# maps & policy delegation

Ralf Hildebrandt

Linuxforum 2005
Copenhagen

# What are maps?

- Postfix uses lookup tables to store and look up information for:
  - access control
  - address rewriting
  - and even for content filtering
- all Postfix lookup tables store information as (key, value) pairs
- works just like a phonebook:
  - key: Name
  - value: phone number

# What are maps?

- The (key, value) query interface completely hides the complexities of LDAP or SQL from Postfix
- This is a good example of connecting complex systems with simple interfaces

# Where are maps used?

- local aliasing:

```
alias_maps = hash:/etc/postfix/aliases
```

- address rewriting:

```
virtual_alias_maps = hash:/etc/postfix/virtual
```

- routing table:

```
transport_maps = hash:/etc/postfix/transport
```

- content filtering:

```
header_checks = regexp:/etc/postfix/header_checks
```

# What kind of maps are there?

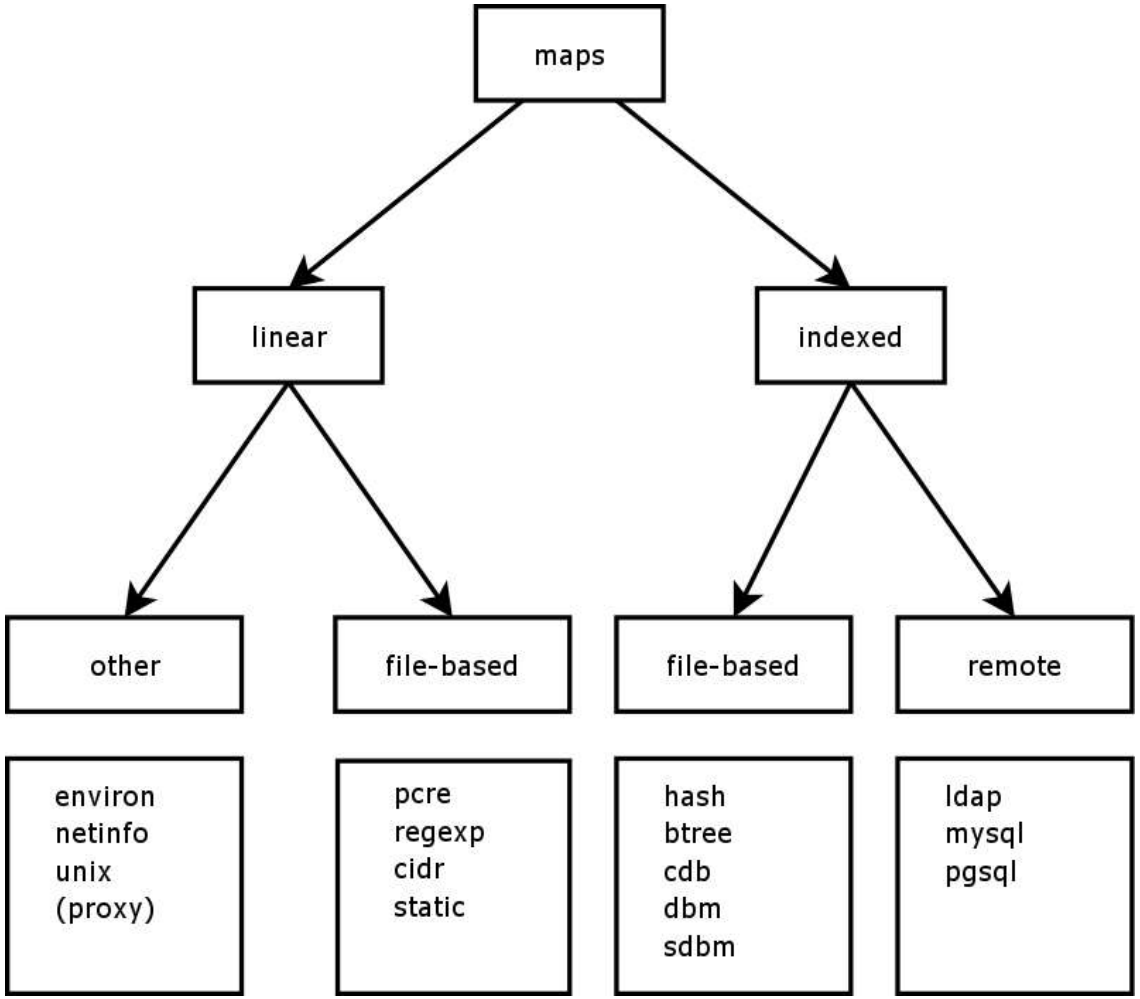
indexed maps

- `hash`
- `btree`
- `cdb`
- `*sql`
- `ldap`

linear maps

- `pcre`
- `regexp`
- `cidr`
- `environ`
- `nis`
- `nisplus`

# A maps overview



# Why use *SQL or LDAP at all?

- file-based maps are opened before going to the `chroot()` jail
- for a change in a file-based map to take effect, the daemon exits and upon rebirth opens the altered map

