Using Puppet to Perform Configuration Management in Oracle Solaris 11.4

E72062-02
August 2023

ORACLE

Using Puppet to Perform Configuration Management in Oracle Solaris 11.4,

E72062-02

Primary Author: Cathleen Reiher, Alta Elstad, Sharon Veach

Contents

Using This Documentation

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.

1

Using Puppet to Manage System Configuration in Oracle Solaris

What's New in Puppet in Oracle Solaris 11.4

Puppet 7.21.0. Oracle Solaris 11.4 Support Repository Update (SRU) 57 introduces the Puppet 7.21.0 software.

When Puppet 6.26 is installed on a system, an update to Oracle Solaris 11.4 SRU 57 replaces the Puppet 6.26 Agent with the Puppet 7.21.0 Agent.

> **Note:**
>
> Future Oracle Solaris 11.4 SRUs might include updated versions of Puppet 7.

Earlier versions of Puppet have been included with the following Oracle Solaris 11 software:

- Oracle Solaris 11.4 SRU 45 through SRU 56 supports the Puppet 6.26.0 Agent component.
- Oracle Solaris 11.4 through SRU 44 supports the Puppet 5.5 Master and Puppet Agent components.
- Oracle Solaris 11.3 supports the Puppet 3.6 software.

> **Note:**
>
> Do not install a different version of the Puppet software on your system.

For information about alternate ways to run the Puppet Server on a system that runs at least Oracle Solaris 11.4 SRU 57, see `https://puppet.com/docs/puppet/7/puppet_index.html`.

While the Puppet software is updated automatically, you must update the Puppet configuration on your system manually. For information about maintaining and updating your Puppet configuration, see Puppet Documentation.

The Puppet web site does not include information about updating Oracle Solaris SMF service properties. Updating Puppet software on an Oracle Solaris 11.4 system migrates existing SMF service property values automatically. However, you must remove obsolete property values and update property values that require changes manually. See Configuring the Puppet Agents.

Puppet Features in Oracle Solaris

An Oracle Solaris installation does not include the Puppet software by default. So, you must install the Puppet Image Packaging System (IPS) package on any node that you want to run the Puppet Agent. See Installing Puppet.

This section introduces the Puppet features and functionality that are available with Oracle Solaris 11.4.

Puppet Utilities

The Puppet package includes the following utilities:

Facter
Puppet uses Facter to discover facts about a particular system, such as OS type, CPUs, or memory size. Facter gathers and sends system information to the Puppet Server, which compiles the information into catalogs. A catalog describes the system state for a specific set of resources and lists all of the resources and resource dependencies to manage. See Gathering Information About a System by Using Facter and Facter documentation on the Puppet web site.

Hiera
Hiera is a cross-platform, key-value lookup tool that you can use to manage configuration data. Using Hiera to maintain site-specific data rather than using a Puppet manifest avoids repetition and enables you to create a more generic type of manifest that you can reuse for multiple systems.
Puppet classes can request required data and use Hiera to act as a site-wide configuration file. When Hiera is loaded, Puppet uses this Hiera configuration file instead of the `/etc/puppetlabs/puppet/hiera.yaml` global file. 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)
├── 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/
$ 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

The Puppet software package installs 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:user
online          8:36:42 svc:/application/puppet:upgrade
```

For information about enabling and using these services, see Configuring the Puppet Agents.

Puppet Configuration File

The `/etc/puppetlabs/puppet/puppet.conf` Puppet Agent configuration file defines many system resources and default values.

Puppet uses `svc:/application/puppet` service property values to generate the `puppet.conf` configuration file. Only use the SMF commands to apply property value changes you want to the `puppet.conf` file. See *Managing System Services in Oracle Solaris 11.4* and the `svccfg`(8) man page.

Puppet Resources and Resource Types

Puppet uses resources to represent the desired state of a system configuration. For example, a resource might specify when and how to run services, specify which software packages to install, and specify certain components of networking and naming service configuration.

Each resource has a resource type that is defined by a name and a set of attributes (or parameters) and values that you can specify in a Puppet manifest. The parameters and their values depend on the configuration type being managed.

The following example `puppet describe` command obtains information about the `zone` resource:

```
$ puppet describe zone

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

See 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.

