Using Puppet to Perform Configuration Management in Oracle® Solaris 11.3

Part No: E77676
September 2018

ORACLE®

Using Puppet to Perform Configuration Management in Oracle Solaris 11.3

Part No: E77676

Référence: E77676

Accès aux services de support Oracle

Les clients Oracle qui ont souscrit un contrat de support ont accès au support électronique via My Oracle Support. Pour plus d'informations, visitez le site `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` ou le site `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` si vous êtes malentendant.

Contents

Using This Documentation

- **Overview** – Provides information about how to configure and administer various network components in the Oracle Solaris operating system (OS), such as datalinks, IP interfaces and addresses, naming and directory services, reactive profiles, and wireless networks.
- **Audience** – System administrators who are responsible for managing network configuration in corporate datacenters.
- **Required knowledge** – Basic and advanced network administration concepts and practices.

Product Documentation Library

Documentation and resources for this product and related products are available at http://www.oracle.com/pls/topic/lookup?ctx=E53394-01.

Feedback

Provide feedback about this documentation at http://www.oracle.com/goto/docfeedback.

♦ ♦ ♦ CHAPTER 1

About Performing Configuration Management With Puppet in Oracle Solaris

Puppet is cross-platform software that you can use to automate and enforce the configuration management of most major subsystems in Oracle Solaris, including Oracle Solaris servers and their subsystems. You can use Puppet to perform several common system configuration tasks. Puppet enables you to standardize and enforce resource configurations across your entire IT infrastructure.

This chapter provides an overview of Puppet's infrastructure, as well as a basic description of how Puppet is implemented in Oracle Solaris.

This chapter includes the following topics:

For information that is beyond the scope of this guide, including background information and more complete descriptions of the various Puppet offerings, go to the Puppet web site.

You can find specific information about the Puppet version that is supported in Oracle Solaris 11.3 in the Puppet 3.6 Reference Manual.

Highlights of Puppet Support in Oracle Solaris 11.3

The following Puppet features and functionality are supported in Oracle Solaris:

- **Puppet installation**

  The Puppet software package (`system/management/puppet`) is not installed by default on your Oracle Solaris system. You must individually install the same Puppet Image Packaging

System (IPS) package on the Puppet master and all of the nodes that will run the Puppet agent.

See Chapter 2, “Getting Started With Puppet in Oracle Solaris”.

- **Puppet modules and utilities**

  When you install the Puppet IPS package, you get all of the core Puppet modules, as well as other modules that are specific to the Oracle Solaris release.

  To display a complete list of all of the installed and available modules that are included in a Puppet installation, run the following command:

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

  For detailed information about a specific Puppet module, refer to the `README` file that is included with that module. You can also go to the Puppet web site to search for more information about a given module.

  When you install Puppet, you also get the following utilities, which are designed to work in concert with Puppet:

  - **Facter** – Is a utility that Puppet uses to *discover* facts about a particular system, for example, OS type, CPUs, memory size, and so on. The information that Facter gathers about a system is sent to the Puppet master, which the Puppet master then uses to compile *catalogs* that describe a desired system state for a specific set of resources. The catalog lists all of the resources that must be managed and any dependencies between those resources. See “Gathering Information About a System by Using Facter” on page 36.
  - **Hiera** – Is a cross-platform, key/value lookup tool that you use to manage configuration data. You use Hiera along with Puppet to maintain site-specific data that would normally be included in a Puppet manifest. Storing site-specific data in a Hiera configuration file rather than a manifest avoids repetition, which enables you to write more generic manifests that you can reuse for multiple systems.

    Puppet classes can request the data that is needed and Hiera acts as a site-wide configuration file. When Puppet loads Hiera, it uses this configuration file instead of the global file that is located in `/etc/hiera.yaml`. For more information, go to `https://docs.puppet.com/hiera/3.1/`.

- **Puppet agent/master model**

  Puppet uses an agent/master model, where the *Puppet master* manages important configuration information for all of the nodes (physical or virtual) on which the *Puppet agent* is running.

  Nodes that are running the Puppet *agent* poll the Puppet master at regular intervals and make requests for updated configuration information, which the agent then applies to the node. See “How Puppet Works” on page 14.

- **Puppet SMF service**

When you install the Puppet software package, you get a single Puppet SMF service (`svc:/application/puppet`) with the following two instances: `svc:/application/puppet:master`, for the Puppet master, and `svc:/application/puppet:agent`, for the Puppet agent. By default, these service instances are disabled after a Puppet installation. When you enable these service instances, the daemons for these services are started. When these services are disabled, the daemons are halted. See "Configuring the Puppet Master and Agent" on page 22.

- **Puppet configuration file**

  Puppet provides a configuration file (`/etc/puppet/puppet.conf`) for both the master and the agents. This configuration is stored in the SMF repository. Many system resources are defined in the `puppet.conf` file. The file lists the default values that are used by the Puppet master and all of the nodes that are managed by the master.

  The Puppet configuration file is generated through the `svcio` utility by using an *SMF stencil*. See Chapter 6, "Using a Stencil to Create a Configuration File" in *Developing System Services in Oracle Solaris 11.3*.

  To ensure that the configuration within the `puppet.conf` file always matches what is in the SMF repository, never edit this file directly. Instead, use SMF commands to set the appropriate properties in the file. See `svccfg(1M)`. Because stenciling is used to generate the Puppet configuration file, any persistent changes that you make by setting SMF properties are automatically applied to the `puppet.conf` file.

- **Puppet resources and resource types**

  Puppet uses *resources* to represent various aspects of a system, such as when and how services are run, software package management, and certain components of networking and naming service configuration. A resource can also reflect the state in which a certain aspect of a system should be.

  Each resource has a *resource type*, which is defined by a title and a series of attributes and values that you can specify within a Puppet manifest. The values that you can declare depend on the type of configuration that you are managing. See Chapter 3, "Working With Puppet Resources and Resource Types in Oracle Solaris".

