Kea Webinar

Kea lease allocation, client classification and option assignment

Carsten Strotmann
14th October 2020

https://www.isc.org
Welcome

• Welcome to part three of our webinar series "the KEA DHCP Server"
About this Webinar

• Lease allocation
• Client classification
• DHCP options
• DHCP reservations
• Shared subnets
• Questions & Answers
Lease allocation
Kea lease allocation (simplified)

(without shared-networks or reservations)
Kea lease allocation (detail)
Kea lease allocation details

• When searching for a new lease
• Kea 1.8 iterates over all subnets by subnet-id
• Previous versions iterated over subnets in configuration file order
Client classification
DHCP client classes

• Kea DHCP can assign one or more client classes to client requests
• Depending on the client classes, different DHCP information can be send to the client:
  • DHCP-Options
  • IP-Addresses
  • BOOTP-Parameter inside DHCP responses
• Kea can select from multiple subnets / pools with the help of client classes
DHCP client classes

- Client classes can be build from various DHCP identifier information from the client host
- information from the DHCP relay
- information from the DHCP packet path towards the DHCP server
- Client classification examines the incoming DHCP packet's contents and selects one or more class(es) based on configuration criteria
Where do DHCP identifier come from?
Where do DHCP identifier come from?
Where do DHCP identifier come from?
Where do DHCP identifiers come from?
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 prepended with VENDOR_CLASS_ and the result is interpreted as a class
  • 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:
Automatic vendor classing example

"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": [
          [...]
"
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

```json
{
    "client-classes": [{
        "name": "dependent-class",
        "test": "member('KNOWN')",
        "only-if-required": true
    }
}
```
Dynamic 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
  • Logical and conditional expressions can be used to assign classes to the DHCP request
• List of available expressions:
  https://kea.readthedocs.io/en/kea-1.8.0/arm/classify.html#using-expressions-in-classification
Dynamic client classing based on expressions

<table>
<thead>
<tr>
<th>Name</th>
<th>Example expression</th>
<th>Example value</th>
</tr>
</thead>
<tbody>
<tr>
<td>String literal</td>
<td>'example'</td>
<td>'example'</td>
</tr>
<tr>
<td>Hexadecimal string literal</td>
<td>0x5a7d</td>
<td>'ZJ'</td>
</tr>
<tr>
<td>IP address literal</td>
<td>10.0.0.1</td>
<td>0xa0000001</td>
</tr>
<tr>
<td>Integer literal</td>
<td>123</td>
<td>'123'</td>
</tr>
<tr>
<td>Binary content of the option</td>
<td>option[123].hex</td>
<td>'content of the option'</td>
</tr>
<tr>
<td>Option existence</td>
<td>option[123].exists</td>
<td>'true'</td>
</tr>
<tr>
<td>Binary content of the sub-option</td>
<td>option[12].option[34].hex</td>
<td>'content of the sub-option'</td>
</tr>
<tr>
<td>Sub-Option existence</td>
<td>option[12].option[34].exists</td>
<td>'true'</td>
</tr>
<tr>
<td>Client class membership</td>
<td>member(&quot;foo\bar&quot;)</td>
<td>'true'</td>
</tr>
<tr>
<td>Known client</td>
<td>known</td>
<td>member(KOWN)</td>
</tr>
<tr>
<td>Unknown client</td>
<td>unknown</td>
<td>not member(KOWN)</td>
</tr>
<tr>
<td>DHCPv4 relay agent sub-option</td>
<td>relay[123].hex</td>
<td>'content of the RAI sub-option'</td>
</tr>
<tr>
<td>DHCPv4 Relay Options</td>
<td>relay[6].option[codec].hex</td>
<td>(value of the option)</td>
</tr>
<tr>
<td>DHCPv4 Relay Peer Address</td>
<td>relay[6].peeraddr</td>
<td>2001:DB8:1</td>
</tr>
<tr>
<td>DHCPv4 Relay Link Address</td>
<td>relay[6].linkaddr</td>
<td>2001:DB8:1</td>
</tr>
<tr>
<td>Interface name of packet</td>
<td>pkt.iface</td>
<td>eth0</td>
</tr>
<tr>
<td>Source address of packet</td>
<td>pkt.src</td>
<td>10.1.2.3</td>
</tr>
<tr>
<td>Destination address of packet</td>
<td>pkt.dst</td>
<td>10.1.2.3</td>
</tr>
<tr>
<td>Length of packet</td>
<td>pkt.len</td>
<td>513</td>
</tr>
<tr>
<td>Hardware address in DHCPv4 packet</td>
<td>pkt4.mac</td>
<td>0x010203040506</td>
</tr>
<tr>
<td>Broadcast length in DHCPv4 packet</td>
<td>pkt4.bcast</td>
<td>6</td>
</tr>
</tbody>
</table>
Client classification example (1/3)

