Using Puppet to Perform Configuration Management in Oracle® Solaris 11.4

Part No: E72062
October 2019

ORACLE®

Using Puppet to Perform Configuration Management in Oracle Solaris 11.4

Part No: E72062

Référence: E72062

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 using Puppet to perform configuration management in the Oracle Solaris operating system (OS).
- **Audience** – System administrators and developers.
- **Required knowledge** – Basic and advanced system administrator and developer skill set.

Product Documentation Library

Documentation and resources for this product and related products are available at http://www.oracle.com/pls/topic/lookup?ctx=E37838-01.

Feedback

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

♦ ♦ ♦ CHAPTER 1

Using Puppet to Manage System Configuration in Oracle Solaris

Puppet is a cross-platform tool that you can use to automate and enforce the configuration of most major subsystems in Oracle Solaris. Use Puppet to automate tasks such as provisioning, software management, and system configuration. Puppet enables you to standardize and enforce resource configurations across your entire IT infrastructure. Puppet can scale from simple deployments to more complex infrastructures, such as cloud deployments.

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

This chapter includes the following topics:

For information about Puppet that is not specific to Oracle Solaris, refer to the following resources:

- Puppet web site
- Puppet Documentation
- Puppet Resource Library
- Glossary of Puppet Vocabulary

What's New in Puppet in Oracle Solaris 11.4

Puppet 5.5. Oracle Solaris 11.4 includes Puppet 5.5. Oracle Solaris 11.3 includes Puppet 3.6. If Puppet is already installed on the system, Puppet is automatically updated to Puppet 5.5 when you update to Oracle Solaris 11.4. You cannot choose to install a different version of Puppet.

While Puppet is updated automatically, you must update the Puppet configuration manually. See Puppet Documentation for how to maintain and update the Puppet configuration.

The Puppet web site does not include information about updating Oracle Solaris SMF service property values. Any SMF service property values that are already configured are migrated automatically when you update. However, you must manually remove any obsolete values and update any values that need to be changed. See "Configuring the Puppet Master and Agents" on page 24.

Puppet Features in Oracle Solaris

Puppet is not installed on your Oracle Solaris system by default. You must install the Puppet Image Packaging System (IPS) package on the Puppet master and on all of the nodes that will run the Puppet agent as described in "Installing Puppet" on page 21.

This section introduces Puppet features and functionality that are included in Oracle Solaris.

Puppet Utilities

The following utilities are installed when you install Puppet:

Facter

Facter is a utility that Puppet uses to *discover* facts about a particular system such as OS type, CPUs, or memory size. The information that Facter gathers about a system is sent to the Puppet master, and the Puppet master then compiles the information into *catalogs* that describe a desired system state for a specific set of resources. A 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 38 and Facter documentation on the Puppet web site.

Hiera

Hiera is a cross-platform, key/value lookup tool that you use to manage configuration data. 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 in 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 Hiera configuration file instead of the global file that is located in `/etc/puppetlabs/puppet/hiera.yaml`. See Hiera documentation on the Puppet web site.

Puppet Modules

When you install the Puppet IPS package, you get the core Puppet modules plus other modules that are specific to the Oracle Solaris release. For example, you get modules to support Oracle Solaris ZFS, networking, services, and zones.

Use the following command to list modules that are installed on this system:

```
$ puppet module list
```

Use the following command to list modules that are available, as shown in Searching modules from the command line:

```
$ puppet module search search_term
```

Use the following methods to get detailed information about a specific Puppet module:

- For installed modules, see the `README` file for that module at the path given by the `puppet module list` command.

```
$ puppet module list
/usr/puppetlabs/puppet/modules
├── oracle-solaris_providers (v2.0.1)
├── puppet-staging (v3.0.0)
├── puppetlabs-concat (v4.1.1)
├── puppetlabs-inifile (v2.1.0)
├── puppetlabs-ntp (v7.0.0)
├── puppetlabs-rsync (v1.0.0)
└── puppetlabs-stdlib (v4.23.0)
$ ls /usr/puppetlabs/puppet/modules
concat/             ntp/                solaris_providers/  stdlib/
inifile/            rsync/              staging/
$ less /usr/puppetlabs/puppet/modules/solaris_providers/README.md
# solaris_providers Module for Puppet
...
```

- Search for the module on Puppet web sites. For example, select a module name from the Puppet Supported Modules Compatibility Matrix. Select “Solaris” in the Operating System field at the top of the page and select the Search button.

Puppet SMF Services

When you install the Puppet software package, you get the following Puppet SMF services:

```
$ svcs puppet
```

```
STATE          STIME    FMRI
disabled        8:17:58 svc:/application/puppet:agent
disabled        8:17:58 svc:/application/puppet:main
disabled        8:17:58 svc:/application/puppet:master
disabled        8:17:58 svc:/application/puppet:user
online          8:36:42 svc:/application/puppet:upgrade
```

See "Configuring the Puppet Master and Agents" on page 24 for information about enabling and using these services.

