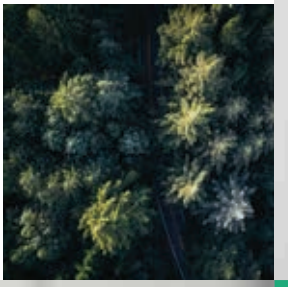


ホワイトペーパー

コンテナ向けの永続的ストレージを容易に実現

アプリケーション開発のスピードアップと
DevOpsの効率化を同時に実現

NetApp®

課題：DevOps の成熟 3

大規模コンテナが抱える障害：ステートフル アプリケーション向け
ストレージの管理 3

コンテナ、データの永続性、ステートレス アプリケーションと
ステートフル アプリケーションの概要 4

コンテナ向け永続的ストレージの基本的なプロビジョニングによって
プロジェクトを開始 4

進歩は見られるが相変わらず待ち時間が長いという問題 4

ストレージ クラス、ストレージ プール、Trident：
オンデマンドの永続的ストレージ 5

ストレージクラスのカタログでプロビジョニングが簡単に 5

DevOps の未来 6

作業方法を変革 6

お客様とビジネスをサポート 7

関連資料 7

課題：DevOps の成熟

たゆみない革新。時期を逃さない迅速な製品リリース。運用の合理化。満足度の高い顧客の増加。

いずれも、多くの組織に共通する目標です。業績が向上し、売り上げや最終収益の拡大に向かっていることを示す目安でもあります。

そうした目標達成への道のりの裏側では、ビジネス変革のさなかにある組織によくある例として、問題や不満も溜まりつつあります。迅速な開発で顧客のニーズに応えたいアプリケーション開発者は、サービス チケットの発行や、ストレージやコンピューティング リソースの割り当てに時間がかかることを不満に思っているかもしれません。一方、インフラと運用（I&O）のチームは、発行されるサービス チケットへの対応やインフラの活用方法の管理に追われています。

目標を掲げながらも、ビジネス成長に伴う課題を抱えている組織は、即応性に優れた無駄のない手法で製品開発を効率化し、スピードアップを追求しなければなりません。即応性に優れた無駄のない手法には大きなメリットがありますが、プロセスに合った IT インフラで必要な効率性とスピードを実現すれば、さらに大きなメリットが得られます。

そこで DevOps の登場となるわけです。DevOps は、アプリケーション開発と IT インフラ運用を短期間で同時に変革する手法です。

DevOps が成熟している組織には、6 つの重要な能力に長けているという特徴がよく見られます。その能力とは次のとおりです。

- **コード / アーティファクト / バイナリ管理：**ソフトウェア コンポーネントの維持と管理のリポジトリ
- **構成管理：**インフラとソフトウェアのシステムを既知の方法で設定、管理
- **クラウド / PaaS：**パブリック、プライベート、ハイブリッドの各クラウド インフラでソフトウェア開発をサポート
- **コンテナ：**軽量ながら拡張性に優れたアプリケーション ランタイム環境
- **分析：**インフラの自動監視と自動管理
- **継続的統合 / 継続的導入（CI / CD）：**開発者がコードを記述して自動で導入できるエンドツーエンドのプロセス

スピードアップと効率化を模索するアプリケーション開発チームやインフラ運用チームでは、以上 6 つの領域のうちで、特にコンテナの重要性が増しています。本ホワイトペーパーでは、コンテナを用いて DevOps の成熟を目指すこの動向を取り上げるとともに、考慮すべき主な問題のひとつ、ステートフル アプリケーション向け永続的ストレージの管理に注目します。

コンテナが抱える大きな障害：ステートフル アプリケーション向け永続的ストレージの管理

DevOps チームが、コンテナ化されたアプリケーションを実際の本番環境で運用することを検討し始めたとき、さまざまな課題が生じました。なかでも無視できないものが、コンテナ向け永続的データ ストレージの管理です。

コンテナで実現するデータ永続性の考え方や、ステートフル アプリケーションとステートレス アプリケーションの違いなどを理解したい場合は、4 ページのコラムを参照してください。

