Oracle® Solaris 11.3 での Puppet を使用した構成管理の実行

Part No: E77933
2016 年 7 月

Part No: E77933

ドキュメントのアクセシビリティについて

オラクルのアクセシビリティについての詳細情報は、Oracle Accessibility ProgramのWeb サイト(http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc)を参照してください。

Oracle Supportへのアクセス

サポートをご契約のお客様には、My Oracle Supportを通して電子支援サービスを提供しています。詳細情報は(http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info) か、聴覚に障害のあるお客様は (http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs)を参照してください。

目次

このドキュメントの使用

- **概要** – データリンク、IP インタフェースとアドレス、ネームサービスとディレクトリサービス、リアクティブプロファイル、無線ネットワークなど、Oracle Solaris オペレーティングシステム (OS) のさまざまなネットワークコンポーネントを構成および管理する方法に関する情報を提供します。
- **対象読者** – 企業のデータセンターでネットワーク構成の管理を担当するシステム管理者。
- **前提知識** – ネットワーク管理の概念と実践に関する基本的および高度な理解。

製品ドキュメントライブラリ

この製品および関連製品のドキュメントとリソースは http://www.oracle.com/pls/topic/lookup?ctx=E62101-01 で入手可能です。

フィードバック

このドキュメントに関するフィードバックを http://www.oracle.com/goto/docfeedback からお聞かせください。

♦ ♦ ♦ 第 1 章

Oracle Solaris での Puppet を使用した構成管理の実行について

Puppet は、Oracle Solaris 内のほとんどの主要なサブシステム (Oracle Solaris サーバーとそのサブシステムを含む) の構成管理を自動化および適用するために使用できるクロスプラットフォーム対応ソフトウェアです。Puppet を使用すると、いくつかの一般的なシステム構成タスクを実行できます。Puppet では、IT インフラストラクチャー全体にわたってリソース構成を標準化および適用できます。

この章では、Puppet インフラストラクチャーの概要について説明したあと、Oracle Solaris での Puppet の実装方法に関する基本的な説明を提供します。

この章の内容は、次のとおりです。

このガイドの範囲を超える詳細情報 (さまざまな Puppet サービスの背景情報やより完全な説明を含む) については、`http://docs.puppet.com` を参照してください。

Oracle Solaris 11.3 でサポートされている Puppet バージョンに関する具体的な情報は、`https://docs.puppet.com/puppet/3.6/reference/` で見つけることができます。

Oracle Solaris 11.3 での Puppet サポートの重要な事項

Oracle Solaris では、次の Puppet 機能がサポートされています。

- **Puppet のインストール**

  Puppet ソフトウェアパッケージ (`system/management/puppet`) は、デフォルトでは Oracle Solaris システムにインストールされません。Puppet マスターと、Puppet

エージェントを実行するすべてのノードに同じ Puppet Image Packaging System (IPS) パッケージを個別にインストールする必要があります。

第2章「Oracle Solaris での Puppet の使用開始」を参照してください。

- **Puppet モジュールおよびユーティリティー**

  Puppet IPS パッケージをインストールすると、すべてのコア Puppet モジュールだけでなく、Oracle Solaris リリースに固有のその他のモジュールが得られます。

  Puppet のインストールに含まれているインストール済みで使用可能なすべてのモジュールの完全なリストを表示するには、次のコマンドを実行します。

  ```
  % pkg list -a *puppet*
  ```

  特定の Puppet モジュールの詳細は、そのモジュールに含まれている `README` ファイルを参照してください。また、`https://puppet.com/` にアクセスして特定のモジュールの詳細を検索することもできます。

  Puppet をインストールすると、Puppet と連携して動作するように設計された次のユーティリティーも得られます。

  - **Facter** – Puppet が特定のシステムに関するファクト (OS タイプ、CPU、メモリーサイズなど) を検出するために使用するユーティリティーです。Facter がシステムに関して収集した情報は、Puppet マスターに送信されます。次に、Puppet マスターがこれを使用して、特定のリソースセットの望ましいシステム状態を記述したカタログをコンパイルします。このカタログには、管理する必要のあるすべてのリソースと、これらのリソース間の依存関係が一覧表示されています。36 ページの「Facter を使用したシステムに関する情報の収集」を参照してください。
  - **Hiera** – 構成データを管理するために使用するクロスプラットフォームのキー/値検索ツールです。Hiera は、通常は Puppet マニフェストに含まれるサイト固有のデータを保持するために Puppet とともに使用します。サイト固有のデータをマニフェストにではなく Hiera 構成ファイル内に格納すると繰り返しが避けられ、それにより、複数のシステムのために再利用できるより一般的なマニフェストを記述できます。

    Puppet クラスは必要なデータを要求でき、Hiera はサイト全体の構成ファイルとして機能します。Puppet が Hiera をロードすると、Hiera はこの構成ファイルを `/etc/hiera.yaml` にあるグローバルファイルの代わりに使用します。詳細は、`https://docs.puppet.com/hiera/3.1/` を参照してください。

- **Puppet エージェント/マスターモデル**

  Puppet は、エージェント/マスターモデルを使用します。ここで、*Puppet* マスターは、*Puppet* エージェントが実行されているすべてのノード (物理または仮想) の重要な構成情報を管理します。

  Puppet エージェントを実行しているノードは Puppet マスターを定期的にポーリングし、更新された構成情報を要求します。次に、エージェントがこれをそのノードに適用します。14 ページの「Puppet の動作のしくみ」を参照してください。

- **Puppet SMF サービス**

Puppet ソフトウェアパッケージをインストールすると、Puppet マスター用の `svc:/application/puppet:master` と Puppet エージェント用の `svc:/application/puppet:agent` の 2 つのインスタンスを備えた 1 つの Puppet SMF サービス (`svc:/application/puppet`) が得られます。デフォルトでは、Puppet のインストール後、これらのサービスインスタンスは無効になっています。これらのサービスインスタンスを有効にすると、これらのサービスのデーモンが起動されます。これらのサービスが無効になると、これらのデーモンが停止されます。24 ページの「Puppet マスターおよびエージェントの構成」を参照してください。

- **Puppet 構成ファイル**

  Puppet には、マスターとエージェントの両方の構成ファイル (`/etc/puppet/puppet.conf`) が用意されています。この構成は、SMF リポジトリ内に格納されています。`puppet.conf` ファイルには、多くのシステムリソースが定義されています。このファイルには、Puppet マスターとマスターによって管理されるすべてのノードが使用するデフォルト値が一覧表示されています。

  Puppet 構成ファイルは、*SMF* ステンシルを使用して `svcio` ユーティリティーによって生成されます。『Oracle Solaris 11.3 でのシステムサービスの開発』 の 第 6 章, 「ステンシルを使用した構成ファイルの作成」を参照してください。

  `puppet.conf` ファイル内の構成を常に SMF リポジトリの内容に一致させるため、このファイルは決して直接編集しないでください。代わりに、SMF コマンドを使用してファイル内の適切なプロパティーを設定します。`svccfg(1M)`を参照してください。Puppet 構成ファイルの生成には刷り出しが使用されるため、SMF プロパティーを設定することによって行なった永続的な変更はすべて、`puppet.conf` ファイルに自動的に適用されます。

- **Puppet リソースおよびリソースタイプ**

  Puppet は、リソースを使用して、サービスが実行される時期や方法、ソフトウェアパッケージの管理、ネットワークおよびネームサービス構成の特定のコンポーネントなどのシステムのさまざまな側面を表します。リソースはまた、システムの特定の側面があるべき状態を反映することもできます。

  各リソースには、Puppet マニフェスト内で指定できるタイトルと一連の属性および値によって定義されるリソースタイプがあります。宣言できる値は、管理している構成のタイプによって異なります。第3章「Oracle Solaris での Puppet リソースおよびリソースタイプの操作」を参照してください。

- **Puppet プロバイダ**

  Puppet プロバイダは、リソースの一般的な定義を、特定のプラットフォーム上でそのリソースを実装するために必要なアクションに変換します。これらのクロスプラットフォーム機能は、構成設定を指定された構成の適用に必要なプラットフォーム固有のコマンドに変換する Puppet Resource Abstraction Layer (RAL) によって有効になります。

  たとえば、Oracle Solaris システムにソフトウェアパッケージをインストールしている場合、Puppet は Red Hat Enterprise Linux システムの間は IPS を使用し、パッケージのインストールには RPM (Red Hat Package Manager) を使用します。

Oracle Solaris でサポートされている主なプロバイダのいくつかを次に示します。

- IPS パッケージのインストール、コマンド、パブリッシャー、ファセット、およびメディエータ
- SVR4 パッケージのインストール
- ブート環境
- データリンクプロパティー
- アグリゲーション
- Etherstub
- IP ネットワークインタフェース
- ネームサービス
- Oracle Solaris ゾーン、Oracle Solaris カーネルゾーン、および共有ストレージ上のゾーン (ZOSS) バッキングストア
- SMF 管理コマンド
- SMF プロパティー
- TCP/IP チューニング可能値
- 仮想ローカルエリアネットワーク (VLAN)
- 仮想ネットワークインタフェースカード (VNIC)
- ZFS データセットの作成およびプロパティーの操作 (ほとんどの `vdev` タイプでの `zpool` の作成と削除を含む)

31 ページの「Puppet リソースおよびリソースタイプについて」を参照してください。

- **Puppet コマンド行インタフェース (CLI)**

  マスターとエージェントノードの間の初期ハンドシェークなどのいくつかのアクションを実行するには、Puppet コマンド行インタフェース (CLI) を使用します。CLI はまた、テスト用にドライランを実行するためにも使用することがあります。また、CLI を使用して、Puppet に関する問題のトラブルシューティングやデバッグを行うこともできます。

  Puppet CLI を使用して実行する可能性のあるその他のタスクには、次のものがあります。

  - 証明書の管理
  - レポートの生成および管理
  - プラグインへのアクセス
  - リソースの管理
  - ステータスの表示

  Puppet CLI を使用してアクションを実行するには、次の構文を使用します。

  ```
  # puppet subcommand [options] action [options]
  ```

使用可能なすべての Puppet サブコマンドとその使用法は、次のように表示します。

```
# puppet help
```

特定のサブコマンドのヘルプは、次のように表示します。

```
# puppet help subcommand
```

次の部分的な例は、agent サブコマンドに関する情報を表示する方法を示しています。

```
# puppet help agent
puppet-agent(8) -- The puppet agent daemon
========

SYNOPSIS
--------
Retrieves the client configuration from the puppet master and applies it to
the local host.

This service may be run as a daemon, run periodically using cron (or something
similar), or run interactively for testing purposes.


USAGE
-----
puppet agent [--certname <name>] [-D|--daemonize|--no-daemonize]
  [-d|--debug] [--detailed-exitcodes] [--digest <digest>] [--disable [message]] [--
enable]
  [--fingerprint] [-h|--help] [-l|--logdest syslog|<file>|console]
  [--no-client] [--noop] [-o|--onetime] [-t|--test]
  [-v|--verbose] [-V|--version] [-w|--waitforcert <seconds>]


DESCRIPTION
-----------
This is the main puppet client. Its job is to retrieve the local
machine's configuration from a remote server and apply it. In order to
successfully communicate with the remote server, the client must have a
certificate signed by a certificate authority that the server trusts;
the recommended method for this, at the moment, is to run a certificate
authority as part of the puppet server (which is the default). The
client will connect and request a signed certificate, and will continue
connecting until it receives one.
. . .
```

