Red Hat Satellite 6.10

Red Hat Satellite 5 から Red Hat Satellite 6 への移行

Satellite 5 から Satellite 6 への移行ガイド

Last Updated: 2023-11-28

Red Hat Satellite 6.10 Red Hat Satellite 5 から Red Hat Satellite 6 への移行

Satellite 5 から Satellite 6 への移行ガイド

Red Hat Satellite Documentation Team
satellite-doc-list@redhat.com

法律上の通知

Copyright © 2023 Red Hat, Inc.

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

本書は、既存の Satellite 5.6 または 5.7 サーバーを新しい Satellite 6 Server に移行する方法を説明します。ここでは、必要な準備および前提条件、クライアントの移行に使用するブートストラップスクリプト、移行の完了後に使用していた Satellite 5.6 または 5.7 デプロイメントをリタイアさせる方法を説明します。

目次

第1章 はじめに

Satellite では、Satellite 5 Server から Satellite 6 Server へ移行できます。Satellite 6 は Satellite 5 とは大きく異なります。そのため、(バージョン 4.x から 5.x へなどの) インプレースアップグレードプロセスは、バージョン 5 から 6 への移行には適用されません。新規サーバーに Satellite 6 をインストールして、クライアントをその新規 Satellite 6 のインスタンスに移行する必要があります。したがって、これはアップグレードプロセスではなく移行プロセスと呼ばれます。Red Hat は、Satellite サブスクリプションを二重で 1 年間提供するため、Satellite 5 クライアントから新しいシステムへ移行する場合に **限り**、移行の前に Satellite 6 Server を構築してテストできます。

移行のワークフロー

Satellite 5 から Satellite 6 への移行プロセスは、以下のような手順になります。

1. ドキュメントを読み、Satellite 6 製品の基礎を理解する。s390 システムに Satellite 5.8 を実行している場合は、x86_64 アーキテクチャーシステムに新規の Satellite 6 をビルドする必要があります。Satellite 6 は、s390 システムでは実行できません。

2. 新しいマシンに Satellite 6 をインストールし、マニフェストでアクティベートし、CDN の Red Hat コンテンツと同期する。

3. クライアントシステムにブートストラップをインストールする。

4. 前提条件を満たしていることを確認して、ブートストラップスクリプトを実行し、クライアントを移行する。

サポート対象の Red Hat Enterprise Linux バージョン

Satellite 6 は、Satellite 5 とは異なり、Red Hat Enterprise Linux 4 以前で稼働しているクライアントをサポートしません。移行を開始する前に、すべてのクライアントが以下の Red Hat Enterprise Linux バージョンで設定されていることを確認してください。

- Red Hat Enterprise Linux 5.7 以降
- Red Hat Enterprise Linux 6.1 以降
- Red Hat Enterprise Linux 7.0 以降

第2章 SATELLITE 5 と SATELLITE 6 の比較

2.1. 設計および概念

2.1.1. Red Hat Satellite 5

Red Hat Satellite 5 はライフサイクル管理ツールで、大量のシステムをデプロイ、管理、監視する機能があります。Satellite 5 は、オンラインまたはオフラインモードで設定でき、オリジナルのプール型サブスクリプションアプローチを使用してクライアントシステムに Red Hat のソフトウェアを配布します。プール型サブスクリプションの概念は、クライアントが Red Hat Network Classic からエンタイトルメントを使用する方法と同様です。

機能および特徴

Satellite 5 でよく使われる機能としては、キックスタートファイルとアクティベーションキーを使用して多数のシステムをプロビジョニングし、予測可能な状態にインストールして設定する機能などが挙げられます。このプロビジョニングプロセスでは、システムは指定された組織やソフトウェア、設定チャンネルに関連付けられ、事前に定義されたシステムグループにシステムを追加します。管理者は、Satellite 5 のプロビジョニング機能を使用すると、システムの一貫性を保ちつつ、数千ものシステムをプロビジョニングできます。

また別の機能として、ローカルまたはリモートの環境にプロビジョニングした多数のシステムのソフトウェアや設定ファイルを管理する機能があります。Satellite 5 におけるソフトウェアおよび設定ファイルを管理する概念としてよく知られているものにチャンネルがあります。ソフトウェアと設定はすべてチャンネル経由で管理され、配布されており、ソフトウェアや設定コンテンツへのアクセスが必要なクライアントは、1 つ以上の関連チャンネルに関連付けられている必要があります。さらに、チャンネルをクローンする機能により、管理者は多くの企業で必要とされる開発環境や実稼働環境を構築できます。

Red Hat Satellite 5 を使用すると、サーバーまたはクライアントシステムが公共のインターネットにアクセスすることなく Red Hat Network の恩恵を受けられます。これにより、システムの信頼性、セキュリティー、パフォーマンスを最大化するために必要なツール、サービス、情報リポジトリーを一体化できます。

2.1.2. Red Hat Satellite 6

Red Hat Satellite 6 は、Red Hat のライフサイクル管理プラットフォームをさらに進化させた製品で、グローバルエンタープライズ用のシステムおよびコンテンツの管理を専用に行うツールとして、システム管理者の想定通りの機能を提供します。Satellite 6 は、Satellite 5 のお客様が求めるユースケースに対応するだけでなく、はるかに大規模な環境で、コンテンツを連合 (フェデレーション) し、プロビジョニング時の効果的なシステム管理を行い、ライフサイクル管理へのよりシンプルなアプローチを実現する機能を提供します。さらに Satellite 6 では、証明書ベースのエンタイトルメントや統合されたサブスクリプション管理への特有のアプローチをさらに進化させています。

2.1.3. 概念の比較

以下の表は、重要な概念と、その概念が Satellite 5 および Satellite 6 でどのように実装されているかを示します。

表2.1 Satellite 5 および Satellite 6 概念の比較

概念	説明	Satellite 5	Satellite 6
オープンソースプロジェクト	単一のプロジェクトアプローチとモジュール式アプローチ	Spacewalk	Foreman、Katello、Puppet、Candlepin、および Pulp
サブスクリプションの種類	プールベースまたはチャンネルベースと証明書ベース。サブスクリプション管理はここ何年かの間に、プールベースまたはチャンネルベースのアプローチから、より具体的な証明書ベースのアプローチへと改善されてきました。証明書ベースのサブスクリプション管理では、クライアントが使用するサブスクリプションに対して、全体的な制御が改善されました。	エンタイトルメント	サブスクリプション
サブスクリプションの方法 (または Satellite サブスクリプションの使用)	Satellite が Red Hat コンテンツを同期して配信できるようにする方法。証明書はインストール時にアクティベートされ、マニフェストはインストール後にアップロードされます。	証明書ファイル	マニフェストファイル
組織管理	Satellite 5 および 6 の両方に、複数組織の概念がありますが、Satellite 6 には、ロケーションのコンテキストを設定する機能があります。	組織	組織およびロケーション

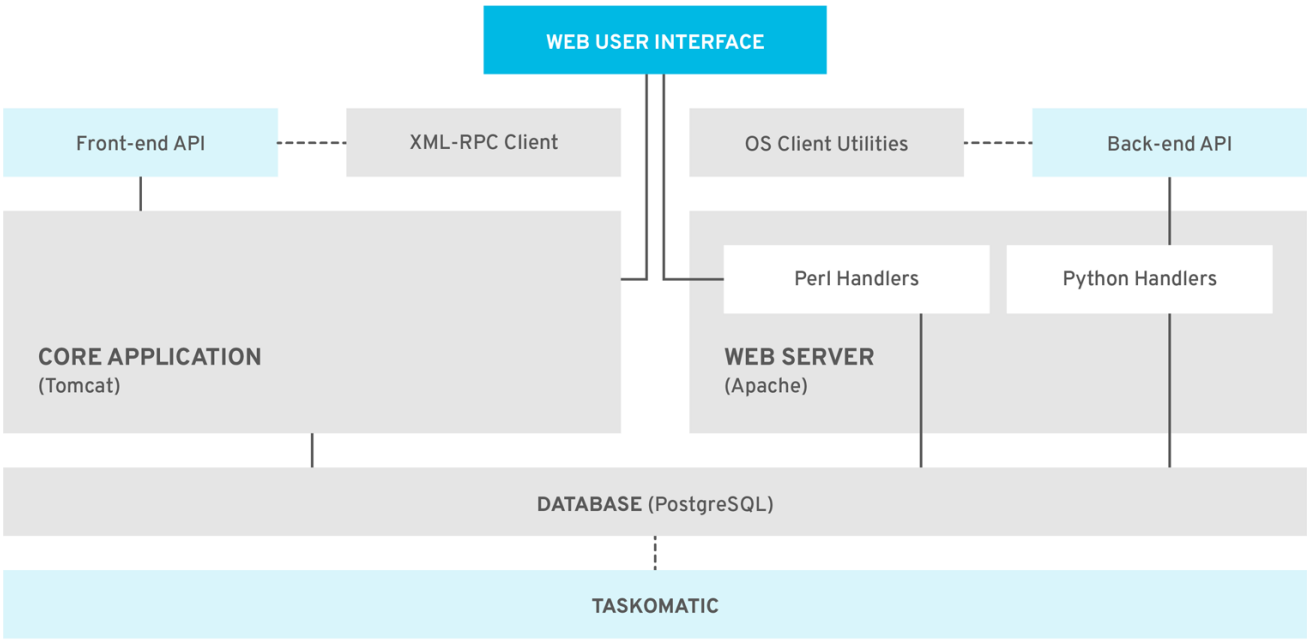
概念	説明	Satellite 5	Satellite 6
ソフトウェアおよび設定コンテンツ	チャンネルによる配信と環境ベースで公開してプロモートしたコンテンツビューによる配信。Satellite 6 のコンテンツビューには、環境に公開およびプロモートしたソフトウェアリポジトリーや設定モジュールで選択したものが表示されます。クライアントシステムは、環境を関連付けることでそのソフトウェアおよび設定を使用します。	ソフトウェアチャンネル	製品およびリポジトリー
設定		設定チャンネル	Puppet リポジトリー
Proxy サービス		Red Hat Satellite Proxy Server	Red Hat Satellite Capsule Server
コマンドラインツール		各種 CLI ツール	Hammer
仮想およびクラウドプロバイダー		KVM および Xen	OpenStack、Red Hat Enterprise Virtualization、KVM、VMware、EC2
データベースサポート		組み込み PostgreSQL、管理 PostgreSQL、外部 PostgreSQL、Oracle Database 10g Release 2 または 11g (Standard または Enterprise Edition)	6.0 用の組み込み PostgreSQL

