scimd

SCIM provisioning for software that never got it

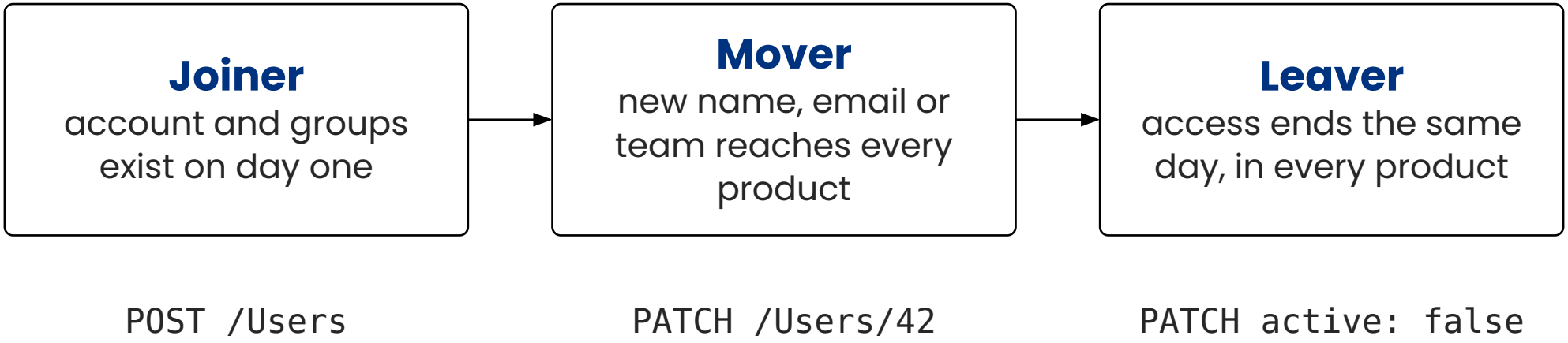
SCIMD

Simon Hauser
Helsinki Systems

Email simon.hauser@helsinki-systems.de
Matrix @simon.hauser:helsinki-systems.de

Joiners, movers, leavers

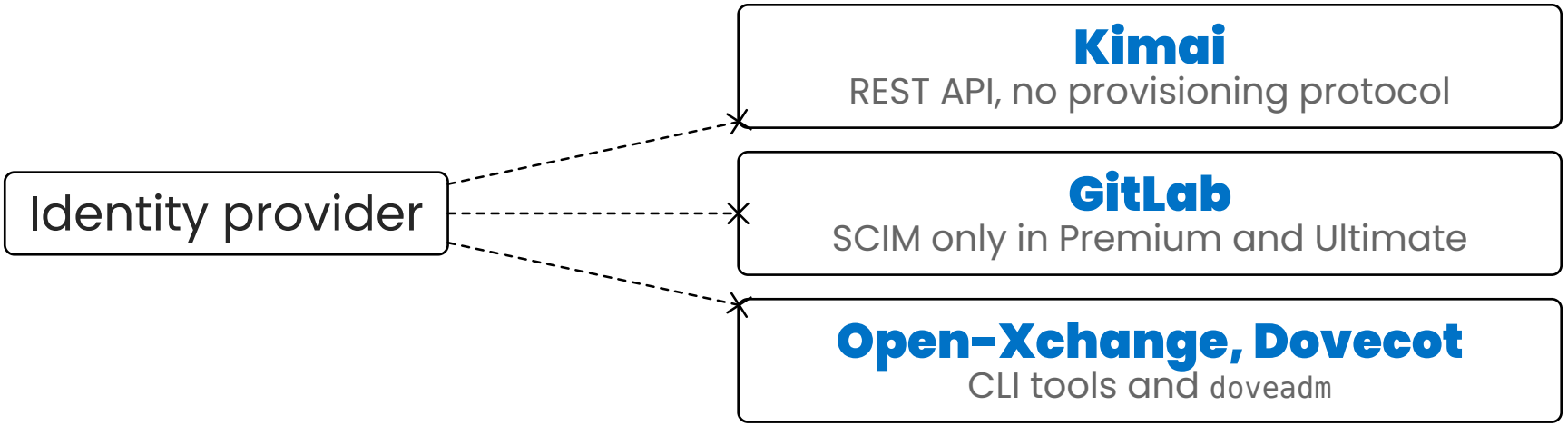
HELSINKI SYSTEMS
Network & IT-Solutions



SCIM 2.0, defined in RFC 7643 and RFC 7644, is how an identity provider pushes these changes to a product. Entra ID, Okta and authentik are examples of IdPs that act as SCIM clients.

