DevOps

Caltech | Center for Technology & Management Education

Post Graduate Program in DevOps

simplilearn

Getting Started with Puppet

Learning Objectives

By the end of this lesson, you will be able to:

- Describe the Puppet architecture
- Set up Puppet master and agent
- Write Puppet manifests

Introduction to Puppet

What Is Puppet?

Puppet is one of the most widely used open source DevOps tools to centralize and automate the configuration management process. It is developed by Puppet Labs.

Why Puppet for Configuration Management?

Has a large community along with large installed bases that include Google, Red Hat, Siemens, and more

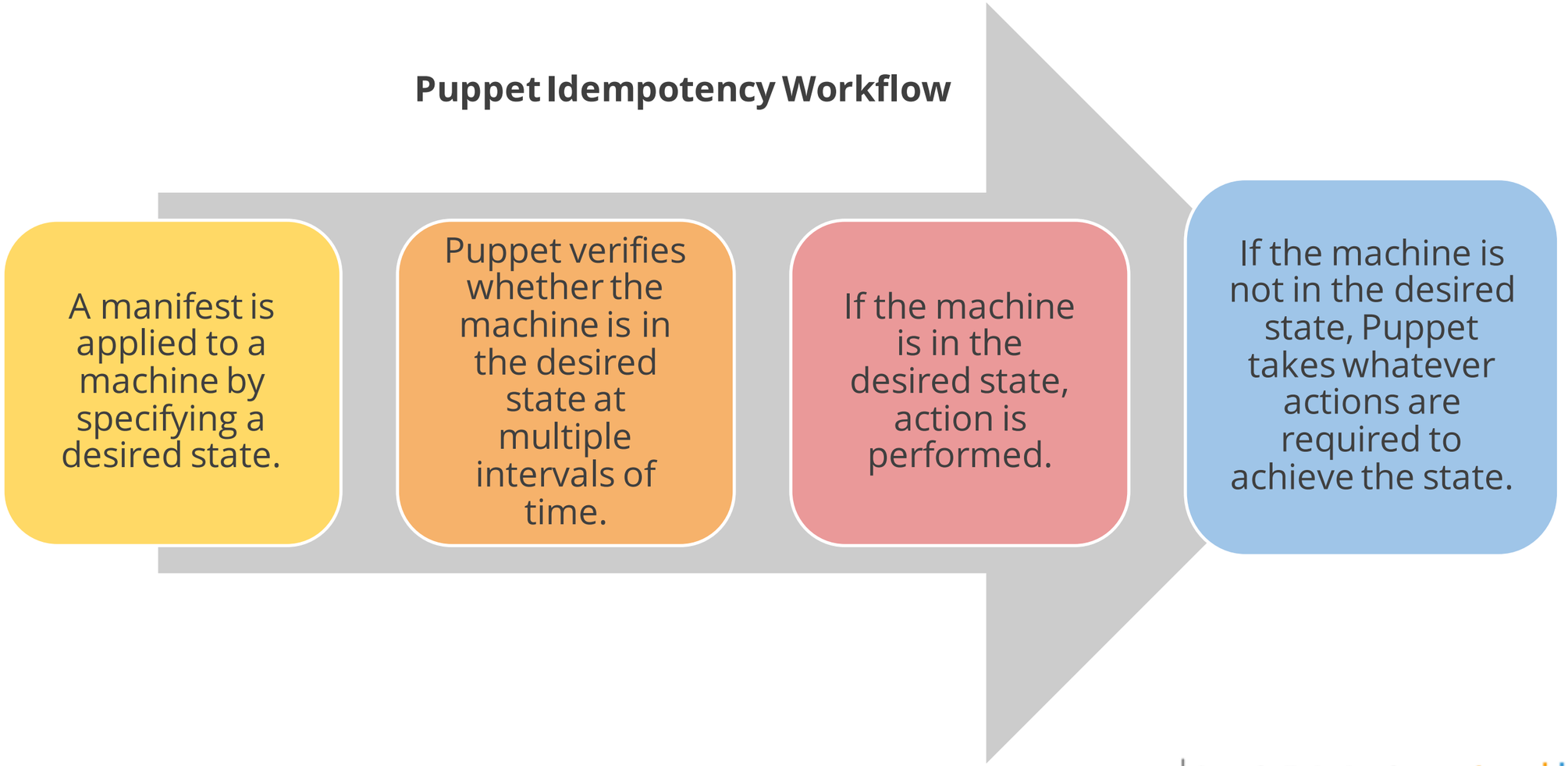
Uses its own DSL, which is specifically designed for configuration management

Provides in depth documentation and comprehensive references for the resource types and the language

Offers flexible platform support ranging from Linux, Red Hat, Windows, and more

Puppet Idempotency

Puppet allows users to apply the same manifest to a machine multiple times to achieve a desired resource state on a system with the same results every time.



Use Cases of Puppet

Puppet can be used in various ways to simplify and improve system administration workflow.

Puppet can be used to:

1. Define infrastructure as code
2. Manage multiple servers at once
3. Enforce system configuration
4. Automate cloud processes
5. Track asset and license inventory

Advantages and Disadvantages of Puppet

Advantages

- Language is clean and easy to learn
- Open source with cross-platform support
- Lets users schedule jobs on a periodic basis
- Allows to override an instruction with another instruction
- Has a large and active community

Disadvantages

- For progressive tasks, Ruby may be required that is complex to understand
- Non-programmers may find it difficult to understand
- Model-driven approach may be hard to control

Core Components of Puppet

Puppet Components and Terminologies

Puppet Master	Puppet master runs on a server that manages all other nodes.
Puppet Agent	Puppet agent runs on the client servers that are to be managed by the Puppet master.
Config Repository	Config repository is where all nodes and server related configurations are stored and accessed when needed.

Puppet Components and Terminologies

Facts	Facts are global variables that contain all the machine-level information.
Catalogue	Catalogue is the format in which all the configurations written in the puppet language are compiled and saved.
Resources	In Puppet package, a resource may represent packages, files, users, and commands.

Puppet Components and Terminologies

Puppet Class	Puppet classes are named blocks of Puppet code that represent a collection of resources.
Nodes	All the clients or the servers that have to be managed are called nodes.
Manifests	All the Puppet programs with the extension of .pp that are written to manage target host machines are called manifests.