2.2. システムアーキテクチャー

2.2.1. Red Hat Satellite 5

Red Hat Satellite 5 は、Spacewalk と呼ばれるオープンソースプロジェクトをベースにしており、以下のアーキテクチャーに用意されている複数の重要なコンポーネントで設定されています。

図2.1 Red Hat Satellite 5 システムアーキテクチャー



Web UI

Satellite Web UI は Apache Web サーバーを使用して、Satellite 操作の主なエントリーポイントを提供します。

フロントエンド API

フロントエンド API は、XML-RPC API を介して Satellite 5 と対話する機能を提供します。この API により、システム管理者は、反復タスクを実行するスクリプトの作成や、Satellite に関連するサードパーティーのアプリケーションの開発が可能になります。フロントエンド API は、XML-RPC を使用する Web UI 機能のほとんどを公開します。

バックエンド API

バクエンドは、複数のクライアントユーティリティー (**rhn_register**、**yum**) が接続する一連の API を提供します。これは文書化されておらず、クライアントユーティリティーでのみ使用されます。

Taskomatic

Taskomatic は、Red Hat Satellite 5 では独立したサービスで、さまざまな非同期ジョブ (セッションテーブルの削除や、エラータが公開された場合の通知メール送信など) を実行します。このジョブの多くは定期的に実行しますが、その実行頻度は調整できます。

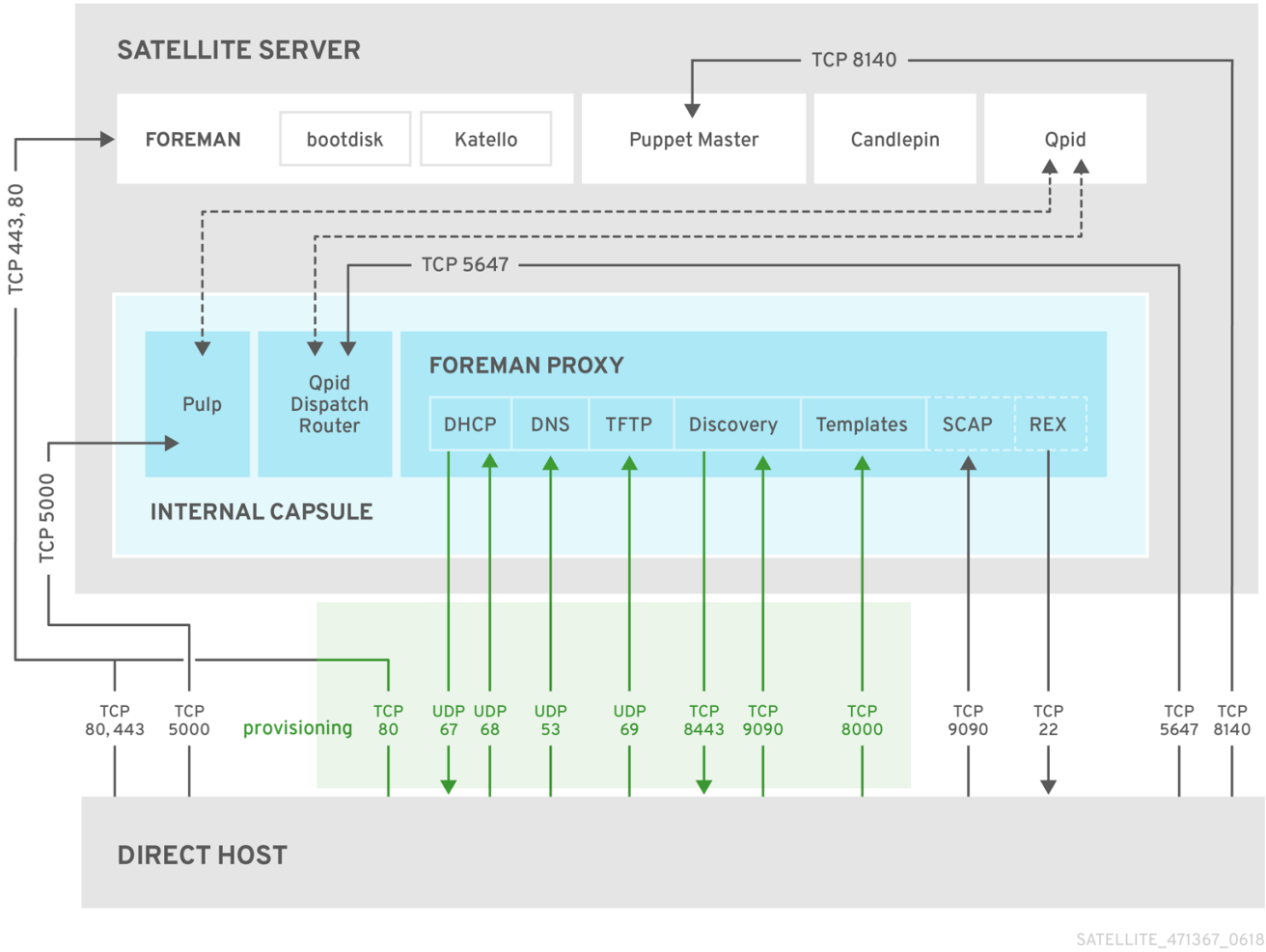
検索サーバー

Satellite には、スタンドアロンの検索サーバーが含まれており、数百もの項目をリストに挙げることなく、システム、パッケージ、エラータをすばやく見つけることができます。検索エンジンには、Apache の Lucene 検索エンジンライブラリーが使用されていますが、このライブラリーにより、多数のクエリー言語が使用され、より関連する検索結果を得ることができます。

2.2.2. Red Hat Satellite 6

Red Hat Satellite 6 は、以下のアーキテクチャーに用意されている複数のオープンソースプロジェクトをベースとしています。

図2.2 Red Hat Satellite 6 システムアーキテクチャー



Foreman

Foreman は、物理システムと仮想システムのプロビジョニングとライフサイクル管理に使用されるオープンソースのアプリケーションです。Foreman は、キックスタートや Puppet モジュールなどの各種の方法を使用して、これらのシステムを自動的に設定します。また、レポート、監査、およびトラブルシューティングに使用される履歴データを提供します。

Katello

Katello は、サブスクリプションとリポジトリーを管理するアプリケーションです。Katello を使用して Red Hat リポジトリーをサブスクライブし、コンテンツをダウンロードできます。コンテンツについては、複数の異なるバージョンを作成し、管理することが可能であり、コンテンツのバージョンは、ユーザーが定義するアプリケーションライフサイクルの各ステージ内にある特定のシステムに適用できます。

Candlepin

Candlepin は、サブスクリプションの管理を行う Katello 内のサービスです。

Pulp

Pulp は、リポジトリーおよびコンテンツの管理を行う Katello 内のサービスです。

Hammer

Hammer は、コマンドラインおよびシェルを提供する CLI ツールで、Web UI とほぼ同様の機能を提供します。

REST API

Red Hat Satellite 6 には REST ベースの API サービスが含まれます。システム管理者や開発者は、このサービスを使用して、カスタムスクリプトや 、Red Hat Satellite へのインターフェイスとなるサードパーティーアプリケーションを作成できます。

Capsule

Red Hat Satellite Capsule Server は、リポジトリーのストレージ、**DNS**、**DHCP**、および Puppet マスター設定など、Satellite の一部の主要機能のプロキシーとして動作します。各 Satellite Server には、統合された Capsule Server の各種サービスが含まれます。

Red Hat Satellite 6 は、x86_64 アーキテクチャーシステムにしかインストールできません。

2.3. コンテンツ管理

2.3.1. Red Hat Satellite 5

Red Hat Satellite 5 アーキテクチャーには、パッケージキャッシングのメカニズムで、Red Hat Satellite の帯域幅要件を低減し、カスタムパッケージのデプロイを可能にする Red Hat Satellite Proxy Server が同梱されています。Satellite Proxy は、クライアントシステムと Satellite Server を仲介する役目を果たします。

クライアントから見ると、Satellite Proxy と Satellite には違いがありません。ただし、Satellite Server から見ると、Satellite Proxy は特殊なタイプの Satellite クライアントになります。

Satellite Proxy サーバーは Satellite 5 専用の機能で、Satellite 6 では使用できません代わりに、Satellite 6 には、ほぼ同様の機能を提供する Capsule の概念が導入されています。