Products without SCIM

HELSINKI SYSTEMS
Network & IT-Solutions



Without SCIM, people fall back to onboarding tickets and manual renames. Offboarding misses a product, and accounts keep working after the person has left.

We can automate every one of these products. None of them speaks SCIM.

What SCIM is

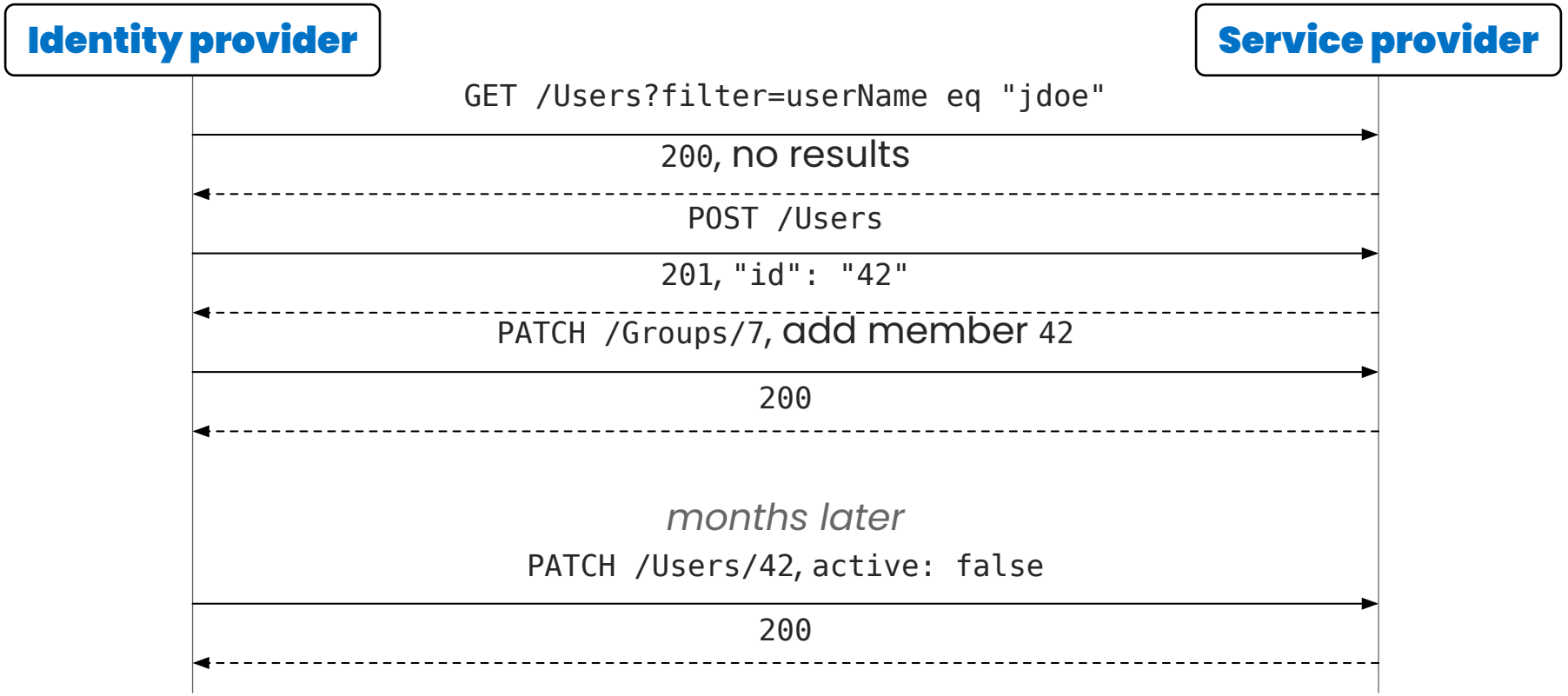
HELSINKI SYSTEMS
Network & IT-Solutions

- **RFC 7643** defines the schema: resources, attributes, extensions.
- **RFC 7644** defines the protocol: REST, JSON, `application/scim+json`.
- The IdP is the client. The product, or a sidecar in front of it, is the service provider.
- RFC 7643 predefines `User` and `Group`. A provider may add its own resource types.

