresql
      tuning: default
      templates:
        - num: 1
          file: ./examples/quickstarts/rails-postgresql.json

  tuningsets: 2
    - name: default
      pods:
        stepping: 3
          stepsize: 5
          pause: 0 s
        rate_limit: 4
          delay: 0 ms
```

1. エンドツーエンドテストのオプション設定。**local** に設定して、過剰に長いログメッセージを回避します。
2. このチューニングセットでは、速度制限やステップ設定、複数の Pod バッチ作成、セット間での一時停止などが可能になります。クラスターローダーは、以前のステップが完了したことをモニターリングしてから、続行します。
3. ステップ設定では、オブジェクトが **N** 個作成されてから、**M** 秒間一時停止します。
4. 速度制限は、次のオブジェクトを作成するまで **M** ミリ秒間待機します。

12.5. 既知の問題

IDENTIFIER パラメーターがユーザーテンプレートで定義されていない場合には、テンプレートの作成は **error: unknown parameter name "IDENTIFIER"** エラーを出して失敗します。テンプレートをデプロイする場合は、このエラーが発生しないように、以下のパラメーターをテンプレートに追加してください。

```
{
  "name": "IDENTIFIER",
  "description": "Number to append to the name of resources",
  "value": "1"
}
```

Pod をデプロイする場合は、このパラメーターを追加する必要はありません。

第13章 CPU マネージャーの使用

13.1. CPU マネージャーの機能

CPU マネージャーは、CPU グループを管理して、ワークロードを特定の CPU に制限します。

CPU マネージャーは、以下のような属性が含まれるワークロードに有用です。

- できるだけ長い CPU 時間が必要な場合
- プロセッサーのキャッシュミスの影響を受ける場合
- レイテンシーが低いネットワークアプリケーションの場合
- 他のプロセスと連携し、単一のプロセッサーキャッシュを共有することに利点がある場合

13.2. CPU マネージャーの設定

CPU マネージャーを設定するには、以下を実行します。

1. オプションで、ノードにラベルを指定します。

```
# oc label node perf-node.example.com cpumanager=true
```

2. ターゲットノードで CPU マネージャーのサポートを有効にします。

```
# oc edit configmap <name> -n openshift-node
```

例を以下に示します。

```
# oc edit cm node-config-compute -n openshift-node
```

出力例

```
...
kubeletArguments:
...
  feature-gates:
  - CPUManager=true
  cpu-manager-policy:
  - static
  cpu-manager-reconcile-period:
  - 5s
  system-reserved: (1)
  - cpu=500m
```

```
# systemctl restart atomic-openshift-node
```

(1) **system-reserved** は必須の設定です。この値は、環境に合わせて調整する必要がある場合があります。

3. コア 1 つまたは複数を要求する Pod を作成します。制限および要求の CPU の値は整数にする必要があります。これは、対象の Pod 専用のコア数です。

```
# cat cpumanager.yaml
```

出力例

```
apiVersion: v1
kind: Pod
metadata:
  generateName: cpumanager-
spec:
  containers:
  - name: cpumanager
    image: gcr.io/google_containers/pause-amd64:3.0
    resources:
      requests:
        cpu: 1
        memory: "1G"
      limits:
        cpu: 1
        memory: "1G"
  nodeSelector:
    cpumanager: "true"
```

4. Pod を作成します。

```
# oc create -f cpumanager.yaml
```

5. Pod がラベル指定されたノードにスケジュールされていることを確認します。

```
# oc describe pod cpumanager
```

出力例

```
Name:         cpumanager-4gdtn
Namespace:    test
Node:         perf-node.example.com/172.31.62.105
...
    Limits:
      cpu:     1
      memory:  1G
    Requests:
      cpu:        1
      memory:     1G
...
QoS Class:       Guaranteed
Node-Selectors:  cpumanager=true
                 region=primary
```

6. **cgroups** が正しく設定されていることを確認します。一時停止プロセスの PID を取得します。

```
# systemd-cgls -l
```

出力例

```
├─1 /usr/lib/systemd/systemd --system --deserialize 20
├─kubepods.slice
│ ├─kubepods-pod0ec1ab8b_e1c4_11e7_bb22_027b30990a24.slice
│ │ ├─docker-
b24e29bc4021064057f941dc5f3538595c317d294f2c8e448b5e61a29c026d1c.scope
│ │ │ └─44216 /pause
```

QoS 階層 **Guaranteed** の Pod は、 **kubepods.slice** に配置されます。他の QoS の Pod は、**kubepods** の子である **cgroups** に配置されます。

```
# cd /sys/fs/cgroup/cpuset/kubepods.slice/kubepods-
pod0ec1ab8b_e1c4_11e7_bb22_027b30990a24.slice/docker-
b24e29bc4021064057f941dc5f3538595c317d294f2c8e448b5e61a29c026d1c.scope
# for i in `ls cpuset.cpus tasks` ; do echo -n "$i "; cat $i ; done
```

出力例