Puppet Architecture

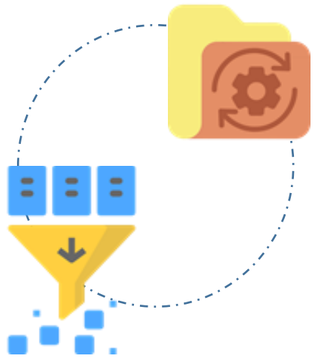
Puppet Architecture

Users can configure machines with agent-master Puppet architecture or with standalone Puppet architecture.

Puppet configures systems in two stages:

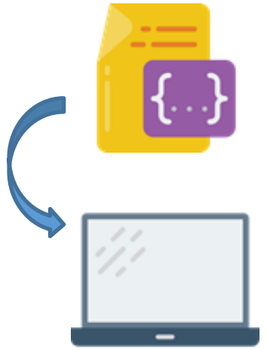
Stage 1

Compile a catalogue

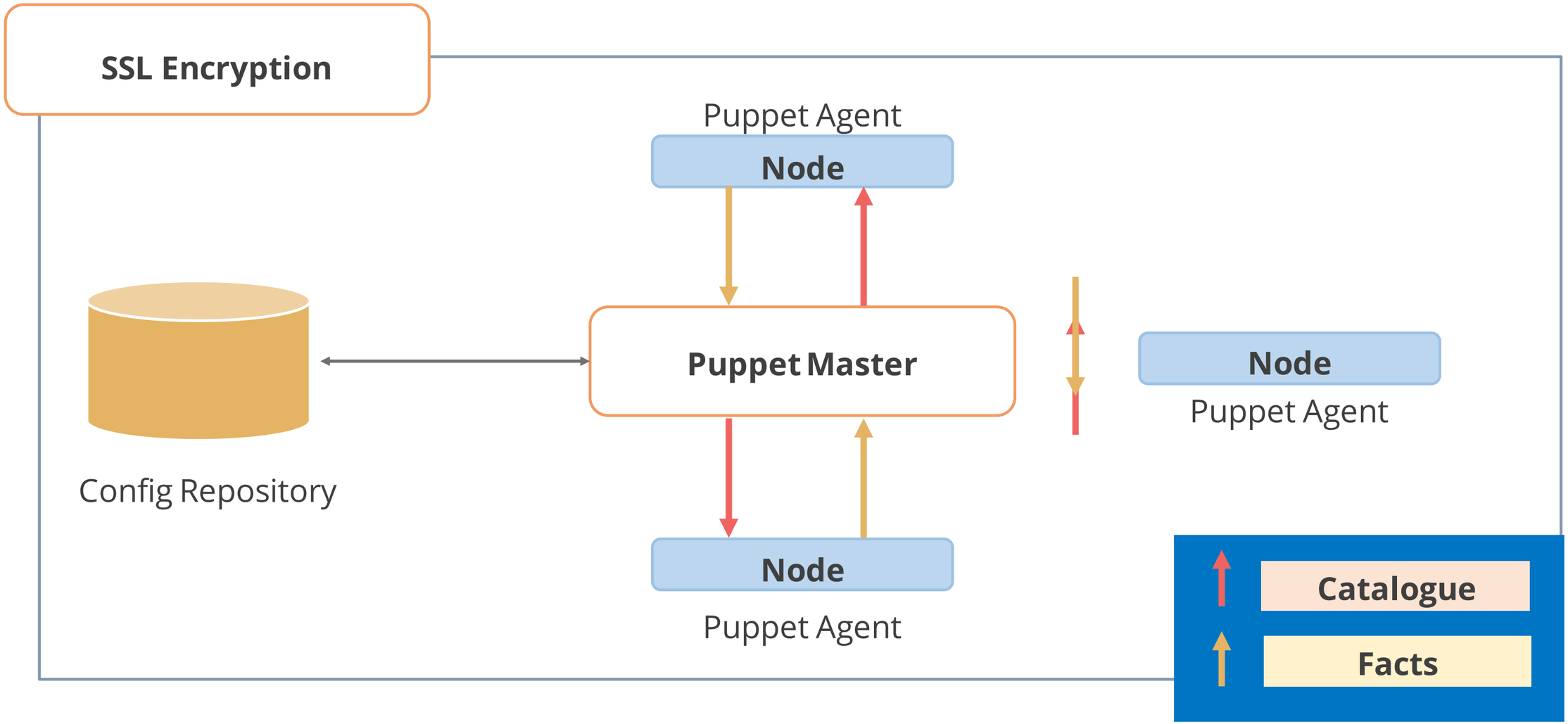


Stage 2

Apply the catalogue

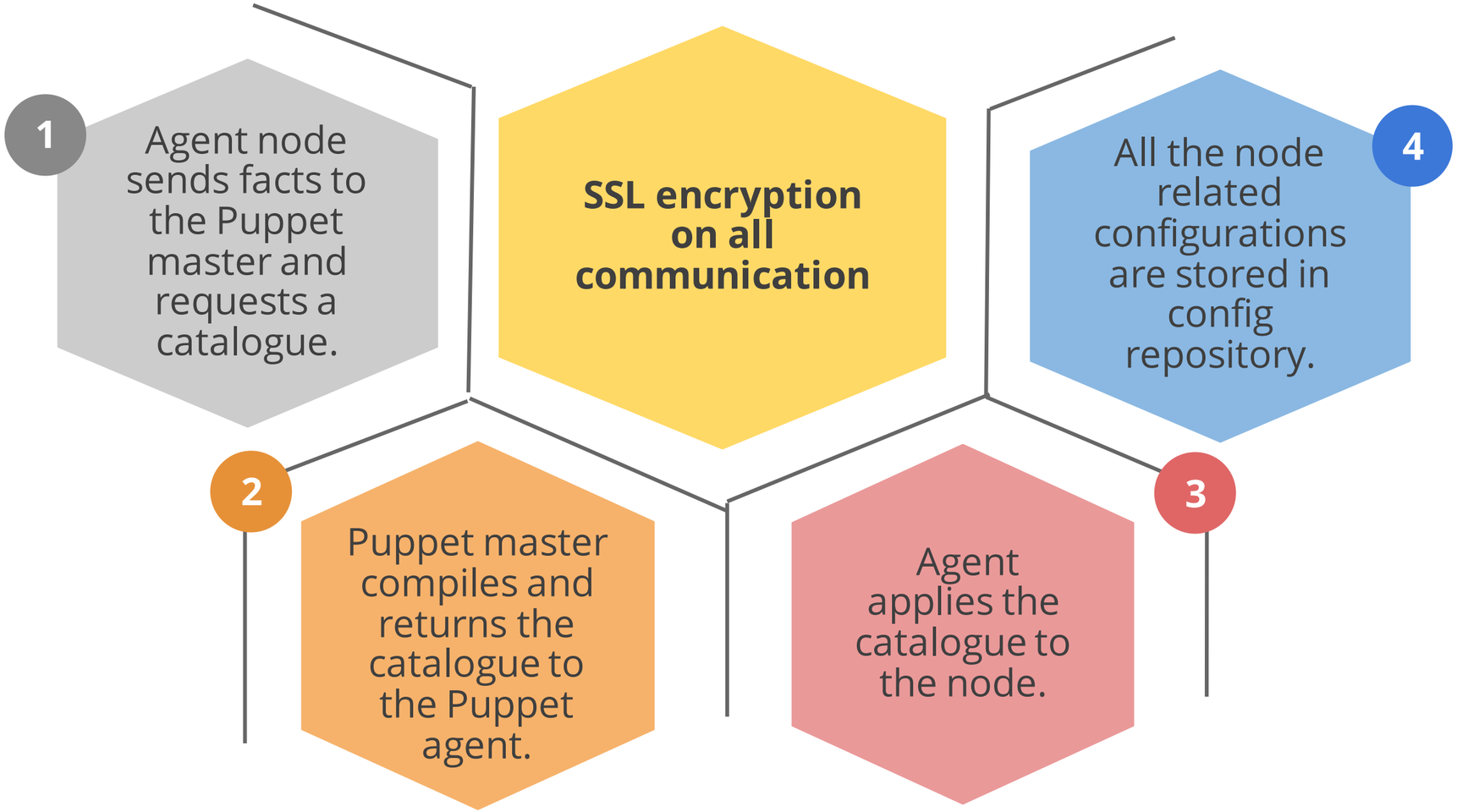


Agent-Master Architecture

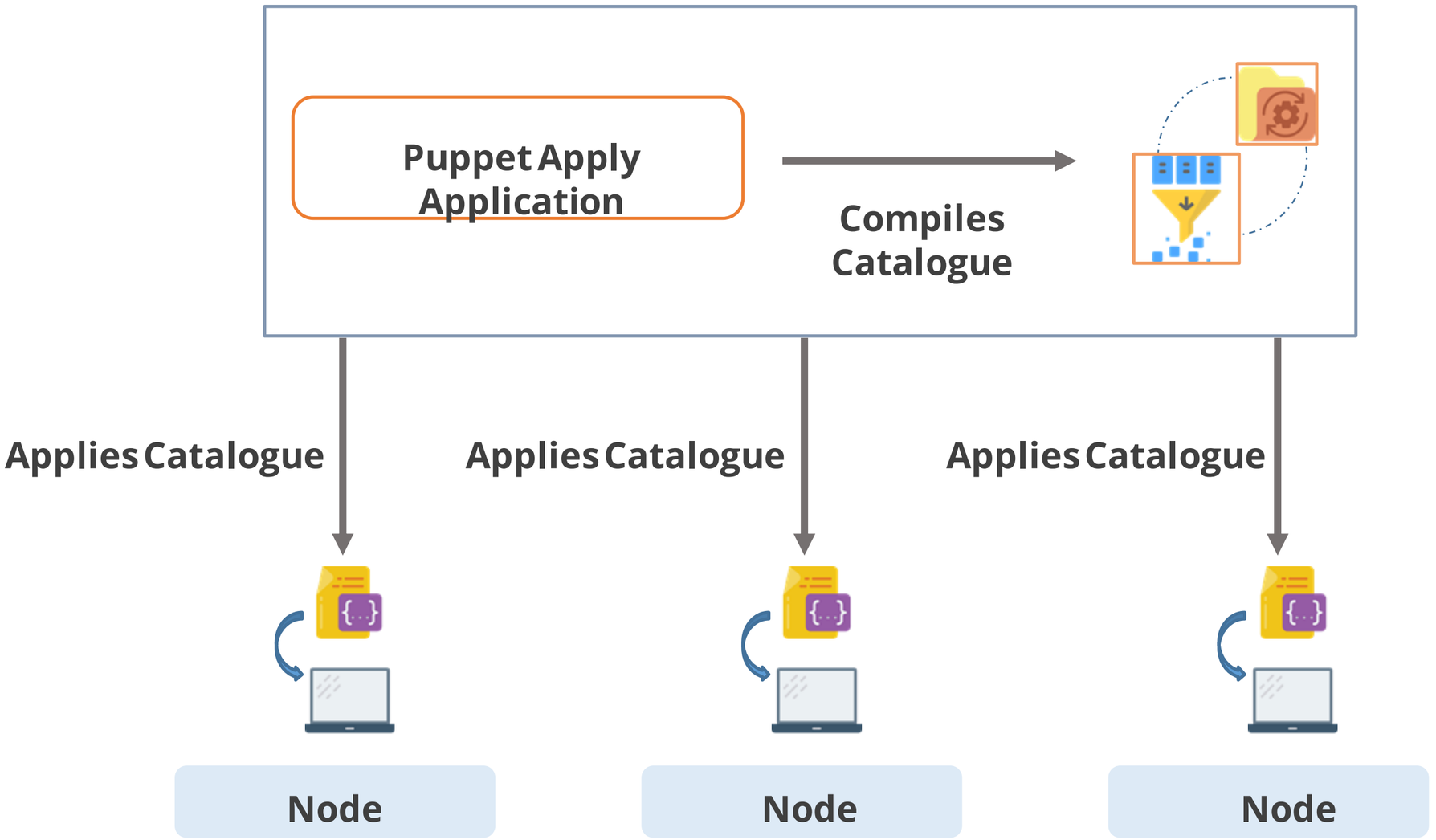


Agent-Master Architecture

Communication workflow of agent-master architecture is shown below:



Puppet Stand-Alone Architecture



Source: https://success.docker.com/article/networking#dockerbridgenetworkdriver

Puppet Stand-Alone Architecture

Communication workflow of the stand-alone architecture is shown below:

1. Managed nodes run the “Puppet apply” application as a scheduled task.
2. Puppet apply application compiles the catalogues of the managed nodes.
3. Puppet apply application applies the compiled catalogue to the managed nodes.
4. Puppet apply application stores the report on a disc indicating that the configuration is complete.

Source: https://success.docker.com/article/networking#dockerhostnetworkdriver

Agent-Master vs. Stand-Alone Architecture

Differential Parameter	Agent-Master Architecture	Stand-Alone Architecture
Configuration information	All agent nodes are only aware of their own configuration.	All nodes have complete configuration knowledge of their cluster nodes.
Ease of reporting	Agents send the reports directly to Puppet master, which can be configured with any number of report handlers to pass the reports to other services.	Managed nodes handle their own information. In order to send the information to other services, node will have to authorize directly. This raises security concerns.

