Kea DHCP - Template Classes

Using dynamic classes in client classification

Carsten Strotmann and the ISC Team

Welcome

Welcome to our Webinar on using template classes to create dynamic classes in Kea DHCP

In this Webinar

- Recap: Client Classification
- Static client classing
- Automatic vendor classing
- Dynamic classing using templates classes
- Examples of template classes

All content © 2023 Internet Systems Consortium, Inc.

Recap of Client classification

DHCP client classes

- Kea DHCP can assign one or more client classes to client requests
- Kea DHCP can be configured to inspect the properties of incoming DHCP requests, and to assign the DHCP packet to one or more client classes

DHCP client classes

- Depending on the client classes, Kea DHCP can control the response data send back to the client …
    - DHCP-Options
    - IP-Addresses
    - Lease parameters (lease time)
    - BOOTP-Parameter inside DHCP responses
- Kea can select from multiple subnet / pools with the help of client classes
- With the *Limits* Hook, Kea DHCP can limit the number of leases assigned to a class
- Kea DHCP can drop packets based on client classification

Client classing can be done in different ways in Kea DHCP

- Static client classing
- Automatic vendor classing
- (**New**) Dynamic client classes based on template classes
- Hooks

All content © 2023 Internet Systems Consortium, Inc.

Static client classing

Client classing based on expressions

- DHCP requests can be assigned one or more client classes
  - Expressions can be used to extract information from the DHCP request message
  - The expression in the `test` statement must evaluate into an *TRUE* or *FALSE* value
  - Logical and conditional expressions can be used to assign classes to the DHCP request
- List of available expressions

  https://kea.readthedocs.io/en/latest/arm/classify.html#using-expressions-in-classification

List of classification values

Name	Example expression	Example value
String literal	'example'	'example'
Hexadecimal string literal	0x5a7d	'Z}'
IP address literal	10.0.0.1	0x0a000001
Integer literal	123	'123'
Binary content of the option	option[123].hex	'(content of the option)'
Option existence	option[123].exists	'true'
Binary content of the sub-option	option[12].option[34].hex	'(content of the sub-option)'
Sub-Option existence	option[12].option[34].exists	'true'
Client class membership	member('foobar')	'true'
Known client	known	member('KNOWN')
Unknown client	unknown	not member('KNOWN')
DHCPv4 relay agent sub-option	relay4[123].hex	'(content of the RAI sub-option)'
DHCPv6 Relay Options	relay6[nest].option[code].hex	(value of the option)
DHCPv6 Relay Peer Address	relay6[nest].peeraddr	2001:DB8::1
DHCPv6 Relay Link Address	relay6[nest].linkaddr	2001:DB8::1
Interface name of packet	pkt.iface	eth0
Source address of packet	pkt.src	10.1.2.3

Client classification example (1/2)

- In this example we classify a client based on its parameter request list (PRL, the DHCP options the client requests from the DHCP server).
    - The PRL is often unique for different operating systems, devices or DHCP client software versions used

```
"Dhcp4": {
   "client-classes": [
        {   "name": "foo",
            "test": "hexstring(substring(option[55].hex,0,4),":") == '01:02:06:0c'",
            "option-data": [{
                    "name": "domain-name", "data": "foo.example.com" }]
        },
        {   "name": "bar",
            "test": "not(hexstring(substring(option[55].hex,0,4),":") == '01:02:06:0c')",
            "option-data": [{
                    "name": "domain-name", "data": "bar.example.com" }]
        }
    ],
[...]
```

Client classification example (2/2)

- The client class is used further down in the configuration file to select a subnet inside a shared network
    - `foo` clients get IP addresses from the 1st subnet
    - `bar` clients get IP addresses from the 2nd subnet

```
        "shared-networks": [
            {
                "name": "kea-dhcp01",
                "relay": { "ip-address": "192.0.2.1" },
                "subnet4": [
                    {
                        "subnet": "192.0.2.0/24",
                        "client-class": "foo", # <-- all 'foo' Clients will
                                               # get IP addresses from this subnet
                        "option-data": [{
                                "name": "routers", "data": "192.0.2.1" }],
                        "pools": [{
                                "pool": "192.0.2.60 - 192.0.2.250" }]
                    },
                    {
                        "subnet": "10.0.0.0/24",
                        "client-class": "bar", # <-- "bar" clients will
                                               # get IP addresses from this subnet
                        "option-data": [
[...]
```

All content © 2023 Internet Systems Consortium, Inc.