```
cpuset.cpus 2
tasks 44216
```

7. 対象のタスクで許可される CPU 一覧を確認します。

```
# grep ^Cpus_allowed_list /proc/44216/status
```

出力例

```
Cpus_allowed_list:    2
```

8. システム上の別の Pod (この場合は **burstable** QoS 階層にある Pod) が、 **Guaranteed** Pod に割り当てられたコアで実行できないことを確認します。

```
# cat /sys/fs/cgroup/cpuset/kubepods.slice/kubepods-burstable.slice/kubepods-burstable-
podbe76ff22_dead_11e7_b99e_027b30990a24.slice/docker-
da621bea7569704fc39f84385a179923309ab9d832f6360cccbff102e73f9557.scope/cpuset.cpus

0-1,3
```

```
# oc describe node perf-node.example.com
```

出力例

```
...
Capacity:
 cpu:     4
 memory:  16266720Ki
 pods:    40
Allocatable:
 cpu:     3500m
 memory:  16164320Ki
 pods:    40
```

```
---
  Namespace                Name                   CPU Requests  CPU Limits  Memory Requests
Memory Limits
  ---------               ----                  ------------  ----------  ---------------  -------------
  test                    cpumanager-4gdtn          1 (28%)       1 (28%)      1G (6%)          1G (6%)
  test                    cpumanager-hczts          1 (28%)       1 (28%)      1G (6%)          1G (6%)
  test                    cpumanager-r9wrq          1 (28%)       1 (28%)      1G (6%)          1G (6%)
...
Allocated resources:
  (Total limits may be over 100 percent, i.e., overcommitted.)
  CPU Requests  CPU Limits  Memory Requests  Memory Limits
  ------------  ----------  ---------------  -------------
  3 (85%)       3 (85%)     5437500k (32%)   9250M (55%)
```

この仮想マシンには、CPU コアが 4 基あります。**system-reserved** は 500 ミリコアに設定して、**Node Allocatable** の数になるようにノードの全容量からコアの半分を引きます。

Allocatable CPU が 3500 ミリコアであることを確認できます。これは、それぞれがコアを 1 つ受け入れるので、CPU マネージャー Pod の内 3 つを実行できるという意味になります。1 つのコア全体は 1000 ミリコアに相当します。

4 つ目の Pod をスケジュールしようとすると、システムは Pod を受け入れますが、スケジュールはされません。

```
# oc get pods --all-namespaces |grep test
```

出力例

```
test          cpumanager-4gdtn           1/1      Running         0        8m
test          cpumanager-hczts           1/1      Running         0        8m
test          cpumanager-nb9d5           0/1      Pending         0        8m
test          cpumanager-r9wrq           1/1      Running         0        8m
```

第14章 HUGE PAGE の管理

14.1. HUGE PAGE の機能

メモリーは Page と呼ばれるブロックで管理されます。多くのシステムでは、1 ページは 4Ki です。メモリー 1Mi は 256 ページに、メモリー 1Gi は 262,144 ページに相当します。CPU には、内蔵のメモリー管理ユニットがあり、ハードウェアでこのようなページリストを管理します。トランスレーションルックアサイドバッファー (TLB: Translation Lookaside Buffer) は、仮想から物理へのページマッピングの小規模なハードウェアキャッシュのことです。ハードウェアの指示で渡された仮想アドレスが TLB にあれば、マッピングをすばやく決定できます。そうでない場合には、TLB ミスが発生し、システムは速度が遅く、ソフトウェアベースのアドレス変換にフォールバックされ、パフォーマンスの問題が発生します。TLB のサイズは固定されているので、TLB ミスの発生率を減らすには Page サイズを大きくする必要があります。

Huge Page とは、4Ki より大きいメモリーページのことです。x86_64 アーキテクチャーでは、2Mi と 1Gi の 2 つが一般的な Huge Page サイズです。別のアーキテクチャーではサイズは異なります。Huge Page を使用するには、アプリケーションが認識できるようにコードを書き込む必要があります。Transparent Huge Pages (THP) は、アプリケーションによる認識なしに、Huge Page の管理を自動化しようとしますが、制約があります。特に、ページサイズは 2Mi に制限されます。THP では、THP のデフラグが原因で、メモリー使用率が高くなり、断片化が起こり、パフォーマンスの低下につながり、メモリーページがロックされてしまう可能性があります。このような理由から、アプリケーションは THP ではなく、事前割り当て済みの Huge Page を使用するように設計 (また推奨) される場合があります。

OpenShift Container Platform では、Pod のアプリケーションが事前に割り当てられた Huge Page を割り当て、消費することができます。以下のトピックでは、その方法について説明します。

14.2. 前提条件

1. ノードは、Huge Page の容量をレポートできるように Huge Page を事前に割り当てる必要があります。ノードは、単一サイズの Huge Page のみを事前に割り当てることができます。

14.3. HUGE PAGE の消費

Huge Page は、リソース名の **hugepages-<size>** を使用してコンテナーレベルのリソース要件で消費可能です。この場合、サイズは特定のノードでサポートされる整数値を使用した最もコンパクトなバイナリー表記です。たとえば、ノードが 2048KiB ページサイズをサポートする場合、これはスケジュール可能なリソース **hugepages-2Mi** を公開します。CPU やメモリーとは異なり、Huge Page はオーバーコミットをサポートしません。