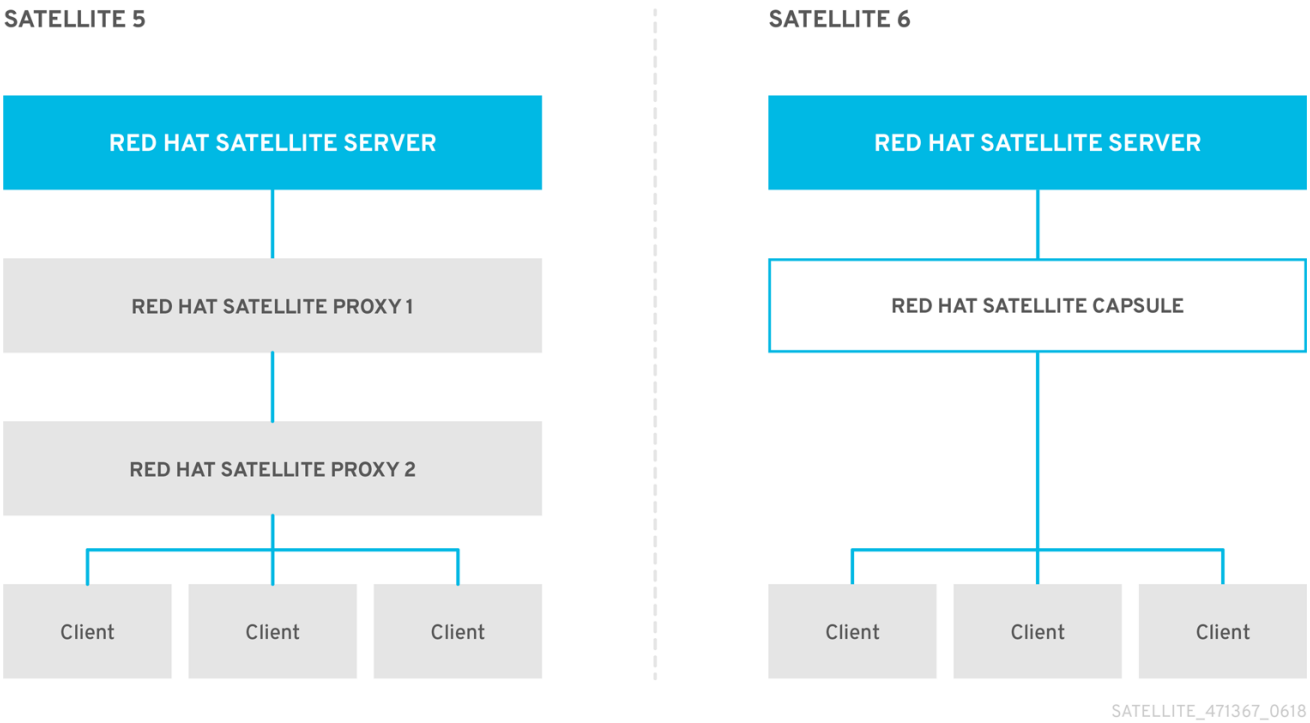
2.3.2. Red Hat Satellite 6

Satellite 6 アーキテクチャーには、Satellite 5 で Proxy サーバーが提供したのと同等レベルの機能を Satellite 6 に提供する Capsule Server が含まれます。

重要

Capsule Server は、Proxy サーバーと同じように階層を作ることはできません。以下の図でそれを説明します。

図2.3 Satellite 5 Proxy および Satellite 6 Capsule Server の比較



Satellite 6.0 とともに配信された Capsule Server の最初のリリースでは、以下の機能を提供します。

- ミラーリポジトリーコンテンツ (Pulp ノード)。コンテンツは、ホストで使用される前に Pulp ノードで使用されます。
- ミラー Puppet コンテンツ (Puppet マスター)
- DHCP、DNS、および TFTP を使用して、Identity Management (IdM) と統合します。

2.4. 切断されているコンテンツ管理

Satellite 5 と Satellite 6 との間の主な違いは、切断した (ネットワークに接続していない) コンテンツ管理のエリアになります。両バージョンの Satellite でプロビジョニングを行い、インターネットに直接接続しなくてもホストを同期することができますが、その方法は少々異なります。

2.4.1. Red Hat Satellite 5

Red Hat Satellite 5 は、**katello-disconnected** ユーティリティーと同期ホストを使用して、切断されているコンテンツの管理を行います。この場合、インターネット接続のある中間システムが同期ホストとして機能する必要があります。この同期ホストは、Satellite Server とは分離したネットワークに置かれます。

同期ホストは、Red Hat コンテンツ配信ネットワーク (CDN) からコンテンツをインポートします。その後、DVD、CD などのメディアや、外部ハードドライブにコンテンツをエクスポートしてから、ネットワークから切断されている Satellite Server に移行します。

2.4.2. Red Hat Satellite 6

Red Hat Satellite 6 では、インターネットを使用する 2 つ目の Satellite と、Inter-Satellite Synchronization (ISS) 機能を使用して、ネットワークに接続されていないコンテンツの管理を実現します。

ネットワークに接続している Satellite は、Red Hat Content Delivery Network (CDN) からコンテンツをインポートします。その後、DVD、CD などのメディアや、外部ハードドライブにコンテンツをエクスポートしてから、ネットワークから切断されている Satellite Server に移行します。この Inter-Satellite Synchronization (ISS) 機能により、完全更新または増分更新を作成できるようになります。詳細は **コンテンツ管理ガイド** の Satellite Server 間でのコンテンツ同期 を参照してください。

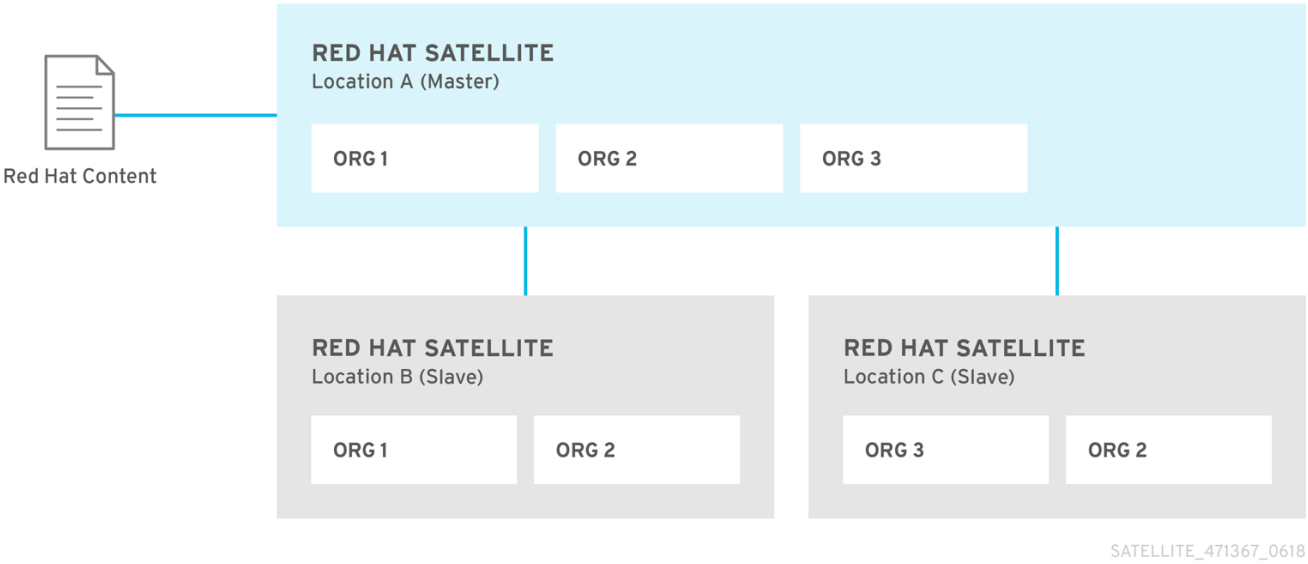
2.5. 組織の構造

2.5.1. Red Hat Satellite 5

Red Hat Satellite 5 は、複数の異なる組織にシステム、コンテンツ、および設定をまとめることができます。これは複数の部門 (たとえば Finance (財務)、Marketing (マーケティング)、および Sales (営業)) をかかえる企業には有用です。各組織は、独自の設定およびシステムプロファイルがあり、個別の管理 Satellite として機能します。ただし、Satellite は複数の組織間でコンテンツ (ソフトウェアチャネル) を共有できます。

Red Hat Satellite 5 には、複数の Satellite Server 間でコンテンツを同期する機能があります。これにより、管理者は、Satellite が地理的に離れていても、同じソフトウェアチャンネルを共有できます。たとえば、企業のオフィスの場所が 3 つに分かれていて、各オフィスで Satellite が必要になった場合に、選択した親となる Satellite Server からコンテンツを同期します。