特定のサブコマンドのアクションのヘルプは、次のように表示します。

```
# puppet help subcommand action
```

- **Puppet の特権と承認**

  Puppet を構成および管理するには、`Puppet Management` 権利プロファイルが割り当てられているか、または `root` 役割になる必要があります。`Puppet Management` 権利プロファイルには、`solaris.smf.manage.puppet` および `solaris.smf.value.puppet` 特権が含まれています。ユーザー権利および特権の動作のしくみの詳細は、『Oracle Solaris 11.3 でのユーザーとプロセスのセキュリティー保護』の「ユーザー権管理」を参照してください。

Oracle Solaris での Puppet の一般的な使用法

Puppet を使用すると、Oracle Solaris 内のほとんどの主要なサブシステムを管理したり、複数のノード (物理と仮想の両方) のためのいくつかの一般的なシステム構成タスクを自動化したりできます。Puppet は単純な配備から、クラウド配備や OpenStack などのより複雑なインフラストラクチャーまで拡張できます。Puppet を使用する可能性のあるその他のいくつかの方法には、プロビジョニング、システム構成、ソフトウェア管理などがあります。

Puppet の動作のしくみ

Puppet には、システムに必要なソフトウェアや構成を定義したあと、初期設定後に指定された状態を維持する機能があります。

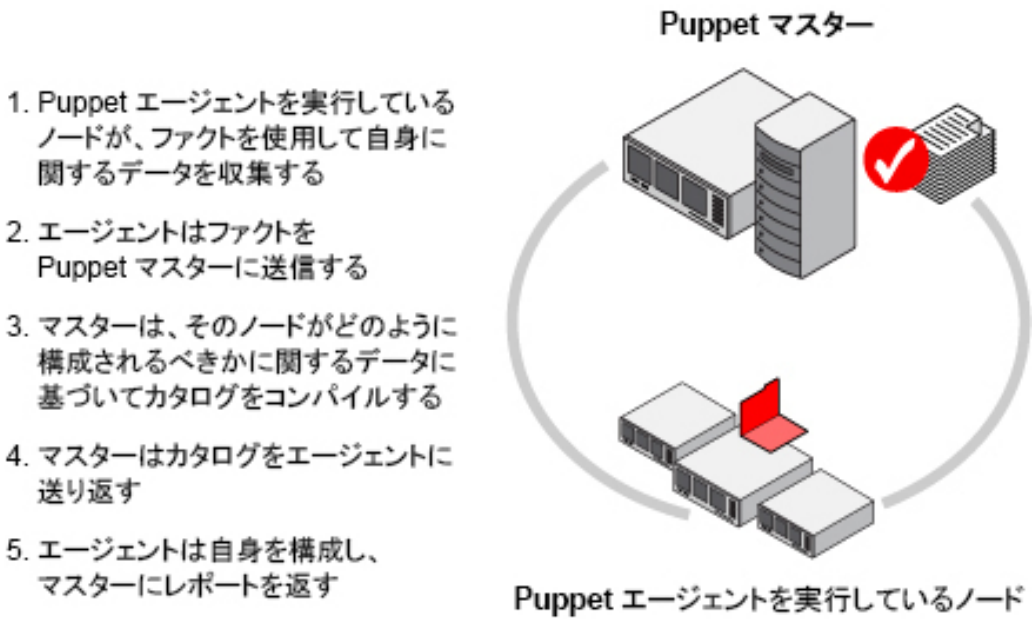
特定の環境またはインフラストラクチャーの構成パラメータを定義するには、Ruby に似た宣言型のドメイン固有言語 (DSL) を使用します。Puppet は、Puppet ソフトウェアパッケージのインストール時にインストールされる *Facter* と呼ばれるユーティリティーを使用して、システムに関する情報を検出します。36 ページの「Facter を使用したシステムに関する情報の収集」を参照してください。

Puppet マスターは、マニフェストを使用して制御するすべてのノードの重要な構成情報を管理するシステムです。18 ページの「Puppet マニフェスト」を参照してください。

マスターが制御するノードは、Puppet がインストールされており、*Puppet* エージェント (デーモン) を実行しているノードです。エージェントがノードに関して収集した構成情報は、Puppet マスターに送信されます。Puppet マスターはそのあと、そのノードがどのように構成されるべきかに基づいてカタログをコンパイルします。各ノードは、その情報を使用して、必要なすべての構成の更新を自身に適用します。

Puppet は、プルモードを使用して動作します。ここで、エージェントはマスターを定期的にポーリングして、サイト固有の構成やノード固有の構成を取得します。このインフラストラクチャーでは、管理対象ノードは Puppet エージェントアプリケーションを (通常はバックグラウンドサービスとして) 実行します。詳細は、`https://docs.puppet.com/puppet/3.6/reference/architecture.html` を参照してください。

次の図は、Puppet マスター/エージェントトポロジをさらに詳細に説明しています。



Puppet マスターについて

Puppet マスターは、指定されたサーバー上で実行されるデーモンであり、Puppet の構成データおよび権限のプライマリソースです。マスターは、Puppet インフラストラクチャーの一部であるすべてのノードに手順を提供します。 コンポーネント構成のある側面はほかのコンポーネントの構成に依存するため、Puppet マスターとして指定されているサーバーはシステムの構成全体を認識している必要があります。Puppet では、マスターを独自のユーザーおよびグループとして実行させることによってマスターへのアクセスを制限します。16 ページの「Puppet ユーザーおよびグループの機能」を参照してください。

マスターは、次を含むいくつかのアクションに責任を負っています。

- エージェント用のカタログのコンパイル

- ファイルサーバーからのファイルの転送
- セントラルインスタンスへのレポートの送信

注記 - マスターは、`root` 特権を必要としないその他のアクションも実行する可能性があります。

Puppet エージェントについて

ターゲットシステム (またはノード) 上で実行される *Puppet* デーモンは、*Puppet* エージェントと呼ばれます。エージェントは、Puppet マスターからプルした構成カタログを適用できるように、それが有効になっているノードの適切な特権を持っている必要があります。エージェントは、マスターにはじめて接続したときに SSL (Secure Socket Layer) 証明書を要求することによって、マスターサーバーから通信特権を取得します。そのあと、エージェントは構成の更新がないかどうかマスターをポーリングするたびに、その証明書が有効な場合にのみそれらの更新を受信します。

各ターゲットノード上で実行される Puppet エージェントは、システムの構成のほとんどの側面を変更する機能を備えている必要があります。この要件によって、マスターが示したそのエージェントのあるべき状態が適用されます。Puppet エージェントはシステムへのアクセスを大量に必要とするため、`root` ユーザー、または `Puppet Management` 権利プロファイルが割り当てられたユーザーとして実行されます。

注記 - マスターは、正しくない構成情報を誤って受信することのないように、エージェントに対する認証も実行する必要があります。

Puppet ユーザーおよびグループの機能

Puppet ユーザーおよびグループは、モジュールがマスターから取得する必要のある情報にのみアクセスできるようにするために、セキュリティー上の理由から使用されます。また、Puppet ユーザーおよびグループにより、Puppet モジュールが攻撃または改ざんされることも防止されます。Puppet ユーザーはマスター上でタスクを実行し、Puppet グループのメンバーです。設定プロセス中にマスターの SMF サービスインスタンスを有効にすると、この特権ユーザーおよびグループが自動的に作成され、マスターデーモンに割り当てられます。第2章「Oracle Solaris での Puppet の使用開始」を参照してください。

Puppet ユーザーを通して、Puppet マスターは次のタスクを実行します。

- 構成マニフェストを Puppet マニフェストのディレクトリ内に格納します。
- エージェントからの SSL 証明書を受け入れます。

- ファイルをエージェントに転送します。
- カタログを作成します。

Puppet の暗号化および通信方式

Puppet は、SSL および TLS (Transport Layer Security) 暗号化プロトコルに基づく OpenSSL ツールキットとインタフェースを取ります。Puppet は、エージェントとマスターの認証および検証に標準の SSL/TLS 暗号化テクノロジと標準の SSL 証明書を使用します。Puppet はまた、サーバーとエージェントの間のトラフィックフローの暗号化にも SSL/TLS を使用します。使用されるデフォルトのハッシュは SHA-256 です。

Puppet の暗号化方式では、次のことを実行します。

- すべてのエージェントをマスターに対して認証します
- すべてのエージェントでマスターを認証します
- マスターとエージェントの間の盗聴する通信を防ぎます

Puppet は、TLS クライアント側の X.509 証明書を使用して、相互のホスト認証を実行します。デフォルトでは、この情報は Puppet 構成ファイル (`puppet.conf`) で定義されている `/etc/puppet/ssl` ディレクトリ内に格納されます。このデフォルトの場所は SMF コマンドを使用して変更でき、これはそのあと、サイトの構成ファイルに反映されます。鍵、証明書、署名された要求だけでなく、署名を待っている要求に対しても個別のディレクトリが存在します。これらのディレクトリは、マスターとエージェントの両方に存在します。

Puppet は独自の認証局 (CA) を使用するため、CA に対するシステムのデフォルト設定 (`/etc/certs/CA`) を使用する必要はありません。マスターは初期化されると、独自の CA 証明書と非公開鍵を生成し、証明書失効リスト (CRL) を初期化してから、サーバー証明書と呼ばれる別の証明書を生成します。この証明書は SSL および TLS 通信に使用され、エージェントに送信されます。マスターとエージェントの交換中に、この CA はマスター上の `/etc/puppet/ssl/ca/signed` ディレクトリおよびエージェント上の `/etc/puppet/ssl/certs` ディレクトリ内に格納されます。

Puppet の用語

Puppet は、状態を定義する宣言型のドメイン固有言語 (DSL) を使用します。Puppet コードは、マニフェスト内に書き込まれます。そのコードでは、ファイル、パッケージ、サービスなどのシステムのさまざまな側面を定義するリソースを宣言します。リソースはクラスにグループ化され、各クラスでは、リソースの動作に影響を与える可能性のあるパラメータが公開されています。クラスと構成ファイルが次に、モジュールに整理されます。これらの Puppet の基本的な用語は、以降のセクションでより詳細

に説明されています。より完全な定義については、Puppet の用語集 (https://docs.puppet.com/references/glossary.html) を参照してください。

Puppet マニフェスト

特定の構成に対して宣言する必要のあるさまざまなリソースは、マニフェストと呼ばれるファイル内に格納されます。マニフェストは Puppet コードを含み、Puppet のインフラストラクチャーの中心です。これらのマニフェストは Puppet マスター上にあります。リソース定義を保存する場合は常に、マニフェスト内に保存します。各マニフェストは .pp ファイル拡張子で終わる必要があります。

すべてのノードに適用されるグローバル構成を定義するには、Puppet site.pp マニフェストを使用します。サイトマニフェストには、特定のノードに適用されるノード固有のコードも含めることができます。ノード定義 (または ノード文) は、一致するノードのカタログにのみ含まれる Puppet コードのブロックです。この機能を使用すると、特定のノードに特定の構成を割り当てることができます。詳細は、https://docs.puppet.com/puppet/latest/reference/lang_node_definitions.html を参照してください。