```
kind: Pod
metadata:
  generateName: hugepages-volume-
spec:
  containers:
  - securityContext:
      privileged: true
    image: rhel7:latest
    command:
    - sleep
    - inf
    name: example
    volumeMounts:
    - mountPath: /hugepages
```

```
    name: hugepage
  resources:
    limits:
      hugepages-2Mi: 100Mi 1
volumes:
- name: hugepage
  emptyDir:
    medium: HugePages
```

1. **hugepages** のメモリー量は、実際に割り当てる量に指定します。この値は、ページサイズで乗算した **hugepages** のメモリー量に指定しないでください。たとえば、Huge Page サイズが 2MB と仮定し、アプリケーションに Huge Page でバックアップする RAM 100 MB を使用する場合には、Huge Page は 50 に指定します。OpenShift Container Platform により、計算処理が実行されます。上記の例にあるように、**100MB** を直接指定できます。

プラットフォームによっては、複数の Huge Page サイズをサポートするものもあります。特定のサイズの Huge Page を割り当てるには、Huge Page の起動コマンドパラメーターの前に、Huge Page サイズの選択パラメーター **hugepagesz=<size>** を指定してください。**<size>** の値は、バイトで指定する必要があります。その際、オプションでスケール接尾辞 [**kKmMgG**] を指定できます。デフォルトの Huge Page サイズは、**default_hugepagesz=<size>** の起動パラメーターで定義できます。詳細は、Huge Page および Transparent Huge Pages を参照してください。

Huge Page 要求は制限と同じでなければなりません。制限が指定されているにもかかわらず、要求が指定されていない場合には、これがデフォルトになります。

Huge Page は、Pod のスコープで分割されます。コンテナーの分割は、今後のバージョンで予定されています。

Huge Page がサポートする **EmptyDir** ボリュームは、Pod 要求よりも多くの Huge Page メモリーを消費することはできません。

shmget() で **SHM_HUGETLB** を使用して Huge Page を消費するアプリケーションは、**proc/sys/vm/hugetlb_shm_group** に一致する補助グループで実行する必要があります。

第15章 GLUSTERFS ストレージでの最適化

15.1. データベースのコンバージョンモードに関するガイド

アプリケーションにコンバージドモードを使用する場合には、お使いのワークロードの種類によって、gluster-block と GlusterFS モードを使い分けられるように、このトピックで説明されているガイドとベストプラクティスに従うようにしてください。

15.2. テスト済みのアプリケーション

OpenShift Container Platform 3.10 では、これらの SQL データベースを使用する場合および使用しない場合に関連して広範なテストが行われました。

- Postgresql SQL v9.6
- MongoDB noSQL v3.2

これらのデータベースのストレージは、コンバージドモードのストレージクラスターから取得しています。

Postgresql SQL ベンチマークについては、pgbench がデータベースのベンチマークに使用されていました。MongoDB の場合は、noSQL ベンチマーク YCSB Yahoo!Cloud Serving Benchmark ベンチマークに使用され 、workloada、workloadb、workloadf がテストされました

15.3. サポートマトリックス

表15.1 表タイトル: GlusterFS

データベース	ストレージバックエンド: **GlusterFS**	**off** にするパフォーマンス変換	**on** にするパフォーマンス変換
Postgresql SQL	○	• performance.stat-prefetch • performance.read-ahead • performance.write-behind • performance.readdir-ahead • performance.io-cache • performance.quick-read • performance.open-behind	• performance.strict-o-direct

MongoDB noSQL	○	• performance.stat-prefetch • performance.read-ahead • performance.write-behind • performance.readdir-ahead • performance.io-cache • performance.quick-read • performance.open-behind	• performance.strict-o-direct

表15.2 表タイトル: gluster-block

データベース	ストレージバックエンド: **gluster-block**
Postgresql	○
MongoDB	○

上述のように GlusterFS のパフォーマンス変換は、コンバージドモードの最新イメージで提供されるデータベースプロファイルにすでに含まれています。

15.4. テスト結果

Postgresql SQL データベースの場合は、GlusterFS と gluster-block のパフォーマンスはほぼ同じ結果となりました。MongoDB noSQL データベースの場合は、gluster-block のパフォーマンスのほうが優れています。したがって、MongoDB noSQL データベースには、gluster-block ベースのストレージを使用します。