データ ストレージの処理が課題であることは、アプリケーションのコンテナ化が始まった当初から明らかでした。Cloud Native Computing Foundation（CNCF）の調査では、回答者の半数近く（42%）が、コンテナの導入ではストレージとリソースの管理が大きな課題になると答えています（図 1）[1]。さらにその多くが、ストレージの永続性が課題の焦点であるとしています。ネットワーク ストレージへのアクセスのしやすさを求める回答者もいました。

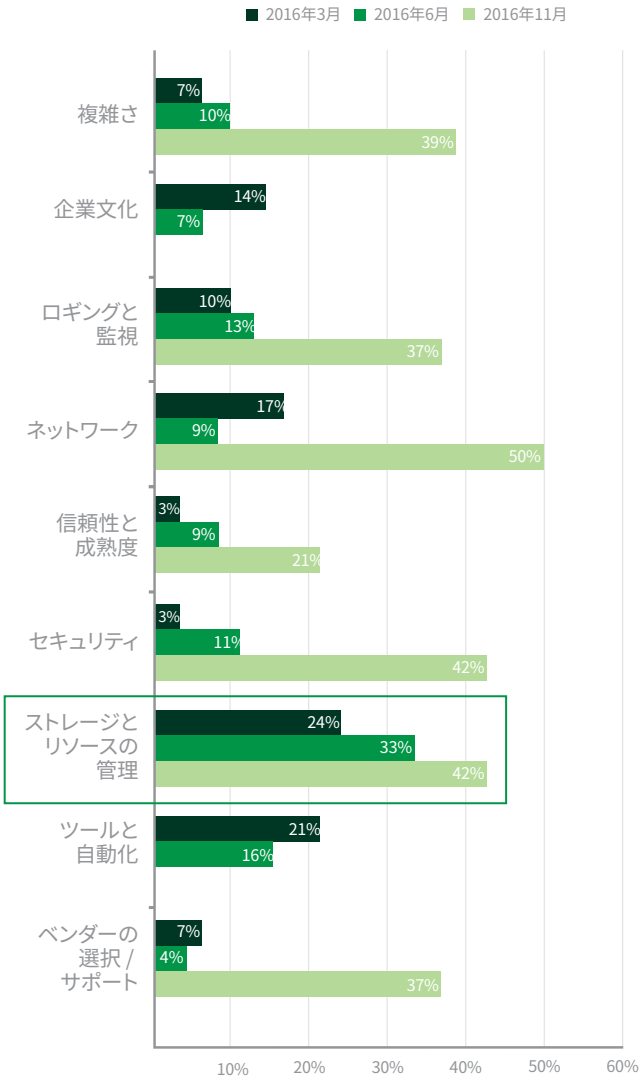


図 1）コンテナ導入に関する課題（回答は複数選択可）
出典：Cloud Native Computing Foundation

このことから、コンテナ プラットフォームにおける永続的ストレージの課題への取り組みが始まりました。初期の、永続的ストレージのプロビジョニングに関する取り組みは好調な滑り出しを見せましたが、やや柔軟性に欠け、手作業が多すぎる点は変わりませんでした。DevOps パイプラインの高速化も実現できないように思われました。特に、アプリケーションをコード化してテストし、数百や数千のコンテナに導入する必要がある場合は尚更です。

数千のコンテナにストレージを手動でプロビジョニングすることは、拡張性の点で無理があると見られ始めました。煩雑すぎてミスを招きやすく、維持するのが難しすぎるからです。もっと良い方法が必要でした。

コンテナが登場するまで、エンタープライズ アプリケーションの多くは、一元化された共有エンタープライズ ストレージに接続することで永続的データ ストレージのニーズを満たしていました。コンテナ環境でも、同じような共有ストレージ機能に簡単にアクセスできれば、ステートフル アプリケーションを問題なく運用できるのではないでしょうか。しかし、この問題はまだ続いています。簡単に解決するには、どうすれば良いのでしょう。

コンテナ向け永続的ストレージの基本的なプロビジョニングによってプロジェクトを開始

Docker や Kubernetes などのコンテナ環境は元々、ストレージの永続性に関する課題に、使いやすい半自動メカニズムで対応していました。このメカニズムを用いると、ユーザは、特定の永続的ストレージ ボリュームを作成するよう「要求」して、1 つ以上のコンテナ プロセスに使用することができます。ただしこれには、ストレージ管理者が最初に、基盤にあるさまざまなネットワーク ストレージ リソースから永続的ボリュームを作成する必要がありました。