Puppet Command-Line Interface

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

- Initial handshake between the Puppet Server and Puppet Agent nodes
- Trial run for testing purposes
- Manage certificates
- Generate and manage reports
- Access plug-ins
- Manage resources
- Display status

- Troubleshoot and debug Puppet issues

The `puppet` command has the following syntax:

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

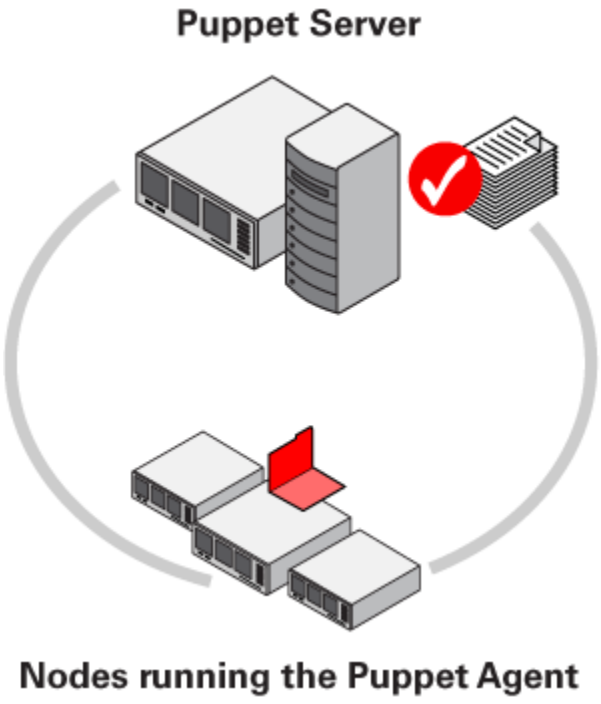
The `puppet help` command lists and describes all its subcommands.

The `puppet help` *subcommand* command provides usage information about the specified subcommand.

The `puppet help` *subcommand action* command provides usage information about the specified subcommand and its specified action.

How Puppet Works

Puppet enables you to define the software and configuration that a system requires and then maintain that specified state. The following graphic shows the Puppet agent-server architecture where a Puppet Server (server) node controls the configuration information for a fleet of managed nodes that run the Puppet Agent (agent).



The Puppet Server controls the configuration information. Each Puppet Agent-managed node configures itself by requesting its own configuration from the server.

Puppet uses the Facter utility, which is included in the Puppet software package, to discover information about a system. See Gathering Information About a System by Using Facter.

The server uses manifests to describe the resources to configure for each agent-managed node. Also, you can create a site manifest to define a global configuration that applies to all of the agent-managed nodes.

Typically, agent-managed nodes run the Puppet Agent application as a background service. The agent collects configuration information about itself and sends that information to the

server for compilation into a catalog. The catalog describes the configuration of each agent-managed node. The agent-managed node uses the catalog to apply any configuration updates.

Puppet uses a pull mode where agents poll the server at regular intervals to retrieve site-specific and node-specific configuration information. See Overview of Puppet's Architecture.

Puppet Agent-Server Model

Puppet uses an agent-server model, where the Puppet Server (server) manages important configuration information for all of the physical and virtual nodes on which the Puppet Agent (agent) runs.

Each agent-managed node regularly polls the server to acquire and apply updated configuration information.

The Puppet Server

The Puppet Server (server) node is the primary source of Puppet configuration data and authority. The server is a Ruby and Clojure application that runs on the Java Virtual Machine (JVM). The server uses Ruby to compile catalogs and to serve files by using several JRuby interpreters. The server uses Clojure to provide a certificate authority. Note that the configuration of some components depends on the configuration of other components. As a result, the server must house information about all the components on each managed node.

The server has the following responsibilities:

- Compiles the catalog for each Puppet Agent-managed node
- Transfers files from a file server
- Sends reports to a central server

The server uses the `puppet` user to perform the following tasks:

- Stores configuration manifests in the Puppet manifests directory
- Accepts Secure Socket Layer (SSL) certificates from Puppet Agents (agents)
- Transfers files to agents
- Creates catalogs

The `puppet` user runs the Puppet Server. The `puppet` user is a member of the `puppet` group.

Puppet Agents

The Puppet Agent (agent) is a daemon that runs on a managed node. To apply the configuration that the agent pulls from the Puppet Server (server), the agent must be able to modify most of the configuration on the system. For this reason, the agent runs as the `root` user or a user that is assigned the `Puppet Management` rights profile.