図2.4 Red Hat Satellite 5 のトポロジーの例

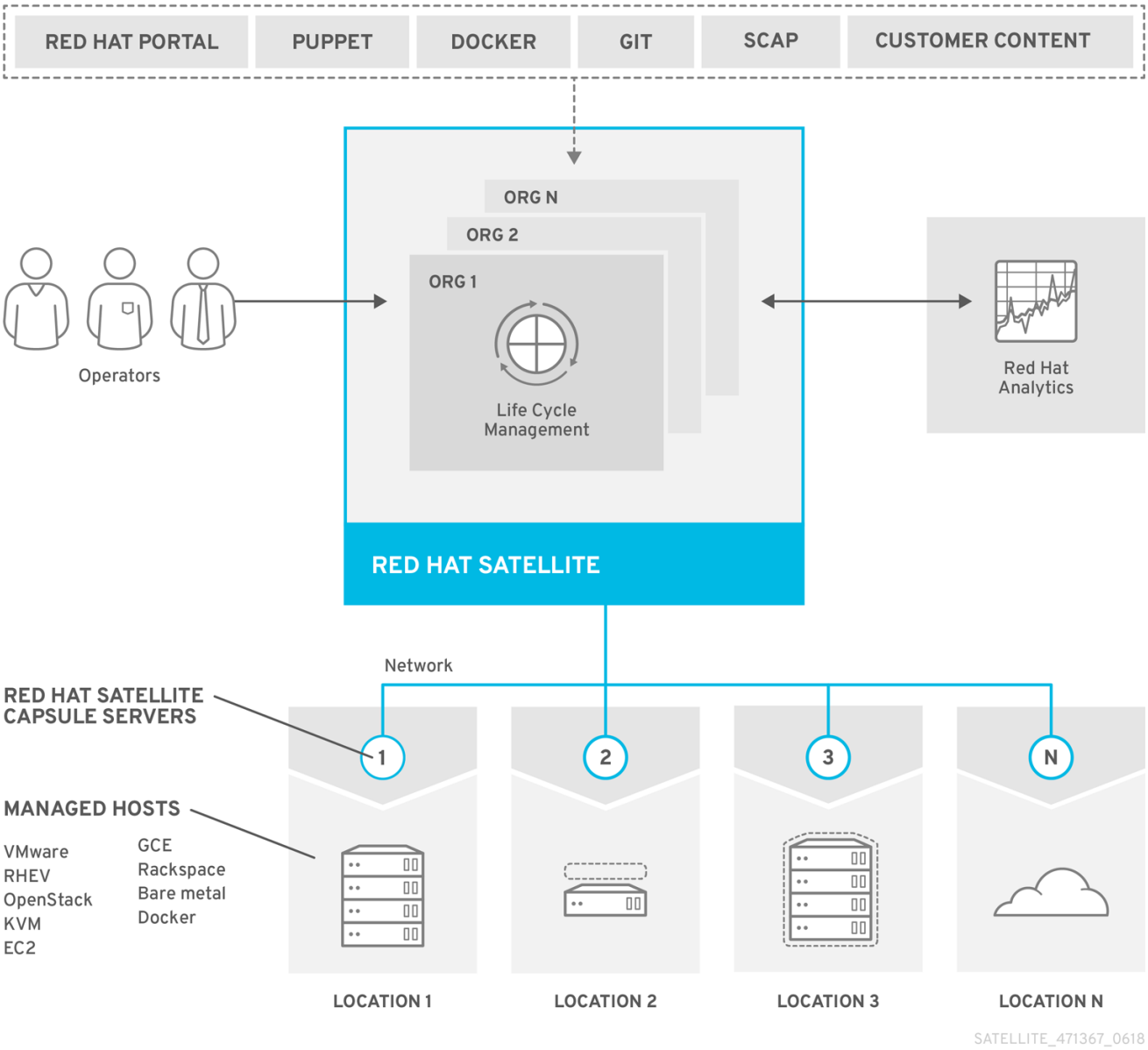


システム管理操作はすべて、登録している Satellite で行われます。

2.5.2. Red Hat Satellite 6

Red Hat Satellite 6 は、組織とロケーションの管理に対して統合的なアプローチを取ります。システム管理者は、1 台の Satellite Server に、組織とロケーションをそれぞれ複数定義します。たとえば、3 つの国 (米国、英国、および日本) に 3 つの組織 (Finance (財務)、Marketing (マーケティング)、および Sales (営業)) がある会社について考えてみましょう。この例では、Satellite Server がすべての地理的な場所 (ロケーション) にあるすべての組織を管理しているため、システムを管理するためのコンテキストを 9 つ作成します。さらに、ユーザーはロケーションを定義し、そのロケーションをネストして階層を作成できます。たとえば、Satellite 管理者が、米国のロケーションをさらにボストン、フェニックス、サンフランシスコなどの都市に分けるような場合です。

図2.5 Red Hat Satellite 6 のトポロジーの例



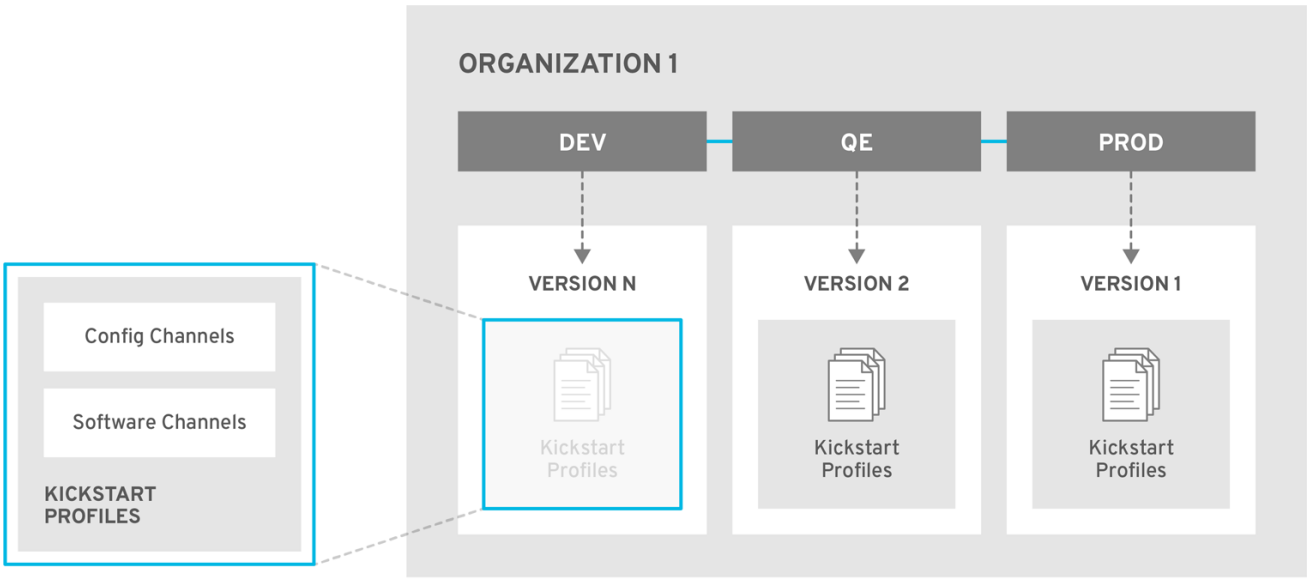
コンテンツと設定は、メインの Satellite Server と、特定のロケーションに割り当てられた Satellite Capsule との間で同期されますが、メインの Satellite Server が管理機能を保持します。

2.6. アプリケーションのライフサイクル

2.6.1. Red Hat Satellite 5

Red Hat Satellite 5 におけるアプリケーションのライフサイクルは、パスのステージを進んでいきます。たとえば、アプリケーションの場合は、開発からテストに進み、その後一般提供 (GA) と進みます。Red Hat Satellite 5 の各ステージでは、ライフサイクルのある特定の時点で、システムの定義を使用してシステムを管理します。Red Hat Satellite 5 のシステムの定義は、キックスタートプロファイルで使用されているソフトウェアチャンネルと設定チャンネルのセットになります。

図2.6 Red Hat Satellite 5 におけるアプリケーションのライフサイクル

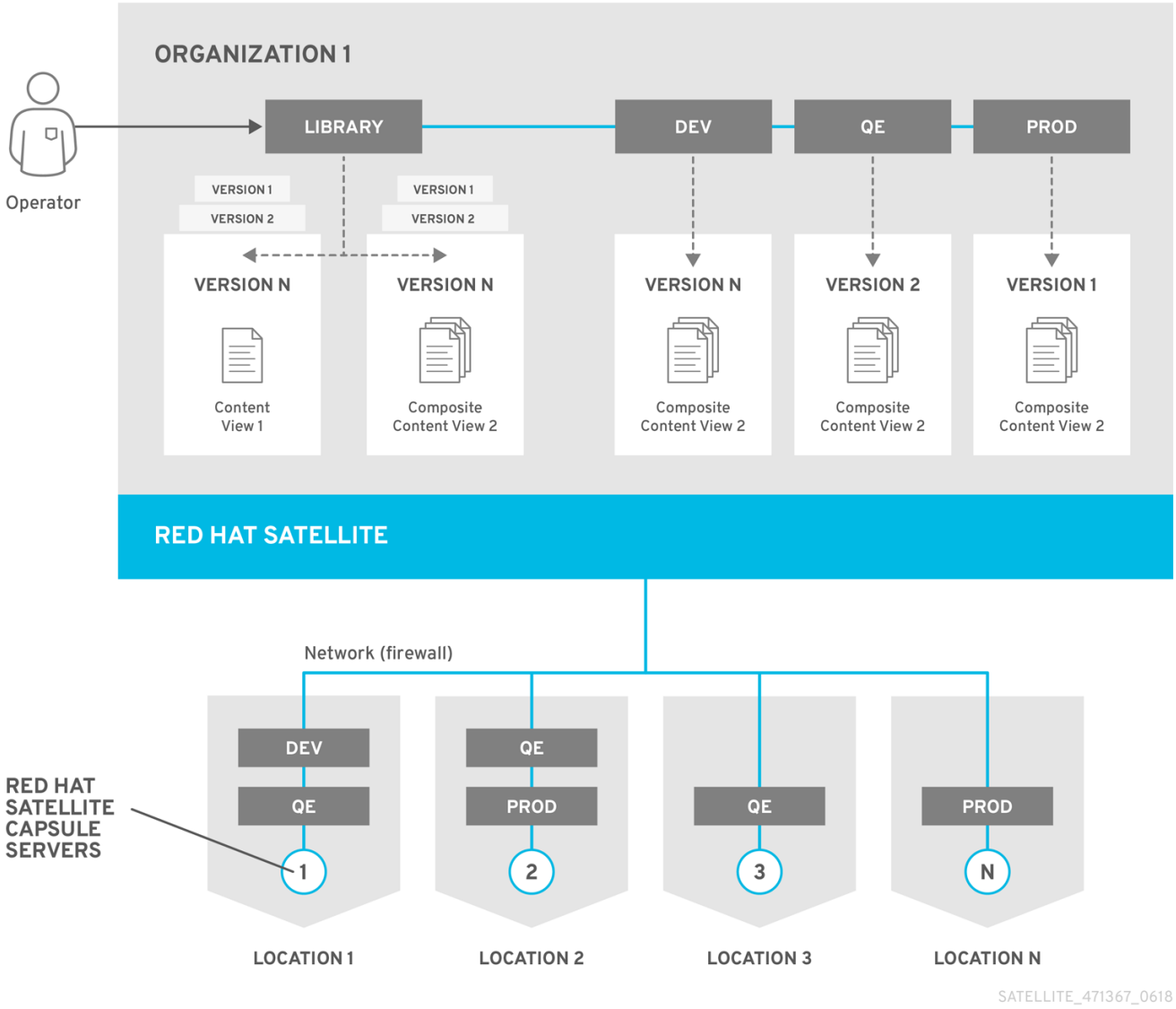


2.6.2. Red Hat Satellite 6

Red Hat Satellite 6 のアプリケーションライフサイクルは、ライフサイクル環境とコンテンツビューという、2 つの重要なコンポーネントに分かれています。

