



Solr-Rollenaktualisierung: Benutzerverwaltung und Bugfix

Einleitung

Im Rahmen dieser Anpassungen wurde die bestehende Ansible-Rolle für Solr um ein flexibles Benutzer-Management erweitert und ein Fehler in den Integrationstests behoben. Ziel war es, die Rolle mandantenfähig zu machen, zusätzliche Nutzer mit eigenen Rollen zu unterstützen und gleichzeitig auf die offiziellen Berechtigungsnamen der Solr-Dokumentation zu achten ¹.

Zusammenfassung der Änderungen

- **Bugfix in den Integrationstests:** Der Status des Docker-Containers wird jetzt mit dem Jinja-Filter `trim` ausgewertet. Dadurch werden Zeilenumbrüche entfernt und der Status „running“ korrekt erkannt. Dies verhindert falsch negative Testergebnisse.
- **Neues Benutzer-Management:** Über die Variable `solr_additional_users` können nun beliebig viele zusätzliche Nutzer definiert werden. Für jeden Core wird automatisch eine Administratorrolle erstellt (`core-admin-<core>`), falls keine eigenen Rollen angegeben werden. Passwörter werden mit doppeltem SHA256 und Salt gesichert.
- **Aktualisiertes Security-Template:** Veraltete Berechtigungen wie `delete` und `backup` wurden entfernt. Stattdessen werden die offiziellen Berechtigungen der Rule-Based Authorization Plugins verwendet ¹. Für jeden Core gibt es drei neue Rollen: eine Vollzugriffs-Rolle für Core-Admins, eine Lese/Schreib-Rolle für Kunden sowie eine reine Lese-Rolle für Support.
- **Neue Default-Variablen:** In den Defaults wurden die Variablen `solr_additional_users` und `solr_core_admin_role_prefix` eingeführt, die standardmäßig eine leere Liste bzw. den Prefix „core-admin“ setzen.
- **Optionale Web-UI:** Für eine benutzerfreundliche Verwaltung wurde ein Konzept für eine Web-Oberfläche skizziert, die neue Benutzer anlegen, Rollen zuweisen, Nutzer sperren und die Änderungen per Ansible-Lauf synchronisieren kann.

Geänderte Dateien

defaults/main.yml

Die Datei definiert nun zwei neue Variablen und enthält einen Kopfkommentar mit maximal fünf Unterpunkten, die die Verbesserungen erläutern. Die Variablen ermöglichen die Definition zusätzlicher Nutzer und die Konfiguration des Rollenpräfixes:

```
#####
# Default variable definitions for Solr role enhancements.
# Fixes/Improvements:
# 1. Added ``solr_additional_users`` to allow provisioning of extra Solr
users.
```

```

# 2. Added ``solr_core_admin_role_prefix`` to configure per-core
administrator role names.
# 3. Provided documentation comments for each variable to aid understanding.
# 4. Ensured sensible defaults (empty list and "core-admin") for backward
compatibility.
# 5. Maintains existing behaviour for pre-defined users while enabling
extensibility.
#####
---
# Minimal defaults overridden for dynamic user management

solr_additional_users: []
solr_core_admin_role_prefix: "core-admin"

```

tasks/integration_tests.yml

Die Integrationstests wurden um eine Versionserhöhung auf 1.2.0 ergänzt und der Kopfkommentar fasst die vorgenommenen Korrekturen zusammen. Der entscheidende Fix ist das Trimmen des Container-Status:

```

---
# Version: 1.2.0
# Description: Integration tests for Solr installation
# Fixes/Improvements:
# 1. Trimmed container status output using ``| trim`` to avoid false
negatives when checking ``running`` state.
# 2. Clarified comments around test result summary for maintainability.
# 3. Maintained existing tests (authentication, core operations, performance,
security) for backward compatibility.
# 4. Added version bump to indicate changes.
# 5. Provided this header to summarise modifications for future contributors.

```

Die Logik des Tests bleibt unverändert; lediglich der Vergleich `container_status.stdout | trim == 'running'` stellt sicher, dass der Rückgabewert korrekt ausgewertet wird.

tasks/user_management.yml

Für die dynamische Verwaltung neuer Nutzer wurde dieses neue Taskfile erstellt. Es verfügt über einen umfassenden Kopfkommentar und verarbeitet zusätzliche Benutzer, generiert sichere Hashes und erstellt Rollen-Mappings:

```

---
# Version: 1.1.0
# Description: Manage dynamic Solr users beyond the built-in admin/support/
customer.
# Fixes/Improvements:
# 1. Introduced support for additional users via ``solr_additional_users``
and automatic per-core admin roles.

```

```

# 2. Securely generates double SHA256 hashed credentials with random salts.
# 3. Assigns default or custom roles to new users and cleans up temporary
files.
# 4. Exposes user and role mappings as Ansible facts for integration with
``security.json``.
# 5. Provides warnings for incomplete user definitions to aid
troubleshooting.