Puppet Configuration File

Puppet provides a configuration file (`/etc/puppetlabs/puppet/puppet.conf`) for both the master and the agents. 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.conf` configuration file is generated from `svc:/application/puppet` service property values. Do not edit the `puppet.conf` file directly. Instead, use SMF commands to set the appropriate properties in the file as described in *Managing System Services in Oracle Solaris 11.4* and the `svccfg`(8) man page. Changes that you make by setting SMF property values are applied to the `puppet.conf` file.

Puppet Resources and Resource Types

Puppet uses *resources* to represent the configuration of a system, such as when and how services are run, which software packages are installed, and certain components of networking and naming service configuration. A resource reflects the desired state of that system configuration.

Each resource has a *resource type*, which is defined by a name and a set of attributes and values that you can specify within a Puppet manifest. The values that you can specify depend on the type of configuration that you are managing. You can get information about a resource by using the `puppet describe` command as shown in the following example:

```
$ puppet describe zone

zone
====
Manages Solaris zones.
...
```

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, to install a software package on an Oracle Solaris system, Puppet uses IPS, while on an Oracle Linux system, Puppet uses RPM.

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

- IPS package installation, commands, publishers, facets, and mediators
- 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 Local Area Networks (VLANs)
- Virtual Network Interface Cards (VNICs)
- ZFS dataset creation and property manipulation, including ZFS pool creation and deletion for most virtual device types

See also “Puppet Resources and Resource Types” on page 31.

Puppet Command-Line Interface

Use the Puppet command-line interface (CLI) to perform tasks such as the following:

- Initial handshake between the master and agent nodes

- Trial run for testing purposes
- Manage certificates
- Generate and manage reports
- Access plug-ins
- Manage resources
- Display status
- Troubleshoot and debug issues with Puppet

The Puppet CLI has the following syntax:

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

The following command displays all of the available Puppet subcommands and their usage:

```
# puppet help
```

The following command displays help for a specific subcommand:

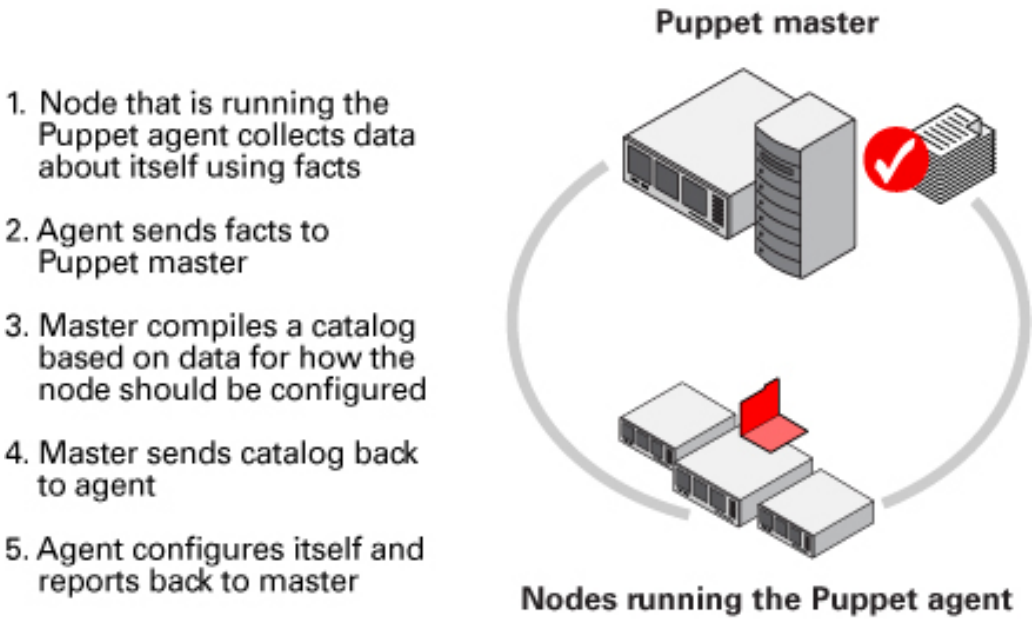
```
# puppet help subcommand
```

The following command displays help for a specific action of a subcommand:

```
# puppet help subcommand action
```

How Puppet Works

Puppet enables you to define the software and configuration that a system requires and then maintain that specified state. The Puppet master controls the configuration information; each managed agent node requests its own configuration from the master and configures itself.



Puppet discovers information about a system by using the Facter utility, which is installed when you install the Puppet software package. See "Gathering Information About a System by Using Facter" on page 38.