アプリケーションライフサイクルは、**ライフサイクル環境** に分けられ、それぞれがライフサイクルの各ステージを表します。このライフサイクルの環境は **環境パス** にリンクされます。コンテンツは、環境パスの次のライフサイクルステージにプロモートできます。たとえば、アプリケーションの特定のバージョンで環境を完了し、このバージョンをテスト環境にプロモートしたら、次のバージョンの開発を開始できます。

図2.7 4 つの環境を含む環境パス



コンテンツビューとは、マネージドのコンテンツセレクションで、この中にはオプションのフィルター機能とともに、1 つ以上の yum、Puppet リポジトリーが含まれます。フィルターを包括的または排他的に使用して、ライフサイクルの管理用にコンテンツのシステムビューを調整することができます。また、フィルターを使用してクライアントシステムで利用できるようにコンテンツをカスタマイズできます。コンテンツビューのバージョンは、アプリケーションライフサイクル期間に環境パス上でプロモートされます。公開されたコンテンツビューがライフサイクル環境と共に使用されます。

第3章 SATELLITE 5 から 6 への移行

本章では、ブートストラップスクリプトを使用して、以前の Satellite 5 から Satellite 6 へクライアントを移行し、移行プロセスの完了後に以前の Satellite インスタンスをリタイアさせる方法を説明します。

移行に使用するブートストラップスクリプトでは、コンテンツ登録、製品の証明書、および Puppet の設定を行います。ブートストラップスクリプトには、登録先 (RHN、Satellite 5、SAM、RHSM) や登録の有無に関係なく、Red Hat Enterprise Linux システムを Satellite 6 にサブスクライブするという利点があります。

ブートストラップスクリプトでは、以下のことができます。

- Satellite 5 から Satellite 6 にクライアントを移行する
- Satellite 6 インスタンスのクライアントを、別のインスタンスに移行する
- 新規 Red Hat Enterprise Linux システムを Satellite 6 に登録する

3.1. 前提条件

- s390 システムに Satellite 5.8 を実行している場合は、x86_64 アーキテクチャーシステムに新規の Satellite 6 をビルドする必要があります。Satellite 6 は、s390 システムでは実行できません。
- 完全に機能する、最新の Satellite 6 で開始するようにしてください。
- 移行プロセスを開始する前に、Satellite 6 インスタンスを必要な Red Hat コンテンツと同期させる必要があります。
- ホストにアクティベーションキーを設定しておきます。アクティベーションキーの設定方法は **コンテンツ管理ガイド** の アクティベーションキーの管理 を参照してください。
- 任意で、環境に適したホストコレクションを設定します。ホストコレクションの作成方法は **ホストの管理** の ホストコレクションの設定 を参照してください。

3.2. 移行のワークフロー

ワークフローには、以下の手順が含まれます。

1. クライアントへのブートストラップスクリプトのインストール
2. Red Hat Enterprise Linux System の移行
    a. クライアントシステムでブートストラップスクリプトを実行する
    b. 管理者として Web UI で Puppet 証明書を承認する
    c. クライアントシステムで移行プロセスを完了したことを確認する

3.3. 移行の実行

Satellite 6 では、ブートストラップスクリプトを使用して、以前の Satellite 5 インスタンスから既存のクライアントを移行します。

ブートストラップスクリプトは、コンテンツの登録、製品の証明書、および Puppet 設定を処理します。

3.3.1. ブートストラップスクリプトのインストール

ブートストラップスクリプトのパッケージである **katello-client-bootstrap** は、デフォルトで Satellite Server のベースシステムにインストールされ、スクリプトは **/var/www/html/pub/** ディレクトリーにインストールされ、クライアントで使用できるようになります。これには、以下の形式の URL からアクセスできます。

```
satellite.example.com/pub/bootstrap.py
```

このスクリプトには readme ファイルがあります。Satellite CLI でこのファイルを表示するには、以下のコマンドを実行します。

```
$ less /usr/share/doc/katello-client-bootstrap-version/README.md
```

ブートストラップのクライアントへのインストール

スクリプトが必要になるのは一度だけで、さらには **root** ユーザー向けのものなので、これを **/root** に保存して使用後に削除します。**root** として、以下のようにクライアントにブートストラップスクリプトをインストールします。

1. **root** ディレクトリーにいることを確認します。

   ```
   # cd
   ```

2. スクリプトをダウンロードします。

   ```
   # curl -O http://satellite.example.com/pub/bootstrap.py
   ```

   これにより、スクリプトが現行ディレクトリーにインストールされます。

3. スクリプトを実行可能にします。

   ```
   # chmod +x bootstrap.py
   ```

4. スクリプトが実行できることを確認するには、以下のように使用ステートメントを表示します。

   ```
   # ./bootstrap.py -h
   ```

5. 移行プロセスが完了したら、スクリプトを削除できます。

   ```
   # cd
   # rm bootstrap.py
   ```

3.3.2. Red Hat Enterprise Linux 6 システムの移行

Red Hat Enterprise Linux 6 システムの移行

1. ご使用の環境に適した値を使用して、以下のようにブートストラップコマンドを出力します。

--server オプションの場合は、Satellite Server または Capsule Server の FQDN 名を指定します。**--location**、**--organization**、および **--hostgroup** の場合は、オプションへの引数として、ラベルではなく引用符で囲まれた名前を使用します。

```
# bootstrap.py --login=admin \
--server satellite6.example.com \
--location="Example Location" \
--organization="Example Organization" \
--hostgroup="Example Host Group" \
--activationkey=activation_key
```

スクリプトは、**--login** オプションを使用して入力した Satellite ユーザー名に対応するパスワードの入力を求めるプロンプトを出します。

2. スクリプトが実行し、進捗の通知が **stdout** に送信されます。Puppet 証明書の承認を求めるプロンプトを出しているのを確認します。以下に例を示します。

```
[NOTIFICATION], [2016-04-26 10:16:00], [Visit the UI and approve this certificate via Infrastructure->Capsules]
[NOTIFICATION], [2016-04-26 10:16:00], [if auto-signing is disabled]
[RUNNING], [2016-04-26 10:16:00], [/usr/bin/puppet agent --test --noop --tags no_such_tag --waitforcert 10]
```

3. クライアントは、管理者が Puppet 証明書を承認するまで待機します。以下のように Puppet 証明書に署名します。

   a. Web UI で、**インフラストラクチャー** > **Capsules** に移動します。

   b. **--server** オプションで指定した FQDN に対応する Capsule 名の右側にある **証明書** を選択します。

   c. **アクション** コラムで、**署名** を選択して、クライアントの Puppet 証明書を承認します。

   d. クライアントに戻り、残りのブートストラップ処理が完了するのを確認します。

4. Web UI で **ホスト** > **すべてのホスト** に移動して、そのクライアントが、適切なホストグループに接続していることを確認します。

ブートストラップスクリプトの使用方法の詳細は **ホストの管理** の ブートストラップスクリプトを使用したホストの Red Hat Satellite への登録 を参照してください。

3.4. 使用しなくなった SATELLITE 5 SERVER のリタイア

本セクションでは、Red Hat Satellite 5 Server から新規に Satellite 6 Server への完全移行を実行した後に、不要となった Satellite 5 Server をリタイアする方法について説明します。Satellite 5 から新しい RHSM システムに移行した場合は、「新しい Satellite 6 へのサブスクリプションの移行」 に進んでください。

Satellite 5 を新規の RHSM システムに移行していない場合は、**コンテンツ管理ガイド** の サブスクリプションの管理 の説明に従ってマニフェストを新規作成し、新しい Satellite 6 に追加します。オプションで、以下の Python スクリプトを実行して、以前の Satellite のプロファイルを RHN から削除します。

```
#!/usr/bin/env python

import getpass
```

```
import os
import sys
import libxml2
import xmlrpclib
from optparse import OptionParser

DEFAULT_SERVERFQDN="xmlrpc.rhn.redhat.com"
parser = OptionParser()
parser.add_option("-l", "--login", dest="login", help="Login user for RHN Satellite/Hosted",
metavar="LOGIN")
parser.add_option("-p", "--password", dest="password", help="Password for specified user. Will
prompt if omitted", metavar="PASSWORD")
parser.add_option("-s", "--server", dest="serverfqdn", help="FQDN of satellite server - omit https://
(default: %s)" % DEFAULT_SERVERFQDN, metavar="SERVERFQDN",
default=DEFAULT_SERVERFQDN)
(options, args) = parser.parse_args()

if not options.login:
    print "Must specify login option.  See usage:"
    parser.print_help()
    print "\nExample usage: ./decommissionServer.py -l admin -p password"
    sys.exit(1)
else:
    login = options.login
    password = options.password
    serverfqdn = options.serverfqdn

if not password: password = getpass.getpass("%s's password:" % login)

SATELLITE_URL = "https://%s/rpc/api" % serverfqdn
SATELLITE_LOGIN = login
SATELLITE_PASSWORD = password
SYSTEMID_FILE = "/etc/sysconfig/rhn/systemid"

# Check For root
# Have to be root to open the SYSTEMID_FILE which is chmod'd 0600
if os.getuid() != 0:
    print "This script requires root-level access to run."
    sys.exit(1)

# Log into Satellite and get an authentication token
client = xmlrpclib.Server(SATELLITE_URL, verbose=0)
key = client.auth.login(SATELLITE_LOGIN, SATELLITE_PASSWORD)

# Parse /etc/sysconfig/rhn/systemid to ge the system ID
# Use the systemid and delete the systems RHN profile
parsed_file = libxml2.parseDoc(file(SYSTEMID_FILE).read())
systemid = parsed_file.xpathEval('string(//member[* ="system_id"]/value/string)').split('-')[1]
print systemid
try:
    client.system.deleteSystems(key,int(systemid))
    print "The system with SystemID " + systemid + " was successfully deleted"
except xmlrpclib.Fault, e:
    print "XMLRPC fault \n\t%s" % e
```

```
# Logout of Satellite
client.auth.logout(key)
```