`GET /Users?filter=...`	search
`POST /Users`	create
`GET /Users/{id}`	read
`PUT /Users/{id}`	replace
`PATCH /Users/{id}`	update
`DELETE /Users/{id}`	delete
`POST /Bulk`	batch
`POST /.search`	search via body
`GET /ServiceProviderConfig`	features
`GET /ResourceTypes`	resource types
`GET /Schemas`	attributes

A provisioning run

HELSINKI SYSTEMS
Network & IT-Solutions



Before it creates an account, the IdP searches for it.

A user

HELSINKI SYSTEMS
Network & IT-Solutions

```
{
  "schemas": ["urn:ietf:params:scim:schemas:core:2.0:User"],
  "id": "42",
  "externalId": "b1e5c9d2-7f3a-4c1e-9a6b-2d8f0e4c7a15",
  "userName": "jdoe",
  "displayName": "Jane Doe",
  "active": true,
  "emails": [{ "value": "jane.doe@example.com", "type": "work", "primary": true }],
  "groups": [{ "value": "7", "display": "Engineering" }]
}
```

- `id` belongs to the provider, `externalId` to the IdP.
- Most IdPs deprovision with `active: false`, some with `DELETE`.

A PATCH

HELSINKI SYSTEMS
Network & IT-Solutions

```
{
  "schemas": ["urn:ietf:params:scim:api:messages:2.0:PatchOp"],
  "Operations": [
    { "op": "replace", "path": "active", "value": false },
    { "op": "replace", "path": "emails[type eq \"work\"].value",
      "value": "jane@example.org" },
    { "op": "add", "value": { "title": "CTO" } }
  ]
}
```

- Three operations, three path styles, and all three are legal.
- `path` is optional for `add` and `replace`, so a provider has to accept both forms.

Filters, discovery and errors

HELSINKI SYSTEMS
Network & IT-Solutions

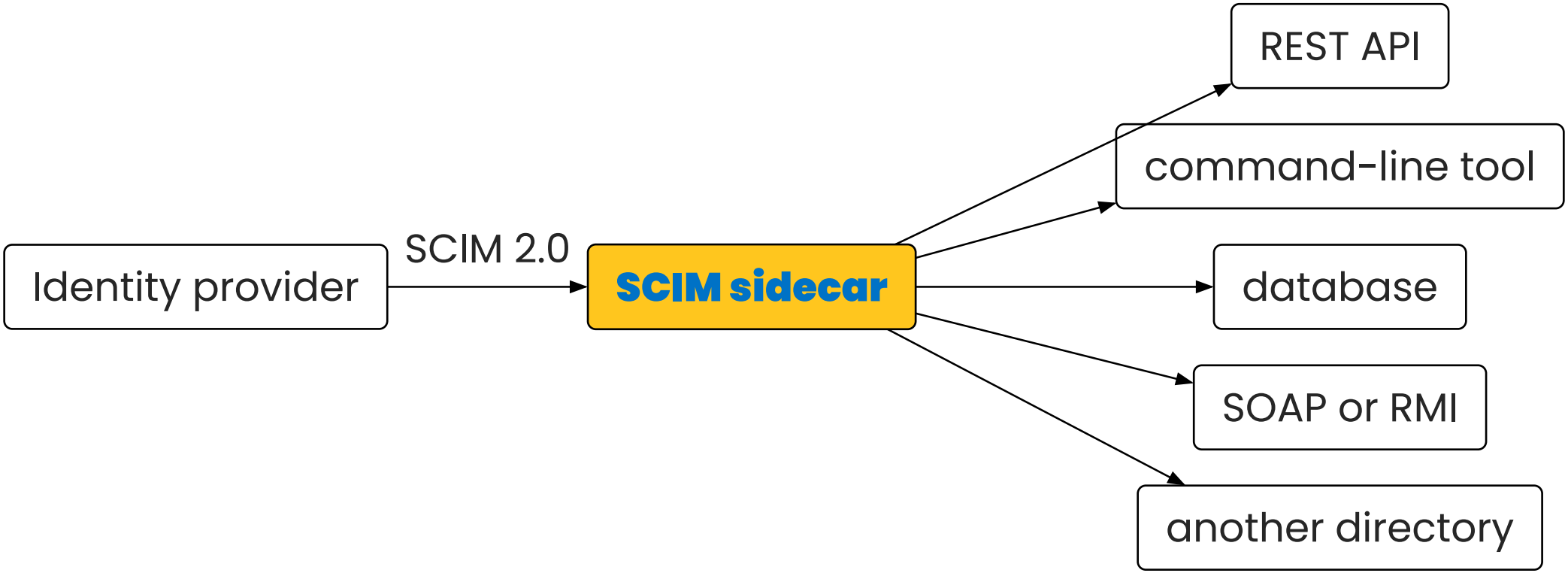
```
userName eq "jdoe"
emails[type eq "work" and value co "@example.com"]
meta.lastModified gt "2026-01-01T00:00:00Z" and not (active eq false)
```