Automatic vendor classing

Automatic vendor classing

- Kea DHCP automatically assigns a vendor client class if a vendor option (DHCPv4 option 60 or DHCPv6 option 16) is set in the DHCP request
- The content of that option is added to the string `VENDOR_CLASS_` and the result is interpreted as a class name
  - For example, modern cable modems send this option with value `docsis3.0`, so the packet belongs to class `VENDOR_CLASS_docsis3.0`

Automatic vendor classing example

- Example subnet selection based on the vendor option
  - A client **must** be in any of the client classes listed to get a lease from this subnet
  - The vendor options used in this exercise are examples and not the real-world vendor option values:

```
        "shared-networks": [
            {
                "name": "kea-net01",
                "relay": { "ip-address": "192.0.2.1" },
                "subnet4": [
                    {
                        "subnet": "192.0.2.0/24",
                        "client-class": "VENDOR_CLASS_windowsCE", # <-- Windows CE Clients will get
                                                                  # an IP from this subnet
                        "option-data": [{
                                "name": "routers", "data": "192.0.2.1" }],
                        "pools": [{
                                "pool": "192.0.2.60 - 192.0.2.220" }]
                    },
                    {
                        "subnet": "10.0.0.0/24",
                        "client-class": "VENDOR_CLASS_fedoraLinux", # <-- Fedora-Linux Clients will
                                                                    # get an IP from this subnet
                        "option-data": [
[...]
```

All content © 2023 Internet Systems Consortium, Inc.

Dynamic classes from templates

Dynamic classes from templates

- Kea DHCP can dynamically create new classes from configuration templates
  - This is similar to the *spawning classes* feature in ISC DHCP
- Dynamic classes use a `template-test` function with an expression
- The expression must return a string (not a true/false value as with `test`)
  - The string is attached to the name of the template class to create the name for the new dynamic class
  - The dynamic class name is created from the string `SPAWN`, underscore, then the name of the template class (`fingerprint` in the example), underscore and the value of the string returned from the `template-test` expression

Example of a simple dynamic class

- This example below will take the first 3 bytes from the clients hardware address (MAC-Address, which contains the hardware *Organizationally unique identifier* of the vendor)

Example of a simple dynamic class

- For the hardware address `5c:1b:f4:81:01:01` (`5c:1b:f4` is Apple, Inc.) the configuration below will create a class named `SPAWN_oui-vendor_5c:1b:f4`:

```
"Dhcp4": {
  "client-classes": [
    {
      "name": "oui-vendor",
      "template-test": "hexstring(substring(pkt4.mac, 0, 3), ':')"
    }
 [...]
```

Use of dynamic classes in Kea DHCP

- This configuration below will assign the DHCP option `captive-portal` (RFC 8910: https://www.rfc-editor.org/rfc/rfc8910) to all Apple machines with the Vendor OUI `5c:1b:f4`:

```
"Dhcp4": {
  "client-classes": [
    {
      "name": "oui-vendor",
      "template-test": "hexstring(substring(pkt4.mac, 0, 3), ':')",
    },
    {
      "name": "SPAWN_oui_vendor_5c:1b:f4",
      "option-data": [
          {
              "name": "captive-portal",
              "data": "https://example.org/captive-portal/api"
          }
      ]
 }]}
```

Use of dynamic classes in Kea DHCP

- The created dynamic class can be used in the Kea DHCP configuration as any client class - in this example a DHCP4 pool is assigned based on the `oui-vendor` dynamic class:

```
 "subnet": [
   {
     "client-class": "SPAWN_oui-vendor_5c:1b:f4",
     "subnet": "192.0.2.0/24",
     "pools": [ "pool": "192.0.2.100-192.0.2.159" ]
   }
   ]
[...]
```

Example of grouping dynamic classes into static classes

- Sometimes multiple dynamic classes should be grouped into a static client class
  - Large vendors use multiple OUIs, operating systems use different PRL but still belong to the same class of machines
- Example configuration for DHCP client fingerprinting using dynamic client classes

```
   [...]
    "client-classes": [{
       "name": "fingerprint",
       "template-test": "ifelse(hexstring(option[55].hex,':') == '', 'NOPRL', hexstring(option[55].hex,
   { "name": "SPAWN_fingerprint_01:02:06:0c:0f:1a:1c:79:03:21:28:29:2a:77:f9:fc:11" },
   { "name": "SPAWN_fingerprint_01:79:03:06:0f:6c:72:77:fc:5f:2c:2e" },
   { "name": "SPAWN_fingerprint_01:79:03:06:0f:6c:72:77:fc:5f:2c" },
   { "name": "Client-Linux","test": "member('SPAWN_fingerprint_01:02:06:0c:0f:1a:1c:79:03:21:28:29:2a:7
   { "name": "Client-macOS","test": "member('SPAWN_fingerprint_01:79:03:06:0f:6c:72:77:fc:5f:2c:2e') or
   ],
[...]
```