3.4.1. 新しい Satellite 6 へのサブスクリプションの移行

本セクションは、使用していた Satellite 5 Server から新しい Satellite 6 Server へサブスクリプションを移行する方法を説明します。

1. 新しいサブスクリプションを設定します。
   a. カスタマーポータル にログインし、**サブスクリプション** に移動します。
   b. **インベントリー** に移動して **Satellite 組織** をクリックします。
   c. 移行する Satellite 5 Server の名前の左側にあるチェックボックスを選択し、**選択した項目を削除** をクリックします。警告ボックスが表示されるので **削除** をクリックしてユニットを削除します。
   d. 新しい Satellite 6 Server の名前をクリックし、移行サブスクリプション名の左側にあるチェックボックスを選択します。**選択項目の削除** をクリックします。警告ボックスが表示されるので、**削除** をクリックしてサブスクリプションを削除します。
   e. **サブスクリプションのアタッチ** をクリックし、適切な Satellite サブスクリプションを選択し、**選択項目のアタッチ** をクリックします。
   f. 新しい Satellite 6 Server で以下のコマンドを実行し、サブスクリプションマネージャーを更新します。

```
# subscription-manager refresh
```

2. 新しい Satellite 6 Server でリポジトリーを有効にします。
   a. 移行する Satellite 5 Server で、以下のコマンドを使用して、有効にしたチャンネルを表示します。

```
# spacewalk-channel -l
```

   b. 新しい Satellite 6 Server で、以下のコマンドを使用して、組織で有効になっているリポジトリーをリスト表示します。

```
# hammer product list --organization "organization_name"
```

   c. 新しい Satellite 6 Server で、必要なリポジトリーがすべて有効になるように設定します。
   Web UI をご利用の場合

   Content → Red Hat Repositories に移動して、有効にするリポジトリーを選択します。

   CLI をご利用の場合

   名前または ID 番号を使用してリポジトリーを有効にします。任意で、リリース番号とベースアーキテクチャーを追加できます。

```
# hammer repository-set enable \
--product "product_name" \
```

```
--name "repository_name" \
--organization "org_name" \
--releasever "" \
--basearch "x86_64"
```

第4章 SATELLITE 6 API への移行

Red Hat Satellite 6 が Satellite 5 と大きく違う点の 1 つに API が挙げられます。Satellite 5 は XMLRPC ベースの API を使用しますが、Satellite 6 は REST ベースの API を使用します。この基本的な違いにより、Satellite 6 REST API を使用する前に、Satellite 5 API と統合した既存のスクリプトまたはツールを確認するか、少なくても部分的に書き直す必要があります。

本セクションは、各製品内で同じユースケースを実現するための比較を紹介します。このチュートリアルは、特定のプログラミング言語を対象にしておらず、どのスクリプトも HTTPS で安全を確保していません。ここでは、Satellite 6 API への移行を開始するために Satellite 5 API スクリプトをメンテナンスするところから説明します。

重要

Satellite 6 移行ツールでは、Satellite 5 API またはスクリプトは Satellite 6 に移行されません。移行プロセスを開始するには、このセクションから始めてください。

詳しい情報

API の詳細は **API ガイド** を参照してください。お使いの Satellite Server では、以下の場所で参照できます。

- Satellite 6: https://satellite6.example.com/apidoc/v2.html
- Satellite 5: https://satellite5.example.com/rpc/api

4.1. API スクリプトの例

本セクションでは、例を挙げて、以下の点を説明します。

1. Red Hat Satellite のシステムおよびホスト
    a. 認証し、ログインするユーザーに利用可能なシステム (Satellite 5) またはホスト (Satellite 6) のリストを要求する
2. ユーザーとロール
    a. 認証し、ログインしたユーザーに表示可能なユーザーのリストを要求する
    b. **example** ユーザーが存在する場合は削除する
    c. **example** ユーザーを新規作成し、そのロールを **Administrator** に設定する

本セクションでは、同じ結果を実現するための方法を全部で 5 つ紹介します。そのうち 2 つは Satellite 5 用で、残りの 3 つは Satellite 6 用です。

- Satellite 5 の Python スクリプト
- Satellite 6 Python スクリプト
- Satellite 6 の Ruby スクリプト
- Satellite 5 の **spacecmd** の例
- Satellite 6 の **hammer** の例

注記

spacecmd コマンドは、Satellite 5.6 ではサポートされていません。Satellite 5.7 ではサポートされているため、完全を期すためにここに示します。

全部で 10 個の例が提供されます。最初に、システムおよびホストをリストを表示する簡単なユースケースを紹介し、その後、ユーザーやロールなどの複雑なユースケースを紹介します。

4.1.1. システムおよびホストのリスト表示

本セクションでは、例を挙げて、ログインしたユーザーに利用可能なシステムおよびホストのリストを表示する方法を説明します。

Satellite 5 で Python を使用して利用可能なシステムをリスト表示

この例では、Python スクリプトを使用して Satellite 5 に接続し、認証し、利用可能なシステムのリストを取得します。

```
#!/usr/bin/python
import xmlrpclib

# Define Satellite location and login details
SATELLITE_URL = "http://localhost/rpc/api"
SATELLITE_LOGIN = "admin"
SATELLITE_PASSWORD = "password"

client = xmlrpclib.Server(SATELLITE_URL, verbose=0)

# Authenticate and get session key
key = client.auth.login(SATELLITE_LOGIN, SATELLITE_PASSWORD)

# Get list of systems available to user
list = client.system.listSystems(key)
for system in list:
  print system.get('id')
  print system.get('name')

# Logout
client.auth.logout(key)
```

Satellite 6 で Python を使用して利用可能なホストをリスト表示

この例では、Python スクリプトを使用して Satellite 6 に接続し、認証し、利用可能なシステムのリストを取得します。

```
#!/usr/bin/python
import json
import requests

# Define Satellite location and login details
SAT_API = "https://localhost/api/v2/"
USERNAME = "admin"
PASSWORD = "changeme"
SSL_VERIFY = False
```

```
def get_json(location):
    """
    Performs a GET using the passed URL location
    """

    r = requests.get(location, auth=(USERNAME, PASSWORD), verify=SSL_VERIFY)

    return r.json()

def main():
    # List all hosts available to the user
    hosts = get_json(SAT_API + "hosts/")

    # Pretty Print the returned JSON of Hosts
    print json.dumps(hosts, sort_keys=True, indent=4)

if __name__ == "__main__":

    main()
```

Satellite 6 で Ruby を使用して利用可能なホストをリスト表示

この例では、Ruby スクリプトを使用して Satellite 6 に接続し、認証し、利用可能なシステムのリストを取得します。

```
#!/usr/bin/env ruby
require 'json'
require 'rest-client'

url = 'https://localhost/api/v2/'
$username = 'admin'
$password = 'changeme'

def get_json(location)
    response = RestClient::Request.new(
        :method => :get,
        :url => location,
        :user => $username,
        :password => $password,
        :headers => { :accept => :json,
        :content_type => :json }
    ).execute
    results = JSON.parse(response.to_str)
end

hosts = get_json(url+"hosts/")
#puts JSON.pretty_generate(hosts)

puts "Hosts within Satellite are:"
hosts['results'].each do |name|
     puts name['name']
end

exit()
```

Satellite 5 でコマンドラインを使用して利用可能なシステムをリスト表示

Satellite 5 で以下のコマンドを使用して、利用なシステムをリスト表示します。

```
# spacecmd -u username -p password system_list
```

セッションは以下のようになります。

```
# spacecmd -u admin -p password system_list

INFO: Spacewalk Username: admin
INFO: Connected to https://localhost/rpc/api as admin
test_02.example.com
```

Satellite 6 でコマンドラインを使用して利用可能なホストをリスト表示

Satellite 6 で以下のコマンドを使用して、利用可能なホストをリスト表示します。

```
# hammer host list
```

セッションは以下のようになります。

```
# hammer host list

[Foreman] password for admin:
---|-----------------|------------------|------------|--------------|------------------
ID | NAME            | OPERATING SYSTEM  | HOST GROUP | IP           | MAC
---|-----------------|------------------|------------|--------------|------------------
1  | test.example.com | RHEL Server 6.5  |            | 10.34.34.235 | e4:1f:13:6b:ed:0c
```

4.1.2. ユーザーの削除および作成

本セクションでは、ユーザーの検索、作成、削除する方法を説明します。

Satellite 5 で Python を使用してユーザーを管理

この例では、Python スクリプトを使用して Satellite 5 Server に接続し、認証する方法を説明します。ここでは、特定ユーザー (**example**) を検索し、そのユーザーが存在する場合は削除して再作成し、管理者権限を付与する方法を説明します。

```
#!/usr/bin/python
import xmlrpclib

# Define Satellite location and login details
SATELLITE_URL = "http://localhost/rpc/api"
SATELLITE_LOGIN = "admin"
SATELLITE_PASSWORD = "password"

client = xmlrpclib.Server(SATELLITE_URL, verbose=0)

# Authenticate and get session key
key = client.auth.login(SATELLITE_LOGIN, SATELLITE_PASSWORD)

# Get list of users
list = client.user.list_users(key)
print "Existing users in Satellite:"
```

```
for user in list:
  print user.get('login')

# Look for user example and if found, delete the user
for user in list:
  if user.get('login') == 'example':
     deleteuser = client.user.delete(key, 'example')
     if deleteuser == 1:
        print "User example deleted"

# Create a user called example
createuser = client.user.create(key, 'example', 'password', 'Example', 'User', "root@localhost")
if createuser == 1:
   print "User example created"
   # Admin Org Admin role to the example user
   adminrole = client.user.addRole(key, 'example', 'org_admin')
   if adminrole == 1:
      print "Made example an Org Admin"

# Logout
client.auth.logout(key)
```