Kubernetes：実践的な基本のプロビジョニング

Kubernetes は、このプロセスに、PersistentVolume（PV）と、対応する PersistentVolumeClaim（PVC）を関連付ける、コードベースのメカニズムを採り入れました。

まず管理者は、静的な永続的ボリューム（容量が 8GB のオールフラッシュ ストレージのボリュームなど）を作成します。すると、アプリケーション ユーザ（または開発者）は、その永続的ボリュームを数行のコードでリクエスト（要求）できるようになります。

この、PV と PVC を関連付けて要求する方法は、コンテナ環境にとって魅力的でした。コンテナ化されたアプリケーションが、永続的なネットワーク ストレージをコード経由で消費できるようになったからです。

ネットワーク ストレージを消費できるようになり、滑り出しは好調に見えたものの、この基本的なプロビジョニング方法には、いくつもの課題が残っていました。最大の課題は、自動化の 必要性です。

進歩は見られるが相変わらず待ち時間が長いという問題

事前にプロビジョニングした PV と、対応する PVC を関連付けて使用する方法は、コンテナの永続的ストレージの問題を解決する最初の一歩として、大変優れていました。ただし、ストレージ プロビジョニング プロセスの多くは依然として静的な手作業であり、開発と運用のどちらでも、要求を繰り返さなければなりませんでした。これは、作業を自動化して、CI / CD に引き渡す業務を減らしたいと考えているアプリケーション開発チームや DevOps チームにとっては非効率的です。

コンテナ、データの永続性、ステートレス アプリケーションとステートフル アプリケーションの概要

コンテナは本来ステートレスであり、コンテンツに永続性はありません。これは、コンテナに関連付けられたアプリケーションやプロセスを素早く開始、停止、リスタートできるのは、一定のセッション中だということです。デフォルトでは、コンテナの運用中に作成したデータは、コンテナを削除すると同時に削除されます。

ところがあいにく、DevOps チームがコンテナ プラットフォームをベースに開発したアプリケーションや、本番環境向けに導入するアプリケーションが増えたことから、1 つのコンテナのライフサイクルが終了しても、データを維持し永続させるニーズが多方面に生じました。

多くの開発者はすぐに、ほぼすべてのアプリケーションに、永続的ボリューム（PV）を必要とするプロセスやマイクロサービスが 1 つはあることに気付きました。永続的ボリューム（PV）があれば、コンテナのライフサイクルを超えてデータの状態を維持できます。

データの永続性を必要とするステートフル アプリケーションには、たとえば次のものがあります。

- **データベース環境：**データベース コンテナには、データストアを格納するための永続的ストレージが必要です。しかし、コンテナが本来永続的でないことを考えれば、コンテナだけでこれを実現することはできません。ローカル ストレージもふさわしい方法ではありません。コンテナを移動したり削除したりすれば、データにアクセスできなくなります。
- **環境データやセッション データ：**ステートフル アプリケーションでは、アプリケーション環境の属性やクライアント セッション データ（の状態）の収集や保存が行われることもよくあります。これは、クライアントの動作を強化する履歴データとして機能します。これにより、次にクライアントがアプリケーションに働きかけると、関連するデータを提供したり、以前のセッションで作成したデータをスムーズに使用したりできます。

データの状態を保存する必要性と同じように重要なのが、データ共有の必要性です。開発、テスト、運用のどの段階でも、一元管理されたネットワーク ストレージ リソースから同じデータセットにアクセスできなければなりません。アプリケーション コンテナの作成や移行を行っている組織が、これに気付くのに時間はかかりませんでした。ネットワーク ストレージに格納された永続的ボリュームなら、コンテナ化されたアプリケーションをエンタープライズクラスのストレージ機能で強力に保護し、アプリケーションの可用性、信頼性、セキュリティ、データ保護を向上させることも可能です。