Example of grouping dynamic classes into static classes

- The first part of the configuration defines the template for the dynamic class
    - The template test will create a class with either the name `SPAWN_fingerprint_NOPRL` if not option 55 (Parameter Request List) is available, or a class with the name `SPAWN_fingerprint_<hex-of-PRL-list>` with the hexadecimal representation of the PRL

```
   [...]
    "client-classes": [{
       "name": "fingerprint",
       "template-test": "ifelse(hexstring(option[55].hex,':') == '', 'NOPRL', hexstring(option[55].hex,
   { "name": "SPAWN_fingerprint_01:02:06:0c:0f:1a:1c:79:03:21:28:29:2a:77:f9:fc:11" },
   { "name": "SPAWN_fingerprint_01:79:03:06:0f:6c:72:77:fc:5f:2c:2e" },
   { "name": "SPAWN_fingerprint_01:79:03:06:0f:6c:72:77:fc:5f:2c" },
   { "name": "Client-Linux","test": "member('SPAWN_fingerprint_01:02:06:0c:0f:1a:1c:79:03:21:28:29:2a:7
   { "name": "Client-macOS","test": "member('SPAWN_fingerprint_01:79:03:06:0f:6c:72:77:fc:5f:2c:2e') or
   ],
[...]
```

Example of grouping dynamic classes into static classes

- The second part of the configuration defines the possible dynamic classes generated from the template.
    - Defining the possible dynamic classes is required as the Kea DHCP configuration parser needs to know the names of the classes to be matched in the static client classification rules below

```
   [...]
    "client-classes": [{
       "name": "fingerprint",
       "template-test": "ifelse(hexstring(option[55].hex,':') == '', 'NOPRL', hexstring(option[55].hex,
   { "name": "SPAWN_fingerprint_01:02:06:0c:0f:1a:1c:79:03:21:28:29:2a:77:f9:fc:11" },
   { "name": "SPAWN_fingerprint_01:79:03:06:0f:6c:72:77:fc:5f:2c:2e" },
   { "name": "SPAWN_fingerprint_01:79:03:06:0f:6c:72:77:fc:5f:2c" },
   { "name": "Client-Linux","test": "member('SPAWN_fingerprint_01:02:06:0c:0f:1a:1c:79:03:21:28:29:2a:7
   { "name": "Client-macOS","test": "member('SPAWN_fingerprint_01:79:03:06:0f:6c:72:77:fc:5f:2c:2e') or
   ],
[...]
```

Example of grouping dynamic classes into static classes

- In the 3rd part of the configuration the different dynamic classes are collected into groups by static client classing using the `member` expression which tests the membership inside a client class

```
   [...]
    "client-classes": [{
       "name": "fingerprint",
       "template-test": "ifelse(hexstring(option[55].hex,':') == '', 'NOPRL', hexstring(option[55].hex,
   { "name": "SPAWN_fingerprint_01:02:06:0c:0f:1a:1c:79:03:21:28:29:2a:77:f9:fc:11" },
   { "name": "SPAWN_fingerprint_01:79:03:06:0f:6c:72:77:fc:5f:2c:2e" },
   { "name": "SPAWN_fingerprint_01:79:03:06:0f:6c:72:77:fc:5f:2c" },
   { "name": "Client-Linux","test": "member('SPAWN_fingerprint_01:02:06:0c:0f:1a:1c:79:03:21:28:29:2a:7
   { "name": "Client-macOS","test": "member('SPAWN_fingerprint_01:79:03:06:0f:6c:72:77:fc:5f:2c:2e') or
   ],
[...]
```

Lease limiting and rate limiting with dynamic classes

- The *Limits* hook library (https://kea.readthedocs.io/en/latest/arm/hooks.html#lim limits-to-manage-lease-allocation-and-packet-processin paid support contract) can limit leases and DHCP responses
  - Lease limiting: allow a maximum of `n` leases assigned at any one tim
  - Rate limiting: allow a maximum of `n` packets per `time_unit` to rece response.
- With the combination of template classes and the *limits* hook library, limits can be dynamically assigned to group of clients

Lease limiting and rate limiting with dynamic classes

- The example below limits the number of leases given to dynamic classes created out of a template classes for each vendor OUI.
  - Only 2 leases are allowed for each vendor OUI, and only 10 request packets per minute will be answered by this Kea DHCP server

```
{
  "Dhcp6": {
    "client-classes": [{
        "name": "oui-vendor",
        "template-test": "hexstring(substring(option[1].hex, 4, 3), ':')",
        "user-context": {
          "limits": {
            "address-limit": 2,
            "rate-limit": "10 packets per minute"
          }
}}]}}
```

All content © 2023 Internet Systems Consortium, Inc.