- **Puppet providers**

  Puppet *providers* translate the general definitions for a resource into the actions that are required to implement that resource on a specific platform. These cross-platform capabilities are enabled by the Puppet Resource Abstraction Layer (RAL), which translates configuration settings into the platform-specific commands that are required to apply the specified configuration.

  For example, if you are installing a software package on an Oracle Solaris system, Puppet uses IPS, while on a Red Hat Enterprise Linux system, Puppet uses RPM (Red Hat Package Manager) to install the package.

The following are some of the key providers that are supported in Oracle Solaris:

- IPS package installation, commands, publishers, facets, and mediators
- SVR4 package installation
- Boot environments
- Datalink properties
- Aggregations
- Etherstubs
- IP network interfaces
- Naming services
- Oracle Solaris Zones, Oracle Solaris Kernel Zones, and Zones on Shared Storage (ZOSS) backing stores
- SMF administrative commands
- SMF properties
- TCP/IP tunables
- Virtual area networks (VLANs)
- Virtual network interface cards (VNICs)
- ZFS dataset creation and property manipulation (including `zpool` creation and deletion for most `vdev` types)

See "About Puppet Resources and Resource Types" on page 29.

- **Puppet command-line interface (CLI)**

  You use the Puppet command-line interface (CLI) to perform several actions, for example, the initial handshake between the master and agent nodes. You might also use the CLI to perform a *dry run* for testing purposes. You can also use the CLI to troubleshoot and debug issues with Puppet.

  Other tasks that you might perform with the Puppet CLI include the following:

  - Managing certificates
  - Generating and managing reports
  - Accessing plug-ins
  - Managing resources
  - Displaying status

  You use the following syntax to perform actions with the Puppet CLI:

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

  Display all of the available Puppet subcommands and their usage as follows:

  ```
  # puppet help
  ```

Display help for a specific subcommand as follows:

```
# puppet help subcommand
```

The following partial example shows how you would display information about the `agent` subcommand:

```
# 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.
. . .
```

Display help for a specific subcommand's action as follows:

```
# puppet help subcommand action
```

- **Puppet privileges and authorizations**

To configure and administer Puppet, you must be assigned the `Puppet Management` rights profile, or you must assume the `root` role. The `Puppet Management` rights profile includes the `solaris.smf.manage.puppet` and `solaris.smf.value.puppet` privileges. See "User Rights Management" in *Securing Users and Processes in Oracle Solaris 11.3* for details about how user rights and privileges work.

Common Uses for Puppet in Oracle Solaris

You can use Puppet to manage most major subsystems in Oracle Solaris, as well as automate several common system configuration tasks for multiple nodes (both physical and virtual). Puppet can scale from simple deployments to more complex infrastructures, such as cloud deployments and OpenStack. Some other ways that you might use Puppet include provisioning, system configuration, and software management.

How Puppet Works

Puppet provides the ability to define which software and configuration a system requires and then maintain a specified state after an initial setup.

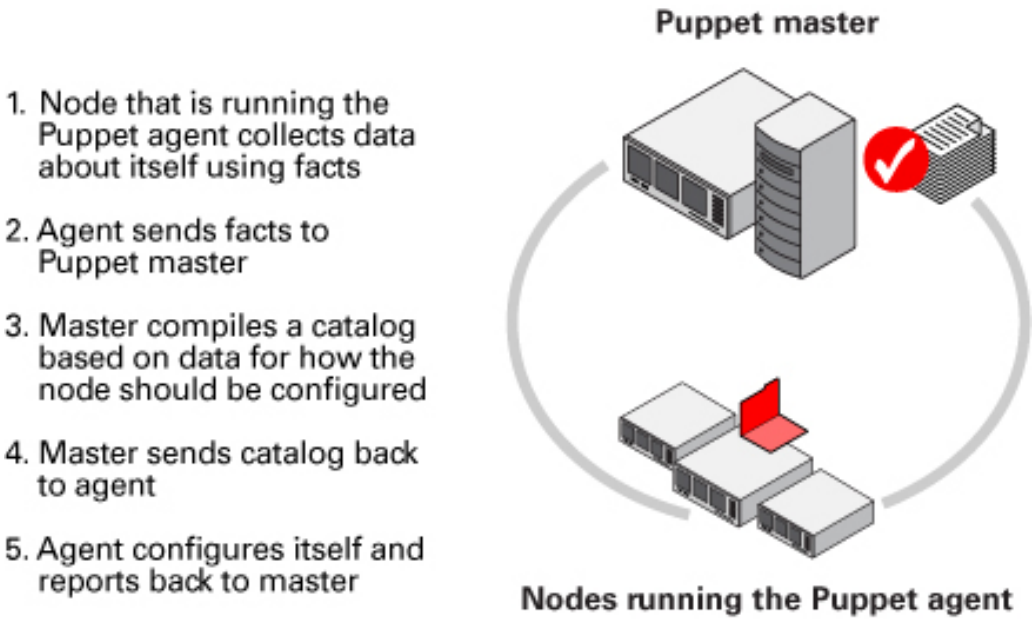
You use a declarative Domain Specific Language (DSL) that is similar to Ruby to define configuration parameters for a specific environment or infrastructure. Puppet discovers information about a system by using a utility called *Facter*, which is installed when you install the Puppet software package. See "Gathering Information About a System by Using Facter" on page 36.

The *Puppet master* is the system that manages important configuration information for all of the nodes that it controls by using manifests. See "Puppet Manifests" on page 17.

The nodes that the master controls are those that have Puppet installed on them and are running the *Puppet agent*, which is a daemon. The configuration information that the agent collects about a node is sent to the Puppet master. The Puppet master then complies a catalog based on how the node should be configured. Each node uses that information to apply any necessary configuration updates to itself.

Puppet works by using a *pull mode*, where agents poll the master at regular intervals to retrieve site-specific and node-specific configurations. In this infrastructure, managed nodes run the Puppet agent application, typically as a background service. For more information, go to Overview of Puppet's Architecture.

The following figure describes the Puppet master/agent topology in more detail.