The Puppet master uses manifests to declare the resources that are needed to configure each specific managed node. You can also create a site manifest to define global configuration that applies to all of the managed nodes.

Managed nodes run the Puppet agent application, typically as a background service. The Puppet agent collects configuration information about itself and sends that information to the Puppet master. The Puppet master compiles a catalog of how the agent node should be configured. Each managed agent node uses that catalog to apply any necessary configuration updates to itself.

Puppet works by using a *pull mode*: Agents poll the master at regular intervals to retrieve site-specific and node-specific configuration information. For more information, See Overview of Puppet's Architecture.

Puppet Agent/Master Model

Puppet uses an agent/master (client/server) model, where the Puppet master manages important configuration information for all of the physical and virtual nodes 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.

The Puppet Master

The *Puppet master* is the primary source of configuration data and authority for Puppet. The Puppet master is a daemon that runs on a designated system and provides instructions for all of the nodes that it manages. Because some component configuration depends on the configuration of other components, the Puppet master requires information about all components of each managed node.

The following are some of the actions for which the master is responsible:

- Compiling the catalog for each managed agent
- Transferring files from a file server
- Sending reports to a central server

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

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

The master daemon runs as the `puppet` user. The `puppet` user is a member of the `puppet` group. Running the master daemon as the `puppet` user helps ensure that Puppet modules can access only the information that they require from the Puppet master and helps prevent Puppet modules from being exploited or compromised. The `puppet` user is automatically assigned to the master daemon when you enable the `puppet:master` SMF service instance during the setup process. See Chapter 2, “Getting Started With Puppet in Oracle Solaris”.

Puppet Agents

The *Puppet agent* is a daemon that runs on a managed node. To apply the configuration that the Puppet agent pulls from the Puppet master, the Puppet agent must have the ability to modify most of the configuration on the system. For this reason, the Puppet agent runs as the `root` user or a user that is assigned the `Puppet Management` rights profile.

The time interval at which agents poll the master is configurable per agent as shown in "Configuring the Puppet Master and Agents" on page 24.

The Puppet agent gains communication privileges from the Puppet master system by requesting a Secure Socket Layer (SSL) certificate the first time the agent contacts the master system. Subsequently, the Puppet agent receives configuration updates from the master only if the certificate is still valid.

The master also authenticates to the agents so that the agents do not receive incorrect configuration information.

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

The Puppet 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/puppetlabs/puppet/ssl` directory. This `ssl` directory contains separate directories for keys, certificates, and signed requests, as well as those requests that are awaiting a signature. These directories exist on both the master and the agent. For more information, see Directories: SSLdir.

The Puppet master 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/puppetlabs/puppet/ssl/ca/signed` directory on the master and in the `/etc/puppetlabs/puppet/ssl/certs` directory on the agent.

Puppet Manifests

Puppet uses a Declarative Domain Specific Language (DSL) that is similar to Ruby to define *states*. Puppet configuration specifications are recorded in files called *manifests*. Manifests have a `.pp` file extension and are located on the Puppet master. Manifests declare resources that define various aspects of a system, such as files, software packages, and services. Resources are grouped into classes, which expose parameters that can affect their behavior. Classes and configuration files are organized into modules. See also the Puppet Glossary and the Puppet language Resources.

Use the Puppet `site.pp` manifest to define global configuration that applies to all of the managed agent nodes. A site manifest can also include node-specific code. A *node definition* (or *node statement*) is a block of Puppet code that is only included in the catalogs of the nodes named after the `node` keyword. This feature enables you to assign specific configurations to specific nodes. For more information, see the Puppet language Node definitions.

A manifest can group several resources together into a *class*. Then you can use the class to apply the resources to the specified nodes. A Puppet class can include resources, variables, and additional, advanced attributes. When you assign a class to a node, that node gets all of the configurations that are part of that class. Include class declarations in a manifest as described in "Writing Puppet Classes" on page 45 and Chapter 4, "Writing Puppet Manifests, Classes, and Modules in Oracle Solaris".

Puppet *modules* are self-contained collections of files and directories that can contain Puppet manifests and other objects, including files and templates. Puppet uses modules to find the classes and types that can be used for configuration management within your IT infrastructure. Puppet loads classes and defined types that are stored in modules. Declare these classes and types by name in a manifest as described in "Writing Puppet Modules" on page 47.

Puppet Privileges and Authorizations

Use one of the following methods to gain the privilege you need to configure and administer Puppet. See *Securing Users and Processes in Oracle Solaris 11.4* for more information about roles and profiles, including how to determine which role or profile you need.

Roles

Use the `roles` command to list the roles that are assigned to you. Use the `su` command with the name of the role to assume that role. As this role, you can execute any commands that are permitted by the rights profiles that are assigned to that role.