また、いくつかのリソースをグループにまとめるマニフェストを記述することもできます。この場合は、クラスを使用して、指定されたノードにリソースを適用します。第4章「Oracle Solaris での Puppet マニフェスト、クラス、およびモジュールの記述」を参照してください。

Puppet クラス

クラスは、まとめてバンドルされる一連の構成です。Puppet クラスには、リソースや変数だけでなく、追加の高度な属性を含めることができます。ノードにクラスを割り当てると、そのノードは、このクラスの一部である構成のすべてを取得します。クラス宣言は、マニフェスト内に含めます。43 ページの「Puppet クラスの記述」を参照してください。

Puppet モジュール

Puppet モジュールは、ファイルとディレクトリの自己完結型のコレクションであり、Puppet マニフェストやその他のオブジェクト (ファイルとテンプレートを含む) を含めることができます。モジュール内にある情報はパッケージ化され、Puppet が理解して使用できる方法で整理されています。モジュールは、Puppet が IT インフラストラクチャー内の構成管理に使用できるクラスとタイプを検索する方法です。Puppet は、特定のモジュール内に格納されているすべてのクラスまたは定義済みのタイプを自動的

にロードします。マニフェスト内では、これらのクラスまたはタイプのすべてを名前で宣言できます。45 ページの「Puppet モジュールの記述」を参照してください。

Puppet の追加のリファレンス

Puppet の詳細は、次のドキュメントを参照してください。

- Puppet に関する一般的な情報については、`https://puppet.com/` を参照してください。
- Puppet のリファレンス情報については、`https://docs.puppet.com/puppet/3/reference/` を参照してください。
- Puppet のその他のドキュメントリソースについては、`https://puppet.com/resources/books` を参照してください。

♦ ♦ ♦ 第 2 章

Oracle Solaris での Puppet の使用開始

この章では、Oracle Solaris で Puppet をインストール、構成、および有効化する方法について説明します。

この章の内容は、次のとおりです。

注記 - Puppet での構成管理を管理するには、`Puppet Management` 権利プロファイルが割り当てられているか、または `root` 役割になる必要があります。`Puppet Management` 権利プロファイルに関連付けられた特権には、`solaris.smf.manage.puppet` と `solaris.smf.value.puppet` が含まれます。『Oracle Solaris 11.3 でのユーザーとプロセスのセキュリティー保護』 の 「割り当てられている管理権利の使用」を参照してください。

Puppet のインストール前タスク

マスターと、Puppet エージェントを実行するノードに Puppet IPS パッケージをインストールする前に、次のタスクを実行します。

- Puppet マスターとして機能するサーバーを指定します。

  Puppet をいずれかのノードにインストールする前に、Puppet を (1 つまたは複数の) マスターサーバーにインストールして構成してください。
- Puppet エージェントを実行するノードを指定します。
- 完全修飾ドメイン名を使用してすべてのホストを解決できるように、マスターとエージェントの両方でドメインネームシステム (DNS) プロトコルを構成します。『Oracle Solaris 11.3 ディレクトリサービスとネームサービスでの作業: DNS と NIS』 の 第 3 章, 「DNS サーバーとクライアントサービスの管理」を参照してください。

- Puppet マスター上の時間管理が正確に構成されていることを確認します。22 ページの「Puppet マスター上での NTP の構成」を参照してください。

Puppet マスター上での NTP の構成

Puppet マスターサーバーは認証局として機能するため、推奨されるベストプラクティスとして、Puppet をインストールする前にマスター上で NTP (Network Time Protocol) を構成して正確な時間が維持されるようにします。そうしないと、マスターがエージェントで期限切れとして処理される証明書を発行する可能性があります。NTP の管理の詳細は、『Oracle Solaris 11.3 でのクロック同期と Web キャッシュを使用したシステムパフォーマンスの拡張』を参照してください。

▼ Puppet マスター上で NTP を構成する方法

Puppet IPS パッケージをインストールする前に、Puppet マスター上で次の手順を実行します。

1. **`Puppet Management` 権利プロファイルが割り当てられた管理者になるか、または `root` 役割になります。**

2. **`/etc/inet/ntp.client` ファイルを編集したあと、その情報を `/etc/inet/ntp.conf` にコピーすることによって構成ファイルを作成します。**

   この手順では、1 つのタイムサーバーに障害が発生した場合に備えて、次の 4 つのタイムサーバーが使用されます。

   ```
   # echo "server 0.pool.ntp.org" > /etc/inet/ntp.conf
   # echo "server 1.pool.ntp.org" >> /etc/inet/ntp.conf
   # echo "server 2.pool.ntp.org" >> /etc/inet/ntp.conf
   # echo "server 3.pool.ntp.org" >> /etc/inet/ntp.conf
   ```

3. **`/etc/inet/ntp.conf` ファイルに必要な構成パラメータを追加します。**

   ```
   # echo "driftfile /var/ntp/ntp.drift" >> /etc/inet/ntp.conf
   # echo "statsdir /var/ntp/ntpstats/" >> /etc/inet/ntp.conf
   # echo "filegen peerstats file peerstats type day enable" >> /etc/inet/ntp.conf
   # echo "filegen loopstats file loopstats type day enable" >> /etc/inet/ntp.conf
   ```

4. **初期の時間同期を強制的に実行します。**

   ```
   # ntpdate 0.pool.ntp.org
   ```

5. **`ntp` SMF サービスを有効にします。**

   ```
   # svcadm enable ntp
   ```

6. **NTP が動作していることを確認します。**

   ```
   # ntpq -p
   ```

注記 - NTP の起動には 15 - 60 分、またはそれ以上かかる場合があります。

次の手順 代わりの方法として、Puppet マニフェストを使用して NTP 構成を指定することもできます。第4章「Oracle Solaris での Puppet マニフェスト、クラス、およびモジュールの記述」を参照してください。

Puppet のインストール

Oracle Solaris では、Puppet IPS パッケージはデフォルトではインストールされません。Puppet を使用するには、Puppet マスターと、Puppet エージェントを実行するすべてのノードに `system/management/puppet` パッケージを個別にインストールする必要があります。マスターと各ノードの両方に同じソフトウェアパッケージをインストールします。Oracle Solaris 11.3 でサポートされている Puppet バージョンは 3.6.2 です。

Puppet IPS パッケージがシステムにインストールされているかどうかを判定するには、次のコマンドを入力します。

```
# pkg list puppet
pkg list: no packages matching the following patterns are installed:
puppet
```

Puppet IPS パッケージに関するより詳細な情報を次のように表示できます。

```
# pkg info -r puppet
Name: system/management/puppet
       Summary: Puppet - configuration management toolkit
   Description: Puppet is a flexible, customizable framework designed to help
                system administrators automate the many repetitive tasks they
                regularly perform. As a declarative, model-based approach to IT
                automation, it lets you define the desired state - or the "what"
                - of your infrastructure using the Puppet configuration
                language. Once these configurations are deployed, Puppet
                automatically installs the necessary packages and starts the
                related services, and then regularly enforces the desired state.
      Category: System/Administration and Configuration
         State: Not installed
     Publisher: solaris
       Version: 3.6.2
 Build Release: 5.11
        Branch: 0.175.3.8.0.5.0
Packaging Date: Mon May 09 22:30:56 2016
          Size: 427.92 kB
          FMRI: pkg://solaris/system/management/puppet@3.6.2,5.11-0.175.3.8.0.5.0:
20160509T223056Z
```

前の出力は、Puppet パッケージがシステムにインストールされていないことを示しています。

次のように、Puppet IPS パッケージを最初にマスターにインストールしてから、各ノードにインストールします。

```
# pkg install puppet
```

Puppet がインストールされていることを確認します。

```
# pkg info puppet
Name: system/management/puppet
       Summary: Puppet - configuration management toolkit
   Description: Puppet is a flexible, customizable framework designed to help
                system administrators automate the many repetitive tasks they
                regularly perform. As a declarative, model-based approach to IT
                automation, it lets you define the desired state - or the "what"
                - of your infrastructure using the Puppet configuration
                language. Once these configurations are deployed, Puppet
                automatically installs the necessary packages and starts the
                related services, and then regularly enforces the desired state.
      Category: System/Administration and Configuration
         State: Installed
     Publisher: solaris
       Version: 3.6.2
 Build Release: 5.11
        Branch: 0.175.3.0.0.30.0
Packaging Date: Fri Aug 21 17:26:04 2015
          Size: 426.20 kB
          FMRI: pkg://solaris/system/management/puppet@3.6.2,5.11-0.175.3.0.0.30.0:
20150821T172604Z
```

前の出力に従って、Puppet が現在システムにインストールされていることを確認できます。

Puppet マスターおよびエージェントの構成

Puppet マスターと、マスターが制御するノード (エージェント) の両方に Puppet をインストールしたら、マスターとエージェントを構成する準備ができました。

注記 - 次のタスクを実行するには、`Puppet Management` 権利プロファイルが割り当てられているか、または `root` 役割になる必要があります。

Puppet 構成が SMF によって管理される方法

Puppet のインストールの一部として、Puppet マスターと、Puppet エージェントを実行するノードの両方に Puppet SMF サービスがインストールされます。この SMF サービスには、Puppet マスター用の `svc:/application/puppet:master` と Puppet エージェント用の `svc:/application/puppet:agent` の 2 つのインスタンスがあります。Puppet が SMF によって管理されるようにすると、システム更新中に構成を保持するために役立つ階層化された構成を利用できます。

次の出力に示すように、デフォルトでは、両方の Puppet SMF サービスインスタンスが無効になっています。

```
# svcs -a | grep puppet
disabled       Feb_18   svc:/application/puppet:agent
disabled       Feb_18   svc:/application/puppet:master
```

初期の Puppet 構成ファイル (`etc/puppet/puppet.conf`) は、SMF ステンシルを使用して生成されます。これにより、SMF リポジトリ内に格納されているすべての構成が Puppet 構成ファイル内に格納されている構成に正しくマップするようになります。

Puppet は、`application/puppet` サービスインスタンスで設定されたプロパティーからではなく、`/etc/puppet/puppet.conf` ファイルから構成情報を読み取ります。必要な構成ファイルを提供するために、各 puppet インスタンスは、ステンシルファイルと `configfile` プロパティーグループを提供します。`configfile` プロパティーグループは `svcio` ユーティリティーに、動作して指定された構成ファイルを作成するよう指示します。ステンシルファイルはそのあと、正しい形式を使用してサービスプロパティー値のデータを構成に書き込むために使用されます。ステンシルの詳細は、『Oracle Solaris 11.3 でのシステムサービスの開発』 の 第 6 章,「ステンシルを使用した構成ファイルの作成」を参照してください。