About the Puppet Master

The Puppet master is a daemon that runs on a designated server and is the primary source of configuration data and authority for Puppet. The master provides instructions for all of the nodes that are part of the Puppet infrastructure. Because some aspects of component configuration depend on the configuration of other components, the server that is designated as the Puppet master is required to be aware of the system's entire configuration. Puppet restricts access to the master by having the master run as its own user and group. See “Function of the Puppet User and Group” on page 16.

The master is responsible for several actions, including the following:

- Compiling the catalog for the agents
- Transferring files from a file server
- Sending reports to a central instance

Note - The master might also perform other actions that do not require `root` privileges.

About the Puppet Agent

The *Puppet daemon* that runs on a target system (or node) is known as the *Puppet agent*. The agent must have the appropriate privileges for the node on which it is enabled so that it can apply the configuration catalogs that it pulls from the Puppet master. The agent gains communication privileges from the master server by requesting an Secure Socket Layer (SSL) certificate the first time that it contacts the master. Subsequently, whenever the agent polls the master for configuration updates, it only receives updates if its certificate is valid.

The Puppet agent that runs on each of the target nodes must have the ability to modify most aspects of the system's configuration. This requirement enforces the state in which the master has indicated the agent should be. Because so much access to the system is required by the puppet agent, it is run as the `root` user or a user who is assigned the `Puppet Management` rights profile.

Note - Note that the master must also authenticate to the agents so that they do not inadvertently receive incorrect configuration information.

Function of the Puppet User and Group

The Puppet *user* and *group* are used for security purposes to ensure that a module only has access to the information that it requires from the master. The Puppet user and group also prevent the Puppet module from being exploited or compromised. The Puppet user performs tasks on the master and is a member of the Puppet group. This privileged user and group are automatically created and assigned to the master daemon when you enable the master SMF service instance during the setup process. See Chapter 2, “Getting Started With Puppet in Oracle Solaris”.

Through the Puppet user, the Puppet master performs the following tasks:

- Stores configuration manifests in the puppet manifests directory.
- Accepts SSL certificates from agent.
- Transfers files to agents.
- Creates catalogs.

Puppet Encryption and Communication Methods

Puppet interfaces with the OpenSSL toolkit, which is based on SSL and the Transport Layer Security (TLS) cryptographic protocol. Puppet uses standard SSL/TLS encryption technology and standard SSL certificates for agent and master authentication and verification. Puppet also

makes use of SSL/TLS to encrypt the traffic flow between server and agents. SHA-256 is the default hash that is used.

Puppet's encryption method does the following:

- Authenticates any agent to the master
- Authenticates the master on any agent
- Prevents communication eavesdropping between master and agents

Puppet uses a TLS client-side X.509 certificate to perform mutual host authentication. By default, this information is stored in the `/etc/puppet/ssl` directory, which is defined in the puppet configuration file (`puppet.conf`). You can change the default location by using SMF commands, which would then be reflected in the site configuration file. Note that there are separate directories for keys, certificates, and signed requests, as well as those requests that are awaiting a signature. These directories exists on both the master and the agent.

Because Puppet uses its own certificate authority (CA), you do not need to use the system's default setting for CAs (`/etc/certs/CA`). When the master is initialized, it generates its own CA certificate and private key, initializes the Certificate Revocation List (CRL), then generates another certificate, called the *server certificate*. This certificate is used for SSL and TLS communications and is sent to the agent. During the master and agent exchange, the CA is stored in the `/etc/puppet/ssl/ca/signed` directory on the master and in the `/etc/puppet/ssl/certs` directory on the agent.

Puppet Terminology

Puppet uses a Declarative Domain Specific Language (DSL) that defines *states*. Puppet code is written in manifests. In that code, you declare resources that define various aspects of a system, such as files, packages, services, and so on. Resources are grouped into classes, which expose parameters that can affect their behavior. Classes and configuration files are then organized into modules. These core Puppet terms are described in more detail in the following sections. For more complete definitions, consult the Puppet Glossary.

Puppet Manifests

The various resources that you need to declare for a specific configuration are stored in files that are called *manifests*. Manifests contain Puppet code and are central to Puppet's infrastructure. These manifests are located on the Puppet master. Whenever you want to save a resource definition, you save it in a manifest. Note that each manifest must end with a `.pp` file extension.

You use a Puppet `site.pp` manifest to define global configuration that applies to all of the nodes. A site manifest can also include node-specific code that applies to certain nodes. A *node definition* (or *node statement*) is a block of Puppet code that is only included in the catalogs of the nodes that it matches. This feature enables you to assign specific configurations to specific nodes. For more information, go to Node Definitions.

You can also write manifests that group several resources together. In this case, you would use a *class* to apply the resources to the specified nodes. See Chapter 4, "Writing Puppet Manifests, Classes, and Modules in Oracle Solaris".

Puppet Classes

A *class* is a set of configurations that are bundled together. A Puppet class can include resources, variables, as well as additional, advanced attributes. When you assign a class to a node, that node gets all of the configurations that are part of the class. You include class declarations within a manifest. See "Writing Puppet Classes" on page 43.

Puppet Modules

Puppet *modules* are self-contained collections of files and directories that can contain Puppet manifests and other objects, including files and templates. The information that is within a module is packaged and organized in a way that Puppet can understand and use. Modules are how Puppet finds the classes and types that can be used for configuration management within your IT infrastructure. Puppet automatically loads any class or defined type that is stored within a given module. You can declare any of these classes or types by name within a manifest. See "Writing Puppet Modules" on page 45.

Additional Puppet References

For more in-depth information about Puppet, refer to the following documentation:

- For general information about Puppet, go to the Puppet web site
- For Puppet reference information, see the Puppet 3.6 Reference Manual
- For other Puppet documentation resources, see the Puppet Resource library

♦ ♦ ♦ CHAPTER 2

Getting Started With Puppet in Oracle Solaris

This chapter describes how to install, configure, and enable Puppet in Oracle Solaris.

