Puppet Cheatsheet

A comprehensive cheat sheet covering essential Puppet concepts, syntax, and commands for effective infrastructure management and automation in DevOps and Cloud environments.

Puppet Fundamentals

Core Concepts

Puppet Agent: The client application that runs on managed nodes and applies configurations.
Puppet Master: The central server that compiles catalogs and serves them to agents.
Catalog: A document describing the desired state of a node.
Manifests: Files containing Puppet code that define resources and configurations.
Modules: Reusable collections of manifests, templates, and other files.
Resources: Represent individual components of a system (e.g., files, packages, services).

Facts: Information about a node, such as its hostname, IP address, operating system, etc. Facts are automatically discovered by Facter.

Classes: Reusable blocks of Puppet code that define a specific configuration. Classes are the primary means of organizing Puppet code.

Puppet Workflow

1. **Agent Requests Catalog:** Puppet Agent sends facts to the Puppet Master.
2. **Master Compiles Catalog:** The Puppet Master uses facts and manifests to compile a catalog.
3. **Catalog Sent to Agent:** The Puppet Master sends the compiled catalog to the Agent.
4. **Agent Applies Catalog:** The Puppet Agent applies the configuration defined in the catalog.
5. **Agent Reports Status:** The Agent sends a report back to the Puppet Master about the configuration run.

Basic Syntax

Resource Declaration	`file { '/tmp/example.txt': ensure => present, content => 'Hello, world!', }`
Variable Assignment	`$hostname = $facts['hostname']`
Conditional Statements	`if $osfamily == 'RedHat' { package { 'httpd': ensure => installed, } }`

Puppet Resources

Common Resource Types

file: Manages files and directories.
package: Manages software packages.
service: Manages system services.
user: Manages user accounts.
group: Manages group accounts.
cron: Manages cron jobs.
exec: Executes arbitrary commands.

File Resource Attributes

`ensure`	Specifies whether the file should be present, absent, a directory, a link, etc.
`path`	The path to the file.
`content`	The content of the file.
`source`	The source file to copy content from (used for templates).
`owner`	The owner of the file.
`group`	The group of the file.
`mode`	The permissions of the file (e.g., '0644').

Package Resource Attributes

`ensure`	Specifies whether the package should be installed, absent, or a specific version.
`name`	The name of the package.
`provider`	The package provider (e.g., yum, apt, gem).

Puppet Modules & Classes

Module Structure

A Puppet module typically has the following directory structure:

```
module_name/
├── manifests/
│   └── init.pp
├── files/
├── templates/
└── metadata.json
```

`manifests/init.pp`: Contains the main class definition.
`files/`: Contains static files to be copied to managed nodes.
`templates/`: Contains templates to generate dynamic configuration files.
`metadata.json`: Contains metadata about the module (e.g., name, version, dependencies).

Defining Classes

Basic Class Definition

```
class mymodule {
  # Resource declarations go here
  file { '/tmp/example.txt':
    ensure  => present,
    content => 'This file is managed by Puppet.',
  }
}
```

Class Parameters

```
class mymodule (
  $param1 = 'default_value',
  $param2,
) {
  # Use parameters in resource declarations
  file { '/tmp/example.txt':
    ensure  => present,
    content => "Parameter 1 is ${param1}",
  }
}
```

Including Classes

`include`

```
include mymodule
```

Simplest way to include a class. Can only be used once per class.

`require`

```
class {'mymodule':
  require => Class['othermodule'],
}
```

Ensures that the class is applied before the current class.

`contain`

```
contain mymodule
```

Similar to include, but allows classes to be declared multiple times.

Advanced Puppet Features

Templates

Puppet uses Embedded Ruby (ERB) templates to generate dynamic configuration files. Templates are located in the `templates/` directory of a module.

Example (mytemplate.erb):

```
ServerName <%= @hostname %>
DocumentRoot <%= @docroot %>
```

To use a template in a manifest:

```
file { '/etc/httpd/conf/httpd.conf':
  ensure  => present,
  source  => 'puppet:///modules/mymodule/mytemplate.erb',
}
```

Facts and Variables

Accessing Facts

```
$osfamily = $facts['os']['family']
if $osfamily == 'RedHat' {
  # Do something specific to RedHat systems
}
```

Custom Facts

Custom facts can be created in Ruby or as executable scripts. They are stored in the `lib/facter` directory of a module.

Variables

```
$myvariable = 'somevalue'
file { '/tmp/example.txt':
  ensure  => present,
  content => "The variable is ${myvariable}",
}
```

Hiera

Hiera is a key-value lookup tool for Puppet. It allows you to externalize data from your Puppet code.

Example (hiera.yaml):

```
---
:backends:
  - yaml
:yaml:
  :datadir:
/etc/puppetlabs/code/environments/%
{environment}/data
:hierarchy:
  - "nodes/%{::trusted.certname}"
  - common
```

Example (common.yaml):

```
ntp::servers:
  - 0.pool.ntp.org
  - 1.pool.ntp.org
```

Using Hiera data in Puppet:

```
class ntp {
  $servers = hiera('ntp::servers', [])
  package { 'ntp':
    ensure => installed,
  }

  file { '/etc/ntp.conf':
    ensure  => present,
    content =>
template('ntp/ntp.conf.erb'),
    require => Package['ntp'],
  }
}
```