次の例は、`configfile` プロパティーグループに含まれているすべての puppet サービスプロパティーを示しています。

```
# svcprop -g configfile puppet
svc:/application/puppet:master/:properties/puppet_stencil/mode astring 0444
svc:/application/puppet:master/:properties/puppet_stencil/path astring /etc/puppet/
puppet.conf
svc:/application/puppet:master/:properties/puppet_stencil/stencil astring puppet.stencil
svc:/application/puppet:agent/:properties/puppet_stencil/mode astring 0444
svc:/application/puppet:agent/:properties/puppet_stencil/path astring /etc/puppet/
puppet.conf
svc:/application/puppet:agent/:properties/puppet_stencil/stencil astring puppet.stencil
```

前の出力では、puppet サービスの両方のインスタンスが、同じ値を持つ同じ `configfile` プロパティーを備えています。各 puppet サービスインスタンスは、構成ファイルへのパス、構成ファイルのモード、およびステンシルファイルへのパスを提供します。

次の例は、これらのインスタンスプロパティーが親サービスから継承されていることを示しています。

```
# svccfg -s puppet listprop -l all puppet_stencil
puppet_stencil          configfile  manifest
puppet_stencil/mode     astring     manifest              0444
puppet_stencil/path     astring     manifest              /etc/puppet/puppet.conf
puppet_stencil/stencil  astring     manifest              puppet.stencil
```

`puppet.conf` ファイルに対して構成の変更を行う場合は、このファイルを手動で編集しないでください。次の例に示すように、代わりに SMF コマンドを使用します。

```
# svccfg -s puppet:agent
svc:/application/puppet:agent> setprop config/report=true
svc:/application/puppet:agent> setprop config/pluginsync=true
svc:/application/puppet:agent> refresh
svc:/application/puppet:agent> exit
```

SMF コマンドを使用して行なった変更はすべて、Puppet エージェントサービスインスタンスを再起動したときに `puppet.conf` ファイルに自動的に反映されます。

```
# svcadm restart puppet:agent
# cat /etc/puppet/puppet.conf
```

```
# WARNING: THIS FILE GENERATED FROM SMF DATA.
#     DO NOT EDIT THIS FILE.  EDITS WILL BE LOST.
#
# See puppet.conf(5) and http://docs.puppetlabs.com/guides/configuring.html
# for details.

[agent]

logdest = /var/log/puppet/puppet-agent.log
pluginsync = true
report = true
```

svccfg(1M) および svcadm(1M) のマニュアルページを参照してください。

puppet.conf ファイルに適用されるすべての構成設定の完全なリストについては、https://docs.puppet.com/puppet/3.6/reference/configuration.html を参照してください。

▼ Puppet マスターおよびエージェントを構成する方法

1 つの Puppet マスターは、Puppet エージェントを実行している多数のノードを制御できます。特定のインフラストラクチャーによっては、複数の Puppet マスターで数千のノードを制御するよう指定することもできます。次の手順では、1 つのマスターと 1 つのエージェントを構成する方法について説明します。

始める前に 次の手順を実行する前に、次を実行します。

- マスターと、Puppet エージェントを実行するすべてのノードの両方に Puppet IPS パッケージをインストールします。23 ページの「Puppet のインストール」を参照してください。
- マスター上で、Puppet マスターの SMF サービスインスタンスを構成して有効にします。

```
# svccfg -s puppet:master setprop config/server=master.company.com
# svcadm enable puppet:master
# svcs puppet:master
```

この出力は、マスターの SMF サービスインスタンスがオンラインであることを示します。これで、Puppet エージェントを実行するノードを構成する準備ができました。

1. **ノード上で、次の手順を実行します。**

   a. **エージェントの SMF config/server プロパティーの値を、マスターを指すように設定します。**

```
# svccfg -s puppet:agent setprop config/server=master.company.com
# svccfg -s puppet:agent refresh
```

   変更を有効にするには、SMF サービスをリフレッシュする必要があります。

注記 - エージェントが証明書要求を発行し、それがマスター上で正常に署名されるまで、エージェントサービスインスタンスを有効にしないでください。

b. エージェントからマスターへの接続をテストします。

```
# puppet agent --test
```

エージェント上で `--test` オプションを指定して `puppet agent` コマンドを実行すると、新しい SSL 鍵が作成され、エージェントとマスターの間の認証に対する要求が設定されます。

2. マスター上で、次の手順を実行します。

a. マスターに接続しようとしているエージェントからの未処理の証明書要求をすべて表示します。

```
# puppet cert list
```

このコマンドの出力は、エージェントによって発行された要求を示します。

b. 要求を発行しているエージェントの証明書に署名します。

```
# puppet cert sign agent
```

注記 - Puppet では証明書への手動での署名が推奨されますが、証明書への手動での署名が絶対に必要とはいえない環境を使用している場合は、代わりに特定の CSR に自動的に署名するように CA Puppet マスターを構成できます。`https://docs.puppet.com` にある Puppet のドキュメントを参照してください。

3. エージェントからマスターへの接続を再テストします。

```
# puppet agent --test
```

この手順により、マスターとエージェントの間の認証が実行されたことが確認されます。

4. Puppet エージェントの SMF サービスインスタンスを有効にします。

```
# svcadm enable puppet:agent
# svcs puppet:agent
```

この出力は、エージェントの SMF サービスインスタンスがオンラインであることを示します。

例 1 Puppet マスターおよびエージェントの構成

次の例は、Puppet マスターおよびエージェントを構成する方法を示しています。

```
# svcs -a | grep puppet
```

```
disabled       16:04:54 svc:/application/puppet:agent
disabled       16:04:55 svc:/application/puppet:master

# svccfg -s puppet:master setprop config/server=master.company.com
root@master:~# svcadm enable puppet:master
root@master:~# svcs puppet:master
STATE          STIME    FMRI
online         17:38:42 svc:/application/puppet:master

# svccfg -s puppet:agent setprop config/server=master.company.com
# svccfg -s puppet:agent refresh

# puppet agent --test
Info: csr_attributes file loading from /etc/puppet/csr_attributes.yaml
Info: Creating a new SSL certificate request for agent.company.com
Info: Certificate Request fingerprint (SHA256): E0:1D:0F:18:72:B7:CE:A7:83:E4:48
:D5:F8:93:36:15:55:0A:B9:C8:E5:B1:CE:D9:3E:0A:68:01:BE:F7:76:47
Exiting; no certificate found and waitforcert is disabled

# puppet cert list
"agent.company.com" (SHA256) E0:1D:0F:18:72:B7:CE:A7:83:E4:48 :D5:F8:93:36:15:55:
0A:B9:C8 :E5:B1:CE:D9:3E:0A:68:01:BE:F7:76:47

# puppet cert sign agent.company.com
Notice: Signed certificate request for agent.company.com
Notice: Removing file Puppet:SSL:CertificateRequest agent at '/etc/puppet/ssl/ca/requests/
solaris.pem'

# puppet agent --test
Info: Caching certificate for agent.company.com
Info: Caching certificate_revocation_list for ca
Info: Caching certificate for agent.company.com
Info: Retrieving plugin
Info: Caching catalog for agent.company.com
Info: Applying configuration version '1400782295'
Notice: Finished catalog run in 0.18 seconds

# svcadm enable puppet:agent
# svcs puppet:agent
STATE          STIME    FMRI
online         18:20:32 svc:/application/puppet:agent
```

次の手順 Puppet をインストールし、必要なすべての構成および検証タスクを実行したら、Puppet を使用してシステム構成を管理する準備ができました。

Puppet を使用したリソースの宣言の詳細は、第3章「Oracle Solaris での Puppet リソースおよびリソースタイプの操作」を参照してください。

Puppet マニフェストを記述する手順については、第4章「Oracle Solaris での Puppet マニフェスト、クラス、およびモジュールの記述」を参照してください。

さまざまな Oracle Solaris システム構成の例については、第5章「Oracle Solaris での Puppet を使用したシステム構成の管理」を参照してください。

Oracle Solaris での Puppet に関する問題のトラブルシューティング

Puppet マスターおよびエージェントサービスは、ほとんどのアクティビティーを `syslog` サービスに記録します。`syslog` 構成は、これらのメッセージがどこに保存されるかを指定します。Oracle Solaris では、デフォルトの場所は `/var/adm/messages` ディレクトリです。ただし、Puppet サービスのログは次の場所に格納されます。

Puppet デーモンの場合、ログは次の場所に格納されます。

```
/var/log/puppet/puppet-master.log
```

```
/var/log/puppet/puppet-agent.log
```

Puppet SMF サービスインスタンスの場合、ログは次の場所に格納されます。

```
/var/svc/log/application-puppet:agent.log
```

```
/var/svc/log/application-puppet:master.log
```

Puppet エージェントのログには、次のコマンドを実行することによってアクセスできます。

```
# svcs -Lv puppet:agent
```

Puppet マスターのログには、次のようにアクセスできます。

```
# svcs -Lv puppet:master
```

次の例は、このコマンドで表示できる情報のタイプを示しています。

```
# svcs -Lv puppet:master
svc:/application/puppet:master (Puppet version 3.6.2)
[ Jul  1 09:47:23 Disabled. ]
[ Jul  1 09:47:26 Rereading configuration. ]
```

♦ ♦ ♦ 第 3 章

3 Oracle Solaris での Puppet リソースおよびリソースタイプの操作

この章では、Puppet リソースおよびリソースタイプについてより詳細に説明したあと、Oracle Solaris で一般的に使用されるリソースタイプの例を示します。また、Puppet コマンド行インタフェース (CLI) を使用してリソースを一覧表示、表示、および変更する方法に関する情報も提供されます。

この章の内容は、次のとおりです。

Puppet リソースおよびリソースタイプについて

Puppet は、リソースおよびリソースタイプを使用してシステムの構成を記述します。リソースは、コレクションにグループ化されます。高レベルでは、リソースはタイプと呼ばれます。これは、Puppet がシステム上で管理できる特定のリソース (ユーザー、パッケージング、ネットワークなど) を記述します。

プラットフォーム固有のレベルでは、リソースはプロバイダと呼ばれます。プロバイダは、指定されたタイプのプラットフォーム固有の実装です。プロバイダは、リソースの一般的な定義を、その特定のプラットフォーム上でリソースを実装するために必要なアクションに変換します。

リソースタイプおよびプロバイダは、Puppet Resource Abstraction Layer (RAL) の中心的な部分です。Oracle Solaris には、特定の Puppet バージョンに適用されるいくつかの組み込み型の Puppet リソースタイプおよびプロバイダのほか、ネットワークやネームサービスなどの特定の側面を管理するために使用できる、個別の Puppet モジュールで配布される追加のリソースタイプが含まれています。また、特定の Oracle Solaris 機能 (たとえば、Oracle Solaris ゾーン) の管理に固有のタイプも存在します。

先に説明したように、Puppet は宣言型言語を使用して、システムリソースとその状態を記述します。これらの情報は、マニフェスト内に書き込まれます。Puppet マスター上にある Puppet サイトマニフェスト (`site.pp`) を使用すると、すべてのノードに共通のグローバル構成を定義できます。

また、特定のノードに関連する構成を定義するために、メインの Puppet マニフェスト内にノード固有のコードを含めることもできます。第4章「Oracle Solaris での Puppet マニフェスト、クラス、およびモジュールの記述」を参照してください。