You can configure the time interval at which each agent polls the server. See Configuring the Puppet Server and Agents.

The first time an agent contacts the server, the agent requests an SSL certificate to obtain communication privileges. Subsequently, the agent receives configuration updates from the server only if the certificate is still valid.

To ensure that agents do not receive incorrect configuration information, the agent must authenticate with the server.

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 Puppet Agent (agent) and Puppet Server (server) authentication and verification. Puppet also uses SSL/TLS to encrypt the traffic flow between the server and agents. The default hash is SHA-256.

The Puppet encryption method performs the following tasks:

- Authenticates any agent to the server
- Authenticates the server on any agent
- Prevents communication eavesdropping between the server 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 for those requests that await a signature. These directories exist on the server and on each agent. See Directories: SSLdir.

The server generates its own CA certificate and private key, initializes the Certificate Revocation List (CRL), and then generates another certificate called the *server certificate*. This certificate handles SSL and TLS communications and is sent to the agent. During the server and agent exchange, the CA is stored in the `/etc/puppetlabs/puppet/ssl/ca/signed` directory on the server and in the `/etc/puppetlabs/puppet/ssl/certs` directory on the agent.

Agents automatically request certificates through the server's HTTP endpoint. Use the `puppetserver ca` command to inspect requests and to sign new certificates.

Puppet Manifests

Puppet uses a declarative Domain Specific Language (DSL) that is similar to Ruby to define states. Puppet records configuration specifications in files called manifests, which use a `.pp` file extension and are located on the Puppet Server. 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 resource behavior. Classes and configuration files are organized into modules. See the Puppet Glossary and the Puppet language Resources.

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

A manifest can group several resources together into a class that you can use to apply resources to specific 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 and 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.

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.

2

Getting Started With Puppet in Oracle Solaris

Installing Puppet

Puppet is not installed on your Oracle Solaris system by default. You must install the Puppet software package, `system/management/puppet`, on each managed node that runs the Puppet Agent.

The following command determines whether the Puppet package is installed on this system:

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

The `pkg info -r puppet` command shows 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 7.21.0.

```
$ pkg info -r puppet
           Name: system/management/puppet
        Summary: Puppet agent - The Puppet daemon that runs on the target system
                 (node).
    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: 7.21.0
         Branch: 11.4.63.0.0.153.0
Packaging Date: Fri Aug 11 06:54:55 2023
           Size: 5.53 MB
           FMRI: pkg://solaris/system/management/
puppet@7.21.0-11.4.63.0.0.153.0:20230811T065455Z
    Project URL: http://puppetlabs.com/
     Source URL: https://github.com/puppetlabs/puppet
```

Preparing to Install Puppet

Prior to installing the Puppet IPS package on the managed nodes that run the Puppet Agent (agent), perform the following tasks:

- Designate a system to function as the Puppet Server (server).

  Ensure that you install and configure Puppet on the server or servers before you install Puppet on any of the agent-managed nodes.

- Designate the agent-managed nodes.
- Configure the Domain Name System (DNS) protocol on both the server 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*.

Install Puppet

Install the Puppet package on each node you want to manage using the Puppet Agent:

```
$ sudo pkg install puppet
           Packages to install:  2
            Services to change:  2
       Create boot environment: No
Create backup boot environment: No

DOWNLOAD                                  PKGS         FILES    XFER (MB)   SPEED
Completed                                  2/2     1269/1269      1.9/1.9  143k/s

PHASE                                          ITEMS
Installing new actions                   1326/1326
Updating package state database                Done
Updating package cache                          0/0
Updating image state                           Done
Creating fast lookup database                  Done
Updating package cache
```

Verify that Puppet is installed:

```
$ pkg list puppet
NAME (PUBLISHER)                          VERSION                    IFO
system/management/puppet (solaris)        7.21.0-11.4.63.0.0.153.0   i--
```

Configuring the Puppet Agents

After installing Puppet on the managed Puppet Agent (agent) nodes that the Puppet Server (server) controls, configure the agents. One server can manage the configuration of many agents.