• configuration for dynamic client classing based on the vendor option (Option 60)

```json
"Dhcp4": {
  "client-classes": [  
    {
      "name": "windows",
      "test": "substring(option[60].hex,0,3) == 'win'",
      "option-data": [{
        "name": "domain-name", "data": "win.example.com" }
      }
    },
    {
      "name": "other",
      "test": "not(substring(option[60].hex,0,3) == 'win')",
      "option-data": [{
        "name": "domain-name", "data": "other.example.com" }
      }
    }
  ],
  [...]
}
```
Client classification example (2/3)

• the client class is used to select a subnet inside a shared network
• windows clients get IP addresses from the 1st subnet
• client with other operating systems get IP addresses from the 2nd subnet
"shared-networks": [
    {
        "name": "kea-lab01",
        "relay": { "ip-address": "192.0.2.1" },
        "subnet4": [
            {
                "subnet": "192.0.2.0/24",
                "client-class": "windows", # all Windows 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": "other", # non Windows Clients will
                # get IP addresses from this subnet
                "option-data": [
                    {...}
                ]
            }
        ]
    }
]
Classification via hooks

- Client classification via complex expressions can hurt the DHCP server performance
- Alternative: writing a custom hook for client classification
Debugging client classing (1/3)

• 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 (2/3)

- 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 (3/3)

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

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

• DHCP options can be configured in different scopes in the Kea configuration
  • global
  • class
  • subnet
  • pools
  • reservations
Global DHCP options (1/2)

"Dhcp4": {
    "option-data": [{
        "name": "domain-name-servers",
        "code": 6,
        "space": "dhcp4",
        "csv-format": true,
        "data": "192.0.2.1, 192.0.2.2"
    },
    ...
}
}
Global DHCP options (2/2)

- if the default values are used, the fields code, space and csv-format can be omitted

  "Dhcp4": {
      "option-data": [
        {
          "name": "domain-name-servers",
          "data": "192.0.2.1, 192.0.2.2"
        },
        ...
      ]
  }
"client-classes": [{
    "name": "Zimbutsio-Server",
    "test": "option[vendor-class-identifier].text == 'Zimbutsio'",
    "option-data": [ {
        "name": "log-servers",
        "data": "192.0.2.42"
    }]
}]
[...]

Client class options
Defining custom DHCPv4 options (1/2)

• sometimes it is required to define custom DHCP options that are not part of the DHCP standards.

• These can be vendor specific options, or new DHCP options that are not yet implemented in Kea DHCP
Defining custom DHCPv4 options (2/2)

```json
{
    "Dhcp4": {
        "option-def": [{
            "name": "my-message",
            "code": 234,
            "type": "string",
            "array": false,
            "record-types": "",
            "space": "dhcp4",
            "encapsulate": "" },
        "option-data": [{
            "name": "my-message",
            "space": "dhcp4",
            "csv-format": true,
            "data": "Hello World" }
        },
        [...]
    }
}
```
Option assignment order

Client-class options are assigned in the order in which the client classes are evaluated (specified in the
DHCP reservations
Why DHCP reservations

- Security policies
- stable addressing (server)
- IP bound licenses
- captive portal
  (KNOWN vs. UNKNOWN clients)
DHCP reservations

• Kea DHCP supports reservations of client leases based on
  • hardware interface address (MAC-Address)
  • DHCP Unique ID (DUID)
  • Relay-Circuit-ID (DHCPv4)
  • Client-ID / Hostname (DHCPv4)
  • flex.id
DHCP reservation parameter

- Alongside IP-Address leases, reservations can also reserve a number of DHCP parameters for a client
  - hostname
  - DHCP options
  - reservation-client-classes
  - boot-file-name (BOOTP/DHCPv4)
  - next-server (BOOTP/DHCPv4)
  - server-hostname (BOOTP/DHCPv4)
Global vs. Subnet reservation (1/2)