This chapter contains the following topics:

Note - You must be assigned the `Puppet Management` rights profile or assume the `root` role to administer configuration management with Puppet. Privileges that are associated with the `Puppet Management` rights profile include `solaris.smf.manage.puppet` and `solaris.smf.value.puppet`. See "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

Puppet Pre-Installation Tasks

Prior to installing the Puppet IPS package on the master and the nodes that will run the Puppet agent, perform the following tasks:

- Designate a server that will function as the Puppet master.

  You should install and configure Puppet on the master server, or servers, before you install Puppet on any of the nodes.
- Designate the nodes that will run the Puppet agent.
- Configure the Domain Name System (DNS) protocol on both the master and agents so that all of the hosts can be resolved by using a fully qualified domain name. See Chapter 3, "Managing DNS Server and Client Services" in *Working With Oracle Solaris 11.3 Directory and Naming Services: DNS and NIS*.
- Ensure that time-keeping on the Puppet master is configured accurately. See "How to Configure NTP on the Puppet Master" on page 20.

How to Configure NTP on the Puppet Master

Because the Puppet master server acts as the certificate authority, a recommended best practice is to configure the Network Time Protocol (NTP) to accurately keep time on the master *prior* to installing Puppet. Otherwise, the master could issue certificates that the agents could treat as expired. For more information about managing NTP, see *Enhancing System Performance Using Clock Synchronization and Web Caching in Oracle Solaris 11.3*.

Perform the following procedure on the Puppet master prior to installing the Puppet IPS package.

1. **Become an administrator who is assigned the `Puppet Management` rights profile or assume the `root` role.**

2. **Create a configuration file by editing the `/etc/inet/ntp.client` file and then copying the information to `/etc/inet/ntp.conf`.**

   In this procedure, the following four time servers are used, in the event that one time server fails.

   ```
   # 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. **Add the required configuration parameters to the `/etc/inet/ntp.conf` file.**

   ```
   # 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. **Force an initial time synchronization.**

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

5. **Enable the `ntp` SMF service.**

   ```
   # svcadm enable ntp
   ```

6. **Verify that NTP is working.**

   ```
   # ntpq -p
   ```

Note - NTP start-up can take from 15 to 60 minutes, or longer.

Next Steps As an alternative, you could also specify NTP configuration by using a Puppet manifest. See Chapter 4, “Writing Puppet Manifests, Classes, and Modules in Oracle Solaris”.

Installing Puppet

In Oracle Solaris, the Puppet IPS package is not installed by default. To use Puppet, you must individually install the `system/management/puppet` package on the Puppet master and all of the nodes that will run the Puppet agent. You install the same software package on both the master and each of the nodes. The Puppet version that is supported in Oracle Solaris 11.3 is 3.6.2.

To determine whether the Puppet IPS package is installed on a system, type the following command:

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

You can display more detailed information about the Puppet IPS package as follows:

```
# 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
```

The previous output indicates that the Puppet package is not installed on the system.

Install the Puppet IPS package on the master first, then install the package on each of the nodes as follows:

```
# pkg install puppet
```

Verify that Puppet is installed:

```
# 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
```

Per the previous output, you can see that Puppet is now installed on the system.

Configuring the Puppet Master and Agent

After installing Puppet on both the Puppet master and the nodes (agents) that the master will control, you are ready to configure the master and the agents.

Note - To perform the following tasks, you must be assigned the `Puppet Management` rights profile or assume the `root` role.

How Puppet Configuration Is Managed Through SMF

As part of the Puppet installation, the Puppet SMF service is installed on both the Puppet master and the nodes that the will run the Puppet agent. This SMF service has two instances: `svc:/`

`application/puppet:master`, which is for the Puppet master, and `svc:/application/puppet:agent`, which is for the Puppet agent. Having Puppet managed through SMF enables you to take advantage of a layered configuration, which helps preserve your configuration during system updates.

By default, both of the Puppet SMF service instances are disabled, as shown in the following output:

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

The initial Puppet configuration file (`etc/puppet/puppet.conf`) is generated by using an SMF stencil, which ensures that any configuration that is stored in the SMF repository correctly maps to the configuration that is stored in the Puppet configuration file.

Puppet reads configuration information from the `/etc/puppet/puppet.conf` file rather than from the properties that are set in the `application/puppet` service instances. To provide the required configuration file, each puppet instance provides a stencil file and a `configfile` property group. The `configfile` property group instructs the `svcio` utility to run and to create the specified configuration file. The stencil file is then used to write data from service property values to the configuration using the correct format. For more information about stencils, see Chapter 6, “Using a Stencil to Create a Configuration File” in *Developing System Services in Oracle Solaris 11.3*.

The following example shows all of the puppet service properties that are included in the `configfile` property group:

```
# 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
```

In the previous output, both instances of the puppet service have the same `configfile` properties with the same values. Each puppet service instance provides the path to the configuration file, the mode of the configuration file, and the path to the stencil file.

The following example shows that these instance properties are inherited from the parent service:

```
# 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
```

When making configuration changes to the `puppet.conf` file, do not manually edit the file. Instead, use SMF commands, as shown in the following example:

```
# 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
```

Any changes that you make by using SMF commands are automatically reflected in the `puppet.conf` file when you restart the Puppet agent service instance:

```
# 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
```

See the `svccfg`(1M) and `svcadm`(1M) man pages.

For a complete list of all of the configuration settings that apply to the `puppet.conf` file, see Configuration: How Puppet is Configured.

How to Configure the Puppet Master and Agent

One Puppet master can control many nodes that are running the Puppet agent. Depending on your particular infrastructure, you can also designate more than one Puppet master to control thousands of nodes. The following procedure describes how to configure one master and one agent.

Before You Begin Prior to performing the following procedure, do the following:

- Install the Puppet IPS package on both the master and all of the nodes that will run the Puppet agent. See “Installing Puppet” on page 21.

- On the master, configure and enable the Puppet master SMF service instance.

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

The output should indicate that the SMF service instance for the master is online. You are now ready to configure the nodes that will run the Puppet agent.

1. **On the node, do the following:**

   a. **Set the value of the SMF `config/server` property for the agent so that it points to the master.**

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

You must refresh the SMF service for the changes to take effect.