Oracle Solaris uses Service Management Facility (SMF) services to configure Puppet. When you install the `system/management/puppet` package, the following SMF service instances run on the 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:user
online          8:36:42 svc:/application/puppet:upgrade
```

- The `agent` instance controls the configuration of a particular node.
- The `main` instance holds shared configuration values.
- The `user` instance is used by the `puppet apply` command and by other `puppet` subcommands.
- The `upgrade` instance performs migration and cleanup steps, if needed.

How to Configure Puppet Agents

1. Become an administrator who is assigned the `Puppet Management` rights profile.
2. Set Puppet Server properties on the Puppet Agent.

   While the `puppet:agent` service is disabled, set the `ca_server` and `server` properties as shown in the example of Configuring the Puppet Agent.

   Refresh the `puppet:agent` service.

   > **Note:**
   >
   > Do not enable the `puppet:agent` service instance until after the Puppet Agent (agent) requests the certificate and the agent successfully signs in to the Puppet Server (server).

3. Test the connection from the agent to the server.

   On the agent, run the `puppet agent --test` command to create a new SSL key and request authentication between the agent and the server.
4. On the node that runs the Puppet Server, identify any outstanding certificate requests coming from agents that are attempting to connect to the server.

   Run the following command on the server:

   ```
   $ puppetserver ca list
   ```

   This command output should show that the agent is issuing a request.
5. On the server, sign the certificate for the agent that makes the request.

   ```
   $ puppetserver ca sign agent
   ```

   > **Note:**
   >
   > It is best to sign certificates manually for Puppet. However, if your environment does not require manually signed certificates, configure the CA Puppet Server to sign certain CSRs automatically. See SSL configuration: autosigning certificate requests.

6. Retest the connection from the agent to the server.

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

   This step ensures that the authentication between the server and the agent has occurred.
7. Enable the SMF service instance for the agent.

   ```
   $ svcadm enable puppet:agent
   $ svcs puppet:agent
   ```

   The output should show that the SMF `puppet:agent` service instance is online.

Example 2-1 Configuring a Puppet Agent

Do not edit the `/etc/puppetlabs/puppet/puppet.conf` Puppet configuration file manually. Such changes are not saved. SMF property values generate this Puppet configuration file. This configuration file is updated when you update the associated SMF property values, as shown in the following example. For descriptions of the properties shown here and other configuration values that you can set, see Short list of important settings and Configuration Reference.

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

In this example, the host name of the Puppet Server is `pupsvr` and the fully qualified domain name of the Puppet Agent is `agent.example.com`.

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

Verify that the property values are correct:

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

When you refresh the `puppet:agent` service instance and that instance comes online, any changes that you make by setting SMF property values are reflected in the `puppet.conf` file.

Test the connection on the `agent.example.com` agent:

```
$ puppet agent --test
Info: csr_attributes file loading from /etc/puppetlabs/puppet/csr_attributes.yaml
Info: Creating a new SSL certificate request for agent.example.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
```

Identify outstanding authentication requests on the server:

```
$ puppetserver ca 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:

```
$ puppetserver ca 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'
```

Retest the connection on the agent:

```
$ puppet agent --test
Info: Caching certificate for agent.example.com
Info: Caching certificate_revocation_list for ca
Info: Caching certificate for agent.example.com
Info: Retrieving plugin
```

```
Info: Caching catalog for agent.example.com
Info: Applying configuration version '1400782295'
Notice: Finished catalog run in 0.18 seconds
```

Enable the `puppet:agent`service:

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

View the following abridged example configuration file:

```
$ 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 = pupsvr
logdest = /var/log/puppetlabs/puppet/puppet-agent.log
runinterval = 1d
server = pupsvr
```

Next Steps

After you install Puppet and perform all of the necessary configuration and validation tasks, you are ready to use Puppet to manage system configurations.

For more information about using Pupput in Oracle Solaris, see:

- Working With Puppet Resources and Resource Types in Oracle Solaris
- Writing Puppet Manifests, Classes, and Modules in Oracle Solaris
- Using Puppet to Manage System Configuration in Oracle Solaris

Troubleshooting Puppet Issues in Oracle Solaris

The `puppet:agent` service logs most activity to the `syslog` service. The `syslog` configuration specifies where to save these messages. In Oracle Solaris, the default log location is the `/var/adm/messages` directory.