- **ストレージへのリクエスト：**アプリケーション開発者や QA 担当者は、コードを使った作業やスプリントのテスト中に作業を中断して、ストレージ管理者に PV の作成を依頼しなければならないことが考えられます。そのため、大規模なコンテナ アプリケーションの拡張時には、何百、何千ものストレージへのリクエストが継続的に発生するかもしれません。
- **ストレージ割り当ての待ち時間に関する不満：**開発チームや QA チームは、要求が承認されるまで待機状態になることが考えられます。PV は手動作成なので、管理者がその時間を予定に組み込む間も待機状態になると思われます。
- **ストレージの使い方が非効率的：**開発や QA をサポートするために PV 用ストレージを事前にプロビジョニングすると、プロビジョニングしたストレージ容量が極端に少なかったり（アンダープロビジョニング）、極端に多すぎたり（オーバープロビジョニング）することがあります（図 2）。ボリューム 1 つにリザーブする IOPS が少なすぎたり多すぎたりする場合も考えられます。
  - **アンダープロビジョニングのリスク：**前もって PV 用にプロビジョニングしたストレージが少なすぎると、使用可能な容量よりも多くの容量を必要とする開発者の作業が制約されるというボトルネックが生じるおそれがあります。
  - **オーバープロビジョニングのリスク：**オーバープロビジョニングされたストレージ リソースが 1 つ以上の PV 専用になっていると、必要な容量やパフォーマンスがはるかに少ない PVC が PV に関連付けられた場合に、ストレージが無駄になるおそれがあります（オーバープロビジョニングして、ほとんど使用されていないストレージ サイロという従来の問題に聞き覚えはないでしょうか）。

- **非効率的な管理作業：**アプリケーション開発が複数進行していると、関係しているストレージ管理者やクラスタの管理者は、1 つ以上の DevOps パイプラインで自分たちが新たなボトルネックになっていると気付くことがあります。新しいボリューム要求が土壇場で複数発生しても、管理者は静的な PV ごとに、いくつもの手順を手動で実行して設定と作成を行わなければなりません。インフラの自動化とインフラ全体の消費状況の監視に携わる管理者にとっても、この方法は何の役にも立ちませんでした。

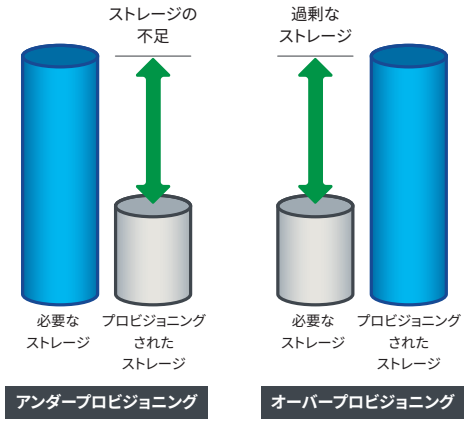


図 2）アンダープロビジョニングとオーバープロビジョニングのリスク

これを、従来型のストレージ プロビジョニングのようだと感じているのは、一部のユーザだけではありません。ネットアップのテクニカル マーケティング エンジニア、Andrew Sullivan は、2017 年の Tech Field Day のセッション「DevOps Through Desired State」の終わりに、こうした従来型プロビジョニングから抜け出したがっている DevOps チームに対して、次のような同情の言葉を述べています。**「ストレージ リソースに関して我慢はしません。私はいつも、ストレージ チームに文句を言って容量を増やしてもらっています。なぜ自分でストレージをプロビジョニングしなければならないのでしょう。2017 年の今も、ストレージ消費の方法が 1989 年と同じだなんて、もってのほかです」**[2]

ありがたいことに、コンテナ環境を支えるネットアップなどのベンダー各社は、DevOps チームにはストレージをオンデマンドで消費し、必要なときに必要な場所にストレージを動的にプロビジョニングできる優れた方法が必要なことを知っていました。

ストレージ クラス、ストレージ プール、Trident：オンデマンドの永続的ストレージ

前項で Andrew Sullivan が言おうとしたのは、ストレージ リソースのプロビジョニング方法は改善が必要だということです。これには、元々コンテナに期待されていた、手動操作ほぼゼロで、機能をオンデマンドで動的に構築して導入するという考えが反映されています。

Kubernetes の場合は、別の考え方である StorageClass を通じた動的なストレージ プロビジョニングが導入されることになりました。また、Kubernetes 対応 Trident など、ストレージ プロビジョニングを自動化する優れた機能が利用されることにもつながりました。