Agent-Master vs. Stand-Alone Architecture

Differential Parameter	Agent-Master Architecture	Stand-Alone Architecture
Ease of updating configuration	Only Puppet master has to be updated and the configuration will be pushed automatically to the agent nodes.	The new configuration code and data have to be synced with each and every node separately.
Ease of reporting	Puppet agents don't compile their own catalogues. So, they don't require heavy resources from the machine they are running on.	Managed nodes need to store their own catalogue and configuration data. Hence, they require more resources from the host machine.

Assisted Practice

Installing and Setting Up Puppet on Ubuntu

Problem Statement:

Install and set up Puppet on Ubuntu.

Assisted Practice: Guidelines

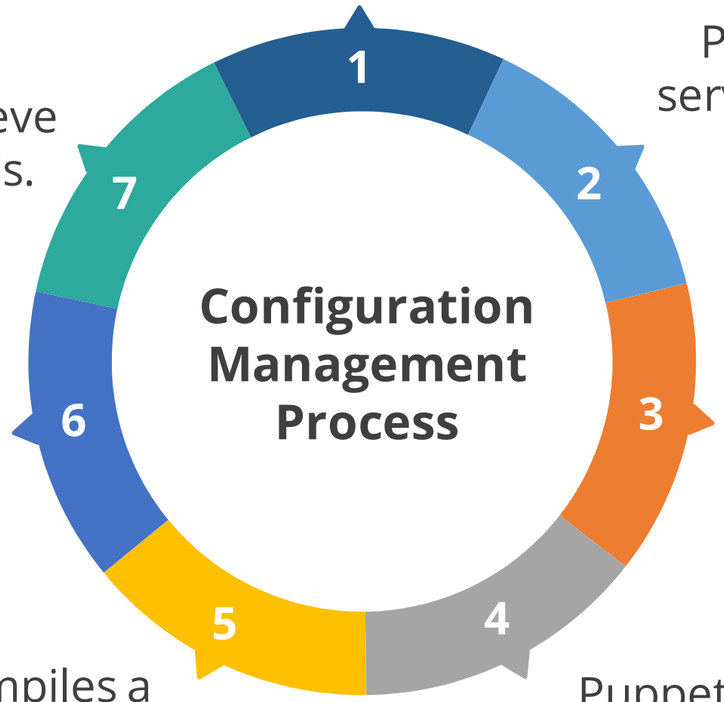
Steps to perform:

1. Download and install Puppet labs repository on the servers where you want to use Puppet
2. Update the system to get the list of new packages
3. To verify the installation, check the version of the Puppet installed
4. Install Puppet server on the Puppet master server
5. Install Puppet agent on the client machine or server

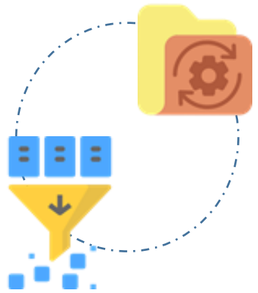
How Does Puppet Work?

How Does Puppet Work?

1. Puppet master is installed on the server that is used to manage other machines or servers.
2. Puppet agent is installed on the servers or machines that have to be managed (Nodes).
3. The user defines configuration parameters for a specific environment using Puppet's domain specific language.
4. Puppet agent will collect configuration information about the nodes using factor utility and send it to the Puppet master.
5. Puppet master compiles a catalog based on how the nodes should be configured and sends it to Puppet agent.
6. Puppet agents use the catalog to apply the necessary configuration updates to the nodes.
7. Puppet agents poll the Puppet master at regular intervals to retrieve node-specific configuration details.



Functions of Puppet Master

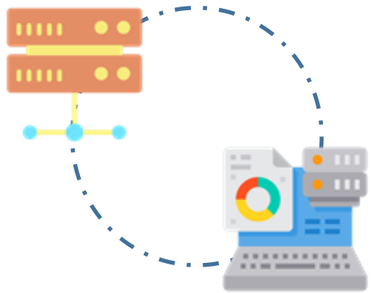


1

Compiling catalog for the Puppet agents

Sending catalog to the Puppet agents

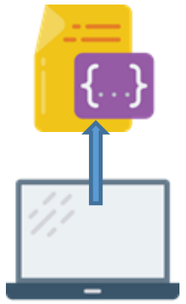
2



3

Storing the configuration information in a central repository

Functions of Puppet Agent

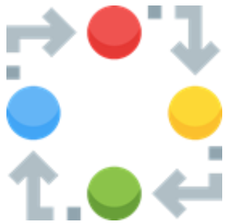


1

Pulling catalogs from the Puppet master

Applying configuration updates to the nodes

2



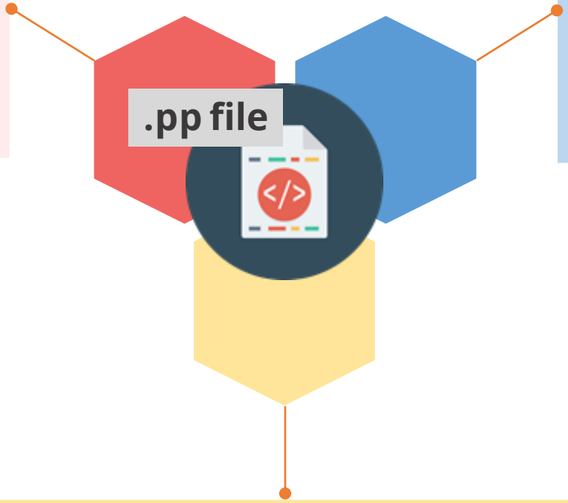
3

Polling the Puppet master after certain intervals for configuration updates

Introduction to Puppet Language

Introduction to Puppet Language

Puppet uses its own Domain Specific Language (DSL) language to write manifests.