• DHCP reservations can optionally be defined on a global scope
• global reservations can be used to assign a fixed hostname or other options to a client
• Kea does not prevent the definition of DHCP parameters on the global level that are only useful in an subnet scope (like IP address or IPv4 default route). Be careful!
Global vs. Subnet reservation (2/2)

• The common case is to have reservations in the subnet or shared-subnet scope
• Kea 1.9 will allow for reservations to be defined on a global and subnet level
Example of global reservation

"Dhcp4": {

    # This specifies global reservations. They will apply to all subnets that
    # have global reservations enabled.

    "reservations": [
        { "hw-address": "aa:bb:cc:dd:ee:ff", "hostname": "hw-host-dynamic" },
        { "hw-address": "01:02:03:04:05:06", "hostname": "hw-host-fixed", "ip-address": "192.0.1.77" }, # risky!
        { "circuit-id": "'office042'", "hostname": "circuit-id-host" },
    [...]
in-pool vs out-of-pool reservations

• Host reservations can be inside a dynamic DHCP pool or outside a dynamic DHCP pool
• Reservations that are inside a pool can lead to DHCP conflicts
  (https://kea.readthedocs.io/en/latest/arm/dhcp4-srv.html#conflicts-in-dhcpv4-reservations)
  and also might result in a performance loss
  (see DHCP tuning)
Dynamically manage DHCP reservations

• Small Kea deployments (small = a few hundred client machines) can have the DHCP reservations inside the Kea configuration file.

• Larger deployments might want to change the DHCP reservations dynamically and programatically via the API.

• The Host Commands hook (part of the Premium hooks package) adds a number of new commands to Kea used to query and manipulate host reservations.
Dynamically manage DHCP reservations

• the Host Commands hook requires a database (-> next webinar) for storing the host reservations

• If reservations are specified in both file and database, file reservations take precedence over the ones in the database.
# Host Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reservation-add</td>
<td>add a new reservation to the Kea DB</td>
</tr>
<tr>
<td>reservation-get-all</td>
<td>get all reservation information (can be huge)</td>
</tr>
<tr>
<td>reservation-get</td>
<td>get information on a single reservation (by address or identifier)</td>
</tr>
<tr>
<td>reservation-get-page</td>
<td>get all reservation information from a subnet by pages (used for GUI display)</td>
</tr>
<tr>
<td>reservation-get-by-hostname</td>
<td>get the reservation information for one host by its hostname</td>
</tr>
<tr>
<td>reservation-get-by-id</td>
<td>get the reservation information for one host by its identifier (global, since 1.9.0)</td>
</tr>
<tr>
<td>reservation-del</td>
<td>delete a reservation from the database</td>
</tr>
</tbody>
</table>
Example command file to add a reservation (1/2)

• this command snippet can be used to create a new reservation inside the Kea Host database

```json
$ cat reservation-add.json
{
  "command": "reservation-add",
  "service": [ "dhcp6" ],
  "arguments": {
    "reservation": {
      "duid": "01:02:03:04:05:06:07:08:09:0A",
      "hostname": "foo.example.com",
      "ip-addresses": [ "2001:db8:1::1" ],
      "option-data": [{
        "data": "4491",
        "name": "vendor-opts"
      }, {
        "data": "3000:1::234",
        "name": "tftp-servers",
        "space": "vendor-4491"
      }],
      "subnet-id": 1
    }
  }
}
```
Example command file to add a reservation (2/2)

- the curl command can be used to send the request towards the Kea API

```bash
$ curl -s -X POST -H "Content-Type: application/json" \\
-d @reservation-add.json http://127.0.0.1:8000/ | jq

[ 
  {
    "result": 0,
    "text": "Host added."
  }
]
```
Example command file retrieving all reservations

• this command snippet can be used to retrieve all reservations from the Kea Host database

```
$ cat reservation-get-all.json
{
  "service": [
    "dhcp6"
  ],
  "command": "reservation-get-all",
  "arguments": {
    "subnet-id": 1
  }
}
$ curl -s -X POST -H "Content-Type: application/json" \
-d @reservation-get-all.json http://127.0.0.1:8000/ | jq
```
Client classing in reservations