```

Im weiteren Verlauf generiert das Taskfile zufällige Salts, hashte Passwörter doppelt mit SHA256 und legt zwei Facts an (`solr_additional_user_hashes`, `solr_additional_user_roles`), die beim Rendern von `security.json` verwendet werden.

templates/security.json.j2

Die Vorlage für die Solr-Sicherheitskonfiguration wurde mit einem ausführlichen Jinja-Kommentar versehen. Darin werden die wichtigsten Änderungen erläutert, insbesondere die Verwendung offizieller Berechtigungen und die Einführung von per-Core-Rollen:

```

{#
  Security configuration template for Solr.
  Fixes/Improvements:
  1. Updated permission names to align with the official Solr rule-based
  authorization plugin; removed deprecated ones (e.g. delete/backup).
  2. Added per-core role definitions (admin, customer, support) to enable
  fine-grained access control to individual cores.
  3. Integrated dynamic additional users via ``solr_additional_users`` and
  corresponding hashed credentials.
  4. Documented variable usage and structure to aid maintainers and promote
  clarity.
  5. Ensured backward compatibility by preserving existing built-in roles and
  credentials.
#}
{
  "authentication": {
    "blockUnknown": true,
    "class": "solr.BasicAuthPlugin",
    "credentials": {
      "{{ solr_admin_user | default('admin') }}": "{{ admin_password_hash }}",
      "{{ solr_support_user | default('support') }}": "{{ support_password_hash }}",
      "{{ solr_customer_user | default('customer') }}": "{{ customer_password_hash }}"
        if solr_additional_user_hashes is defined and
        solr_additional_user_hashes|length > 0 %}, {% endif %}
      {% if solr_additional_user_hashes is defined %}
      {% for username, hash in solr_additional_user_hashes.items() %}
        "{{ username }}": "{{ hash }}"{% if not loop.last %}, {% endif %}
      {% endfor %}
      {% endif %}
    }
  }
}

```

```

        },
        "realm": "{{ solr_auth_realm | default('Solr Eledia') }}",
        "forwardCredentials": {{ solr_forward_credentials | default('false') }} | lower
    }
},
"authorization": {
    "class": "solr.RuleBasedAuthorizationPlugin",
    "permissions": [
        { "name": "all", "role": "admin" },
        { "name": "security-read", "role": "admin" },
        { "name": "security-edit", "role": "admin" },
        { "name": "schema-read", "role": "admin" },
        { "name": "schema-edit", "role": "admin" },
        { "name": "config-read", "role": ["admin", "support"] },
        { "name": "config-edit", "role": "admin" },
        { "name": "core-admin-read", "role": "admin" },
        { "name": "core-admin-edit", "role": "admin" },
        { "name": "collection-admin-read", "role": "admin" },
        { "name": "collection-admin-edit", "role": "admin" },
        { "name": "metrics-read", "role": ["admin", "support"] },
        { "name": "health", "role": ["admin", "support"] },
        /*
         * Custom per-core permissions to provide fine-grained access on a
         single core.
        */
        { "name": "core-{{ solr_core_name }}-admin", "collection": "{{ solr_core_name }}", "path": ["/*"], "method": ["GET", "HEAD", "POST", "PUT", "DELETE"], "role": ["{{ solr_core_admin_role }} | default(solr_core_admin_role_prefix ~ '-' ~ solr_core_name)"] },
        { "name": "core-{{ solr_core_name }}-customer", "collection": "{{ solr_core_name }}", "path": ["/*"], "method": ["GET", "HEAD", "POST", "DELETE"], "role": ["customer"] },
        { "name": "core-{{ solr_core_name }}-support", "collection": "{{ solr_core_name }}", "path": ["/*"], "method": ["GET", "HEAD"], "role": ["support"] },
    ],
    "user-role": {
        "{{ solr_admin_user | default('admin') }}": ["admin"],
        "{{ solr_support_user | default('support') }}": ["support"],
        "{{ solr_customer_user | default('customer') }}": ["customer"]
        {% if solr_additional_user_roles is defined and solr_additional_user_roles|length > 0 %}
        {% endif %}
        {% if solr_additional_user_roles is defined %}
        {% for username, roles in solr_additional_user_roles.items() %}
            ,{{ username }}: {{ roles | to_json }}{% if not loop.last %},{% endif %}
        {% endfor %}
        {% endif %}
    }
}
}

```

Hinweise zur Nutzung

1. **Definition zusätzlicher Nutzer:** Legen Sie in Ihrer Inventardatei `solr_additional_users` als Liste von Dikt-Objekten an. Jeder Eintrag benötigt einen `username` und ein `password`; optional kann ein Feld `roles` angegeben werden. Wird `roles` ausgelassen, erhält der Benutzer automatisch die per-Core-Adminrolle.
2. **Einbindung der Tasks:** Binden Sie das Taskfile `tasks/user_management.yml` vor dem Rendern von `security.json.j2` ein. Dadurch stehen alle generierten Variablen bereit, wenn das Sicherheits-Template verarbeitet wird.
3. **Neu erstellte Rollen:** Verwenden Sie nur die in der Dokumentation vorgesehenen Berechtigungsnamen wie `config-read`, `security-edit` und `health`¹. Die veralteten Berechtigungen `delete`, `backup` etc. sind entfernt.
4. **Playbook ausführen:** Nach Anpassung Ihrer Variablen führen Sie das Playbook wie gewohnt aus. Die Rolle kopiert die aktualisierte `security.json` in den Solr-Container und aktualisiert die Konfiguration mittels API, ohne einen Neustart zu benötigen.
5. **Weiterentwicklung:** Für zukünftige Erweiterungen bietet sich eine Web-UI an, über die Administratoren Benutzer anlegen, sperren und Rollen verwalten können. Diese UI würde letztlich die Variablen in `solr_additional_users` anpassen und das Ansible-Playbook erneut ausführen.

Fazit

Durch das Zusammenspiel der neuen Variablen, des Benutzer-Management-Taskfiles und des aktualisierten Security-Templates ist die Solr-Rolle jetzt mandantenfähig und bietet ein fein granulares Rechtemodell. Die Änderungen orientieren sich strikt an der offiziellen Solr-Dokumentation und entfernen veraltete Berechtigungen. Der behobene Bug in den Integrationstests sorgt zudem für verlässliche Testausführungen.

¹ Rule-Based Authorization Plugins :: Apache Solr Reference Guide

<https://solr.apache.org/guide/solr/latest/deployment-guide/rule-based-authorization-plugin.html>