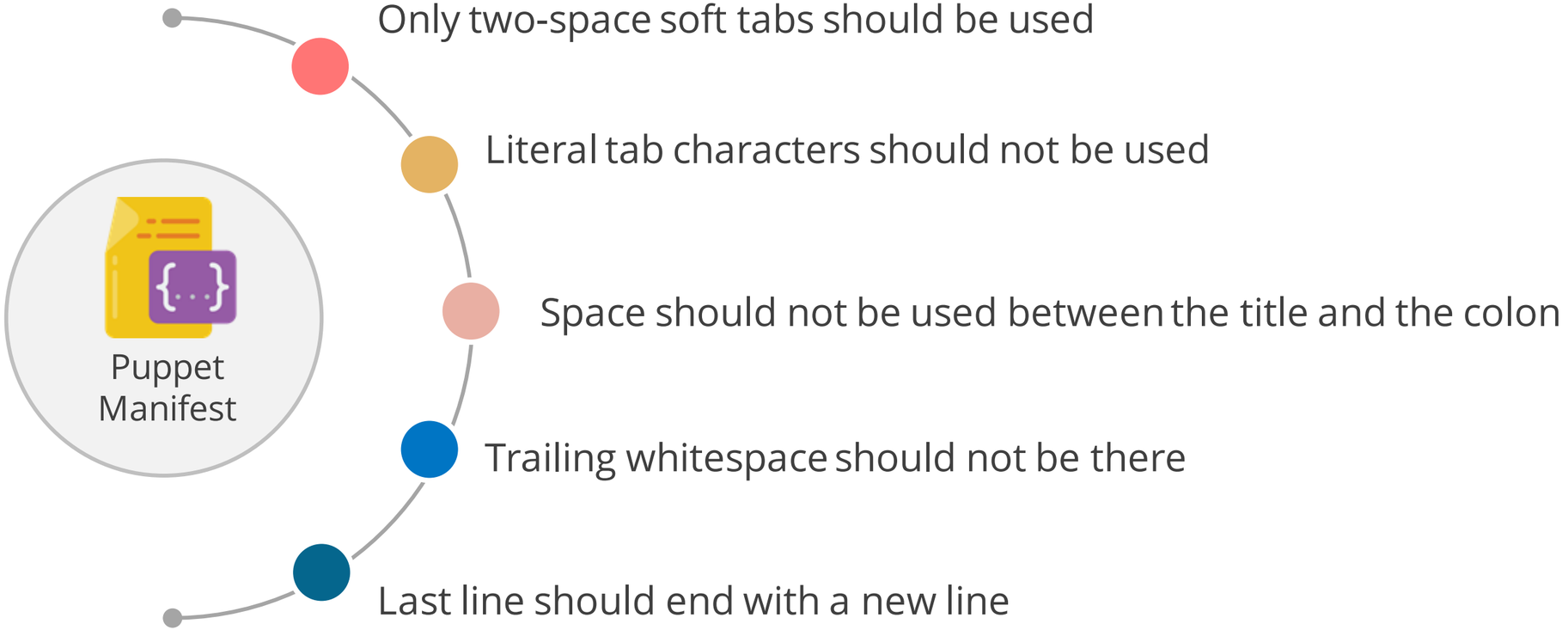


Puppet programming language follows declarative programing paradigm.

Puppet programming language was specifically designed for system admin tasks.

Writing Puppet Manifests

Rules to be followed while writing manifests:



Comments

Comments in Puppet language are represented as hashed sentences. Comments should explain the purpose of the puppet statement.

Good commenting practice

```
# Configures NTP
file { '/etc/ntp.conf': ... }
```

Bad commenting practice

```
/* Creates file /etc/ntp.conf */
file { '/etc/ntp.conf': ... }
```

Values and Data Types

Value	An individual piece of data that may be used for any purpose in Puppet language is called as a value.
Data Type	The format of information present in a value is called a data type.

Puppet Data Types

Strings

Text fragments of any length

Numbers

Integers and floating-point numbers

Boolean

One bit values that represent true and false

Arrays

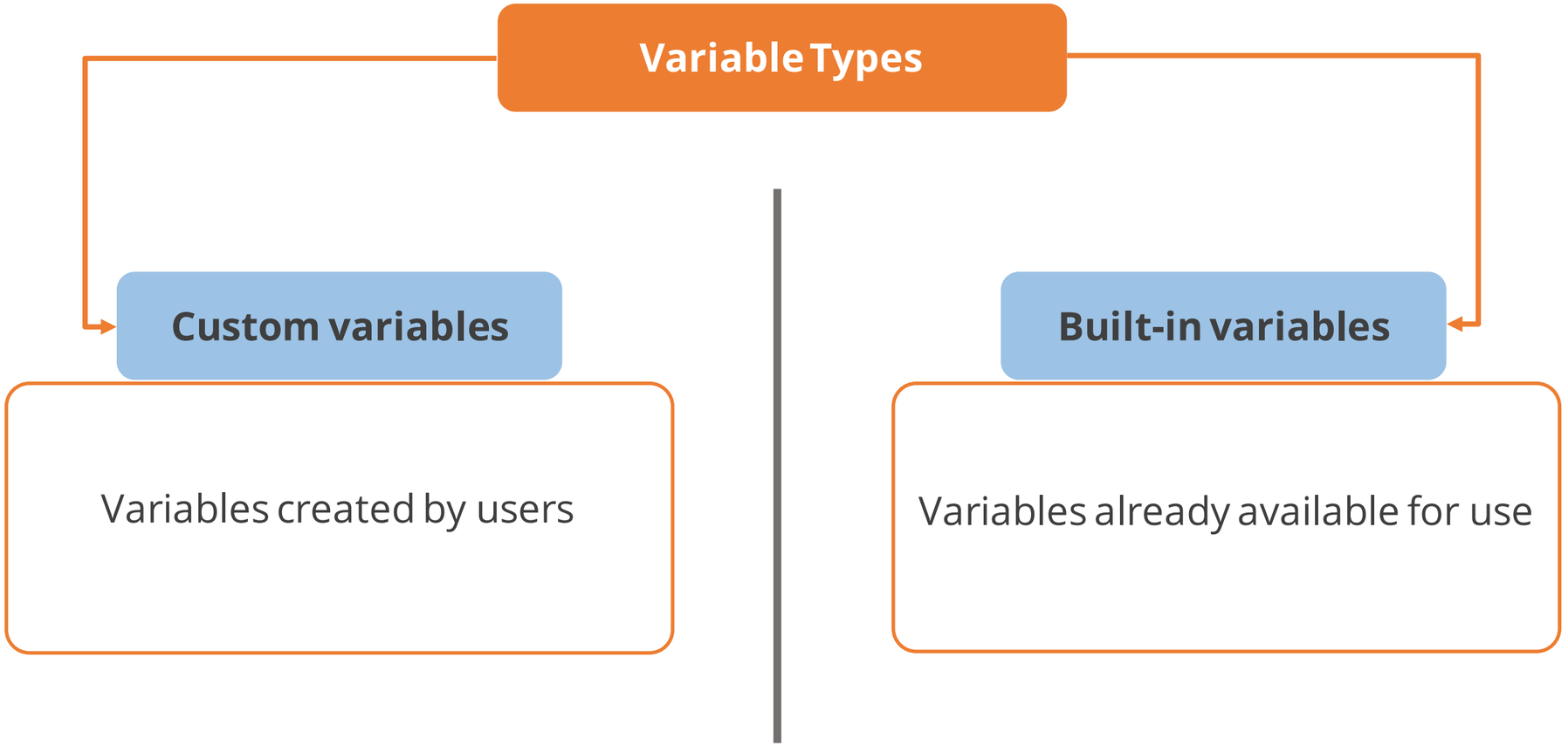
Ordered list of values with fixed length

Hashes

Data in key and value format

Variables and Built-in Variables

Variable is a container with a descriptive name that stores a value. This is done to access that value in the program using the variable name.