Rights profiles

You must have the `Puppet Management` rights profile to administer Puppet. Use the `profiles` command to list the rights profiles that are assigned to you.

Use one of the following methods to execute commands that your rights profiles permit you to execute:

- Use a profile shell such as `pfbash` or `pfksh`.
- Use the `pfexec` command in front of the command that you want to execute. In general, you must specify the `pfexec` command with each privileged command that you execute.

sudo command

Depending on the security policy at your site, you might be able to use the `sudo` command with your user password to execute a privileged command.

♦ ♦ ♦ 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:

Installing Puppet

Puppet is not installed on your Oracle Solaris system by default. You must install the Puppet software package (`system/management/puppet`) on the Puppet master and on each of the managed nodes (agents).

Use the following command to determine whether the Puppet package is installed on this system:

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

The following command displays more information about the Puppet package. In addition to the package summary and description, this output shows that Puppet is not installed and that the version of Puppet that is delivered by this package is version 5.5.0.

```
$ 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: 5.5.0
        Branch: 11.5.0.0.0.22.0
Packaging Date: Tue May 29 21:22:40 2018
          Size: 5.70 MB
          FMRI: pkg://solaris/system/management/puppet@5.5.0-11.5.0.0.0.22.0:
20180529T212240Z
   Project URL: http://puppetlabs.com/
    Source URL: https://github.com/puppetlabs/puppet/archive/5.5.0.tar.gz
```

Perform Pre-Installation Tasks

Prior to installing the Puppet IPS package on the master server and on the managed nodes, 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 managed nodes.
- Designate the managed nodes (agents).
- Configure the Domain Name System (DNS) protocol on both the master and the 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 11.4 Directory and Naming Services: DNS and NIS*.
- Ensure that time-keeping on the Puppet master is configured accurately as described in the following procedure.

▼ How to Configure NTP on the Puppet Master

Perform this procedure on the Puppet master *prior* to installing the `system/management/puppet` Puppet IPS package.

Because the Puppet master server acts as the certificate authority, 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 *Managing Clock Synchronization in Oracle Solaris 11.4*.

Before You Begin Assume the `Puppet Management` rights profile. Use `pfedit` to edit the `ntp.conf` file.

1. **Create an NTP configuration file.**

   Copy server information from the `/etc/inet/ntp.client` file to `/etc/inet/ntp.conf`.

   In this procedure, the following four time servers are specified so that a backup is available if one time server fails:

   ```
   server 0.pool.ntp.org
   server 1.pool.ntp.org
   server 2.pool.ntp.org
   server 3.pool.ntp.org
   ```

2. **Add the required configuration parameters to the NTP configuration file.**

   Add the following parameters to the `/etc/inet/ntp.conf` file:

   ```
   driftfile /var/ntp/ntp.drift
   statsdir /var/ntp/ntpstats/
   filegen peerstats file peerstats type day enable
   filegen loopstats file loopstats type day enable
   ```

3. **Force an initial time synchronization.**

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

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

   ```
   # svcadm enable ntp
   ```

5. **Verify that NTP is working.**

   ```
   # ntpq -p
   ```

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

Next Steps You can also specify NTP configuration by using a Puppet manifest. See Chapter 4, “Writing Puppet Manifests, Classes, and Modules in Oracle Solaris”.

Install Puppet

Use the following command to install the Puppet package on the master first, and then on each agent. The Puppet master can be the same system as one of the Puppet agents.

```
$ pkg install puppet
           Packages to install:         7
           Mediators to change:         1
            Services to change:         2
     Estimated space available: 830.82 GB
Estimated space to be consumed: 349.45 MB
       Create boot environment:        No
Create backup boot environment:        No
          Rebuild boot archive:        No
...
```

Verify that Puppet is installed:

```
$ pkg list puppet
NAME (PUBLISHER)                                  VERSION                    IFO
system/management/puppet                          5.5.0-11.5.0.0.0.22.0      i--
```

Configuring the Puppet Master and Agents

After installing Puppet on the Puppet master and on the agents that the master will control, configure the master and the agents. One Puppet master can manage the configuration of many Puppet agents.

On Oracle Solaris, Puppet is configured by configuring SMF services. When you install the `system/management/puppet` package, you get the following SMF service instances on both the Puppet master and the Puppet agents:

```
$ svcs puppet
STATE          STIME    FMRI
disabled        8:17:58 svc:/application/puppet:agent
disabled        8:17:58 svc:/application/puppet:main
disabled        8:17:58 svc:/application/puppet:master
disabled        8:17:58 svc:/application/puppet:user
online          8:36:42 svc:/application/puppet:upgrade
```

- The `master` instance serves configuration to agents.
- The `agent` instance controls configuration of a particular node.
- The `main` instance holds shared configuration values.
- The `user` instance is used by the `puppet apply` command, as well as many of the less common `puppet` subcommands.
- The `upgrade` instance performs migration and cleanup steps if needed.