- Attribute names are case-insensitive. String values are too, unless the schema marks the attribute `caseExact`.
- `/ServiceProviderConfig`, `/ResourceTypes` and `/Schemas` tell the IdP what it may send.

```
{
  "schemas": ["urn:…:messages:2.0:Error"],
  "status": "400",
  "scimType": "invalidValue",
  "detail": "Kimai cannot rename a user"
}
```

Put SCIM next to the product

HELSINKI SYSTEMS
Network & IT-Solutions



The IdP sees one standard SCIM endpoint. The sidecar uses whatever interface the product has.

What a sidecar has to do

HELSINKI SYSTEMS
Network & IT-Solutions

The same for every product

- resource schemas and discovery
- filter parsing and matching
- reading PATCH operations
- paging and the `ListResponse` envelope
- Bulk with `bulkId` references
- bearer tokens and content negotiation
- RFC 7644 error documents

Different for every product

- map attributes onto the product's fields
- call the product
- handle what the product cannot do

scimd implements the left column once.

Four crates

HELSINKI SYSTEMS
Network & IT-Solutions

`scimd-types`	the data model, the `Error` a service answers with, the `Resource` trait
`scimd-core`	the registry, dispatch, Bulk, discovery, filters, SCIM metrics
`scimd`	actix-web routes and middleware, mountable into an existing app
`scimd-server`	the daemon: sockets, signals, config, `/health`, `/metrics`

Only `scimd` and `scimd-server` know about a web framework, and only `scimd-server` knows about a process. A host that mounts scimd into its own server compiles none of the daemon.

Defining a resource

HELSINKI SYSTEMS
Network & IT-Solutions

```rust
#[derive(Serialize, Deserialize, Clone)]
struct Device {
    id: Option<Uuid>,
    #[serde(rename = "displayName")]
    display_name: String,
    #[serde(rename = "serialNumber")]
    serial_number: String,
    #[serde(rename = "urn:…:Device")]
    lease: Option<Lease>,
    // schemas, meta
}
```

RFC 7643 treats `User` and `Group` as two resource types among many. scimd does too.

```rust
impl Resource for Device {
    type Id = Uuid;
    const RESOURCE_TYPE: &'static str = "Device";
    const ENDPOINT: &'static str = "/Devices";
    const SCHEMA: &'static str =
        "urn:example:params:scim:schemas:Device";

    fn schema() -> Schema {
        Schema::new(Self::SCHEMA, "Device", "", vec![
            Attributes::new("displayName", "string")
                .required(),
            Attributes::new("serialNumber", "string")
                .required()
                .case_exact(),
        ])
    }
    fn extensions() -> Vec<Extension> {
        vec![Extension::optional(Lease::schema())]
    }
    // id(), meta(), set_meta(): one line each
}
```

Writing the service

HELSINKI SYSTEMS
Network & IT-Solutions

```rust
#[async_trait]
pub trait ResourceService<R: Resource>: Send + Sync {
    async fn list(&self, _filter: FilterQuery) -> Result<ListResponse<R>> {
        Err(Error::MethodNotImplemented)
    }
    async fn create(&self, _resource: R) -> Result<R> { /* 501 */ }
    async fn get(&self, _id: R::Id) -> Result<R> { /* 501 */ }
    async fn replace(&self, _id: R::Id, _resource: R) -> Result<R> { /* 501 */ }
    async fn patch(&self, _id: R::Id, _op: PatchOp) -> Result<ValOrEmpty<R>> { /* 501 */ }
    async fn delete(&self, _id: R::Id) -> Result<()> { /* 501 */ }
}
```

- Every method defaults to `501 Not Implemented`. Override the ones the product supports.
- `R::Id` is the resource's own id type: a `String` for `User`, a `Uuid` for `Device`.