システム上で使用可能なすべての Puppet リソースタイプは、次のように表示します。

```
# puppet resource --types
address_object
address_properties
augeas
boot_environment
computer
cron
dns
etherstub
exec
file
filebucket
group
host
interface
interface_properties
ip_interface
ip_tunnel
ipmp_interface
k5login
ldap
link_aggregation
link_properties
macauthorization
mailalias
maillist
mcx
mount
nagios_command
nagios_contact
nagios_contactgroup
nagios_host
nagios_hostdependency
nagios_hostescalation
nagios_hostextinfo
nagios_hostgroup
nagios_service
nagios_servicedependency
nagios_serviceescalation
nagios_serviceextinfo
nagios_servicegroup
nagios_timeperiod
nis
notify
nsswitch
package
pkg_facet
pkg_mediator
pkg_publisher
pkg_variant
```

```
protocol_properties
resources
router
schedule
scheduled_task
selboolean
selmodule
service
solaris_vlan
ssh_authorized_key
sshkey
stage
svccfg
tidy
user
vlan
vni_interface
vnic
whit
yumrepo
zfs
zone
zpool
```

前の出力には、Oracle Solaris 固有のタイプと、その他のプラットフォーム上で使用可能なコア Puppet タイプの両方が含まれています。

Puppet リソースタイプの説明

サポートされている各 Puppet プロバイダで使用可能なすべてのリソースタイプと説明を表示するには、`--list` オプションを指定して `puppet describe` コマンドを次のように使用します。

```
# puppet describe --list
These are the types known to puppet:
address_object  - Manage the configuration of Oracle Solaris ad ...
address_properties - Manage Oracle Solaris address properties
augeas          - Apply a change or an array of changes to the  ...
boot_environment - Manage Oracle Solaris Boot Environments (BEs)
computer        - Computer object management using DirectorySer ...
cron            - Installs and manages cron jobs. Every cron re ...
dns             - Manage the configuration of the DNS client fo ...
etherstub       - Manage the configuration of Solaris etherstub ...
exec            - Executes external commands. Any command in an ...
file            - Manages files, including their content, owner ...
filebucket      - A repository for storing and retrieving file  ...
group           - Manage groups. On most platforms this can onl ...
host            - Installs and manages host entries. For most s ...
interface       - This represents a router or switch interface. ...
interface_properties - Manage Oracle Solaris interface properties
ip_interface    - Manage the configuration of Oracle Solaris IP ...
ip_tunnel       - Manage the configuration of Oracle Solaris IP ...
ipmp_interface  - Manage the configuration of Oracle Solaris IP ...
k5login         - Manage the `.k5login` file for a user. Specif ...
ldap            - Manage the configuration of the LDAP client f ...
link_aggregation - Manage the configuration of Oracle Solaris li ...
link_properties - Manage Oracle Solaris link properties
macauthorization - Manage the Mac OS X authorization database. S ...
```

```
mailalias       - Creates an email alias in the local alias dat ...
maillist        - Manage email lists. This resource type can on ...
mcx             - MCX object management using DirectoryService  ...
mount           - Manages mounted filesystems, including puttin ...
nagios_command  - The Nagios type command. This resource type i ...
nagios_contact  - The Nagios type contact. This resource type i ...
nagios_contactgroup - The Nagios type contactgroup. This resource t ...
nagios_host     - The Nagios type host. This resource type is a ...
nagios_hostdependency - The Nagios type hostdependency. This resource ...
nagios_hostescalation - The Nagios type hostescalation. This resource ...
nagios_hostextinfo - The Nagios type hostextinfo. This resource ty ...
nagios_hostgroup - The Nagios type hostgroup. This resource type ...
nagios_service  - The Nagios type service. This resource type i ...
nagios_servicedependency - The Nagios type servicedependency. This resou ...
nagios_serviceescalation - The Nagios type serviceescalation. This resou ...
nagios_serviceextinfo - The Nagios type serviceextinfo. This resource ...
nagios_servicegroup - The Nagios type servicegroup. This resource t ...
nagios_timeperiod - The Nagios type timeperiod. This resource typ ...
nis             - Manage the configuration of the NIS client fo ...
notify          - Sends an arbitrary message to the agent run-t ...
nsswitch        - Name service switch configuration data
package         - Manage packages. There is a basic dichotomy i ...
pkg_facet       - Manage Oracle Solaris package facets
pkg_mediator    - Manage Oracle Solaris package mediators
pkg_publisher   - Manage Oracle Solaris package publishers
pkg_variant     - Manage Oracle Solaris package variants
protocol_properties - Manage Oracle Solaris protocol properties
resources       - This is a metatype that can manage other reso ...
router          - Manages connected router.
schedule        - Define schedules for Puppet. Resources can be ...
scheduled_task  - Installs and manages Windows Scheduled Tasks. ...
selboolean      - Manages SELinux booleans on systems with SELi ...
selmodule       - Manages loading and unloading of SELinux poli ...
service         - Manage running services. Service support unfo ...
solaris_vlan    - Manage the configuration of Oracle Solaris VL ...
ssh_authorized_key - Manages SSH authorized keys. Currently only t ...
sshkey          - Installs and manages ssh host keys. At this p ...
stage           - A resource type for creating new run stages.  ...
svccfg          - Manage SMF service properties with svccfg(1M) ...
tidy            - Remove unwanted files based on specific crite ...
user            - Manage users. This type is mostly built to ma ...
vlan            - Manages a VLAN on a router or switch.
vni_interface   - Manage the configuration of Solaris VNI inter ...
vnic            - Manage the configuration of Oracle Solaris Vi ...
whit            - Whits are internal artifacts of Puppet's curr ...
yumrepo         - The client-side description of a yum reposito ...
zfs             - Manage zfs. Create destroy and set properties ...
zone            - Manages Solaris zones.
zpool           - Manage zpools. Create and delete zpools. The  ...
```

Puppet リソースの宣言について

リソース宣言は、リソースの望ましい状態を記述した式です。Puppet マスターはそのあと、これらのリソース宣言を取得し、それをカタログにコンパイルします。そのカタログをターゲットシステムに適用する場合、Puppet はそこに含まれているすべてのリソースを管理することにより、実際の状態が望ましい状態に確実に一致するようにします。

これに対して、クラスまたは定義済みのタイプの内容と動作を指定する場合は、Puppet 言語を使用して定義します。クラスまたはタイプを定義すると、それがマニフェスト内で宣言対象として使用可能になります。

Puppet では、基本的なリソース宣言に次の形式を使用します。

```
resource_type { 'title':
  attribute1 => 'value1',
  attribute2 => 'value2',
}
```

すべてのリソース宣言で次の形式が使用されます。

- `resource_type` – 宣言されているリソースのタイプです。`resource_type` は、引用符を含まない単語である必要があります。

  `resource_type` には、中括弧 ({) で始まる開始部が含まれています。
- `title` – Puppet が識別のために使用する識別文字列です。どの `resource_type` にもタイトルが必要であり、その文字列はリソースタイプごとに一意である必要があります。このタイトルは、ノード上で管理しているリソースの名前に一致している必要はありません。

  `title` のあとにはコロン (`:`) を付けます。
- `attribute` – リソースの望ましい状態を記述します。ほとんどのリソースには一連の必須の属性がありますが、一連のオプションの属性を含めることもできます。

  属性と値のペアは、次のもので構成されている必要があります。
  - 属性名。引用符を含まない小文字の単語です。

    各属性名は、リソースのある側面を処理します。各リソースタイプには、独自の一連の使用可能な属性があります。
  - 矢印 (`=>`)。「ファットコンマ」または「ハッシュロケット」とも呼ばれます。
  - 値。任意のデータ型にできます。

    値のデータ型は、その属性が何を受け入れるかによって異なります。
  - 最後のコンマ。
- リソース宣言の最後に終了の中括弧 (}) を含める必要があります。

Puppet 言語では、任意の量の空白文字を使用できます。

詳細は、https://docs.puppet.com/puppet/latest/reference/lang_resources.html#resource-types を参照してください。

コマンド行を使用した Puppet リソースの表示および変更

`puppet resource` コマンドを使用して、システムのリソースの状態を表示および変更できます。このコマンドは、現在のシステム状態を Puppet の宣言型言語に変換します。次にこれを使用して、ほかのシステム上で構成を適用できます。

Puppet リソースの状態の表示

次の例は、zone リソースタイプの状態を表示する方法を示しています。

```
# puppet resource zone
zone { 'global':
  ensure   => 'running',
  brand    => 'solaris',
  iptype   => 'shared',
  zonepath => '/',
}
zone { 'myzone':
  ensure   => 'running',
  brand    => 'solaris-kz',
  iptype   => 'excl',
  zonepath => '/system/volatile/zones/myzone/zonepath',
}
```

前の例では、大域ゾーンとインストールされたカーネルゾーンの 2 つの zone リソースが宣言されています。これらの各リソースには、ensure、brand、iptype、zonepath の 4 つの属性があります。各属性には値が関連付けられています。この値は、Puppet の宣言型言語の中心的なコンポーネントです。

次の例は、service リソースタイプの状態を表示する方法を示しています。

```
# puppet resource service svc:/network
/dns/client:default
service {'svc:/network/dns/client:default':
  ensure => 'running',
  enable =>'true',
}
```

Puppet リソースの状態の変更

puppet resource コマンドは、リソースの状態を変更するためにも使用できます。この方法は、Puppet マニフェスト内で構成を直接変更する代わりに使用します。

たとえば、service リソースタイプの状態を次のように変更します。

```
# puppet resource service svc:/network/dns/client:default enable=false
Notice: /Service[svc:/network/dns/client:default]/enable: enable changed 'true' to 'false'
service { 'svc:/network/dns/client:default':
  ensure => 'stopped',
  enable => 'false',
}
```

Facter を使用したシステムに関する情報の収集

システムに関する情報を収集するには、Facter ユーティリティーを使用します。これらの情報は Puppet マスターに送信されたあと、各ノードに適用されるべき構成の変更

を指定するカタログをコンパイルするために Puppet のリソースプロバイダによって使用されます。

カタログではまた、各リソースのあるべき状態を指定することもできます。これらの定義に基づいて、各システムは必要に応じて独自の構成を適用できます。カタログがシステムに適用されたあと、エージェントはレポートを生成し、そのレポートを Puppet マスターに送信します。このレポートには、ターゲットノード上で現在どのリソースが管理されているかに関する情報のほか、望ましい状態を実現するためにそのノードに加えられたすべての変更が含まれています。14 ページの「Puppet の動作のしくみ」を参照してください。

特定のノードで使用可能なすべてのファクトを一覧表示するには、次のコマンドを入力します。

```
# facter -p
architecture => i86pc
facterversion => 2.1.0
hardwareisa => i386
hardwaremodel => i86pc
hostname => myhost
id => root
interfaces => lo0,net0
ipaddress => 10.0.0.15
ipaddress6 => ::
ipaddress_lo0 => 127.0.0.1
ipaddress_net0 => 10.0.0.5
ipaddress_net1 => 10.0.1.5
...
uptime => 0:22 hours
uptime_days => 0
uptime_hours => 0
uptime_seconds => 1320
virtual => virtualbox
```

または、特定のノード (たとえば、`hostname`) の個々のファクトを次のように表示できます。

```
# facter hostname
myhost
```