Note - Do not enable the agent service instance until after the agent makes the certificate request and it is successfully signed on the master.

---

   b. **Test the connection from the agent to the master.**

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

Running the `puppet agent` command with the `--test` option on the agent creates a new SSL key and sets up a request for authentication between the agent and the master.

2. **On the master, do the following:**

   a. **View any outstanding certificate requests coming from agents that are attempting to connect to the master.**

```
# puppet cert list
```

The output of this command should show a request being made by the agent.

   b. **Sign the certificate for the agent that is making the request.**

```
# puppet cert sign agent
```

Note - Although manually signing certificates is the preferred Puppet practice, if you have an environment where it is not absolutely necessary to manually sign certificates, you can configure the CA Puppet master to automatically sign certain CSRs. See the Puppet documentation at SSL Configuration: Autosigning Certificate Requests.

3. **Retest the connection from the agent to the master.**

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

This step ensures that the authentication between the master and the agent has taken place.

4. **Enable the SMF service instance for the Puppet agent.**

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

The output should indicate that the SMF service instance for the agent is online.

Example 1 Configuring the Puppet Master and Agent

The following example shows how you would configure a Puppet master and agent.

```
# 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
```

Next Steps After you have installed Puppet and performed all of the necessary configuration and validation tasks, you are ready to use Puppet to manage system configuration.

For details about declaring resources with Puppet, see Chapter 3, “Working With Puppet Resources and Resource Types in Oracle Solaris”.

For instructions on writing Puppet manifests, see Chapter 4, “Writing Puppet Manifests, Classes, and Modules in Oracle Solaris”.

For examples of various Oracle Solaris system configurations, see Chapter 5, “Using Puppet to Manage System Configuration in Oracle Solaris”.

Troubleshooting Issues With Puppet in Oracle Solaris

The puppet master and agent services log most activity to the `syslog` service. The `syslog` configuration dictates where these messages are saved. In Oracle Solaris, the default location is the `/var/adm/messages` directory. However, Puppet service logs are stored in the following locations.

For the Puppet daemon, the logs are stored in the following locations:

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

For the Puppet SMF service instances, the logs are stored in the following locations:

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

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

You can access the logs for the Puppet agent by running the following command:

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

You can access the logs for the Puppet master as follows:

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

The following example shows the types of information you can view with this command:

```
# 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. ]
```

♦ ♦ ♦ CHAPTER 3

Working With Puppet Resources and Resource Types in Oracle Solaris

This chapter provides more detailed descriptions of Puppet resources and resource types, as well as examples of resource types that are commonly used in Oracle Solaris. Information about how to use the Puppet command-line interface (CLI) to list, view, and modify resources is also provided.

This chapter includes the following topics:

About Puppet Resources and Resource Types

Puppet uses *resources* and *resource types* to describe a system's configuration. Resources are grouped into collections. At a high level, a resource is referred to as a *type*, which describes a specific resource that Puppet can manage on a system, such as users, packaging, networking, and so on.

At a platform-specific level, a resource is referred to as a provider. *Providers* are the platform-specific implementations of a specified type. Providers translate the general definitions for a resource into the actions that are required to implement the resource on that specific platform.

Resource types and providers are at the heart of the Puppet Resource Abstraction Layer (RAL). Oracle Solaris includes several built-in Puppet resource types and providers that apply to a specific Puppet version, as well as additional resource types that are distributed in individual Puppet modules that enable you to manage certain aspects of networking, naming services, and so on. There are also types that are unique to managing specific Oracle Solaris features, for example, Oracle Solaris Zones.

As previously mentioned, Puppet uses a declarative language to describe system resources and their state. This information is written in manifests. You can use a Puppet site manifest (`site.pp`), which is located on the Puppet master, to define global configuration that is common to all of the nodes.

You can also include node-specific code in a main Puppet manifest to define configuration that pertains to certain nodes. See Chapter 4, “Writing Puppet Manifests, Classes, and Modules in Oracle Solaris”.

Display all of the Puppet resource types that are available on a system as follows:

```
# 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
```

The previous output includes both Oracle Solaris-specific types and core Puppet types, which are available for use on other platforms.

Puppet Resource Type Descriptions

You would use the `puppet describe` command with the `--list` option as follows to display all of the available resource types and descriptions for each of the supported Puppet providers:

```
# 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  ...
```

About Declaring Puppet Resources

A *resource declaration* is an expression that describes the desired state for a resource. The Puppet master then takes these resource declarations and compiles them into a catalog. When Puppet applies that catalog to a target system, it manages every resource that it contains, ensuring that the actual state matches the desired state.

In contrast, when you specify the contents and behavior of a class or a defined type, you *define* it by using the Puppet language. Defining a class or type makes it available to be declared within a manifest.

Puppet uses the following format for basic resource declarations:

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

All resource declarations use the following format:

- `resource_type` – Is the type of resource that is being declared. The `resource_type` must be a word that does not include quotation marks.

  The `resource_type` includes an opening that begins with a curly brace (`{`).
- `title` – Is an identifying string that Puppet uses for identification purposes. Every `resource_type` must have a title, and the string must be unique, per each resource type. Note that the title does not have to match the name of the resource that you are managing on the node.

  The `title` is followed by a colon (`:`).
- `attribute` – Describes the desired state of the resource. Most resources have a set of *required attributes*, but they can also include a set of *optional attributes*..

  Attribute and value pairs must consist of the following:

  - An attribute name, which is a lowercase word with no quotes.

    Each attribute name handles some aspect of the resource. Each resource type has its own set of available attributes.
  - An arrow (`=>`), also called a "fat comma," or “hash rocket”.
  - A value, which can have any data type.

    The data type of the value depends on what the attribute accepts.
  - A trailing comma.
- A closing curly bracket (`}`) must be included at the end of the resource declaration.

Note that you can use any amount of white space in the Puppet language.

For more detailed information, go to Language: Resources.