▼ How to Configure the Puppet Master and Agent

Before You Begin Become an administrator who is assigned the `Puppet Management` rights profile.

1. **On the agent, set server properties.**

   While the `puppet:agent` service is disabled, set the `ca_server` and `server` properties as shown in the example in Example 1, “Configuring the Puppet Master and Agent,” on page 26.

   Refresh the `puppet:agent` service.

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

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

   Run the `puppet agent` command with the `--test` option on the agent to create a new SSL key and request authentication between the agent and the master.

   ```
   $ puppet agent --test
   ```

3. **On the master, 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.

4. **On the master, 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 SSL configuration: autosigning certificate requests.

5. **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.

6. **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

Do not manually edit the `/etc/puppetlabs/puppet/puppet.conf` Puppet configuration file; your edits will be lost. This Puppet configuration file is generated from SMF property values. This configuration file is updated when you update the associated SMF property values, as shown in the following example. See Short list of important settings and Configuration Reference for descriptions of the properties shown here and other configuration values that you can set.

While the `puppet:agent` service is disabled, set the `ca_server` and `server` properties. The `server` property value is the hostname of the master server. Typically, the `ca_server` value is also the hostname of the master server.

```
# svccfg -s puppet:agent
svc:/application/puppet:agent> setprop config/ca_server = host: hostname
svc:/application/puppet:agent> setprop config/server = host: hostname
svc:/application/puppet:agent> setprop config/runinterval = astring: 1d
svc:/application/puppet:agent> refresh
svc:/application/puppet:agent> exit
```

Check that the values are correctly set:

```
# svcprop -p config puppet:agent
```

Any changes that you make by setting SMF property values are reflected in the `puppet.conf` file when you refresh the service instance and that instance is online. See below for an example.

On the agent, test the connection:

```
$ puppet agent --test
Info: csr_attributes file loading from /etc/puppetlabs/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
```

On the master, view outstanding authentication requests:

```
$ puppet cert list
"agent.example.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
```

Sign the outstanding request:

```
$ puppet cert sign agent.example.com
```

```
Notice: Signed certificate request for agent.example.com
Notice: Removing file Puppet:SSL:CertificateRequest agent at '/etc/puppetlabs/puppet/
ssl/ca/requests/solaris.pem'
```

On the agent, retest the connection:

```
$ 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
```

Enable the service:

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

View the configuration file. Parts of the configuration file are omitted for this example:

```
$ cat /etc/puppetlabs/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]

ca_server = hostname
logdest = /var/log/puppetlabs/puppet/puppet-agent.log
runinterval = 1d
server = hostname
```

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.

Puppet service logs are stored in the following locations:

```
$ svcprop -p config/logdest puppet:master
/var/log/puppetlabs/puppet/puppet-master.log
$ svcprop -p config/logdest puppet:agent
/var/log/puppetlabs/puppet/puppet-agent.log
```

Puppet SMF service instance logs are stored in the following locations:

```
$ svcs -L puppet:master puppet:agent
/var/svc/log/application-puppet:master.log
/var/svc/log/application-puppet:agent.log
```

Use the `svcs -Lv` command to view the complete log file of the service:

```
$ svcs -Lv puppet:master
```

The following example shows the kind of information you might see in the service log for the Puppet master:

```
2018-06-14 13:33:05 -0700 Puppet (notice): Signed certificate request for ca
2018-06-14 13:33:05 -0700 Puppet (notice): solaris.local has a waiting certificate
 request
2018-06-14 13:33:05 -0700 Puppet (notice): Signed certificate request for solaris.local
2018-06-14 13:33:05 -0700 Puppet (notice): Removing file Puppet::SSL::CertificateRequest
 solaris.local at
 '/etc/puppetlabs/puppet/ssl/ca/requests/solaris.local.pem'
2018-06-14 13:33:05 -0700 Puppet (notice): Removing file Puppet::SSL::CertificateRequest
 solaris.local at
 '/etc/puppetlabs/puppet/ssl/certificate_requests/solaris.local.pem'
2018-06-14 13:33:05 -0700 Puppet (warning): The WEBrick Puppet master server is
 deprecated and will be removed in a future
 release. Please use Puppet Server instead. See http://links.puppet.com/deprecate-rack-
webrick-servers for more information.
   (at /usr/ruby/2.1/lib/ruby/vendor_ruby/2.1.0/puppet/application/master.rb:207:in
 `main'
2018-06-14 13:33:06 -0700 Puppet (notice): Reopening log files
2018-06-14 13:33:06 -0700 Puppet (notice): Starting Puppet master version 5.5.0
2018-06-14 13:33:33 -0700 Puppet (notice): Caught TERM; exiting
2018-06-14 13:33:44 -0700 Puppet (warning): The WEBrick Puppet master server is
 deprecated and will be removed in a future
```

```
 release. Please use Puppet Server instead. See http://links.puppet.com/deprecate-rack-
webrick-servers for more information.
   (at /usr/ruby/2.1/lib/ruby/vendor_ruby/2.1.0/puppet/application/master.rb:207:in
 `main'
2018-06-14 13:33:44 -0700 Puppet (notice): Reopening log files
2018-06-14 13:33:44 -0700 Puppet (notice): Starting Puppet master version 5.5.0
```