- clients can be associated to a client-class using a reservation (using the Hardware-Address, DUID, Client-ID, Relay-ID)

```json
[...]
    "subnet4": [ 
        { 
            "subnet": "10.0.0.0/24",
            "pools": [ { "pool": "10.0.0.10-10.0.0.200" } ],
            "reservations": [{
                "hw-address": "01:02:03:04:05:06",
                "client-classes": [ "windows", "staff" ]
            }],
        }
    ],
[...]
```
Performance tuning DHCP reservations (1/4)

• Kea DHCP must check for every lease request for conflicts with reservations. This can slow down the DHCP lease assignment process.
• In some cases, where reservations are not in use or used only in certain scopes, some of these checks can be disabled with the reservation-mode configuration parameter.
• The parameter can be specified at global, subnet, and shared-network levels.

```
"Dhcp4": {
    "subnet4": [{
        "subnet": "192.0.2.0/24",
        "reservation-mode": "disabled",
        ...
    }
}
```
Performance tuning DHCP reservations (2/4)

<table>
<thead>
<tr>
<th>reservation-mode</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>all</td>
<td>reservations can be on global, subnet or inside pool scope, all checks enabled</td>
</tr>
<tr>
<td>out-of-pool</td>
<td>reservations in subnets are always outside the pool</td>
</tr>
<tr>
<td>global</td>
<td>only global reservations allowed, not subnet/pool reservations</td>
</tr>
<tr>
<td>disabled(*)</td>
<td>host reservation support is disabled, no checks for collisions</td>
</tr>
</tbody>
</table>

(*) the best performance is achieved when host reservations are disabled (if no reservations are used). In that case Kea can skip all the checks and lookups.
Performance tuning DHCP reservations (3/4)

• Kea currently supports four types of identifiers:
  • hw-address
  • duid
  • client-id
  • circuit-id
  • flex-id

• For each incoming packet, Kea has to extract each identifier type and then query the database to see if there is a reservation by this particular identifier.
Performance tuning DHCP reservations (4/4)

• A parameter called host-reservation-identifiers takes a list of identifier types that Kea will check
  • For best performance the number of identifier types should be kept to a minimum, ideally one

```json
"host-reservation-identifiers": [ "circuit-id", "hw-address" ],
"subnet4": [{
  "subnet": "192.0.2.0/24",
  ...
}]
```
Shared subnets
Shared subnets

• a shared subnet is a physical subnet with multiple IP networks
• one shared subnet definition can contain two or more subnet definitions
• options can be defined on the shared-network, subnet and pool level
• without client classification, Kea might choose an IP address from any pool of all subnets inside the shared network
When to use Shared Subnets

• Shared Subnets are adding complexity to a DHCP server configuration and should only be used if there is a good use case.
  • Shared subnet are sometimes created if a larger number of IP addresses are needed in a network, but because of IPv4 address shortage no continuous range of IPv4 addresses are available.
  • Another use case of shared subnets is a network where addresses from different IPv4 subnets (and possibly different network configuration) should be given to different network devices.
Kea configuration shared subnet example

[...,]

"shared-networks": [
  {
    "name": "kea-lab01",
    "relay": { "ip-address": "192.0.2.1" },
    "subnet4": [
      {
        "subnet": "192.0.2.0/24",
        "option-data": [
          { "name": "routers", "data": "192.0.2.1" }
        ],
        "pools": [
          { "pool": "192.0.2.20 - 192.0.2.190" }
        ]
      },
      {
        "subnet": "10.0.0.0/24",
        "option-data": [
          { "name": "routers", "data": "10.0.0.1" }
        ],
        "pools": [
          { "pool": "10.0.0.10 - 10.0.0.200" }
        ]
      }
    ]
  }
,...]
Next Webinars

• 28th October - Kea DHCP - High Availability and Database Backends
• 18th November - Kea DHCP - Monitoring, Logging, and Stork
• 2nd December - Kea DHCP - Migrating to Kea from ISC DHCP
Resources

• Understanding Client Classification
  https://kb.isc.org/docs/en/understanding-client-classification

• Do I need to use shared-networks or not with Kea DHCP?

• Host Reservation in DHCPv4
  https://kea.readthedocs.io/en/latest/arm/dhcp4-srv.html#host-reservation-in-dhcpv4

• Standard DHCP Options Defined in ISC DHCP and Kea
  https://kb.isc.org/docs/en/aa-01323
Questions and Answers