Viewing and Modifying Puppet Resources by Using the Command Line

You can view and modify the state of a system's resource by using the `puppet resource` command. This command converts the current system state into Puppet's declarative language, which you can then use to enforce configuration on other systems.

Viewing the State of a Puppet Resource

The following example shows how you would view the state of the `zone` resource type:

```
# 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',
}
```

In the previous example, there are two `zone` resources declared: a global zone and an installed kernel zone. Each of these resources has four attributes: `ensure`, `brand`, `iptype`, and `zonepath`. Each attribute has a value associated with it. This *value* is a central component of Puppet's declarative language.

The following example shows how you would view the state of the `service` resource type:

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

Modifying the State of a Puppet Resource

You can also use the `puppet resource` command to modify the state of a resource. You would use this method in lieu of directly modifying the configuration within a Puppet manifest.

For example, you would modify the state of the `service` resource type as follows:

```
# 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',
}
```

Gathering Information About a System by Using Facter

You use the Facter utility to gather information about a system. This information is sent to the Puppet master and then used by Puppet's *resource providers* to compile *catalogs* that specify the configuration changes that should be applied to each of the nodes.

A catalog also specifies the states in which each of the resources should be. Based on these definitions, each system can then apply its own configurations, as appropriate. After the catalog is applied to the system, the agent generates a report and sends that report to the Puppet master. This report contains information about which resources are currently being managed on the target node, as well as any changes that were made to the node to achieve a desired state. See "How Puppet Works" on page 14.

To list all of the facts that are available for a given node, type the following command:

```
# facter -p
architecture => i86pc
facterversion => 2.1.0
hardwareisa => i386
hardwaremodel => i86pc
hostname => myhost
id => root
interfaces => lo0,net0
ipaddress => 203.0.113.15
ipaddress6 => ::
ipaddress_lo0 => 127.0.0.1
ipaddress_net0 => 203.0.113.5
ipaddress_net1 => 203.0.113.5
...
uptime => 0:22 hours
uptime_days => 0
uptime_hours => 0
uptime_seconds => 1320
virtual => virtualbox
```

Or, you can display an individual fact for a given node, for example `hostname`, as follows:

```
# facter hostname
myhost
```

Gathering facts about a system can assist you in determining the types of configuration that you can enforce on a given system. For example, you could declare a file resource that would populate a given file with platform-specific content.

In the following example, the `osfamily` fact is used to declare the platform within the file:

```
$file_contents = $osfamily ? {
```

```
  'solaris' => "Hello Oracle Solaris",
  'redhat' => "Hello RHEL",
}

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

In the previous example, a new `$file_contents` variable was created and a conditional check was provided by using the `osfamily` fact. Then, depending on the platform, you would assign different contents to the file.

For more information, see Facter: Custom Facts.

♦ ♦ ♦ CHAPTER 4

Writing Puppet Manifests, Classes, and Modules in Oracle Solaris

Puppet manifests, classes, and modules are what Puppet uses to define system configuration within your infrastructure. This chapter describes the basics of writing manifests, classes, and modules.

This chapter contains the following topics:

Writing a Puppet Site Manifest

After installing and configuring Puppet, you can write Puppet manifests to control the nodes that are running the Puppet agent. Puppet manifests are written in a Puppet-specific language that is similar to Ruby, where each manifest uses a `.pp` file extension.

The Puppet site manifest (`site.pp`) is the main file that Puppet uses to define global system configuration. A site manifest defines configuration that you want applied to every node, which is ideal for managing system-wide configurations, such as DNS servers, LDAP configuration, and other site-wide settings that are common to all of the nodes.

A site manifest can also include node-specific blocks of code that apply to certain nodes. This capability enables you to assign specific configurations to specific nodes within a site manifest. See "Writing Puppet Manifests That Specify Node-Specific Code" on page 42.

Note - The `site.pp` manifest does not exist on the Puppet master by default. You must initially create this file, and it must be stored in the `/etc/puppet/manifests/` directory on the master.

How to Write a Puppet Site Manifest

The following procedure describes how to write a Puppet site manifest to enforce configuration globally within your infrastructure.

Before You Begin Prior to writing a Puppet site manifest, you will need to do the following:

- Determine which resource types to declare in the manifest. You can obtain this information by using the `puppet describe` *resource-type* command, which displays all of the available attributes and parameters for the specified resource type.

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

  See "Puppet Resource Type Descriptions" on page 31.
- Familiarize yourself with the basic syntax that you use to declare resources within a Puppet manifest. See "About Declaring Puppet Resources" on page 33. For more detailed information, go to Language: Resources.
- Familiarize yourself with the syntax that you use to define specific Oracle Solaris system configuration within a Puppet manifest. See Chapter 5, "Using Puppet to Manage System Configuration in Oracle Solaris" for examples.

1. **Become an administrator who is assigned the `Puppet Management` rights profile or assume the `root` role.**

   See "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

2. **Create a `site.pp` file on the Puppet master.**

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

   This file should always reside in the `/etc/puppet/manifests` directory on the master.

3. **Define the specified configuration within the Puppet site manifest (`site.pp`) and save your changes.**

   See Chapter 3, "Working With Puppet Resources and Resource Types in Oracle Solaris" for more details.

4. **Test the configuration changes that you made to the `site.pp` file before they are permanently applied.**

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

   `apply` Applies the configuration to the Puppet manifest on the master.

`-v`	Indicates to use verbose mode.
`-noop`	Enables you to perform a *dry run* without actually applying your changes. Use this option for testing purposes.

The Puppet agent that is running on each node queries the master for configuration changes at regular intervals and then applies any required changes to the node.

5. **Check the log file (`/var/log/puppet/puppet-agent.log`) on each node to verify that it retrieved the latest configuration changes.**

6. **(Optional) To manually apply the latest configuration changes, run the following command on the node:**