♦ ♦ ♦ 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:

Puppet Resources and Resource Types

A *resource* is a specific piece of a system that Puppet is managing. A *resource type* is the kind of resource, such as a service, file, package, host, interface, or user. A *resource provider* implements support for a specific implementation of a given resource type, such as a resource type on a particular platform. Resource types can be built-in, defined, or custom.

The following are examples of built-in resource types and their providers on Oracle Solaris:

- The `zone` built-in resource type has only one provider: `solaris`.
- One of the many providers for the `package` resource type is Oracle Solaris `pkg`.

  Note that the `sun` provider for the `package` resource type is only for Oracle Solaris 10 packages and earlier.
- One of the many providers for the `service` resource type is Oracle Solaris `smf`.
- The `user_role_add` provider for the `user` resource type implements user and role management on Oracle Solaris.

The `oracle-solaris_providers` IPS package delivers Oracle Solaris providers. This package is installed automatically when you install Puppet.

additional resource types that are distributed in individual Puppet modules that enable you to manage certain configuration such as networking or naming services. Oracle Solaris also includes Puppet resource types that manage specific Oracle Solaris features such as Oracle Solaris Zones.

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 managed nodes.

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

Use the following command to display all of the Puppet resource types that are available on this system. Output from this command includes both Oracle Solaris-specific types and core Puppet types.

```
# puppet resource --types
address_object
address_properties
anchor
augeas
boot_environment
...
zfs
zfs_acl
zone
zpool
```

Use the following command to display all of the available resource types with a description of each type:

```
# puppet describe --list
address_object  - Manage the configuration of Oracle Solaris ad ...
address_properties - Manage Oracle Solaris address properties
anchor          - A simple resource type intended to be used as ...
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 ...
evs             - Manage the configuration of Oracle Solaris Elastic ...
evs_ipnet       - Manage the configuration of IPnet (subnet of  ...
```

```
evs_properties  - Manage global properties of EVS (Elastic V ...
evs_vport       - Manage the configuration of EVS VPort
exec            - Executes external commands. Any command in an ...
file            - Manages files, including their content, owner ...
file_line       - Ensures that a given line is contained within ...
filebucket      - A repository for storing and retrieving file  ...
group           - Manage groups. On most platforms this can  ...
host            - Installs and manages host entries. For most  ...
ilb_healthcheck - Manage Solaris Integrated Load Balancer (ILB) ...
ilb_rule        - Manage Solaris Integrated Load Balancer (ILB) ...
ilb_server      -  Manage Solaris Integrated Load Balancer (ILB ...
ilb_servergroup - Manage Solaris Integrated Load Balancer (ILB) ...
ini_setting     -
ini_subsetting  -
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. Specify ...
ldap            - Manage the configuration of the LDAP client  ...
link_aggregation - Manage the configuration of Oracle Solaris ...
link_properties - Manage Oracle Solaris link properties
macauthorization - Manage the Mac OS X authorization database.  ...
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 putting ...
nagios_command  - The Nagios type command. This resource type ...
nagios_contact  - The Nagios type contact. This resource type ...
nagios_contactgroup - The Nagios type contactgroup. This resource ...
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 ...
nagios_hostgroup - The Nagios type hostgroup. This resource type ...
nagios_service  - The Nagios type service. This resource type  ...
nagios_servicedependency - The Nagios type servicedependency. This  ...
nagios_serviceescalation - The Nagios type serviceescalation. This  ...
nagios_serviceextinfo - The Nagios type serviceextinfo. This resource ...
nagios_servicegroup - The Nagios type servicegroup. This resource  ...
nagios_timeperiod - The Nagios type timeperiod. This resource  ...
nis             - Manage the configuration of the NIS client  ...
notify          - Sends an arbitrary message to the agent run-time ...
nsswitch        - Name service switch configuration data
package         - Manage packages. There is a basic dichotomy  ...
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 ...
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 ...
selmodule       - Manages loading and unloading of SELinux ...
service         - Manage running services. Service support ...
solaris_vlan    - Manage the configuration of Oracle Solaris VLANs ...
ssh_authorized_key - Manages SSH authorized keys. Currently only  ...
sshkey          - Installs and manages ssh host keys. By default ...
stage           - A resource type for creating new run stages.  ...
svccfg          - Manage SMF service properties with svccfg(8).
system_attributes - Manage system attributes on ZFS files. See ...
tidy            - Remove unwanted files based on specific criteria ...
user            - Manage users. This type is mostly built to ...
vlan            - Manages a VLAN on a router or switch.
vni_interface   - Manage the configuration of Solaris VNI ...
vnic            - Manage the configuration of Oracle Solaris ...
whit            - Whits are internal artifacts of Puppet's current ...
yumrepo         - The client-side description of a yum repo ...
zfs             - Manage zfs. Create destroy and set properties ...
zfs_acl         - Manage NFSv4 ACL Specifications on ZFS Files ...
zone            - Manages Solaris zones.
zpool           - Manage zpools. Create and delete zpools. The ...
```