Satellite 6 での Python を使用したユーザー管理

ここでは、先ほどの例と同じタスクを行います。つまり、Python スクリプトを使用して Satellite 6 Server サーバーに接続し、認証する方法を説明します。特定のユーザーを検索し、そのユーザーが存在する場合は削除して再作成し、管理者権限を付与する方法を説明します。

```
#!/usr/bin/python
import json
import requests

# Define Satellite location and login details
SAT_API = "https://localhost/api/v2/"
POST_HEADERS = {'content-type': 'application/json'}
USERNAME = "admin"
PASSWORD = "changeme"
SSL_VERIFY = False

def get_json(location):
   """
   Performs a GET using the passed URL location
   """
   r = requests.get(location, auth=(USERNAME, PASSWORD), verify=SSL_VERIFY)

   return r.json()

def post_json(location, json_data):
   """
   Performs a POST and passes the data to the URL location
   """
   result = requests.post(
      location,
      data=json_data,
      auth=(USERNAME, PASSWORD),
      verify=SSL_VERIFY,
```

```
        headers=POST_HEADERS)

    return result.json()

def delete_json(location):
    """
    Performs a DELETE and passes the id to the URL location
    """
    result = requests.delete(
        location,
        auth=(USERNAME, PASSWORD),
        verify=SSL_VERIFY)

    return result.json()

def main():
    # List all users within the Satellite
    users = get_json(SAT_API + "users/")

    #print json.dumps(users, indent=4)
    print "Users known are:"
    for login in users['results']:
        print login['login']

    # Look for user example and if found, delete the user
    for delete in users['results']:
        if delete['login'] == 'example':
            id = delete ['id']
            id = str(id)

            delete = delete_json(SAT_API + "/users/" + id)
            #print json.dumps(delete, indent=4)
            print "User example deleted"

    # Create a user called example as admin role
    createuser = post_json(SAT_API + "/users/", json.dumps({ "mail": "root@localhost", "firstname":
"Example", "lastname": "User", "login": "example", "password": "redhat", "admin": 'true',
"auth_source_id": 1 }))
    #print json.dumps(createuser, indent=4)
    print "Admin user example created"

if __name__ == "__main__":
    main()
```

Satellite 6 での Ruby を使用したユーザー管理

次に、Ruby を使用して、先ほどの例と同じタスクを実行します。

```
#!/usr/bin/env ruby
require 'json'
require 'rest-client'

url = 'https://localhost/api/v2/'
$username = 'admin'
$password = 'changeme'
```

```
def get_json(location)
    response = RestClient::Request.new(
        :method => :get,
        :url => location,
        :user => $username,
        :password => $password,
        :headers => { :accept => :json,
        :content_type => :json }
    ).execute
    results = JSON.parse(response.to_str)
end

def post_json(location, json_data)
    response = RestClient::Request.new(
        :method => :post,
        :url => location,
        :user => $username,
        :password => $password,
        :headers => { :accept => :json,
        :content_type => :json},
        :payload => json_data
    ).execute
    results = JSON.parse(response.to_str)
end

def delete_json(location)
    response = RestClient::Request.new(
        :method => :delete,
        :url => location,
        :user => $username,
        :password => $password,
        :headers => { :accept => :json,
        :content_type => :json }
    ).execute
    results = JSON.parse(response.to_str)
end

# List all users within the Satellite
users = get_json(url+"users/")

#puts JSON.pretty_generate(users)
puts "Users known are:"
users['results'].each do |name|
    puts name['login']
end

# Look for user example and if found, delete the user
users['results'].each do |name|
    if name['login'] == 'example'
        id = name['id']
        delete = delete_json(url+"users/"+id.to_s)
        #puts JSON.pretty_generate(delete)
        puts "User example deleted"
    end
end
```

```
# Create a user called example as admin role
data = JSON.generate({ :mail => "root@localhost", :firstname => "Example", :lastname => "User",
:login => "example", :password => "password", :admin => 'true', :auth_source_id => 1})
createuser = post_json(url+"users/", data)

#puts JSON.pretty_generate(createuser)
puts "Admin user example created"

exit()
```

Satellite 5 でコマンドラインを使用してユーザーを管理

ここでは、**spacecmd** コマンドを使用して、上の例と同じタスクを実行します。

```
# spacecmd -u admin -p password user_list
# spacecmd -u admin -p password user_delete example
# spacecmd -u admin -p password user_create
# spacecmd -u admin -p password user_addrole example org_admin
```

セッションは以下のようになります。

```
# spacecmd -u admin -p password user_list
INFO: Spacewalk Username: admin
INFO: Connected to https://localhost/rpc/api as admin
admin
example

# spacecmd -u admin -p password user_delete example
INFO: Spacewalk Username: admin
INFO: Connected to https://localhost/rpc/api as admin

Delete this user [y/N]: y

# spacecmd -u admin -p password user_list
INFO: Spacewalk Username: admin
INFO: Connected to https://localhost/rpc/api as admin
admin

# spacecmd -u admin -p password user_create
INFO: Spacewalk Username: admin
INFO: Connected to https://localhost/rpc/api as admin
Username: example
First Name: Example
Last Name: User
Email: root@localhost
PAM Authentication [y/N]: n
Password:
Repeat Password:

# spacecmd -u admin -p password user_list
INFO: Spacewalk Username: admin
INFO: Connected to https://localhost/rpc/api as admin
admin
example
```

```
# spacecmd -u admin -p password user_addrole example org_admin
INFO: Spacewalk Username: admin
INFO: Connected to https://localhost/rpc/api as admin

# spacecmd -u admin -p password user_details example
INFO: Spacewalk Username: admin
INFO: Connected to https://localhost/rpc/api as admin
Username:      example
First Name:    Example
Last Name:     User
Email Address: root@localhost
Organization:  MY ORG
Last Login:
Created:       8/19/14 8:42:52 AM EDT
Enabled:       True

Roles
-----
activation_key_admin
channel_admin
config_admin
monitoring_admin
org_admin
system_group_admin
```

Satellite 6 でコマンドラインを使用してユーザーを管理

ここでは、 **hammer** コマンドを使用して、上の例と同じタスクを実行します。

```
# hammer shell
> user list
> user delete --id 4 --login example
> user create --admin true --firstname Example --lastname User --login example --mail
root@localhost --password redhat --auth-source-id 1
```

セッションは以下のようになります。

```
# hammer shell
[Foreman] password for admin:
Welcome to the hammer interactive shell
Type 'help' for usage information

hammer> user list
---|---------|--------------|---------------
ID | LOGIN   | NAME         | EMAIL
---|---------|--------------|---------------
4  | example | Example User | root@localhost
3  | admin   | Admin User   | root@localhost
---|---------|--------------|---------------

hammer> user delete --id 4 --login example
User deleted
hammer> user list
---|-------|------------|---------------
ID | LOGIN | NAME       | EMAIL
```

```
---|-------|------------|---------------
3  | admin | Admin User | root@localhost
---|-------|------------|---------------

hammer> user create --admin true --firstname Example --lastname User --login example --mail
root@localhost --password redhat --auth-source-id 1
User created
hammer> user list
---|----------|--------------|---------------
ID | LOGIN    | NAME         | EMAIL
---|----------|--------------|---------------
3  | admin    | Admin User   | root@localhost
5  | example  | Example User | root@localhost
---|----------|--------------|---------------
hammer>
```

4.2. SATELLITE 6 API のヒント

本セクションでは、Satellite 6 API での体験を最適化するための一般的なヒントを紹介します。

Satellite 6 API の閲覧

Satellite 6 Web UI にログインしている場合は、GET 要求のデフォルトの結果を /api/v2/<API-NAME> で確認できます。例を示します。

- https://satellite6.example.com/api/v2/users/
- https://satellite6.example.com/api/v2/users/3

コマンドラインで Satellite 6 API 要求の使用

curl コマンドを使用して、Satellite 6 API と対話ができます。例を示します。

例4.1 GET 要求の例

Satellite 6 の組織、ホスト、ユーザーのリストを表示する GET 要求の例

```
# SATUSER=admin
# SATPASS='changeme'
# SATURL="https://localhost"

# curl -k -u $SATUSER:$SATPASS -X GET -H \
'Accept: application/json' $SATURL/api/v2/organizations | json_reformat
# curl -k -u $SATUSER:$SATPASS -X GET -H \
'Accept: application/json' $SATURL/api/v2/hosts | json_reformat
# curl -k -u $SATUSER:$SATPASS -X GET -H \
'Accept: application/json' $SATURL/api/v2/users | json_reformat
# curl -k -u $SATUSER:$SATPASS -X GET -H \
'Accept: application/json' $SATURL/api/v2/users/3 | json_reformat
```

例4.2 DELETE 要求の例

上のコマンドで得られた結果から、 ID が 9 のユーザーを削除します。

```
# curl -k -u $SATUSER:$SATPASS -X DELETE -H \
'Accept: application/json' $SATURL/api/v2/users/9 | json_reformat
```

例4.3 POST 要求の例

example という名前の新規ユーザーを作成する POST 要求の例。**true** フラグを渡して、ユーザーに管理者権限を付与します。

```
# curl -k -u $SATUSER:$SATPASS -X POST \
-d '{ "mail": "root@localhost", "firstname": "Example", \
"lastname": "User", "login": "example", "password": "redhat", \
"admin": 'true', "auth_source_id": 1 }' \
-H 'Accept: application/json' \
-H 'Content-Type: application/json' \
$SATURL/api/v2/users | json_reformat
```