Register it

HELSINKI SYSTEMS
Network & IT-Solutions

```rust
let args = scimd_server::parse_args();
let scim = scimd::ScimBuilder::new()
    .register::<Device, _>(DeviceService::default())
    .with_filter(true, None)
    .with_bulk(true, Some(50), Some(1024 * 1024));

let app = scimd::AppBuilder::new(scim)
    .bearer_token(dotenvy::var("EXAMPLE_SERVICE_BEARER_TOKEN").ok())
    .with_metrics(true)
    .build();

scimd_server::Server::new(app).args(args).run().await?;
```

- `/v2/Devices` exists, and discovery describes `Device` and `Lease`.
- `/ServiceProviderConfig` advertises filter and bulk because the builder enabled them.

Filtering

HELSINKI SYSTEMS
Network & IT-Solutions

```rust
async fn list(&self, query: FilterQuery) -> scimd::Result<ListResponse<Device>> {
    let devices = self.devices().values().cloned().collect();
    scimd::filter::list_response(devices, &query)
}
```

- `list_response` parses the filter, matches it against each resource and pages the result.
- The parser implements the whole grammar of RFC 7644 section 3.4.2.2, `valuePath` brackets included. A parse error answers `400 invalidFilter` with the position where parsing failed.
- Matching honours `caseExact` from the schema, so `serialNumber` compares exactly.
- The AST is public. A backend with its own query language, SQL or LDAP, walks it and filters at the source.

What else scimd handles

HELSINKI SYSTEMS
Network & IT-Solutions

- PATCH bodies parsed into a typed `PatchOp`
- Bulk with `bulkId` references, `failOnErrors` and size limits
- `POST /.search` for one resource type or all of them
- schema extensions, advertised in discovery and added to `schemas` for you
- bearer token authentication and `application/scim+json` negotiation
- RFC 7644 error documents from one `Error` enum
- Prometheus metrics per SCIM operation, not only per HTTP request
- mounting into an existing actix-web app with `configure()`

Bulk with `bulkId`

HELSINKI SYSTEMS
Network & IT-Solutions

```
{
  "schemas": ["urn:ietf:params:scim:api:messages:2.0:BulkRequest"],
  "failOnErrors": 1,
  "Operations": [
    { "method": "POST", "path": "/Devices", "bulkId": "laptop",
      "data": { "schemas": ["urn:example:params:scim:schemas:Device"],
                "displayName": "Jane's laptop", "serialNumber": "C02XK1" } },
    { "method": "PUT", "path": "/Devices/bulkId:laptop",
      "data": { "schemas": ["urn:example:params:scim:schemas:Device"],
                "displayName": "Jane's laptop", "serialNumber": "C02XK1",
                "urn:example:params:scim:schemas:extension:lease:Device":
                  { "leasedUntil": "2027-09-30T00:00:00Z" } } }
  ]
}
```

- Each operation reports its own status. `failOnErrors` sets how many failures stop the batch.
- `bulkId` resolves in request order, so define it before you use it.

Metrics per SCIM operation

HELSINKI SYSTEMS
Network & IT-Solutions

Metric	Labels
`scimd_http_requests_total`	`endpoint`, `method`, `status`
`scimd_http_request_duration_seconds`	`endpoint`, `method`, `status`
`scimd_scim_operations_total`	`resource_type`, `operation`, `source`, `outcome`
`scimd_scim_operation_duration_seconds`	`resource_type`, `operation`
`scimd_bulk_operations_per_request`	
`scimd_registered_resource_types`	

Every label comes from a closed set. `endpoint` is the route pattern (`/v2/Users/{id}`), never a concrete id, so no client request can create a new time series.

Mount into an existing server

HELSINKI SYSTEMS
Network & IT-Solutions

```
let core = scimd::ScimBuilder::new()
    .register::<scimd::models::User, _>(UserService::new())
    .base_path("/scim/v2");
let scim = scimd::AppBuilder::new(core).bearer_token(Some(token)).build();
let scim_routes = scim.configure();

HttpServer::new(move || {
    App::new()
        .service(my_existing_api())
        .configure(scim_routes.clone())
})
```

- `base_path` moves the routes, `meta.location` and the discovery links together.
- Authentication and `Accept` negotiation apply only inside the mount.