Use the following command to display information about a specific resource type, such as the Oracle Solaris specific resource type `zone`.

```
# puppet describe zone
zone
====
Manages Oracle 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
```

For more examples that are specific to Oracle Solaris, see Chapter 5, “Using Puppet to Manage System Configuration in Oracle Solaris”.

Declaring Puppet Resources

A *resource definition* specifies the content and behavior of a resource class or type. A resource class or type must be defined before it can be declared within a manifest.

A *resource declaration* describes the desired state for a resource on the managed system. The Puppet master compiles resource declarations into a catalog for each managed system. Puppet applies each catalog to the corresponding managed system to ensure that the state of that system matches the specified desired state.

Puppet uses the following format for resource declarations:

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

`resource_type`

The type of resource that is being declared. The `resource_type` cannot include quotation marks.

`title`

An identifying string. Every `resource_type` must have a unique title. The title does not have to match the name of the resource.

`attribute`

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/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.

You can use any amount of white space in a resource declaration.

For more detailed information, see 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 preceding example, two `zone` resources are 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 => 2.4.6
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
```

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 preceding 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, go to Facter documentation.

♦ ♦ ♦ 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 the pre-environment 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 44.

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/puppetlabs/code/environments/production/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 Resources and Resource Types" on page 31.
- Familiarize yourself with the basic syntax that you use to declare resources within a Puppet manifest. See "Declaring Puppet Resources" on page 35. 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.**

   See "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.4*.

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

   ```
   # touch /etc/puppetlabs/code/environments/production/manifests/site.pp
   ```

   This file should always reside in the `/etc/puppetlabs/code/environments/production/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/puppetlabs/code/environments/production/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/puppetlabs/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/puppetlabs/code/environments/production/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 preceding 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, go to https://puppet.com/docs/puppet/5.5/lang_classes.html.

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.

Puppet defines 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, go to https://puppet.com/docs/puppet/5.5/lang_classes.html.

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 `/etc/puppetlabs/code/modules` directory on the Puppet master.

Under the `examplecloud` class are several manifests (`/etc/puppetlabs/code/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 preceding 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 preceding 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, go to `https://puppet.com/docs/puppet/5.5/bgtm.html`.

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

- `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/puppetlabs/code/modules` directory:

```
# cd /etc/puppetlabs/code/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/puppetlabs/code/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        463 Mar  3 13:24 ntp.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 https://puppet.com/docs/puppet/5.5/modules_fundamentals.html#example.

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 44.

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 42.

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
    ...
```

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 Oracle Solaris.  The value will be
    validated according to system rules, which in the case of
    Oracle 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 preceding 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: 7.70
        Branch: 11.5.0.0.0.56.0
Packaging Date: Fri Sep 27 17:05:48 2019
          Size: 22.61 MB
          FMRI: pkg://solaris/diagnostic/nmap@7.70-11.5.0.0.0.56.0:20190927T170548Z
```

The output of the preceding 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 Oracle 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 preceding 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/puppetlabs/code/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 preceding command shows that the non-global zone `systemazone` is configured, installed, and running.

♦ ♦ ♦ APPENDIX A

Using MCollective With Puppet for Server Orchestration in Oracle Solaris

The Marionette Collective, or MCollective, is a framework for building server orchestration. MCollective provides the capability for working with large numbers of servers. In Oracle Solaris, you can use MCollective in conjunction with Puppet to more easily and efficiently perform configuration management.

This appendix includes the following topics:

This appendix provides basic information about the MCollective deployment in Oracle Solaris. For in-depth background information and instructions on configuring and using MCollective, go to `https://puppet.com/docs/mcollective/current/index.html`.

About MCollective Support in Oracle Solaris

MCollective is a framework that enables you to build server orchestration or parallel job execution systems. MCollective is primarily used to programmatically execute actions on clusters of servers through the use of state-of-the-art tools, such as publish/subscribe middleware.

MCollective also employs contemporary philosophies, such as real-time discovery of network resources that uses metadata rather than host names to deliver scalable and extremely fast parallel execution environments.