- **オンデマンドの永続的ストレージ：**オープン ソースの Trident プロジェクトは、ネットアップが開発した動的なストレージ プロビジョニングであり、コンテナ化されたアプリケーションが、必要なときに必要なネットアップ ストレージの永続的ボリュームにオンデマンドでアクセスできるようにして、ストレージが割り当てられるまでの待ち時間をなくします。
- **基盤の強力なストレージを活用：**コンテナ環境に Trident を使用すると、基盤にあるネットアップ データ管理プラットフォーム（NetApp HCI、ONTAP®、NetApp SolidFire® Element® OS、SANtricity® など）の強力なストレージ機能を活用できます。
- **Kubernetes 以外もサポート：**Docker や OpenShift にも対応した Trident を通じて、Kubernetes 以外の環境でも同じ機能で永続的ボリュームを利用できます。

ストレージクラスのカタログでプロビジョニングが簡単に

Trident を使用すると、Kubernetes 環境を使用している開発者は、基盤のストレージの仮想プールにストレージ クラスを要求するだけで、永続的ボリュームを動的にプロビジョニングできるようになります。

その仕組みを見てみましょう。ストレージ クラスを使用すると、永続的ボリュームのプロビジョニングがコードによって自動で実行されます。ユーザは、永続的ボリュームを要求する際に、特定の Trident ストレージ クラスを Gold、Silver、Bronze などで指定するだけです。

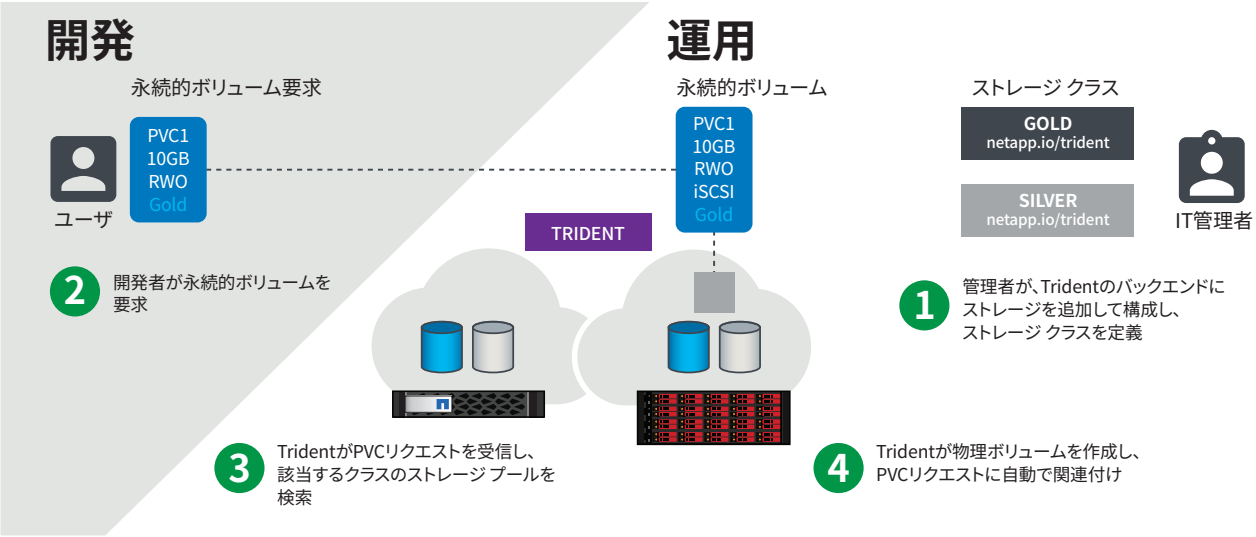


図 3）Trident による永続的ボリュームの動的プロビジョニング

注：基盤のストレージ クラスの属性の設定と命名規則はバックエンドの機能です。通常は、永続的ストレージ クラスのカタログを最初に作成する際に、管理者が設定して命名します。ストレージ クラスは、Dev、Staging、Production のような簡単な名前にすることもできれば、Fast、Slow のような名前にすることもできます。各種の設定機能については、Trident の製品ガイド[3] で詳細をご確認ください。