New values cannot be assigned to a variable that already contains a value.

Types of Built-In Variables

Variable Name	Description
$environment	Puppet agent node's environment
$servername	Puppet master’s Fully-Qualified Domain Name (FQDN)
$serverip	Puppet master’s IP address
$serverversion	The current version of Puppet on the master

Assigning Value to a Custom Variable

Custom variables

- Variable name is prefixed with a dollar sign ($).
- Variable names are case sensitive and generally start with lowercase or an underscore.
- Values are assigned to the variables using the assignment operator (=).

Syntax:

```
$content = "some content\n"
```

Reserved Words and Acceptable Names

Puppet language has reserved some words that can't be used for naming variables, resources, classes, and modules.

Types of reserved words:

Reserved Word	Type	Reserved Word	Type	Reserved Word	Type
and	Expression keyword	**case**	Language keyword	**define**	Language keyword
application	Language keyword	**false**	Boolean value	**elsif**	Language keyword
attr	Reserved for future use	**consumes**	Language keyword	**environment**	Reserved for symbolic namespace use

Conditional Statements

Conditional statements enable users to change or modify the execution flow of the Puppet code. This is done by executing or skipping certain code blocks regardless of their sequence.

Types of conditional statements:

1. If statements
2. Unless statements
3. Case statements
4. Selector expression

If Statements

An if statement takes a boolean condition and executes an arbitrary block of Puppet code only if the condition is true. It can also include elseif and else clauses to execute Puppet code block if the condition is not true.

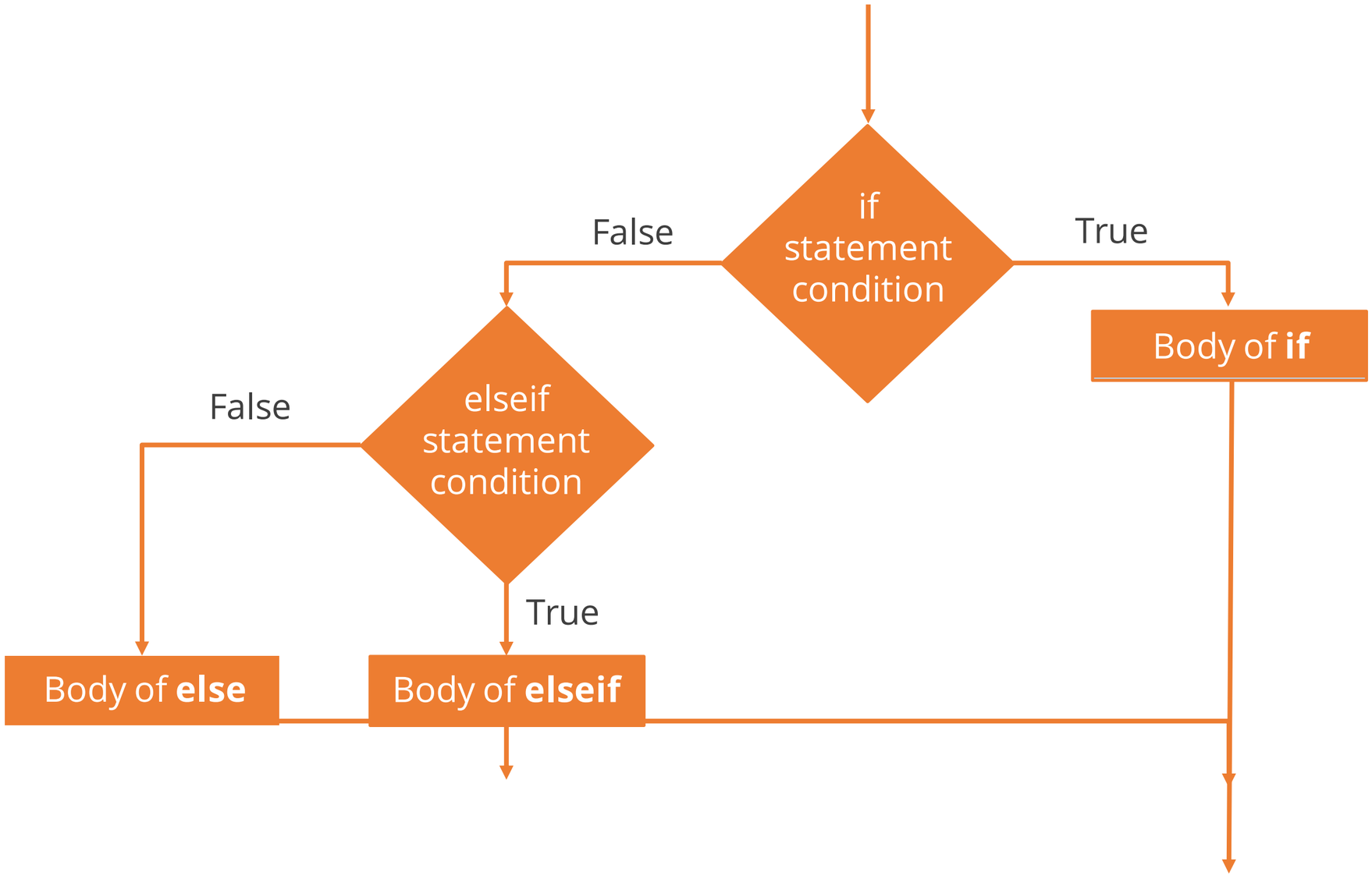
An if statement contains:

1. A condition resolving to a Boolean value
2. A pair of curly braces enclosing the Puppet code block to be executed when the condition is true
3. Optional: any number of elseif clauses
4. Optional: else keyword with Puppet code to be executed if the condition is false

Syntax:

```
$content = "some content\n"
```

Workflow of an If Statement



Unless Statements

Unless statements work like reversed if statements. They take a boolean condition and execute a code block if the condition is false.