The following command shows the location of the service logs:

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

The following command shows the location of the Puppet SMF service instance logs:

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

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

```
$ svcs -Lv puppet:agent
```

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

```
2022-02-10 01:44:21 -0800 Puppet (info): Computing checksum on file /custom-
file.txt
2022-02-10 01:44:21 -0800 /Stage[main]/Main/File[/custom-file.txt] (info):
Filebucketed/
custom-file.txt to puppet with sum 103e5b132c69289dc18042a99b73fad9
2022-02-10 01:44:21 -0800 /Stage[main]/Main/File[/custom-file.txt]/content
(notice):
content changed '{md5}103e5b132c69289dc18042a99b73fad9' to
'{md5}e4b97f0c18e5bb0bb24d6dbe0db326f4'
2022-02-10 01:44:22 -0800 /Stage[main]/Main/User[testuser409]/ensure (notice):
created
2022-02-10 01:44:26 -0800 /Stage[main]/Main/Service[nfs/client]/ensure (notice):
ensure
changed 'stopped' to 'running' (corrective)
2022-02-10 01:44:26 -0800 /Stage[main]/Main/Service[nfs/client] (info):
Unscheduling
refresh on Service[nfs/client]
2022-02-10 01:44:26 -0800 Puppet (notice): Applied catalog in 5.30 seconds
```

3

Working With Puppet Resources and Resource Types in Oracle Solaris

Puppet Resources and Resource Types

A resource is a specific part of a system that Puppet manages. 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 list shows examples of Oracle Solaris built-in resource types and their providers:

- The `zone` built-in resource type has only the `solaris` provider.
- The `package` resource type has many providers including the `pkg` provider.

  Note that the `sun` provider is only for packages for systems that run up to Oracle Solaris 10.
- The `service` resource type has many providers including the `smf` provider.
- The `user` resource type, that implements user and role management, has the `user_role_add` provider.

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

Puppet modules include additional resource types that enable you to manage certain configurations 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 in manifests. Use a Puppet site manifest (`site.pp`) on the Puppet Server to define a 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 Writing Puppet Manifests, Classes, and Modules in Oracle Solaris.

Te following command output shows all of the Puppet resource types that are available on this system, including Oracle Solaris-specific types and core Puppet types:

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

```
zone
zpool
```

The following command output shows 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 ...
```

The following command output shows information about a specific resource type, such as the Oracle Solaris `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
```

For more Oracle Solaris examples, see 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. You must define a resource class or type before you can declare it in a manifest.

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

Puppet uses the following format for resource declarations:

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

`resource_type`
Specifies the type of resource that is being declared. Note that the `resource_type` cannot include quotation marks.

`title`
Specifies a unique identifying string for every `resource_type`. The title does not need to match the name of the resource.

attribute
Specifies the desired state of the resource. Most resources have a set of required attributes (or parameters) and might include a set of optional attributes.
Attribute-value pairs must consist of the following:

- An attribute name that 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 (`=>`) that is also called a “fat comma” or a “hash rocket”.
- A value that can be 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. 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 to 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',
}
```

This example declares two `zone` resources: a global zone and an installed kernel zone. Each of these resources has four attributes (or parameters): `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

The Facter utility gathers information about a system and sends it to the Puppet Server (server) for compilation into catalogs. A catalog specifies the configuration changes to apply to a managed node.

A catalog also includes the specified state of each resource. Based on these definitions, each system can apply its own configurations, as appropriate. After you apply the catalog to the system, the Puppet Agent (agent) generates a report and sends that report to the server. This report contains information about which resources are currently being managed on the target node, as well as about any changes that have been made to the node to achieve a desired state. See How Puppet Works.

The following `facter -p` command lists all of the available facts for this system:

```
# 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 help you to determine the configuration types that you can enforce on the system. For example, you could declare a file resource to populate a given file with platform-specific content.

The following example shows that the `osfamily` fact declares the platform within the file:

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

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

The previous example includes the `$file_contents` variable and a conditional check based on the value of the `osfamily` fact. Depending on the platform type, you would assign different contents to the file.

For more information, see the Facter documentation.

4

Writing Puppet Manifests, Classes, and Modules in Oracle Solaris

Writing a Puppet Site Manifest

After installing and configuring Puppet, you can write Puppet manifests to control the nodes that run the Puppet Agent. Puppet manifests use a Puppet-specific language that is similar to Ruby. 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 a configuration that you want applied to every node. So, using a site manifest is ideal for managing system-wide configurations, such as DNS servers, LDAP configurations, 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.

> **Note:**
>
> The `site.pp` manifest does not exist on the Puppet Server (server) by default. You must initially create this file on the server in the default `/etc/puppetlabs/code/environments/production/manifests` directory.