システムに関するファクトの収集は、特定のシステム上で適用できる構成のタイプを決定するために役立ちます。たとえば、特定のファイルにプラットフォーム固有の内容を設定するファイルリソースを宣言できます。

次の例では、`osfamily` ファクトを使用してファイル内でプラットフォームを宣言しています。

```
$file_contents = $osfamily ? {
  'solaris' => "Hello Oracle Solaris",
  'redhat' => "Hello RHEL",
}

file { '/custom-file.txt':
  ensure => 'present',
  content => $file_contents,
}
```

前の例では、`osfamily` ファクトを使用して新しい `$file_contents` 変数が作成され、条件付きチェックが提供されました。そのあと、プラットフォームに応じて、このファイルに別の内容を割り当てます。

詳細は、`https://docs.puppet.com/facter/` を参照してください。

♦ ♦ ♦ 第 4 章

Oracle Solaris での Puppet マニフェスト、クラス、およびモジュールの記述

Puppet マニフェスト、クラス、およびモジュールは、Puppet がインフラストラクチャー内のシステム構成を定義するために使用するものです。この章では、マニフェスト、クラス、およびモジュールの記述の基本について説明します。

この章の内容は、次のとおりです。

Puppet サイトマニフェストの記述

Puppet をインストールして構成したら、Puppet エージェントを実行しているノードを制御するための Puppet マニフェストを記述できます。Puppet マニフェストは、Ruby に似た Puppet 固有の言語で記述されます。ここで、各マニフェストは `.pp` ファイル拡張子を使用します。

Puppet サイトマニフェスト (`site.pp`) は、Puppet がグローバルシステム構成を定義するために使用するメインのファイルです。サイトマニフェストは、すべてのノードに適用される構成を定義します。これは、DNS サーバー、LDAP 構成、すべてのノードに共通のその他のサイト全体にわたる設定などの、システム全体にわたる構成の管理に最適です。

サイトマニフェストには、特定のノードに適用されるノード固有のコードのブロックも含めることができます。この機能を使用すると、サイトマニフェスト内で特定のノードに特定の構成を割り当てることができます。42 ページの「ノード固有のコードを指定する Puppet マニフェストの記述」を参照してください。

注記 - site.pp マニフェストは、デフォルトでは Puppet マスター上に存在しません。最初にこのファイルを作成し、マスター上の /etc/puppet/manifests/ ディレクトリ内に格納する必要があります。

▼ Puppet サイトマニフェストを記述する方法

次の手順では、構成をインフラストラクチャー内でグローバルに適用するための Puppet サイトマニフェストを記述する方法について説明します。

始める前に Puppet サイトマニフェストを記述する前に、次を実行する必要があります。

- マニフェスト内でどのリソースタイプを宣言するかを決定します。この情報は、指定されたリソースタイプで使用可能なすべての属性とパラメータを表示する puppet describe *resource-type* コマンドを使用して入手できます。

  ```
  # puppet describe resource-type
  ```

  33 ページの「Puppet リソースタイプの説明」を参照してください。
- Puppet マニフェスト内でリソースを宣言するために使用する基本的な構文について理解します。34 ページの「Puppet リソースの宣言について」を参照してください。詳細は、https://docs.puppet.com/puppet/latest/reference/lang_resources.html を参照してください。
- Puppet マニフェスト内で特定の Oracle Solaris システム構成を定義するために使用する構文について理解します。例については、第5章「Oracle Solaris での Puppet を使用したシステム構成の管理」を参照してください。

1. **Puppet Management 権利プロファイルが割り当てられた管理者になるか、または root 役割になります。**

   『Oracle Solaris 11.3 でのユーザーとプロセスのセキュリティー保護』 の 「割り当てられている管理権利の使用」を参照してください。

2. **Puppet マスター上で site.pp ファイルを作成します。**

   ```
   # touch /etc/puppet/manifests/site.pp
   ```

   このファイルは、マスター上の /etc/puppet/manifests ディレクトリ内に常に存在させてください。

3. **Puppet サイトマニフェスト (site.pp) 内で指定された構成を定義し、変更を保存します。**

   詳細は、第3章「Oracle Solaris での Puppet リソースおよびリソースタイプの操作」を参照してください。

4. **site.pp ファイルに加えた構成の変更が永続的に適用される前に、それらの変更をテストします。**

```
# puppet apply -v --noop /etc/puppet/manifests/site.pp
```

apply	マスター上の Puppet マニフェストに構成を適用します。
-v	冗長モードを使用することを示します。
-noop	変更を実際に適用せずにドライランを実行できるようにします。このオプションは、テストのために使用します。

各ノード上で実行されている Puppet エージェントは、構成の変更がないかどうかマスターに定期的に照会したあと、必要なすべての変更をそのノードに適用します。

5. **各ノード上のログファイル (/var/log/puppet/puppet-agent.log) をチェックして、最新の構成の変更が取得されていることを確認します。**

6. **(オプション) 最新の構成の変更を手動で適用するには、ノード上で次のコマンドを実行します。**

```
# puppet agent -t
```

-t (--test) オプションを指定すると、詳細ロギングが有効になります。これにより、エージェントデーモンはフォアグラウンドに残り、マスターサーバーの構成が (構文エラーが発生した場合のように) 無効である場合は終了し、そのあとその構成を 1 回実行したあとに終了します。

使用可能なすべての Puppet サブコマンドを表示するには、/usr/sbin/puppet help agent コマンドを使用します。puppet(8) のマニュアルページも参照してください。

例 2 Puppet マニフェストの記述

次の例は、Puppet サイトマニフェスト内でリソースを宣言する方法を示しています。この例では、site.pp ファイルがすでに作成され、Puppet マスター上の正しいディレクトリ内に格納されていることを前提にしています。

最初に、site.pp ファイル内でリソースを宣言します。この例では、file リソースタイプが宣言されています。このリソースタイプの場合、ensure と content の 2 つの属性が指定されています。これらの 2 つの属性は、そのノード上の root ディレクトリ内に custom-file.txt ファイルが存在し、そのファイルに単語「Hello World」が含まれていることを確認します。

```
file { '/custom-file.txt':
  ensure => 'present',
  content => "Hello World",
}
```

site.pp ファイルを保存したあと、マスター上の構成の有効性を次のようにテストできます。

```
# puppet apply -v --noop /etc/puppet/manifests/site.pp
Notice: Compiled catalog for master in environment production in 0.16 seconds
Info: Applying configuration version '1400794990'
Notice: /Stage[main]/Main/File[/custom-file.txt]/ensure: current_value absent, should be
 present (noop)
Notice: Class[Main]: Would have triggered 'refresh' from 1 events
Notice: Stage[Main]: Would have triggered 'refresh' from 1 events
Notice: Finished catalog run in 0.27 seconds
```

-v オプションは冗長モードを使用することを指定し、-noop オプションは、実際にはどの変更も行われないようにします。テストのために -noop オプションを使用すると、マニフェストに対して変更を実際に適用せずにドライランを実行できます。

各ノード上で実行されている Puppet エージェントは、構成の変更がないかどうかマスターに定期的に照会したあと、必要に応じて新しい変更をすべて適用します。ノードのログファイル (`/var/log/puppet/puppet-agent.log`) をチェックして、そのノードが最新の変更を適用したことを確認できます。

```
# ls -la /custom-file.txt
-rw-------   1 root     root          16 Mar 22 21:50 /custom-file.txt
# cat /custom-file.txt
Hello World
# tail /var/log/puppet/puppet-agent.log
....
2016-03-22 21:50:17 +0000 /Stage[main]/Main/File[/custom-file.txt]/ensure (notice): created
2016-03-22 21:50:17 +0000 Puppet (notice): Finished catalog run in 0.21 seconds
```

前の出力は、その構成がノードに適用されていることを示しています。デフォルトでは、エージェントは構成の変更がないかどうか 30 分間隔でマスターをポーリングします。また、`custom-file.txt` ファイルがノード上に存在するかどうかをチェックすることによって構成を確認することもできます。

オプションで、ノード上で次のコマンドを実行することによって構成の変更を手動で適用します。

```
# puppet agent -t
```

Puppet を使用して Oracle Solaris システム構成を定義する方法を示す具体的な例については、第5章「Oracle Solaris での Puppet を使用したシステム構成の管理」を参照してください。

ノード固有のコードを指定する Puppet マニフェストの記述

さまざまなシステムの構成を管理している場合は、マニフェスト内で条件付きロジックを指定することも考慮できます。これにより、各システムが適切な構成に正しく一致していることが保証されます。

このロジックを適用するには、サイトマニフェスト内で node キーワードを使用します (このマニフェストは .pp ファイル拡張子を持つ 1 つのファイルでも、.pp ファイル拡張子を持つ複数のファイルが含まれたディレクトリでもかまいません)。ノードの宣

言では任意の Puppet コードを指定できますが、変数割り当てとクラス宣言のみを含めることをお勧めします。

次の例は、2 つのノード `agent1.company.com` と `agent2.company.com` に関して同一の構成に一致させる方法を示しています。

```
node ‘agent1.company.com’, ‘agent2.company.com’ {
   # Include resources here
}
```

次の例は、2 つのノードに関して同一の構成に一致させるために使用する構文と、3 番目のノード (`agent3.company.com`) のための別のリソース定義を示しています。

```
node 'agent1.company.com', 'agent2.company.com' {

  # Include resources here
}
node 'agent3.company.com' {

  # Include other resources here

}
```

Puppet はまた、`default` と呼ばれる特殊なノードも提供しています。これにより、既存のノード定義に一致しないすべてのノードのフォールバック構成が可能になります。これらのノードのフォールバック構成は、次のように定義します。

```
node default {
  # Include other resources here
}
```

ノード固有のコードを含むマニフェストの記述の詳細は、https://docs.puppet.com/puppet/3.8/reference/lang_classes.html を参照してください。

Puppet クラスの記述

クラスは、再利用を可能にする Puppet コードのブロックです。クラスを使用すると、マニフェストを読むときの複雑さが軽減されます。クラス定義には、特定のクラスのためのコードが含まれています。最初にクラスを定義してから、そのクラスをマニフェスト内で使用できるようにします。クラス自体は何も評価を実行しません。

次の例は、`examplecloud` という名前のクラス定義に使用される形式を示しています。

```
class examplecloud::analytics {

    package { "system/management/webui/webui-server":
        ensure => installed,
    }

    svccfg { "webui":
        require => Package["system/management/webui/webui-server"],
```

```
        fmri => "system/webui/server:default",
        property => "conf/redirect_from_https",
        value => "false",
        ensure => present,
    }

    service { "system/webui/server":
        require => Package["system/management/webui/webui-server"],
        ensure => running,
    }

}
```

この例では、クラスに `examplecloud` と `analytics` という 2 つの名前空間があります。このクラスで指定されているコードによって、ノード上で `analytics` SMF サービスを有効にする前に特定の IPS パッケージがインストールされ、特定の SMF 構成が適用されることが保証されます。