ストレージ クラスを指定すると、基盤のネットアップ ストレージから指定したクラスで永続的ボリュームが作成され、ユーザの永続的ボリューム要求（PVC）に結び付けられます。ユーザが基盤のストレージのことを意識する必要はありません。すべて Trident が処理し、完了します。

Trident のデモ

Tridentの価値は、動的プロビジョニングが機能している様子を見るとよくわかります。以下のオンライン デモをぜひご覧ください。

- **簡易デモ**（3 分 20 秒）：OpenShift 環境での Trident による動的ストレージ プロビジョニング[4]
- **詳細デモ**（23 分 53 秒）：Kubernetes 環境で永続的データを管理[5]

ITチームやDevOpsチームのメンバーにとって、このプロビジョニング機能には、どういった意味があるのでしょう。

- **開発チームや QA チームの場合：**サービス チケットの発行やストレージ要求の承認を待つ必要がなくなります。業務の引き渡しも不要です。ストレージは、使い慣れたコードを介して動的にプロビジョニングされるので、合意済みの SLA に基づき、必要に応じて消費できます。開発者は、柔軟で動的な自動プロビジョニング システムによって自由にストレージを消費しつつ、運用に関わり続けることができます。
- **IT 管理者やストレージ管理者の場合：**土壇場でのストレージ プロビジョニングの要求や、際限のないサービス チケットの発行に振り回されることがなくなります。ストレージ管理に費やす時間が減って、インフラ拡張の自動化を推進できます。ストレージ消費を予測して管理できるようになり、リソースの監視が簡単になります。
- **IT 担当エグゼクティブの場合：**製品提供の促進、プロセスの改善、リソースの大幅節約が実現します。

DevOps の未来

DARZ は IT サービス総合プロバイダとして、Docker コンテナ サービス ソリューションで DevOps の即応性を実現しています。このソリューションの基盤は、ネットアップのオールフラッシュ ストレージと Trident for Docker です。ユーザは、本格的なオペレーティング システムなしでアプリケーション コンテナのスピン アップやスピン ダウンを素早く行うことができ、必要なコンピューティング システムは 4 分の 1 で済みます。

さらに、無駄のない柔軟な一元管理された環境により、テスト サイクルを短縮して開発を促進し、新製品をスピーディに導入できます。Trident は、Docker ボリュームのコマンド セットを使ってコンテナ ストレージの操作を簡易化し、Docker 環境の永続的データを容易に管理できるようにします。[6]

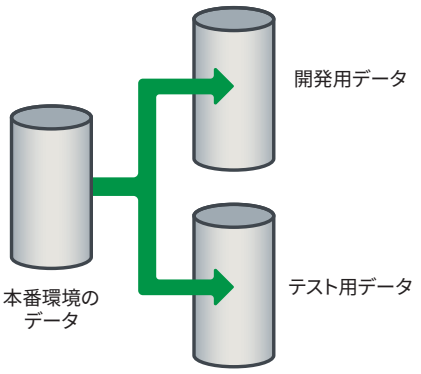
作業方法を変革

Trident のようなツールを使用すると、永続的ボリュームの動的プロビジョニング以外にも多くのことができます。

たとえば、Snapshot ™コピーやクローニングなどの NetApp Storage Efficiency 機能を、コードを介してオンデマンドで利用できます。使用するリソースを最小限に抑え、時間をかけずに多くの作業をこなさなければならない開発や QA チームにとって、この Storage Efficiency 機能は画期的かもしれません。

NetApp Trident経由でSnapshotコピーやクローニングを使用すると、次のことが可能になります。

- **本番環境のデータセット一式をリアルタイムのクローニングで素早く作成：**新しい開発環境やテスト環境を、わずか数行のコードで動的かつ瞬時に作成できます（図 4 を参照）。
- **容易で迅速なデータ リカバリ：**開発に Snapshot コピーを使用すると、データセットを以前のバージョンに素早くロールバックできます。コードのテストにも便利です。テスト用のデータセットを何度も作成しなくても、短時間でテストを繰り返せます。
- **ストレージ容量を削減する Snapshot コピーとクローン：**本番環境のデータのクローンを複数作成して開発やテストに使用すると、多くの場合、ストレージ容量を 40 ～ 90%[7] 削減できます。