Ignoring the filter

HELSINKI SYSTEMS
Network & IT-Solutions

- Before it creates `jdoe`, an IdP asks `GET /Users?filter=userName eq "jdoe"`.
- A provider that ignores `filter` answers with every user it has.
- An IdP that takes the first result now links `jdoe` to someone else's account.

scimd advertises filtering as off by default. While it is off, scimd refuses any request that carries a filter with `invalidFilter`, before the service sees it.

A `400` is better than a wrong answer, because the client cannot detect a wrong answer.

Easy to get wrong

HELSINKI SYSTEMS
Network & IT-Solutions

- `schemas` has to list every schema the body carries. Forget the enterprise extension's URN and IdPs reject the body. scimd adds the URN of every populated extension.
- `USERNAME eq "JDoe"` has to match `"userName": "jdoe"`.
- A missing attribute is not an error. `title eq "CTO"` is false for a user without a title.
- Filters come from the network, and deep nesting means deep recursion. scimd caps the length before it parses and the depth while it parses.

Unsupported attributes and changes

HELSINKI SYSTEMS
Network & IT-Solutions

- IdPs send attributes the product has no field for. Drop those and log them. Failing the whole PATCH would stop provisioning over a field nobody stores.
- Refuse a change the product cannot make. Ignoring a rename leaves the IdP believing it happened.
- Some IdPs deprovision with `DELETE`, some with `active: false`. Either one has to end the person's access.
- `/ServiceProviderConfig` should advertise only what the service implements.

kimaid

HELSINKI SYSTEMS
Network & IT-Solutions

- Kimai is an open source time tracker with a REST API for users and teams, and no SCIM.
- kimaid runs next to Kimai and talks to that API with an API token.
- SCIM `/Users` are Kimai users. SCIM `/Groups` are Kimai teams.

```
let scim = scimd::ScimBuilder::new()
    .register::<scimd::models::User, _>(users::Users::new(kimai.clone(), defaults))
    .register::<scimd::models::Group, _>(groups::Groups::new(kimai))
    .with_filter(true, None)
    .with_patch(true);
```

The rest is configuration: Kimai's URL and API token, the SCIM bearer token and defaults for new users.

What Kimai can’t do

- **No delete endpoint.** `DELETE` deactivates, the same as `active: false`. Repeating it changes nothing.
- **No username in the edit form.** A rename answers `400 invalidValue`.
- **Nowhere to store `externalId`.** The IdP finds users again with `userName eq`.
- **No paging or filtering.** kimaid hands the whole collection to `list_response`.

```
async fn delete(&self, id: String) -> Result<()> {
    let id = map::parse_id(&id)?;
    let (_, mut form) = self.form_for(id).await?;
    if !form.enabled {
        return Ok(());
    }
    form.enabled = false;
    self.kimai.patch_user(id, &form).await?;
    Ok(())
}
```

kimaid's PATCH handler

HELSINKI SYSTEMS
Network & IT-Solutions

```
for operation in &op.operations {
    match operation.attr().as_str() {
        "username" => {
            map::refuse_rename(&current.username, operation.str_value().unwrap_or_default())?;
        }
        "active" => set(&mut form.enabled, operation.bool_value()),
        "emails" => set(&mut form.email, operation.email_address()),
        "displayname" => form.alias = patch::scalar(operation),
        "timezone" => set(&mut form.timezone, operation.str_value()),
        // title, preferredlanguage and locale follow the same pattern.
        // An identity provider sends attributes Kimai has no field for.
        other => tracing::debug!("ignoring patch on '{other}' for user {id}"),
    }
}
```

- `attr()` strips the schema URN, `valuePath` brackets and sub-attributes, and lowercases the result.

Auditing through the sidecar

HELSINKI SYSTEMS
Network & IT-Solutions

- Okta's user import pages through `GET /Users` and `GET /Groups` and compares the result with its own directory.
- For the audit, the sidecar only needs `list`. kimaid's is two lines.
- Kimai has nowhere to store `externalId`, so the IdP matches on `userName`.

In Kimai	In the IdP	Finding
`mmuster`, active	no such person	created by hand, outside provisioning
`jdoe`, active	left in June	access that should have ended
`asmith`, no team	in Engineering	membership drifted

The IdP finds accounts that someone created by hand or that provisioning missed.