クラス宣言は、マニフェスト内で定義されているクラスです。クラス宣言は、Puppet にそのクラス内のコードを評価するよう指示します。

クラス宣言には、通常とリソースライクの 2 つのタイプがあります。

- 通常のクラス宣言の場合は、次の例に示すように `include` キーワードが Puppet コードに含まれています。

  `include` *example_class*

- リソースライクのクラス宣言の場合は、次の例に示すように、リソースが宣言される方法と同様にクラスが宣言されます。

  `class { '`*example_class*`': }`

  クラスパラメータを指定するには、リソースライクのクラス宣言を使用します。これらのパラメータは、クラス属性のデフォルト値をオーバーライドします。

Puppet クラスの記述および割り当ての詳細は、https://docs.puppetlabs.com/puppet/3.8/reference/lang_classes.html を参照してください。

例 3 Puppet マニフェストへのクラス宣言の追加

次のマニフェストの例では、Puppet マスター上の `/puppet/modules` ディレクトリ内にある `exampleloud` という名前のクラス宣言を使用します。

`examplecloud` クラスの下に、さまざまな構成を指定するマニフェスト (`/puppet/modules/examplecloud/manifests`) がいくつか存在します。次の例に示すように、各マニフェストには `examplecloud` クラス宣言が含まれています。

```
# NTP configuration for companyfoo
class examplecloud::ntp {

    file { "ntp.conf" :
        path => "/etc/inet/ntp.conf",
        owner => "root",
        group => "root",
```

```
        mode => 644,
        source => "puppet:///modules/examplecloud/ntp.conf",
    }

    package { "ntp":
        ensure => installed,
    }

    service { "ntp":
        require => File["ntp.conf"],
        subscribe => File["ntp.conf"],
        ensure => running,
    }

}
```

前の例にある `examplecloud` クラスの宣言では、次のことが保証されます。

- NTP パッケージがインストールされる
- 特定の構成ファイル (これは Puppet マスター以外の場所がソースになっています) がインストールされる
- NTP サービスが有効になり、ノード上で実行状態になる

Puppet モジュールの記述

Puppet モジュールは、マニフェストとデータのコレクションであり、ファクト、ファイル、およびテンプレートを含めることができます。モジュールを使用すると、コードをいくつかのマニフェストに分割できるため、Puppet コードを整理したり再利用したりするために役立ちます。すべてのノードのためのグローバル構成を含むメインの `site.pp` マニフェストを除き、ほぼすべての Puppet マニフェストをモジュールに含めてください。複数の Puppet マニフェストがある場合は、それらを整理する方法としてモジュールを使用することを考慮してください。

注意 - IPS を通して提供されるモジュールは、特に Oracle Solaris のために更新されます。これらのモジュールを Puppet Forge モジュールに置き換えないでください。

独自の Puppet モジュールを記述するには、Puppet マスター上で次のコマンドを実行して起動します。

```
# puppet module generate module-name
```

前のコマンドを実行すると、一連の質問がプロンプトに表示されます。Puppet は、ユーザーの応答を使用してモジュールに関する情報を収集したあと、基本的なモジュール構造を作成します。詳細な手順および例については、https://docs.puppet.com/guides/module_guides/bgtm.html を参照してください。

作成した Puppet モジュールをマスター上の `/etc/puppet/modules` ディレクトリに追加します。ここで、基本的なディレクトリツリー構造を次に示します。

module-name/ – モジュールの名前を指定するもっとも外側の (または親の) ディレクトリ構造です。

- `manifests/` – モジュール内のすべてのマニフェストが含まれています。
  - `init.pp` – クラス定義が含まれています。クラス定義の名前はモジュールの名前に一致している必要があります。
  - `other_class.pp` – `my_module::my_defined_type` という名前の定義済みのタイプが含まれています。
  - `my_defined_type.pp` – `my_module::other_class` という名前のクラスが含まれています。
  - `my_module::my_defined_type` – `my_module::my_defined_type` という名前の定義済みのタイプが含まれています。
  - `implementation/` – その下に格納されているクラス名に影響を与える名前を持つディレクトリです。
    - `foo.pp` – `my_module::implementation::foo` という名前のクラスが含まれています。
    - `bar.pp` – `my_module::implementation::bar` という名前のクラスが含まれています。
- `files`/ – 管理対象ノードがダウンロードできる静的ファイルが含まれています。

  `service.conf` – `puppet:///modules/my_module/service.conf` のようなソース URL を含むファイルです。このファイルの内容には、`my_module/service.conf` などのファイル機能を使用してアクセスできます。
- `lib/` – Puppet マスターサーバーと Puppet エージェントサービスの両方で使用されるプラグイン (カスタムファクトやリソースタイプなど) が含まれています。
- `facts.d/` – Ruby ベースのカスタムファクトの代わりに使用できる外部ファクトが含まれています。
- `templates/` – モジュールのマニフェストが使用できるテンプレートが含まれています。
  - `component.erb` – マニフェストが `my_module/component.erb` として表すことのできるテンプレートです。
  - `component.epp` – マニフェストが `my_module/component.epp` として表すことのできるテンプレートです。
- `examples`/ – モジュールのクラスや定義済みのタイプを宣言する方法を示す例が含まれています。
  - `init.pp`
  - `other_example.pp` – 主要なユースケースの例が含まれています。
- `spec`/ – `lib` ディレクトリ内にあるすべてのプラグインのテストが含まれています。

次の例に示すように、`/etc/puppet/modules` ディレクトリの下に `examplecloud` という名前のモジュールが存在します。

```
# cd /etc/puppet/modules
# ls -al
drwxrwxr-x   3 userfoo   staff          3 Mar  4 14:44 .
drwxr-xr-x   5 userfoo   staff          6 Mar 25 06:33 ..
drwxr-xr-x   4 userfoo   staff          4 Mar  3 13:24 examplecloud
# cd examplecloud
# ls -al
drwxr-xr-x   4 userfoo   staff          4 Mar  3 13:24 .
drwxrwxr-x   3 userfoo   staff          3 Mar  4 14:44 ..
drwxr-xr-x   3 userfoo   staff         12 Mar  9 11:55 files
drwxr-xr-x   2 userfoo   staff         12 Mar 24 15:43 manifests
```

examplecloud ディレクトリの下に、そのモジュールのマニフェストを含む manifests ディレクトリが存在します。次の出力に示すように、各マニフェストには 1 つのクラスまたは定義済みのタイプが含まれています。

```
# cd /etc/puppet/modules/examplecloud/manifests
# ls -al
total 52
drwxr-xr-x   2 userfoo   staff         12 Mar 24 15:43 .
drwxr-xr-x   4 userfoo   staff          4 Mar  3 13:24 ..
-rw-r--r--   1 userfoo   staff        552 Mar  3 13:24 analytics.pp
-rw-r--r--   1 userfoo   staff       1097 Mar  3 13:24 compute_node.pp
-rw-r--r--   1 userfoo   staff       1232 Mar  7 12:45 dlmp_aggr.pp
-rw-r--r--   1 userfoo   staff        491 Mar  3 13:24 mysql.pp
-rw-r--r--   1 userfoo   staff       1764 Mar  7 12:45 nameservice.pp
-rw-r--r--   1 userfoo   staff       1073 Mar 24 15:43 neutron_aggr.pp
-rw-r--r--   1 userfoo   staff        463 Mar  3 13:24 ntp.pp
-rw-r--r--   1 userfoo   staff       1814 Mar  3 13:24 openstack_horizon.pp
-rw-r--r--   1 userfoo   staff        690 Mar  3 13:24 rabbitmq.pp
-rw-r--r--   1 userfoo   staff       1688 Mar 14 14:34 storage_ip.pp
```

マニフェストファイルの名前は、そこに含まれるクラスおよび定義済みのタイプの名前にマップされます。examplecloud/manifests ディレクトリの下にある各サブディレクトリには特定の機能があります。

これらの各コンポーネントのより詳細な説明については、https://docs.puppet.com/puppet/3.6/reference/modules_fundamentals.html#example を参照してください。

Puppet Forge サイトには、公式に使用可能なモジュール (より新しいモジュールを含む) のリポジトリのほか、ユーザーがダウンロードできるオーサリングツールやドキュメントが含まれています。

♦ ♦ ♦ 第 5 章

Oracle Solaris での Puppet を使用したシステム構成の管理

この章では、Puppet を使用して Oracle Solaris システム構成を管理できるいくつかの方法を示すエンドツーエンドの例を提供します。

次の例では、マスターサーバーとすべてのノードに Puppet がすでにインストールおよび構成されていることを前提にしています。次の例ではまた、Puppet サイトマニフェストが以前に作成され、このファイルが Puppet マスター上に存在することも前提にしています。

この章の内容は、次のとおりです。

Puppet 構成管理のワークフロー

Puppet を使用して Oracle Solaris システム構成を管理するには通常、次の基本的なプロセスに従います。

1. `puppet describe` コマンドを使用して、構成することを予定している、指定されたリソースタイプの使用可能なすべての属性を表示します。第3章「Oracle Solaris での Puppet リソースおよびリソースタイプの操作」を参照してください。
2. マスター上に存在する Puppet マニフェスト内で適切なリソースを宣言します。

   すべてのノードに適用されるグローバルシステム構成を定義するには、Puppet サイトマニフェスト (`site.pp`) を使用します。

   `node` キーワードを使用して、Puppet サイトマニフェスト内でノード固有の構成を定義することもできます。42 ページの「ノード固有のコードを指定する Puppet マニフェストの記述」を参照してください。

3. ドライランを実行して、Puppet マニフェストで定義されている構成が有効かどうかをテストします。

   この手順は必須ではありませんが、ベストプラクティスとして推奨されます。
4. ノードが構成を適用したことを確認します。

段階的な手順については、40 ページの「Puppet サイトマニフェストを記述する方法」を参照してください。

Puppet を使用したパッケージングの構成

次の例は、マニフェスト内で Puppet `package` リソースタイプを宣言することによって新しい IPS ソフトウェアパッケージ (`nmap`) を追加する方法を示しています。

例 4 Puppet を使用したパッケージングの構成

最初に、インストールする予定のパッケージがすでにインストールされているかどうかを判定します。

```
$ pkg info nmap
pkg: info: no packages matching the following patterns you specified are
installed on the system. Try specifying -r to query remotely:
```

パッケージがインストールされているかどうかをリモートでチェックする場合は、`-r` オプションを次のように使用します。

```
# pkg info -r nmap
Name: diagnostic/nmap
       Summary: Network exploration tool and security / port scanner.
   Description: Nmap is useful for inventorying the network, managing service
                upgrade schedules, and monitoring host or service uptime.
      Category: System/Administration and Configuration
         State: Not installed
     Publisher: solaris
       Version: 6.25
 Build Release: 5.11
        Branch: 0.175.3.0.0.30.0
Packaging Date: Fri Aug 21 16:46:42 2015
          Size: 19.07 MB
          FMRI: pkg://solaris/diagnostic/nmap@6.25,5.11-0.175.3.0.0.30.0:20150821T164642Z
```