クローニングにより、開発とテストのワークフローを素早くサポート

図 4）ネットアップの強力なクローニングで DevOps ワークフローを高速化

ネットアップのクローニングと Snapshot が Trident とどのように連携するのかについては、次の記事をご覧ください。

- Kubernetes 環境でのセルフサービスによるボリューム クローニング[8]
- Trident を介した Snapshot コピーとセルフサービス ボリューム リカバリ[9]
- Tech ONTAP のこちらのポッドキャストでは、ネットアップの Oracle エキスパートである Jeff Steiner が、Docker と Trident を使用して大規模な Oracle データベースをわずか 22 秒で作成する様子を紹介しています。[10]

お客様とビジネスをサポート

開発者とエンジニアのニーズに関する問題の解決は、ネットアップにとって初めての経験ではありません。実を言えば、これがネットアップの伝統芸なのです。弊社はベンダーとして、ストレージ インフラをどのように使えば重要な開発業務やエンジニアリング業務にかかる時間を短縮し、企業のさまざまな目標を素早く達成できるかに早くから気付いていました。

弊社の仕事は、データ管理とストレージ消費のより良い方法を確立することです。ネットアップは、コンテナなどのオープンなエコシステムを、イノベーションをさらに追求するために登場した新しい領域と見ています。このイノベーションは、コンテナのコミュニティ メンバーによって推進されていますが、ネットアップはこのコミュニティの一員として、推進に積極的に関わっています。その目標の下、ネットアップは、コミュニティ メンバーが必要なときに必要な場所で今よりも簡単にストレージを消費できるよう、さまざまな手法の開発に取り組んでいます。現在のネットアップの強みは、各種のオープンなエコシステムでシームレスにストレージを消費できることです。弊社は常に、業界随一の包括的な API セットの開発と、Docker、Kubernetes、OpenShift、OpenStack、Ansible、Chef、Puppet などの環境の統合に取り組んでいます。

そうした取り組みの一例が Trident です。ぜひお客様のコンテナ環境に Trident をお試しいただければと思います。DevOps パイプラインが驚くほど効率化され、リソースを削減できたという声が届く日を楽しみにしています。

関連資料

Trident をはじめとするネットアップの DevOps 統合ソリューションについては、次のリソースで詳細をご確認ください。

ネットアップと Trident に関する詳細

コンテナ向けネットアップ ソリューション	
Trident の機能の概要：Introducing Trident	
Introduction to Kubernetes persistent storage paradigm and Trident	
クローニング：Trident によるボリューム クローニングの紹介（Kubernetes 環境）	
Trident 製品ガイド	
Trident のダウンロード（GitHub）	

ネットアップと DevOps に関する詳細

DevOps 向けネットアップ ソリューション	
thePub（netapp.io）	
ネットアップの Slack チャンネル（netapp.io/slack）	
@NetAppPub	