Demo: kimaid in front of Kimai

HELSINKI SYSTEMS
Network & IT-Solutions

1. `GET /v2/ServiceProviderConfig` and `GET /v2/ResourceTypes`
2. `POST /v2/Users` creates `jdoe`
3. `PATCH /v2/Groups/7` adds `jdoe` to team Engineering
4. `jdoe` appears in Kimai's user list, in Engineering
5. `PATCH active: false` deactivates `jdoe` in Kimai
6. `PATCH userName` answers `400 invalidValue`

If the demo fails, the next three slides show steps 1 to 3, 5 and 6.

Demo: discovery

HELSINKI SYSTEMS
Network & IT-Solutions

```
curl -H "Authorization: Bearer $TOKEN" http://127.0.0.1:5001/v2/ResourceTypes
```


```
{
  "schemas": ["urn:ietf:params:scim:api:messages:2.0:ListResponse"],
  "totalResults": 2,
  "Resources": [
    { "id": "Group", "name": "Group", "endpoint": "/Groups",
      "schema": "urn:ietf:params:scim:schemas:core:2.0:Group" },
    { "id": "User", "name": "User", "endpoint": "/Users",
      "schema": "urn:ietf:params:scim:schemas:core:2.0:User",
      "schemaExtensions": [{
        "schema": "urn:ietf:params:scim:schemas:extension:enterprise:2.0:User",
        "required": false }] }
  ]
}
```

Demo: create and add to a team

HELSINKI SYSTEMS
Network & IT-Solutions

```
curl -X POST -H "Authorization: Bearer $TOKEN" -H "Content-Type: application/scim+json" \
  http://127.0.0.1:5001/v2/Users -d '{
    "schemas": ["urn:ietf:params:scim:schemas:core:2.0:User"],
    "userName": "jdoe", "displayName": "Jane Doe",
    "emails": [{ "value": "jane.doe@example.com", "primary": true }] }'
```

```
{
  "schemas": ["urn:ietf:params:scim:schemas:core:2.0:User"],
  "id": "42", "userName": "jdoe", "displayName": "Jane Doe", "active": true,
  "preferredLanguage": "de", "locale": "de_DE", "timezone": "Europe/Berlin",
  "emails": [{ "value": "jane.doe@example.com", "type": "work", "primary": true }],
  "meta": { "resourceType": "User", "location": "/v2/Users/42" }
}
```

The User schema marks `groups` read-only, so membership changes go through the group:

```
curl -X PATCH ... http://127.0.0.1:5001/v2/Groups/7 -d '{ ..., "Operations": [
    { "op": "add", "path": "members", "value": [{ "value": "42" }] }] }'
```

Demo: deactivate and rename

HELSINKI SYSTEMS
Network & IT-Solutions

```
curl -X PATCH -H "Authorization: Bearer $TOKEN" -H "Content-Type: application/scim+json" \
  http://127.0.0.1:5001/v2/Users/42 -d '{
    "schemas": ["urn:ietf:params:scim:api:messages:2.0:PatchOp"],
    "Operations": [{ "op": "replace", "path": "active", "value": false }] }'
```

```
{ "id": "42", "userName": "jdoe", "active": false, ... }
```

```
curl -X PATCH ... -d '{ ..., "Operations": [
    { "op": "replace", "path": "userName", "value": "jane" }] }'
```

```
{
  "schemas": ["urn:ietf:params:scim:api:messages:2.0:Error"],
  "status": "400", "scimType": "invalidValue",
  "detail": "Kimai cannot rename a user: 'jdoe' cannot become 'jane'"
}
```

Renaming a mailbox

HELSINKI SYSTEMS
Network & IT-Solutions

- Dovecot names a mailbox after the address. After an address change, the next login creates a new, empty mailbox, and the old mail stays behind.
- An admin has to move the mailbox before that first login:

```
doveadm kick "oldname@example.org"      # on every frontend
mv /var/lib/imap/example.org/private/oldname /var/lib/imap/example.org/private/newname
```

- `doveadm kick` only disconnects sessions on its own host. Skip one frontend and a session there survives the rename.

doveadmd

HELSINKI SYSTEMS
Network & IT-Solutions

- doveadmd is a SCIM sidecar on the mail backend. It lists and reads users and renames mailboxes on `PUT` and `PATCH`.
- The kicks go to every frontend over the doveadm HTTP API. The move is a local `rename(2)`, which is atomic and keeps file ownership.
- The order is fixed: kick the old and the new address everywhere, rename, kick again, check that nothing reappeared.
- SQLite maps the SCIM `id` to the mailbox's current address. An unknown `id` on `PUT` is mapped to the address it carries, so doveadmd works against an existing mail store.