An unless statement contains:

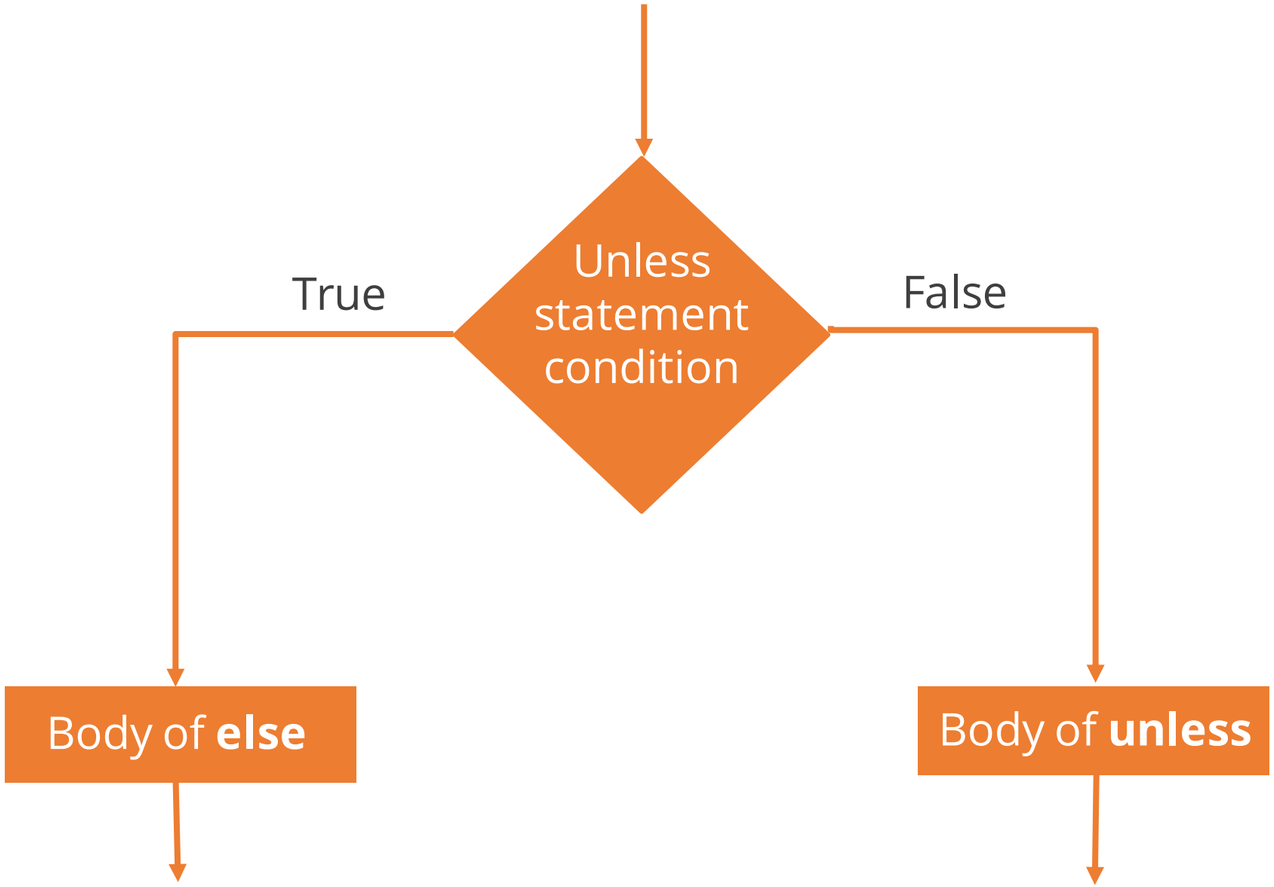
1. A condition that resolves to a boolean value
2. A pair of curly braces enclosing a Puppet code block
3. Optional: an else keyword with a puppet code block within the curly braces

Syntax:

```
unless
$facts['memory']['system']['totalbytes'] >
1073741824
{
  $maxclient = 500
}
```

Note: Unless statements do not include elseif clauses.

Workflow of an Unless Statement



Case Statements

Case statements choose and execute one of several Puppet code blocks.

A case statement contains:

1. A control expression resolving to a value
2. A pair of curly braces
3. Any number of possible matches of cases and corresponding puppet code within the curly braces

Syntax:

```
case $facts['os']['name'] {
  'Solaris':           { include
role::solaris } # Apply the solaris class
  'RedHat', 'CentOS':  { include
role::redhat  } # Apply the redhat class
  /^(Debian|Ubuntu)$/: { include
role::debian  } # Apply the debian class
  default:             { include
role::generic } # Apply the generic class
}
```

Workflow of a Case Statement

1

Puppet takes a control expression, a list of cases, and a Puppet code block.

2

Puppet compares the control expression to each of the cases in the order they are listed.

3

It executes the block of code whose corresponding case matches with the control expression.

4

Rest of the code blocks are ignored. If none of the cases match, Puppet ignores all case statements.

Selector Expressions

Selector expression are like case statements. The only difference is that instead of executing code they return a value.

A selector expression contains:

1. A control expression resolving to a value
2. A question mark keyword
3. A pair of curly braces enclosing a case, hash rocket (=>) keyword, a value, and a trailing comma

Syntax:

```
$rootgroup = $facts['os']['family'] ? {
  'Solaris'                    => 'wheel',
  /(Darwin|FreeBSD)/ => 'wheel',
  default                      => 'root',
}
```