注

[1]「Meeting Challenges in Using and Deploying Containers」執筆者：Cloud Native Computing Foundation、Sarah Conway 氏（2017 年 4 月 27 日）、https://www.cncf.io/blog/2017/04/27/meeting-challenges-using-deploying-containers/ Creative Commons CC-BY 4.0 ライセンスの下、赤丸付きで再掲
[2]「DevOps Through Desired State」発表者：ネットアップ Andrew Sullivan（2017 年 5 月 11 日、Tech Field Day 第 14 日）、https://www.youtube.com/watch?v=btLZl7M6gnY&list=PLinuRwpnsHacYmunO7zyES6SyrsrFfu5O&index=4
[3] 最新の Trident 製品ガイド：https://netapp-trident.readthedocs.io/
[4] オンライン デモ「Using Trident for Dynamic Storage Provisioning with OpenShift」（3 分 20 秒）thePub @ NetApp（2017 年 2 月 17 日）、https://www.youtube.com/watch?v=97VZWYssL2E
[5] オンライン デモ「Managing Persistent Data in Kubernetes」（23 分 53 秒）thePub @ NetApp（2017 年 5 月 15 日）、https://www.youtube.com/watch?v=XIuN91vG2wM
[6] お客様の導入事例『DARZ Docker & Container-as-a-Service Drives Digital Transformation Through DevOps』ネットアップ（2017 年）、https://www.netapp.com/jp/media/cs-darz-devops.pdf
[7] ネットアップ コミュニティのブログ「How NetApp IT Shortened Development Cycles Using FlexClone」執筆者：Gopal Parthasarathy（2015 年 10 月 8 日）、https://community.netapp.com/t5/Technology/How-NetApp-IT-Shortened-Development-Cycles-Using-FlexClone/ba-p/110581
[8]「Trident 18.01 beta 1: Introducing volume cloning to Kubernetes!」執筆者：ネットアップ Garrett Mueller（2017 年 12 月 14 日）、https://netapp.io/2017/12/14/trident-18-01-beta-1-introducing-volume-cloning-kubernetes/（「Trident 18.01 is Here」執筆者：ネットアップ Andrew Sullivan（2018 年 1 月 25 日）、https://netapp.io/2018/01/25/trident-18-01/ もご参照ください）
[9]「Self-Service Data Recovery using Trident and NFS」執筆者：ネットアップ Andrew Sullivan（2018 年 4 月 3 日）、https://netapp.io/2018/04/03/self-service-data-recovery-using-trident-nfs/
[10] Tech ONTAP ポッドキャスト「Episode 99 - Databases as a Service: Containers」（2017 年）、https://soundcloud.com/techontap_podcast/episode-99-databases-as-a-service-containers

本ドキュメントに記載されている製品や機能のバージョンがお客様の環境でサポートされるかどうかについては、ネットアップ サポート サイトで Interoperability Matrix Tool（IMT）を参照してください。NetApp IMT には、ネットアップがサポートする構成を構築するために使用できる製品コンポーネントやバージョンが定義されています。サポートの可否は、お客様の実際のインストール環境が公表されている仕様に従っているかどうかによって異なります。

著作権に関する情報
Copyright © 2019 NetApp, Inc. All rights reserved.Printed in the U.S. このドキュメントは著作権によって保護されています。著作権所有者の書面による事前承諾がある場合を除き、画像媒体、電子媒体、および写真複写、記録媒体、テープ媒体、電子検索システムへの組み込みを含む機械媒体など、いかなる形式および方法による複製も禁止します。

ネットアップの著作物から派生したソフトウェアは、次に示す使用許諾条項および免責条項の対象となります。このソフトウェアは、ネットアップによって「現状のまま」提供されています。ネットアップは明示的な保証、または商品性および特定目的に対する適合性の暗示的保証を含み、かつこれに限定されないいかなる暗示的な保証も行いません。ネットアップは、代替品または代替サービスの調達、使用不能、データ損失、利益損失、業務中断を含み、かつこれに限定されない、このソフトウェアの使用により生じたすべての直接的損害、間接的損害、偶発的損害、特別損害、懲罰的損害、必然的損害の発生に対して、損失の発生の可能性が通知されていたとしても、その発生理由、根拠とする責任論、契約の有無、厳格責任、不法行為（過失またはそうでない場合を含む）にかかわらず、一切の責任を負いません。

ネットアップは、ここに記載されているすべての製品に対する変更を随時、予告なく行う権利を保有します。ネットアップによる明示的な書面による合意がある場合を除き、ここに記載されている製品の使用により生じる責任および義務に対して、ネットアップは責任を負いません。この製品の使用または購入は、ネットアップの特許権、商標権、または他の知的所有権に基づくライセンスの供与とはみなされません。

このマニュアルに記載されている製品は、1 つ以上の米国特許、その他の国の特許、および出願中の特許によって保護されている場合があります。

権利の制限について：政府による使用、複製、開示は、DFARS 252.277-7103（1988 年 10 月）の Rights in Technical Data and Computer Software（技術データおよびコンピュータソフトウェアに関する諸権利）条項の (c) (1) (ii) 項、および FAR 52-227-19（1987 年 6 月）に規定された制限が適用されます。

商標に関する情報
NetApp、NetApp のロゴ、http://www.netapp.com/jp/legal/netapptmlist.aspx に記載されているマークは、NetApp, Inc. の商標です。その他の会社名と製品名は、それを所有する各社の商標である場合があります。

WP-7270-071-JP