How to Write a Puppet Site Manifest

Prior to creating a Puppet site manifest, perform the following tasks:

- Determine which resource types to declare in the manifest. Use the `puppet describe` *resource-type* command to obtain this information. This command shows all of the available attributes, or parameters, for the specified resource type.

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

  See Puppet Resources and Resource Types.
- Familiarize yourself with the syntax to declare resources within a Puppet manifest. See Declaring Puppet Resources and Language: Resources.
- Familiarize yourself with the syntax to define a specific Oracle Solaris system configuration within a Puppet manifest. For examples, see Using Puppet to Manage System Configuration in Oracle Solaris.

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

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 Server (server).

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

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

   See 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 applying them permanently.

   ```
   # puppet apply -v --noop /etc/puppetlabs/code/environments/production/
   manifests/site.pp
   ```

   apply
   Applies the configuration to the Puppet manifest on the server.

   -v
   Indicates to use verbose mode.

   --noop
   Enables you to perform a dry run test without applying your changes.

   The Puppet Agent (agent) that runs on each node queries the server for configuration changes at regular intervals and then applies any required changes to the agent-managed node.

5. Verify that the node retrieved the latest configuration changes by checking the `/var/log/puppetlabs/puppet/puppet-agent.log` log file.

6. Apply the latest configuration changes manually.

   ```
   # puppet agent -t
   ```

   The `-t` (`--test`) option enables verbose logging, which causes the agent daemon to remain in the foreground, exits if the server's configuration is invalid (as in the case of a syntax error), then exits after running the configuration one time.

   Use the `puppet help agent` command to show all of the available Puppet subcommands. Also see the `puppet`(8) man page.

Example 4-1 Writing a Puppet Manifest

The following example shows how to declare resources in a Puppet site manifest. This example begins with the `site.pp` file in the correct directory on the server.

First, declare resources in the `site.pp` file. In this example, the `file` resource type is declared by specifying the `ensure` and `content` attributes. These attributes ensure that a `custom-file.txt` file exists in the `root` directory of 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, test the configuration's validity on the server as follows:

```
# puppet apply -v --noop /etc/puppetlabs/code/environments/production/manifests/site.pp
2022-02-10 01:44:21 -0800 Puppet (info): Computing checksum on file /custom-
file.txt
2022-02-10 01:44:21 -0800 /Stage[main]/Main/File[/custom-file.txt] (info):
Filebucketed/
custom-file.txt to puppet with sum 103e5b132c69289dc18042a99b73fad9
2022-02-10 01:44:21 -0800 /Stage[main]/Main/File[/custom-file.txt]/content
(notice): content changed '{md5}103e5b132c69289dc18042a99b73fad9' to
'{md5}e4b97f0c18e5bb0bb24d6dbe0db326f4'
2022-02-10 01:44:22 -0800 /Stage[main]/Main/User[testuser409]/ensure (notice): created
2022-02-10 01:44:26 -0800 /Stage[main]/Main/Service[nfs/client]/ensure (notice): ensure
changed 'stopped' to 'running' (corrective)
2022-02-10 01:44:26 -0800 /Stage[main]/Main/Service[nfs/client] (info): Unscheduling
refresh on Service[nfs/client]
2022-02-10 01:44:26 -0800 Puppet (notice): Applied catalog in 5.30 seconds
```

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

The agent that runs on each node queries the server for configuration changes at regular intervals and then applies any updates, as needed. Check the node's `/var/log/puppetlabs/puppet/puppet-agent.log` log file 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/puppetlabs/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 server for configuration changes at 30-minute intervals. Also, you can verify the configuration by checking whether the `custom-file.txt` file exists on the node.

Optionally, 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 an Oracle Solaris system configuration, see 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/7/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,
```

```
    }

}
```

This example class has the `examplecloud` and `analytics` name spaces. The code in this class ensures that the following occurs on the node:

- Installs certain IPS packages.
- Applies a certain SMF configuration prior to enabling the `analytics` SMF service.

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

Puppet defines the normal and resource-like class declarations.

- The following normal class declaration example includes the `include` keyword in the Puppet code:

  ```
  include example_class
  ```

- The following resource-type class declaration example declares the class in a similar way as a resource declaration:

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

  Use resource-like class declarations to specify class parameters that override the default values of class attributes (or parameters).

For more information about writing and assigning Puppet classes, go to https://puppet.com/docs/puppet/7/lang_classes.html.

Including a Class Declaration in a Puppet Manifest

The following example manifest uses the `examplecloud` class declaration in the `/etc/puppetlabs/code/modules` directory on the Puppet Server (server).

The `examplecloud` class contains several manifests (`/etc/puppetlabs/code/modules/examplecloud/manifests`) that specify various configurations. The following example shows that each manifest includes the `examplecloud` class declaration:

```
# 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 `examplecloud` class declarations in the previous example ensure the following:

- Installs the NTP package
- Installs a certain configuration file that is sourced from a location other than the server
- Enables the NTP service and puts it 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 primary server:

```
# 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/7/bgtm.html.

You add Puppet modules that you create to the `/etc/puppetlabs/code/modules` directory on the primary server, 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 primary 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/7/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.

5

Using Puppet to Manage Oracle Solaris System Configuration

This chapter provides end-to-end examples of using Puppet to manage Oracle Solaris system configuration.

The examples in this chapter run commands on an installed and configured Puppet Server and on Puppet Agent nodes. In addition, you must create a Puppet site manifest on the Puppet Server.

Puppet Configuration Management Workflow

The following steps show the workflow for using Puppet to manage Oracle Solaris system configuration:

1. Use the `puppet describe` *resource-type* command to list the attributes (or parameters) that you can configure for the specified resource type. See Working With Puppet Resources and Resource Types in Oracle Solaris.
2. Declare the appropriate resources in a Puppet manifest on the Puppet Server.

   Use a Puppet site manifest,`site.pp`, to define global system configuration that applies to all nodes.

   Also, use the `node` keyword to define node-specific configuration information in a Puppet site manifest. See Writing Puppet Manifests That Specify Node-Specific Code.
3. (Optional.) Perform a dry run to verify the validity of the configuration defined by the Puppet manifest.

   While this step is not required, this verification is a recommended best practice.
4. Verify that each node has the configuration applied successfully.

For step-by-step instructions, see How to Write a Puppet Site Manifest.

Using Puppet to Configure Packaging

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

Example 5-1 Configuring Packaging With Puppet

First, determine whether the package that you want to install is installed already.

- Run the following command on the local system:

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

- Run the following command from a remote system:

```
# 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, run the `puppet describe package` command to check for the appropriate attribute (or parameter) to declare for the `package` resource type.

The following example `puppet describe package` command shows excerpted output:

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

The example resource type declaration in the Puppet manifest on the Puppet Server specifies the following configuration information:

- Specifies `nmap` as the package to install.
- Ensures that the `nmap` package is available for installation by setting the `ensure` attribute to `present`.

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

Running the following `pkg info nmap` command verifies that the `nmap` package has been installed on the node:

```
# 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
```

Note that the specified package can be installed in one of the following ways:

- **Automatically -** When the Puppet Agent runs
- **Manually -** When you run the `puppet agent -t` command on the node

Note that if you later uninstall the `nmap` package, Puppet enforces the specified configuration by reinstalling the package on the node.

Using Puppet to Configure ZFS File Systems

The following example shows how to use the `zfs` resource type to define a ZFS file system configuration in a Puppet manifest.

Example 5-2 Configuring ZFS File Systems With Puppet

Display the following list of the attributes (or parameters) that you can declare for the `zfs` resource type:

```
# 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, 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 5-3 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 5-4 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 example in this section shows one way that you can declare the `zone` resource type in a Puppet manifest to define an Oracle Solaris Zones configuration.

Example 5-5 Configuring Oracle Solaris Zones With Puppet

The following example `puppet describe` command shows an excerpted list of the `zone` resource type characteristics:

```
# 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` parameter shown in the previous output enables you to create a zone configuration file resource by running the following `zonecfg -z` *zonename* command:

```
# 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@puppet_server:~# 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@puppet_server:~# cp /tmp/zone.cfg /etc/puppetlabs/code/modules/mycompany
```

The zone that you create becomes configurable when the `zone` resource type is applied, so declare the `zone` resource type in the Puppet manifest as follows:

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

The `ensure` parameter value is `installed`, which is one of the valid `installed` or `running` values. In this example, the `systemazone` zone is created on the node.

The following command verifies that the node applied the configuration to itself successfully:

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

This example output shows that the `systemazone` non-global zone is configured, installed, and running.

Index

Symbols

A

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

W

Z