Checks before the rename

- Renaming onto an existing mailbox is a `409`. doveadmd never merges mail.
- A `userName` and primary email that disagree are a `400`.
- An unsafe address is a `400` from `Address::parse`.
- `POST` and `DELETE` answer `501`. Provisioning does not delete mail.
- doveadmd writes a journal row before the first kick. `dry_run` touches no mail.

```rust
pub struct Address {
    localpart: String,
    domain: String,
}

impl Address {
    /// Refuses `/`, `\`, a leading dot, controls.
    pub fn parse(address: &str) -> Result<Self>;
    /// `<root>/<domain>/private/<localpart>`
    pub fn path(&self, root: &Path) -> PathBuf;
}
```

Creating and deleting mailboxes

```rust
#[derive(Serialize, Deserialize, Clone)]
struct MailUser {
    #[serde(flatten)]
    user: User,
    #[serde(rename = "urn:…:mailbox:User")]
    mailbox: Option<Mailbox>,
}

impl Resource for MailUser {
    // User's endpoint and schema, plus:
    fn extensions() -> Vec<Extension> {
        let mut extensions = User::extensions();
        extensions.push(
            Extension::optional(Mailbox::schema()));
        extensions
    }
}
```

`POST`	creates the mailbox
`active: false`	suspends it
`DELETE`	removes it
`…:quota`	sets its quota

doveadmd only renames. A draft in `example/src/mail.rs` covers the rest of the lifecycle without a new resource type: the mailbox follows the user.

Mailboxes in `example`

HELSINKI SYSTEMS
Network & IT-Solutions

```
curl -X POST ... http://127.0.0.1:5000/v2/Users -d '{
    "schemas": ["urn:ietf:params:scim:schemas:core:2.0:User"], "userName": "jdoe",
    "urn:example:params:scim:schemas:extension:mailbox:User": { "quota": 1024 } }'
curl -X PATCH ... http://127.0.0.1:5000/v2/Users/$ID -d '{ ..., "Operations": [
    { "op": "replace", "value": { "active": false } }] }'
curl -X PATCH ... http://127.0.0.1:5000/v2/Users/$ID -d '{ ..., "Operations": [
    { "op": "replace", "path": "urn:example:params:scim:schemas:extension:mailbox:User:quota",
      "value": 2048 }] }'
curl -X DELETE ... http://127.0.0.1:5000/v2/Users/$ID
```

```
INFO example::mail: created mailbox for jdoe
INFO example::mail: suspended mailbox of jdoe
INFO example::mail: deleted mailbox of jdoe
```

- The create response lists the mailbox URN in `schemas`. scimd adds it because the extension is populated.

kimaid and doveadmd

HELSINKI SYSTEMS
Network & IT-Solutions

	kimaid	doveadmd
reaches the product via	Kimai's REST API	doveadm HTTP API and `rename(2)`
resource types	`User`, `Group`	`User`
implements	list, get, create, replace, patch, delete	list, get, replace, patch
`DELETE` means	deactivate	`501`
runs	next to Kimai	on the mail backend

Takeaways

HELSINKI SYSTEMS
Network & IT-Solutions

- A product without SCIM is a deployment problem you can solve yourself, with one sidecar per product.
- The protocol work is the same every time. scimd does it once.
- The product-specific part is small. Most of the work is deciding what to refuse and making deprovisioning safe to repeat.

github.com/helsinki-systems/scimd, AGPL

```
export EXAMPLE_SERVICE_BEARER_TOKEN=secret
cargo run -p example -- --bind 127.0.0.1:5000   # mail users and the custom Device
curl -H "Authorization: Bearer secret" http://127.0.0.1:5000/v2/ResourceTypes
```

Questions?

Simon Hauser
Email simon.hauser@helsinki-systems.de
Matrix @simon.hauser:helsinki-systems.de

helsinki-systems.de

github.com/helsinki-systems/scimd