Automatic classes

The KNOWN and UNKNOWN classes

- Kea automatically assigns classes based on host reservations
  - All clients **with** a host reservation will be in the `KNOWN` class
  - All client **without** reservation will be in the `UNKNOWN` class
- For example, these classes can be used to separate *guests* from *staff* clients

```
{
    "client-classes": [{
            "name": "dependent-class",
            "test": "member('KNOWN')",
            "only-if-required": true
    }]
}
```

All content © 2023 Internet Systems Consortium, Inc.

Classification via Hooks

Classification via hooks

- Client classification via complex expressions can hurt the DHCP server performance
- Alternative: writing a custom hook for client classification

All content © 2023 Internet Systems Consortium, Inc.

Debugging client classing

Debugging client classing (1/5)

- To debug client classing based on expressions, enable debug logging inside the Kea DHCP server
- Quick option: start Kea DHCP4 in debug mode from the command line. This will automatically enable the highest debugging level
  - On a busy server, this will create too much debug information (see next slide for an alternative)

```
[kea-server]# systemctl stop kea-dhcp4
[kea-server]# kea-dhcp4 -d -c /etc/kea/kea-dhcp4.conf
```

Debugging client classing (2/5)

- Alternative: enable the special `kea-dhcp4.eval` or `kea-dhcp6.eval` debug logger in the Kea configuration file

```
"Logging": {
    "loggers": [ {
        "name": "kea-dhcp4.eval",
        "output_options": [ {
            "output": "/var/log/kea-dhcp4-eval.log"
        } ],
        "severity": "DEBUG",
        "debuglevel": 55
    } ]
}
```

Debugging client classing (3/5)

- Watch for the test evaluation results in the Kea Eval DHCP4 log file

```
[kea-server]# tail -f /var/log/kea-dhcp4-eval.log
```

Debugging client classing (4/5)

- Main Kea DHCP log message

```
2023-05-31 17:32:58.323 INFO  [kea-dhcp4.dhcpsrv/115574.139651153725120]
           EVAL_RESULT Expression fingerprint evaluated to NOPRL
```

Debugging client classing (5/5)

- Example of client classing debug log entries

```
2023-05-31 17:28 DEBUG [kea-dhcp4.eval] EVAL_DEBUG_OPTION Pushing option 55 with value 0x0102060C0F1A1C
2023-05-31 17:28 DEBUG [kea-dhcp4.eval] EVAL_DEBUG_STRING Pushing text string ':'
2023-05-31 17:28 DEBUG [kea-dhcp4.eval] EVAL_DEBUG_OPTION Pushing option 55 with value 0x0102060C0F1A1C
2023-05-31 17:28 DEBUG [kea-dhcp4.eval] EVAL_DEBUG_STRING Pushing text string ':'
[...]
2023-05-31 17:28 DEBUG [kea-dhcp4.eval] EVAL_DEBUG_TOHEXSTRING Popping binary value 0x0102060C0F1A1C790
2023-05-31 17:28 DEBUG [kea-dhcp4.eval] EVAL_DEBUG_STRING Pushing text string 'NOPRL'
2023-05-31 17:28 DEBUG [kea-dhcp4.eval] EVAL_DEBUG_IFELSE_FALSE Popping 'false' (false) and 0x30313A303
```

Debugging client classing (6/6)

- The Kea `dhcp4` logger in *DEBUG* mode will print the list of classes assigned to a DHCP request, also showing the client-id (`cid`) and the hardware-address (`hwtype=1 00:60:6e:43:b0:e5`) (output formatted for readability):

```
2023-06-01 08:23:42.428 DEBUG [kea-dhcp4.dhcp4/117392.139838341371584]
       DHCP4_CLASS_ASSIGNED [hwtype=1 00:60:6e:43:b0:e5], cid=[01:00:60:6e:43:b0:e5],
       tid=0xcf0cd96f: client packet has been assigned to the following class(es):
       ALL, fingerprint, SPAWN_fingerprint_01:02:06:0c:0f:1a:1c:79:03:21:28:29:2a:77:f9:fc:11, Client-I
```

All content © 2023 Internet Systems Consortium, Inc.

Resources

- Facilitating Classification with Template Classes https://kb.isc.org/v1/docs/facilitating-classification-with-template-classes
- Kea DHCP Client Classification documentation https://kea.readthedocs.io/en/latest/arm/classify.html

All content © 2023 Internet Systems Consortium, Inc.

Upcoming ISC Webinar

- The Webinars for the 2nd half of 2023 will be announced on the ISC Website https://www.isc.org/categories/webinars/
- Recordings of previous webinars are available at https://www.isc.org/presentations/

All content © 2023 Internet Systems Consortium, Inc.

Questions / Answers

All content © 2023 Internet Systems Consortium, Inc.