```
# puppet agent -t
```

Specifying the `-t` (`--test`) option enables verbose logging, which causes the agent daemon to remain in the foreground, exits if the master server's configuration is invalid (as in the case of a syntax error), then exits after running the configuration one time.
To display all of the available Puppet subcommands, use the `/usr/sbin/puppet help agent` command. See also the `puppet`(8) man page.

Example 2 Writing a Puppet Manifest

The following example shows how you would declare resources in Puppet site manifest. This example assumes that you have already created a `site.pp` file and that the file is stored in the correct directory on the Puppet master.

First, you would declare resources in the `site.pp` file. In this example, the `file` resource type is declared. For this resource type, two attributes are specified: `ensure` and `content`. These two attributes ensure that a `custom-file.txt` file exists in the `root` directory on the node and that the file includes the words, "Hello World".

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

After saving the `site.pp` file, you can test the configuration's validity on the master as follows:

```
# 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
```

The `-v` option specifies to use verbose mode, and the `-noop` option ensures that no changes are actually made. Using the `-noop` option for testing purposes enables you to perform a *dry run* without actually applying the changes to the manifest.

The Puppet agent that is running on each node queries the master for configuration changes at regular intervals and then applies any new changes, as needed. You can check the node's log file (`/var/log/puppet/puppet-agent.log`) to verify that the node applied the latest changes:

```
# 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
```

The previous output indicates that the configuration is being enforced on the node. By default, agents poll the master for configuration changes at 30-minute intervals. You could also verify the configuration by checking whether the `custom-file.txt` file exists on the node.

Optionally, you would manually apply the configuration changes by running the following command on the node:

```
# puppet agent -t
```

For specific examples that show how to use Puppet to define Oracle Solaris system configuration, see Chapter 5, “Using Puppet to Manage System Configuration in Oracle Solaris”.

Writing Puppet Manifests That Specify Node-Specific Code

If you are managing configuration for a variety of systems, you might consider specifying *conditional logic* in your manifests, which ensures that each system is correctly matched to the appropriate configuration.

To enforce this logic, use the `node` keyword in your site manifest (which can be a single file with a `.pp` file extension or a directory containing several files with a `.pp` file extension). While node declarations enable you to specify any arbitrary Puppet code, it is recommended that they only contain variable assignments and class declarations.

The following example shows how you would match identical configuration for two nodes, `agent1.company.com` and `agent2.company.com`:

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

The following example shows the syntax that you would use to match identical configuration for two nodes, along with a different resource definition for a third node (`agent3.company.com`).

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

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

  # Include other resources here

}
```

Puppet also provides a special node, called `default`, which enables a fallback configuration for any of the nodes that do not match existing node definitions. You would define a fallback configuration for these nodes as follows:

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

For more in-depth information about writing manifests that includes node-specific code, see Language: Node Definitions.

Writing Puppet Classes

Classes are blocks of Puppet code that enable reuse. Using classes makes reading manifests less complicated. A *class definition* contains the code for a specific class. You first define the class, then you make the class available for use within manifests. Note that the class itself does not perform any evaluation.

The following example shows the format that is used for a class definition named `examplecloud`:

```
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,
    }

}
```

In this example, the class has two name spaces: `examplecloud` and `analytics`. The code that is specified in this class ensures that certain IPS packages are installed and that certain SMF configuration is applied prior to enabling the `analytics` SMF service on the node.

A *class declaration* is a class that is defined within a manifest. A class declaration instructs Puppet to evaluate the code within that class.

There are two types of class declarations: normal and resource-like.

- For the normal class declaration, the `include` keyword is included in Puppet code, as shown in the following example:

  `include` *example_class*
- For the resource-type class declaration, the class is declared similarly to how a resource is declared, as shown in this example:

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

  You use resource-like class declarations to specify class parameters. These parameters override the default values of class attributes.

For more in-depth information about writing and assigning Puppet classes, see Language: Classes.

EXAMPLE 3 Including a Class Declaration in a Puppet Manifest

The following example manifest uses a class declaration named `examplecloud`, which is located in the `/puppet/modules` directory on the Puppet master.

Under the `examplecloud` class are several manifests (`/puppet/modules/examplecloud/manifests`) that specify various configurations. Each manifest includes the `examplecloud` class declaration, as shown in the following example:

```
# 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,
    }

}
```

The declarations for the `examplecloud` class in the previous example ensure the following:

- The NTP package is installed
- A certain configuration file (which is sourced from a location other than the Puppet master) is installed
- The NTP service is enabled and in a running state on the node

Writing Puppet Modules

Puppet modules are a collection of manifests and data, which can include facts, files, and templates. Modules help you organize and reuse Puppet code by enabling you to split the code into several manifests. With the exception of the main `site.pp` manifest that contains global configuration for all of the nodes, nearly all Puppet manifests should be included in modules. If you have several Puppet manifests, consider using modules as a way to organize them.

Caution - Modules that are provided through IPS are specifically updated for Oracle Solaris. Do not replace these modules with Puppet Forge modules.

To write your own Puppet module, you would start by running the following command on the Puppet master:

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

Running the previous command prompts you with a series of questions. Puppet uses your responses to gather information about the module and then creates a basic module structure. For further instructions and examples, see Writing Modules.

You add Puppet modules that you create to the `/etc/puppet/modules` directory on the master, where the basic directory tree structure is similar to the following:

module-name/ – Is the outermost (or parent) directory structure that specifies the name of the module.

- `manifests/` – Contains all of the manifests within the module.
  - `init.pp` – Contains a class definition. The name of the class definition must match the name of the module.
  - `other_class.pp` – Contains a defined type named `my_module::my_defined_type`.
  - `my_defined_type.pp` – Contains a class named `my_module::other_class`.
  - `my_module::my_defined_type` – Contains a defined type named `my_module::my_defined_type`.
  - `implementation/` – Is a directory with a name that affects the class names that are stored under it.
    - `foo.pp` – Contains a class named `my_module::implementation::foo`.
    - `bar.pp` – Contains a class named `my_module::implementation::bar`.
- `files`/ – Contains static files that managed nodes can download.