次に、`puppet describe` コマンド (次の部分的な出力例に示されています) を使用して、`package` リソースタイプに対して宣言する適切な属性をチェックします。

```
# puppet describe package

package
=======
Manage packages.  There is a basic dichotomy in package
support right now:  Some package types (e.g., yum and apt) can
retrieve their own package files, while others (e.g., rpm and sun)
```

```
cannot.  For those package formats that cannot retrieve their own files,
you can use the `source` parameter to point to the correct file.
Puppet will automatically guess the packaging format that you are
using based on the platform you are on, but you can override it
using the `provider` parameter; each provider defines what it
requires in order to function, and you must meet those requirements
to use a given provider.
**Autorequires:** If Puppet is managing the files specified as a
package's `adminfile`, `responsefile`, or `source`, the package
resource will autorequire those files.


Parameters
----------

- **adminfile**
    A file containing package defaults for installing packages.
    This is currently only used on Solaris.  The value will be
    validated according to system rules, which in the case of
    Solaris means that it should either be a fully qualified path
    or it should be in `/var/sadm/install/admin`.

- **allow_virtual**
    Specifies if virtual package names are allowed for install and
    uninstall.
    Valid values are `true`, `false`, `yes`, `no`.
    Requires features virtual_packages.

- **allowcdrom**
    Tells apt to allow cdrom sources in the sources.list file.
    Normally apt will bail if you try this.
    Valid values are `true`, `false`.

- **category**
    A read-only parameter set by the package.

- **configfiles**
    Whether configfiles should be kept or replaced.  Most packages
    types do not support this parameter. Defaults to `keep`.
    Valid values are `keep`, `replace`.

- **description**
    A read-only parameter set by the package.

- **ensure**
    What state the package should be in. On packaging systems that can
    retrieve new packages on their own, you can choose which package to
    retrieve by specifying a version number or `latest` as the ensure
    value. On packaging systems that manage configuration files separately
    from "normal" system files, you can uninstall config files by
    specifying `purged` as the ensure value. This defaults to `installed`.
    Valid values are `present` (also called `installed`), `absent`,
    `purged`, `held`, `latest`. Values can match `/./`.
.
.
.
```

そのあと、マスター上の Puppet マニフェスト内でリソースタイプを次のように宣言します。

```
package { 'nmap':
  ensure => 'present',
}
```

前の例では、リソース定義のタイトルが nmap (インストールされるパッケージ) に設定され、ensure 属性の値が present に設定されています。これは、そのパッケージがインストールに使用可能かどうかをチェックします。

この構成は、次のように確認されます。

```
# pkg info nmap
Name: diagnostic/nmap
       Summary: Network exploration tool and security / port scanner.
   Description: Nmap is useful for inventorying the network, managing service
                upgrade schedules, and monitoring host or service uptime.
      Category: System/Administration and Configuration
         State: Installed
     Publisher: solaris
       Version: 6.25
 Build Release: 5.11
        Branch: 0.175.3.0.0.30.0
Packaging Date: Fri Aug 21 16:46:42 2015
          Size: 19.07 MB
          FMRI: pkg://solaris/diagnostic/nmap@6.25,5.11-0.175.3.0.0.30.0:20150821T164642Z
```

前のコマンドの出力は、nmap パッケージが現在ノードにインストールされていることを示しています。このパッケージは、Puppet エージェントが実行されたときにインストールされます。または、構成の変更を手動で適用するために、ノード上で puppet agent -t コマンドを実行できます。

nmap パッケージをアンインストールしても、Puppet はそのパッケージをノードに再インストールすることによって、指定された構成を適用します。

Puppet を使用した ZFS ファイルシステムの構成

次の例は、zfs リソースタイプを使用して Puppet マニフェスト内で ZFS ファイルシステム構成を定義する方法を示しています。

例 5 Puppet を使用した ZFS ファイルシステムの構成

最初に、zfs リソースタイプに対して宣言できるすべての属性のリストを次のように表示します。

```
# puppet describe zfs
zfs
===
Manage zfs. Create destroy and set properties on zfs instances.
**Autorequires:** If Puppet is managing the zpool at the root of this zfs
instance, the zfs resource will autorequire it. If Puppet is managing any
parent zfs instances, the zfs resource will autorequire them.


Parameters
----------
```

```
- **aclinherit**
    The aclinherit property. Valid values are `discard`, `noallow`,
    `restricted`, `passthrough`, `passthrough-x`.

- **aclmode**
    The aclmode property. Valid values are `discard`, `groupmask`,
    `passthrough`.

- **atime**
    The atime property. Valid values are `on`, `off`.

- **canmount**
    The canmount property. Valid values are `on`, `off`, `noauto`.

- **checksum**
    The checksum property. Valid values are `on`, `off`, `fletcher2`,
    `fletcher4`, `sha256`.

- **compression**
    The compression property. Valid values are `on`, `off`, `lzjb`, `gzip`,
    `gzip-[1-9]`, `zle`.

- **copies**
    The copies property. Valid values are `1`, `2`, `3`.

- **dedup**
    The dedup property. Valid values are `on`, `off`.

- **devices**
    The devices property. Valid values are `on`, `off`.

- **ensure**
    The basic property that the resource should be in.
    Valid values are `present`, `absent`.
.
.
.
```

次に、マニフェスト内で、次のパラメータを使用して `zfs` リソースタイプを宣言します。`readonly` と呼ばれる属性が追加され、`on` に設定されています。

```
zfs { 'rpool/test':
  ensure => 'present',
  readonly => 'on',
}
```

ノード上で次のコマンドを実行することによって構成を確認します。

```
# zfs list rpool/test
NAME        USED  AVAIL   REFER   MOUNTPOINT
rpool/test   31K  31.8G     31K   /rpool/test

# zfs get readonly rpool/test
NAME        PROPERTY  VALUE  SOURCE
rpool/test  readonly  on     local
```

Puppet を使用したネットワークパラメータの構成

次の例は、Puppet を使用してネットワーク構成を管理する方法を示しています。この例では、Puppet マニフェスト内でネットワーク関連のさまざまなリソースタイプが宣言されています。

例 6 Puppet を使用したネットワークパラメータの構成

次の例は、Puppet マニフェスト内で複数のネットワーク構成パラメータを指定する方法を示しています。

```
# Force link speed negotiation to be at least 1 Gb
        link_properties { "net0":
            ensure => present,
            properties => { en-100fdx-cap => "0" },
        }

        link_properties { "net1":
            ensure => present,
            properties => { en-100fdx-cap => "0" },
        }

        link_aggregation { "aggr0" :
            ensure => present,
            lower_links => [ 'net0', 'net1' ],
            mode => "dlmp",
        }

        ip_interface { "aggr0" :
            ensure => present,
            require => Link_aggregation['aggr0'],
        }

        ip_interface { "net0":
            ensure => absent,
            before => Link_aggregation['aggr0'],
        }

        address_object { "net0":
            ensure => absent,
            before => Ip_interface['net0'],
        }

        address_object { 'aggr0/v4':
            require => Ip_interface['aggr0'],
            ensure => present,
            address => "${myip}/24",
            address_type => "static",
            enable => "true",
        }

}
```

Puppet を使用したネームサービスの構成

次の例は、Puppet マニフェスト内で service リソースタイプを宣言することにより Puppet を使用してネームサービス構成を管理する方法を示しています。

例 7 Puppet を使用したネームサービスの構成

次の例では、DNS サービスが有効になり、DNS サーバーが構成されています。次に、domainname プロパティーが設定されています。最後に、ネームサービススイッチの値が指定されています。

```
service { "dns/client":
ensure => running,
}

svccfg { "domainname":
  ensure => present
  fmri => "svc:/network/nis/domain",
  property => "config/domainname",
  type => "hostname",
  value => "company.com",
  notify => Service[‘dns/client'],
}

svccfg { "nameserver":
  ensure => present,
  fmri: => "svc:/network/dns/client",
  property => "config/nameserver",
  type => "net_address",
  value => "1.2.3.4"
  notify => Service[‘dns/client'],
}

# nameservice switch
nsswitch { "dns + ldap":
default => "files",
host => "files dns",
password => "files ldap",
group => "files ldap",
automount => "files ldap",
netgroup => "ldap",
}
```

Puppet を使用した Oracle Solaris ゾーンの構成

次の例は、Puppet マニフェスト内で zone リソースタイプを宣言することによって Oracle Solaris ゾーン構成を定義する 1 つの方法を示しています。

例 8 Puppet を使用した Oracle Solaris ゾーンの構成

`puppet describe` コマンド (次の部分的な出力例に示されています) を実行することによって、最初に `zone` リソースタイプに対して宣言できるすべての属性のリストを表示します。

```
# puppet describe zone
zone
====
Manages Solaris zones.


Parameters
----------

- **archive**
    The archive file containing an archived zone.

- **archived_zonename**
    The archived zone to configure and install

- **brand**


    The zone's brand type

- **clone**
    Instead of installing the zone, clone it from another zone.
    If the zone root resides on a zfs file system, a snapshot will be
    used to create the clone; if it resides on a ufs filesystem, a copy of
    the
    zone will be used. The zone from which you clone must not be running.

- **config_profile**
    Path to the config_profile to use to configure a solaris zone.
    This is set when providing a sysconfig profile instead of running the
    sysconfig SCI tool on first boot of the zone.


- **ensure**
    The running state of the zone.  The valid states directly reflect
    the states that `zoneadm` provides.  The states are linear,
    in that a zone must be `configured`, then `installed`, and
    only then can be `running`.  Note also that `halt` is currently
    used to stop zones.
    Valid values are `absent`, `configured`, `installed`, `running`.
.
.
.
- **zonecfg_export**
    Contains the zone configuration information. This can be passed in
    in the form of a file generated by the zonecfg command, in the form
    of a template, or a string.

- **zonepath**
    The path to zone's file system.

Providers
---------
    solaris
```

zonecfg_export 属性 (前の出力に示されています) を使用すると、zonecfg コマンドを使用してゾーン構成ファイルリソースを次のように作成できます。

```
# zonecfg -z testzone1
Use 'create' to begin configuring a new zone.
zonecfg:testzone> create
create: Using system default template 'SYSdefault'
zonecfg:testzone> export -f /tmp/zone.cfg
zonecfg:testzone> exit
root@master:~# cat /tmp/zone.cfg
create -b
set zonepath=/system/zones/%{zonename}
set autoboot=false
set autoshutdown=shutdown
set ip-type=exclusive
add anet
set linkname=net0
set lower-link=auto
set configure-allowed-address=true
set link-protection=mac-nospoof
set mac-address=auto
end
root@master:~# cp /tmp/zone.cfg /etc/puppet/modules/mycompany
```

作成したゾーンは、zone リソースタイプが適用されたときに構成可能になります。Puppet マニフェスト内で zone リソースタイプを次のように宣言します。

```
zone { 'systemazone':
  zonecfg_export => 'puppet:///modules/mycompany/zone.conf',
  ensure => 'running',
}
```

ここでは、ensure 属性の値が installed に設定されています。ensure の値は、ゾーンに対して受け入れ可能なステータス (installed および running) に一致しています。この例では、systemazone と呼ばれるゾーンがノード上に作成されています。

最後の手順として、ノードが自身に構成を適用したことを確認します。

```
# zoneadm list -cv
  ID NAME             STATUS       PATH                         BRAND       IP
   0 global           running      /                            solaris     shared
   - systemazone      running     /system/zones/systemazone  solaris     excl
```

前のコマンドの出力は、非大域ゾーン systemazone が構成およびインストールされ、実行されていることを示しています。

索引

た

な

は

ま

や

ら

C

D

E

F

I

N

O

P

S

Z