Workflow of a Selector Expression

1
Puppet takes a control expression, a list of cases, and a Puppet code block.

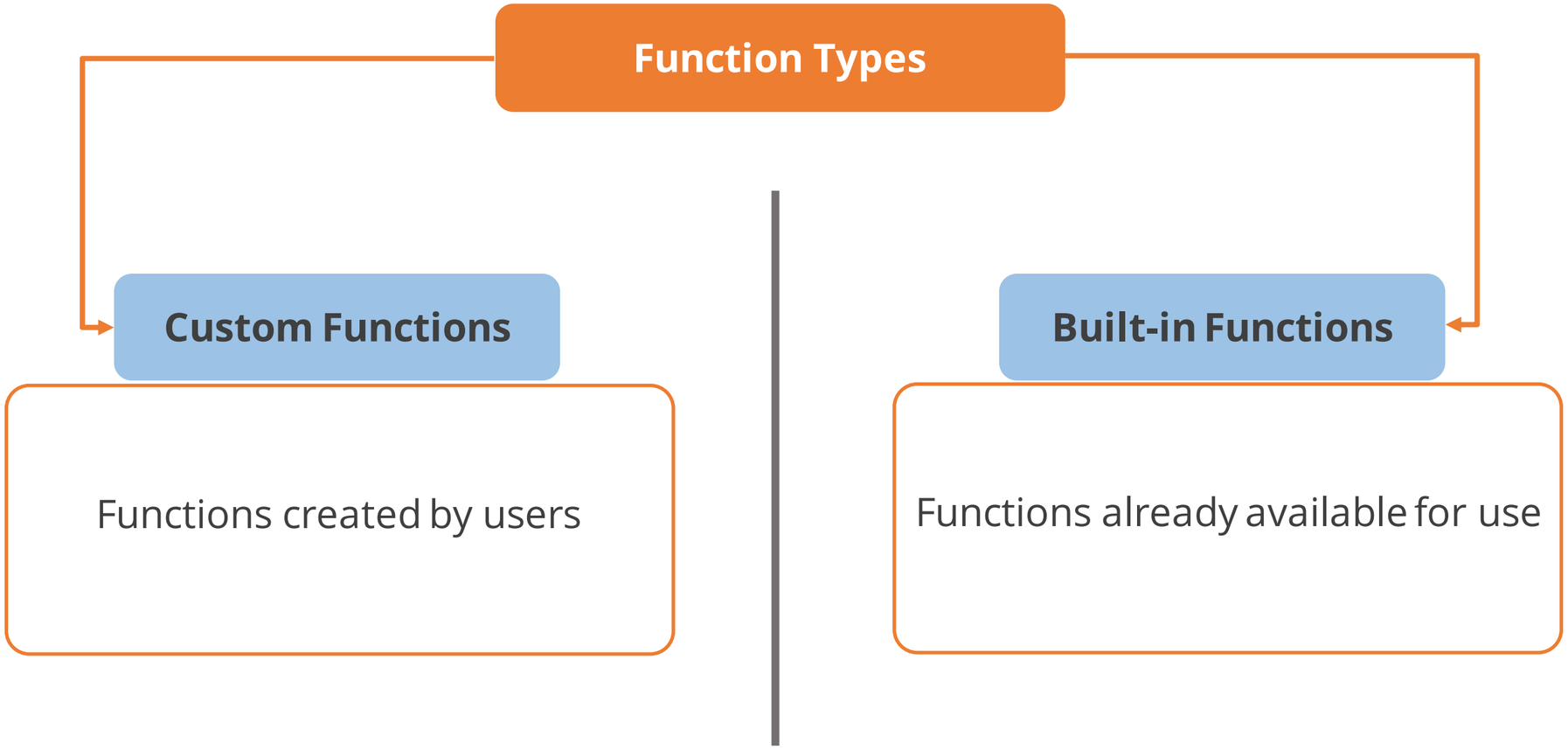
2
Puppet compares the control expression to each of the cases in the order they are listed.

3
When Puppet finds a matching case, it treats the value as the value of expression.

4
Rest of the code blocks are ignored. If none of the cases match, Puppet fails compilation with an error.

Custom and Built-In Functions

A function is a set of statements that take inputs and perform some specified computations to produce an output.



Writing a Custom Function

Custom Functions

- The keyword **function** is used to define a block of code as a function.
- Double colon separated namespace (Puppet::Parser) should be used before the function keyword.
- Name of the function is followed by a pair of parentheses containing comma separated list of parameters.
- A block of Puppet code is included in the function and is enclosed within a pair of curly braces.

Syntax:

```
function <MODULE NAME>::<NAME>(<PARAMETER
LIST>) >> <RETURN TYPE> {
  ... body of function ...
  final expression, which will be the
returned value of the function
}
```

Types of Built-In Functions

Function Name	Description	Function Name	Description	Function Name	Description
alert()	Logs messages on the server at level alert	**convert_to()**	Converts the data type of a given value into a different data type	**include()**	Declares one or more classes
break()	Breaks an innermost iteration	**empty()**	Verifies if the given argument is an empty collection of values	**lookup()**	Retrieves a value of a given key
call()	Calls an arbitrary Puppet function by name	**find_file()**	Finds an existing file	**match()**	Matches an expression against a string

Function Calls

Any custom function or built-in function can be called from anywhere in the Puppet program. A call to any function is an expression that may contain one or more arguments and resolves to a value.

Syntax for calling a function:

```
function_name(argument, argument, ...)
|$parameter, $parameter, ...| { code block }
```

Functions can also be called in chained style using the following syntax:

```
argument.function_name(argument, ...)
|$parameter, $parameter, ...| { code block }
```

Assisted Practice

Writing Puppet Programs Using Conditional Statements

Problem Statement:

Write Puppet code using conditionals to:

- Install SSL if the value of the machine variable is 'production', install nginx if the value of machine variable is 'testing' and for any other value, install OpenSSL.
- To recognize the operating system for a node machine with an apache web server running on it.

Assisted Practice: Guidelines

Steps to perform:

1. Create a text file named conditional1.pp and save it
2. Write if else statement to install SSL, ngnix and openSSL as per the conditions in conditional1.pp file and save it
3. Create another text file named conditional2.pp and save it
4. Write case statements to recognize the operating system for the node machine in conditional2.pp file and save it

Assisted Practice

Creating and Calling Functions

Problem Statement:

- Write a custom Puppet function to add a line in an existing file
- Call the function created in the previous step
- Find out the path of an existing file by calling an built-in function

Assisted Practice: Guidelines

Steps to perform:

1. Create a file and save it with .rb extension
2. Declare and define the function inside the file and save it
3. Add the function call for the custom function in the manifest
4. Add the function call for the built-in function in the manifest

Assisted Practice

Writing a Puppet Manifest

Problem Statement:

Write a basic Puppet manifest to perform the following:

- Modify the file/etc/motd in all agent nodes to add a string
- Stop Postfix service in all agent nodes

Assisted Practice: Guidelines

Steps to perform:

1. Create a file and save it with .pp extension
2. Check if the file is present and provide the required mode permissions
3. If present, add the content
4. Check if the Postfix service is running
5. If running, stop the service

Key Takeaways

- Puppet enables users to define configurations for a system and then maintain a specified state after the initial setup.
- Users can choose between agent-master Puppet architecture and standalone Puppet architecture for configuration management.
- The programs files written to manage the target host machines are called Puppet manifests.
- Puppet has its own declarative, domain specific language that is used to write Puppet manifests.