  `service.conf` – Is a file with a source URL that is similar to `puppet:///modules/my_module/service.conf`. You can access the file's contents by using a file function, for example, `my_module/service.conf`.
- `lib/` – Contains plug-ins, for example custom facts and resource types, which are used by both the Puppet master server and the Puppet agent service.
- `facts.d/` – Contains external facts, which you can use as an alternative to Ruby-based custom facts.
- `templates/` – Contains templates that a module's manifests can use.
  - `component.erb` – Is a template that a manifest can render as `my_module/component.erb`.
  - `component.epp` – Is a template that a manifest can render as `my_module/component.epp`.
- `examples`/ – Contains examples that show how to declare the module's classes and defined types.

  - `init.pp`
  - `other_example.pp` – Includes major use case examples.
- `spec/` – Contains tests for any plug-ins that are in the `lib` directory.

As shown in the following example, a module named `examplecloud` is located under the `/etc/puppet/modules` directory:

```
# 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
```

Under the `examplecloud` directory is the `manifests` directory that contains the manifests for the module. Each manifest contains one class or defined type, as shown in the following output:

```
# 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
```

Manifest file names map to the names of the classes and defined types that they contain. Each subdirectory under the `examplecloud/manifests` directory has a specific function.

For a more comprehensive description of each of these components, go to Module Fundamentals.

The Puppet Forge site includes a repository of publicly available modules, including newer modules, as well as authoring tools and documentation that you can download.

♦ ♦ ♦ CHAPTER 5

Using Puppet to Manage System Configuration in Oracle Solaris

This chapter provides end-to-end examples that show some of the ways in which you can manage Oracle Solaris system configuration with Puppet.

The following examples assume that you have already installed and configured Puppet on the master server and all of the nodes. The following examples also assume that you previously created a Puppet site manifest and that this file exists on the Puppet master.

This chapter includes the following topics:

Puppet Configuration Management Workflow

To manage Oracle Solaris system configuration with Puppet, you typically follow this basic process:

1. Use the `puppet describe` command to display all of the available attributes for the specified resource type that you are planning to configure. See Chapter 3, "Working With Puppet Resources and Resource Types in Oracle Solaris".
2. Declare the appropriate resources in a Puppet manifest that resides on the master.

   You would use a Puppet site manifest (`site.pp`) to define global system configuration that applies to all nodes.

You can also define node-specific configuration within a Puppet site manifest by using the `node` keyword. See "Writing Puppet Manifests That Specify Node-Specific Code" on page 42.

3. Perform a dry run to test whether the configuration that is defined in the Puppet manifest is valid.

   Although this step is not required, it is suggested as a best practice.

4. Verify that the nodes have applied the configuration.

For step-by-step instructions, see "How to Write a Puppet Site Manifest" on page 40.

Using Puppet to Configure Packaging

The following example shows how you would add a new IPS software package (`nmap`) by declaring the Puppet `package` resource type in a manifest.

EXAMPLE 4 Configuring Packaging With Puppet

First, you would determine whether the package that you plan to install is already installed:

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

If you wanted to check remotely whether the package was installed, you would use the `-r` option as follows:

```
# 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
```

Next, you would use the `puppet describe` command (as shown in the following partial example output) to check for the appropriate attribute to declare for the `package` resource type:

```
# 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 `/./`.
...
```

You would then declare the resource type within the Puppet manifest on the master as follows:

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

In the previous example, the resource definition title is set to `nmap` (the package to be installed), and the `ensure` attribute's value is set to `present`, which checks that the package is available for installation.

The configuration is verified as follows:

```
# 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
```

The output of the previous command shows that the `nmap` package is now installed on the node. The package is installed when the Puppet agent runs. Or, you can run the `puppet agent -t` command on the node to manually enforce the configuration changes.

Note that if you were to uninstall the `nmap` package, Puppet would enforce the specified configuration by reinstalling the package on the node.

Using Puppet to Configure ZFS File Systems

The following example shows how you could define ZFS file system configuration in a Puppet manifest by using the `zfs` resource type.

EXAMPLE 5 Configuring ZFS File Systems With Puppet

You would first display a list of all of the attributes that you can declare for the `zfs` resource type as follows:

```
# 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`.
...
```

Next, you would declare the `zfs` resource type, with the following parameters, in the manifest. Note that an additional attribute called `readonly` has been added and it is set to `on`.

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

You would verify the configuration by running the following commands on the node:

```
# 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
```

Using Puppet to Configure Networking Parameters

The following example shows how you might manage network configuration with Puppet. In this example, various network-related resource types are declared in a Puppet manifest.

EXAMPLE 6 Configuring Network Parameters With Puppet

The following example shows how you might specify multiple network configuration parameters in a Puppet manifest.

```
# 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",
        }
}
```

Using Puppet to Configure Naming Services

The following example shows how you might manage naming services configuration with Puppet by declaring the `service` resource type in a Puppet manifest.

EXAMPLE 7 Configuring Naming Services With Puppet

In the following example, the DNS service is enabled and a DNS server is configured. Then, the `domainname` property is set. Finally, the name service switch values are specified.

```
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",
}
```

Using Puppet to Configure Oracle Solaris Zones

The following example shows one way that you could define Oracle Solaris zones configuration by declaring the `zone` resource type in a Puppet manifest.

EXAMPLE 8 Configuring Oracle Solaris Zones With Puppet

By running the `puppet describe` command (as shown in the following partial example output), you would first display a list of all of the attributes that you can declare for the `zone` resource type:

```
# 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
```

The `zonecfg_export` attribute (shown in the previous output) enables you to create a zone configuration file resource by using the `zonecfg` command as follows:

```
# 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
```

The zone that you created becomes configurable when the `zone` resource type is applied. You would declare the `zone` resource type in the Puppet manifest as follows:

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

Here, the `ensure` attribute's value is set to `installed`. The value of `ensure` matches an acceptable status for a zone (`installed`, and `running`). In this example, a zone called `systemazone` is created on the node.

The last step would be to verify that the node applied the configuration to itself:

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

```
- systemazone      running     /system/zones/systemazone  solaris     excl
```

The output of the previous command shows that the non-global zone `systemazone` is configured, installed, and running.

Index

D

E

F

G

H

I

P

R

S

T

U

V

W

Z