例4.4 PUT 要求の例

ID が 10 の **example** ユーザーのメールアドレスを **example@localhost** に変更する PUT 要求の例

```
# curl -k -u $SATUSER:$SATPASS -X PUT \
-d '{ "id": 10, "mail": "example@localhost" }' \
-H 'Accept: application/json' \
-H 'Content-Type: application/json' \
$SATURL/api/v2/users/10 | json_reformat
```

付録A 用語集

以下は、本書全体で使用されている用語です。これらの用語を十分に理解しておくと、Red Hat Satellite 6 を理解するのに役立ちます。

Activation Key (アクティベーションキー)

登録時のアクションを制御するためにキックスタートファイルで使用される登録トークンです。これは Red Hat Satellite 5 のアクティベーションキーと似ていますが、登録後に Puppet でパッケージと設定管理を制御するのに必要な機能のサブセットを提供します。

Application Life Cycle Environment (アプリケーションのライフサイクル環境)

アプリケーションライフサイクル環境は、ソフトウェア開発ライフサイクル (SDLC) のプロモーションパスにおけるステップまたはステージを表します。プロモーションパスは開発パスとしても知られています。パッケージや Puppet モジュールなどのコンテンツは、コンテンツビューの公開およびプロモートにより、ライフサイクル環境間を移動します。すべてのコンテンツビューにはバージョンがあります。つまり、特定のバージョンを標準的なプロモーションパス上でプロモートできます。たとえば、development (開発) から test (テスト) に移行し、その後 production (本番) に移行できます。Red Hat Satellite 5 では、チャンネルをクローン作成することで、この概念を実現していました。

Attach (アタッチ)

RPM コンテンツへのアクセスを提供するサブスクリプションをホストに割り当てるプロセスです。

Capsule

Capsule は、Red Hat Satellite 6 デプロイメントで使用できる追加サーバーであり、他のサービス (Puppet マスター、**DHCP**、**DNS**、**TFTP** など) をローカル化し、コンテンツのフェデレーションと配布を容易にします。

Catalog (カタログ)

カタログは、1 台のコンピューターのシステム状態について説明するドキュメントです。ここでは、管理が必要な全リソースと、そのリソース間の依存関係がリスト表示されます。

Compute Profile (コンピュートプロファイル)

コンピュートプロファイルは、コンピュートリソース上にある新規仮想マシンのデフォルト属性を指定します。

Compute Resource (コンピュートリソース)

コンピュートリソースは、Red Hat Satellite 6 がホストやシステムのデプロイに使用する仮想またはクラウドのインフラストラクチャーです。たとえば、Red Hat Enterprise Virtualization Manager、OpenStack、EC2、VMWare などが挙げられます。

コンテンツ

コンテンツには、ソフトウェアパッケージ (RPM ファイル) および Puppet モジュールが含まれます。コンテンツをライブラリーに同期した後、コンテンツビューを使用してライフサイクル環境にプロモートします。これにより、ホストがコンテンツを使用できるようになります。

コンテンツ配信ネットワーク (CDN)

コンテンツ配信ネットワーク (CDN) は、同じ地理的拠点にサーバーが設置されるコロケーション形態で、Red Hat コンテンツの配信に使用されるメカニズムです。たとえば、ヨーロッパの Satellite がコンテンツを同期する場合は、ヨーロッパにあるソースからコンテンツをプルします。

Content Host (コンテンツホスト)

コンテンツホストは、コンテンツとサブスクリプションに関連するタスクを管理するホストの一部です。

Content View (コンテンツビュー)

コンテンツビューは、製品、パッケージ、および Puppet モジュールを、インテリジェントなフィルタリングやスナップショット作成などの各種機能と組み合わせたコンテンツの定義です。コンテンツビューは、Red Hat Satellite 5 のチャンネルとクローン作成の機能を組み合わせて改良した概念です。

External Node Classifier (外部ノードの分類子)

外部ノードの分類子は、ホストの設定時に Puppet マスターが使用する追加データを提供する Puppet コンストラクトです。Red Hat Satellite 6 は、Satellite デプロイメントで Puppet マスターに対する外部ノードの分類子として機能します。

Facter

Facter は一種のプログラムであり、それを実行するシステムに関する情報 (ファクト) を提供します。たとえば、Facter は合計メモリー、オペレーティングシステムのバージョン、アーキテクチャーなどをレポートできます。Puppet モジュールは、Facter が収集したホストのデータに基づいて、特定の設定を有効にします。

Hammer

Hammer は、Red Hat Satellite 6 のコマンドラインツールです。標準 CLI、またはスクリプトやインタラクティブシェルで Hammer を使用して Red Hat Satellite 6 を管理します。

Hiera

Hiera は、設定データのキーと値のルックアップツールで、Puppet マニフェストからサイト固有のデータを排除できます。

Host

ホストは、Red Hat Satellite 6 が管理するすべてのシステム (物理または仮想システム) を指します。

Host Collection (ホストコレクション)

ホストコレクションは、Satellite 5 のシステムグループに相当するもので、1 つ以上のホストのユーザー定義グループを指します。

Host Group (ホストグループ)

ホストグループは、ホストをビルドするためのテンプレートです。これには、(利用可能な RPM ファイルと Puppet モジュールを定義する) コンテンツビューや、(ソフトウェアと設定を最終的に決定する) 適用対象の Puppet クラスが含まれます。

Location (ロケーション)

ロケーションは、物理的な場所を表すデフォルト設定のコレクションです。これらはネスト化できるため、ロケーションの階層的なコレクションを設定できます。たとえば、中東から始まり、テルアビブ、データセンター東、さらにはラック 22 と詳細化してデフォルトをセットアップできます。

Library (ライブラリー)

ライブラリーには **すべて** のバージョンが含まれるため、ユーザーがデプロイしたソフトウェアの、最近同期したバージョンも含まれます。IT インフラストラクチャーライブラリー (ITIL) の組織 (organization) または部署 (department) のコンテキストでは、これは確定版メディアライブラリーに相当します (以前の名称は確定版ソフトウェアライブラリー)。

Manifest (マニフェスト)

Red Hat カスタマーポータルから Red Hat Satellite 6 にサブスクリプションを送信するメカニズム。これは Red Hat Satellite 5 で使用される証明書と機能的に似ています。
証明書およびサブスクリプションのタイプの詳細は、以下を参照してください。

- サブスクリプション概要およびワークフロー
- RHN Classic からの移行

Organization

組織は、Satellite 6 デプロイメント内の複数のシステム、コンテンツ、その他の機能からなる単独のコレクションです。

Product (製品)

コンテンツリポジトリーのコレクションです。製品には Red Hat 製品、またはソフトウェアと設定コンテンツで設定される、新規に作成される製品が含まれます。

Promote (プロモート)

ソフトウェアと設定コンテンツで設定されるコンテンツビューを 1 つのアプリケーションライフサイクル環境から別のアプリケーションライフサイクル環境に移行する動作を指します。たとえば、Development (開発) から QA、そして Production (本番) への移行などが含まれます。

Provisioning Template (プロビジョニングテンプレート)

プロビジョニングテンプレートは、キックスタートファイル、スニペット、その他のプロビジョニング操作のためのユーザー定義テンプレートです。Satellite 6 では、このテンプレートで、Red Hat Satellite 5 のキックスタートプロファイルと Cobbler スニペットと同様の機能を提供します。

Pulp Node (Pulp ノード)

Pulp ノードは、コンテンツをミラーリングする Capsule Server のコンポーネントです。これは Red Hat Satellite 5 Proxy に似ています。主な違いは、Pulp ノードの場合は、ホストがコンテンツを使用する前に、そのノード上でコンテンツをステージングできる点にあります。

Puppet Agent (Puppet エージェント)

Puppet エージェントは、ホスト上で実行するエージェントで、設定の変更をホストに適用します。

Puppet Master (Puppet マスター)

Puppet マスターは、Puppet エージェントが実行する Puppet マニフェストをホストに提供する Capsule Server のコンポーネントです。

Puppet Module (Puppet モジュール)

Puppet モジュールは、ユーザー、ファイル、サービスなどのリソースを管理するのに使用できるコードとデータの自己完結型のバンドルです。

Repository (リポジトリー)

リポジトリーは、コンテンツのコレクション用にストレージを提供します。たとえば、YUM リポジトリーまたは Puppet リポジトリーなどがあります。

ロール

ロールは、ホストなどの一連のリソースに適用されるパーミッションのコレクションを指定します。

Smart Proxy (スマートプロキシー)

スマートプロキシーは、**DNS**、**DHCP** などの外部サービスと統合できる Capsule Server のコンポーネントです。

Smart Variable (Smart 変数)

スマート変数は Puppet クラスの動作を制御する設定値です。この値は、ホスト、ホストグループ、組織、またはロケーションに設定できます。

Standard Operating Environment (SOE) (標準運用環境 (SOE))

標準運用環境 (SOE) は、アプリケーションがデプロイされるオペレーティングシステムの制御されたバージョンです。

Subscription (サブスクリプション)

サブスクリプションは、Red Hat からコンテンツとサービスを受け取る手段です。

Synchronizing (同期)

同期は、外部リソースのコンテンツを Red Hat Satellite 6 ライブラリーにミラーリングすることを指します。

Synchronization Plans (同期プラン)

同期プランは、コンテンツ同期のスケジュールに基づく実行を可能にします。

User Group (ユーザーグループ)

ユーザーグループは、ユーザーのコレクションに割り当てるロールのコレクションです。これは、Red Hat Satellite 5 のロールに似ています。

ユーザー

ユーザーは、Red Hat Satellite を使用できるように登録されたすべてのユーザーを指します。認証および認可は、組み込みロジック、外部 LDAP リソース、または Kerberos を使用して実行できます。