**What happens if a file-based map changes very often?**

```
Mar  5 12:10:05 mail postfix/smtpd[10388]:\
table cdb:/etc/postfix/no_internet_mail has changed \
-- restarting
```

# How do lookups actually work?

- maps can only find exact matches
- how do the lookups in the different map types differ?
  - linear
  - indexed

# lookups in indexed maps

- create new queries (based on the syntax described in the man pages) by breaking up the input key:
  - `user@sub.domain.tld` mail addresses are broken up into their `user` and `sub.domain.tld` constituent parts
  - `sub.domain.tld` is broken up into `domain.tld` and `tld`
- `parent_domain_matches_subdomains` controls if a `domain.tld map entry` matches `sub.domain.tld` (usually `.domain.tld` is required).

# lookups in linear maps

- each pattern is applied to the **entire** input.
- depending on the application (routing, aliasing, ...), that input is an:
  - entire client hostname,
  - an entire client IP address, or
  - an entire mail address

Thus, no parent domain or parent network search is done, `user@sub.domain.tld` mail addresses are **not** broken up into their user and domain constituent parts, and "`user+foo`" is not broken up into `user` and `foo`.

# How do the precedences differ?

**canonical/virtual**

```
user@domain   address
user          address
@domain*      address
```

**access**

```
user@domain   OK
domain.tld    OK
user@         OK
```

***: Replace `user@site` by address when `site` is `myorigin`, in `mydestination`, in `inet_interfaces` or in `proxy_interfaces`.**

# lookup in an indexed `access(5)`

`Ralf.Hildebrandt@charite.de` is being looked up in an `access(5)` map (e.g by using: `check_sender_access hash:/etc/postfix/sender_access`

1. `Ralf.Hildebrandt@charite.de` is being used as key
2. `charite.de` is being used as key
3. `.charite.de` is being used as key
4. `de` is being used as key
5. `.de` is being used as key
6. `Ralf.Hildebrandt@` is being used as key

So, looking up `Ralf.Hildebrandt@charite.de` causes between 1 and 6 individual queries against the database.

# lookup in a regexp access(5)

Take this regexp map:

```
/LinuxForum 2005/      Rules!
/^ralf/                REJECT
```

- `Ralf.Hildebrandt@charite.de` is being looked up in an `access(5)` map (e.g by using: `check_sender_access regexp:/etc/postfix/sender_access`)
- Does „`Ralf.Hildebrandt@charite.de`“ match `/LinuxForum 2005/`?
- Does „`Ralf.Hildebrandt@charite.de`“ match `/^ralf/`?

It does, since the lookups are case-insensitive by default!

# Caveats

- Don't use `regexp/pcre` where ordinary `hash/btree` lookups suffice:
  `/^pattern@domain$/` in `regexp` is equivalent to `pattern@domain` in `hash/btree/...`
- since every line of the `regexp/pcre` file is tried until a match is found or the end is reached!
- This is the reason for massive CPU consumption with large `header_checks` or `body_checks` files.

# Performance Issues

- This lookup behaviour can cause problems!
- high-latency – `connect()`, `bind()`, query, response
- high-concurrency – many Postfix processes query simultaneously
- a DoS or dictionary attack will directly hit the backend

# Solutions

- export LDAP/*sql to a file-based map to improve performance
  - the database backend cannot handle the high concurrency of queries
  - file-based map has less latency
  - instantaneous updates are rarely needed with email
- `proxymap`
  - consolidate the number of open lookup tables
  - consolidate the number of simultaneous queries

# What is proxymap?

- `proxymap` for consolidation of lookups:
  - overcome `chroot()`
  - consolidate the number of open lookup tables (one open table is shared among multiple processes)

- 1000 `smtpd` processes query
- 1 sql server simultaneously

- 1000 `smtpd` processes query
- 20 `proxymap` processes which query
- 1 sql server simultaneously

# Disadvantages!

- `proxymap` opens only tables that are approved via the `proxy_read_maps`
- `proxymap` is not a trusted daemon process, and must not be used to look up sensitive information such as user or group IDs, mailbox file/directory names or external commands.

**You cannot use it for `aliases`**

# CDB Advantages

CDB (Constant Data Base)

- Advantages
  - indexed file format designed by Daniel Bernstein.
  - CDB is optimized exclusively for read access
  - guarantees that each record will be read in at most two disk accesses.

Yay!

# CDB Disadvantages

- achieved by forgoing support for incremental updates...
- no single-record inserts or deletes are supported!
- CDB databases can be modified only by rebuilding them completely from scratch, hence the "constant" qualifier in the name.
- thus cannot use it e.g. for the maps used by `verify(8)`

You need `tinycdb` (version 0.5 and later) from Michael Tokarev, available from
http://www.corpit.ru/mjt/tinycdb.html

# BerkeleyDB Pro & Contra

BerkeleyDB

- Advantages
  - It comes with your system!
- Disadvantages
  - Berkeley DB version 4 is not supported by Postfix versions before 2.0
  - many commercial UNIXes ship without Berkeley DB support
  - some Linux system libraries use Berkeley DB, as do some third-party libraries such as SASL.