In Oracle Solaris, MCollective complements the use of Puppet by providing mechanisms that enable you to more easily manage system configurations and customizations within a cloud environment, such as an OpenStack deployment.

Some common uses for MCollective include the following:

- Enabling services across a family of client virtual machines (VMs)
- Installing and verifying installation of software packages on a family of client VMs

For more detailed information about using MCollective with Puppet, go to https://puppet.com/docs/mcollective/current/reference/integration/puppet.html.

Deploying MCollective in Oracle Solaris

An MCollective deployment typically consists of the following components: server, client, and middleware. In Oracle Solaris, the MCollective feature is managed through the Service Management Facility (SMF). The middleware component for MCollective is managed through the RabbitMQ open source message broker software. For more information, go to https://puppet.com/docs/mcollective/current/reference/plugins/connector_rabbitmq.html.

The following MCollective features and functionality are supported:

- **MCollective installation**

  MCollective (and its agents) is delivered through the `system/management/mcollective` IPS package. This package is not installed by default on your Oracle Solaris system.
- **MCollective SMF service instance**

  MCollective is managed through a single SMF service, `svc:/application/mcollective:default`.This service is not enabled by default. Enabling the service instance starts the `/usr/sbin/mcollectived` daemon, which is the server application.
- **MCollective agents**

  MCollective agents enable the tool to use Puppet to perform various configuration management tasks on multiple servers, including remote servers. Agents work by leveraging the Puppet framework to provide orchestration for the configuration management of Oracle Solaris servers. The way that MCollective agents are written make is possible for MCollective to make use of Puppet's default behavior if Puppet modules are present on a system.

  The following four agents (plug-ins) are supported:

`facter`	Enables MCollective to use the Facter utility as a source for gathering facts about a system.
`package`	Enables you install, uninstall, update, purge and query the status of software packages on your system.
`puppet`	Enables you to enable, disable, and run the Puppet daemon.

`service` Enables you to stop, start, restart, and query the status of the services on your system.

These plug-ins use the secure client-server communication that is provided by the MCollective framework. The MCollective framework uses the standard SSL/TLS certificates for client and server verification, which is similar to Puppet.

- **MCollective collectives**

  Collectives (also referred to as subcollectives) are part of the MCollective framework. Collectives provide a mechanism to group nodes so that you can perform various configuration actions on the entire group (or collective). For details, go to `https://puppet.com/docs/mcollective/current/reference/basic/subcollectives.html`.

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

  MCollective includes a command-line interface (CLI) that you can use to perform various administrative actions.

  You can display all of the available MCollective subcommands and their usage as follows:

  ```
  $ /usr/bin/mco --help
  ```

  For details, go to `https://docs.puppet.com/mcollective/current/reference/basic/basic_cli_usage.html`.

- **MCollective privileges and authorizations**

  No special privileges are required to administer MCollective in Oracle Solaris. However, to administer Puppet, you must be assigned the `Puppet Management` rights profile or assume the `root` role.

Administering the MCollective Feature in Oracle Solaris

The following examples show how you can administer various aspects of the MCollective feature within your environment.

EXAMPLE 9 Searching for Available MCollective Nodes

This example shows how you would search for available MCollective nodes. In this example, two MCollective nodes are detected.

```
# mco ping
mco-agent                               time=35.18 ms
mco-agent-test                          time=37.10 ms
```

```
---- ping statistics ---- 2 replies max: 37.10 min: 35.18 avg: 36.14
```

EXAMPLE 10 Checking Whether SMF Services Are Running on MCollective Nodes

The following example shows how you would determine the status of SMF services that are running on MCollective nodes.

```
# mco service status repositories-setup
* [ ============================================================> ] 2 / 2
mco-agent: running mco-agent-test: running
Summary of Service Status:
running = 2

Finished processing 2 / 2 hosts in 112.35 ms
```

EXAMPLE 11 Checking the Status of IPS Packages on All of the Nodes in a Collective

You divide your site into *collectives* so that you can perform various configuration actions on all of the systems that are part of a collective.

The following example shows how you would check the status of the `pkg:/system/core-os` IPS package on all of the nodes that are running in a collective named `mcollective`.

```
# mco package status -T mcollective core-os
* [ ============================================================> ] 2 / 2
mco-agent: system/core-os-5.12-5.12.0.0.0.91.2. mco-agent-test:
system/core-os-5.12-5.12.0.0.0.94.0.
Summary of Arch:
No aggregate summary could be computed
Summary of Ensure:
5.12-5.12.0.0.0.94.0 = 1 5.12-5.12.0.0.0.91.2 = 1

Finished processing 2 / 2 hosts in 1194.85 ms
```

Index

N

O

P

R

S

T

U

W

Z