# BerkeleyDB disadvantages II

- If you compile Postfix with a different Berkeley DB implementation, then every Postfix program will dump core because either the system library, SASL library, or Postfix itself ends up using the wrong version.

And you thought switching over from Windows to escape DLL hell was a good idea...

- The more recent Berkeley DB versions have a configure-time switch, "`--with-uniquename`", which renames the symbols in a unique way, so that multiple versions of Berkeley DB can co-exist in the same application.

# LDAP

- Note: Postfix no longer supports the LDAP version 1 interface

(see http://www.postfix.org/LDAP_README.html)

```
alias_maps = ldap:/etc/postfix/ldap-aliases.cf
```

and in /etc/postfix/ldap-aliases.cf you have:

```
server_host = ldap.my.com
search_base = dc=my, dc=com
```

# LDAP II

Upon receiving mail for a local address "ldapuser" Postfix will:

- search the LDAP server listening at port 389 on `ldap.my.com`.
- it will bind anonymously
- search for any directory entries whose `mailacceptinggeneralid` attribute is "`ldapuser`"
- read the"`maildrop`" attributes of those found, and build a list of their maildrops,
- which will be treated as RFC822 addresses
- to which the message will be delivered.

# LDAP III

- You can reduce the number of concurrent ldap clients by using the Postfix `proxymap(8)` service.
- The data is out there...
- all you need is the correct query :)

# mySQL

- the mysql map type allows you to hook Postfix up to a MySQL database.
- Allows for multiple mysql databases:
  - you can use one for a `virtual(5)` table,
  - a different one for an `access(5)` table,
  - and yet another one for an `aliases(5)` table
  - ...if you want.
- You can specify multiple servers for the same database, so that Postfix can switch to a good database server if one goes bad.

# mySQL Performance

- Busy mail servers using mysql maps will generate lots of concurrent mysql clients, so the mysql server(s) should be run with this fact in mind.
- You can reduce the number of concurrent mysql clients by using the Postfix `proxymap(8)` service.
- `http://www.postfix.org/MYSQL_README.html`

# postgresql

- Works just the same

# policy delegation

As of version 2.1, Postfix's `smtpd` can delegate policy decisions to an external server that runs outside Postfix, for example:

- `postgrey` by David Schweikert
  http://isg.ee.ethz.ch/tools/postgrey/
- SPF policy server by Meng Weng Wong
  http://spf.pobox.com/

# But why?

- decisionmaker can reside on remote box
- base decisions on multiple attributes instead of just one
- It's much easier to develop a new feature in few lines of Perl, than trying to do the same in C code.

# `policyd` protocol is simple

- The client request is a sequence of `name=value` attributes separated by newline, and is terminated by an empty line.
- The server reply is one `name=value` attribute and it, too, is terminated by an empty line.

# policyd example

```
request=smtpd_access_policy
protocol_state=RCPT
protocol_name=SMTP
helo_name=some.domain.tld
queue_id=8045F2AB23
sender=foo@bar.tld
recipient=bar@foo.tld
client_address=1.2.3.4
client_name=another.domain.tld
instance=123.456.7
sasl_method=plain
sasl_username=you
sasl_sender=
ccert_subject=solaris9.porcupine.org
ccert_issuer=Wietse Venema
ccert_fingerprint=C2:9D:F4:87:71:73:73:D9:18:E7:C2:F3:C1:DA:6E:04
size=12345
[empty line]
```

# Example II

- The policy server replies with any action that is allowed in a Postfix `smtpd access(5)` table.
- Example:

```
action=450 Service temporarily unavailable
[empty line]
```

- In case of trouble the policy server must not send a reply!
- Instead the server must log a warning and disconnect
- Postfix will retry the request at some later time.

# local and remote use

Postfix can connect to a TCP socket or to a UNIX-domain socket.

Examples:

```
check_policy_service inet:127.0.0.1:9998
check_policy_service unix:/some/where/policy
check_policy_service unix:private/policy
```

# master.cf and main.cf

- master.cf:

```
policy  unix  -  n  n  -  -  spawn
  user=nobody
  argv=/usr/bin/perl /etc/postfix/postfix-policyd.pl
```

- main.cf:

```
    smtpd_recipient_restrictions =
      ...
      check_policy_service unix:private/policy
```

Solaris UNIX-domain sockets do not work reliably...

Use TCP sockets instead

# Write your own?

- skeleton policy server in Perl by Michael Tokarev:
  http://www.corpit.ru/mjt/smtpd-policy-template.pl

# Questions

Now it's time for questions

# Resources

- Postfix website
  http://www.postfix.org/
- The Book of Postfix
  http://www.postfix-book.com/