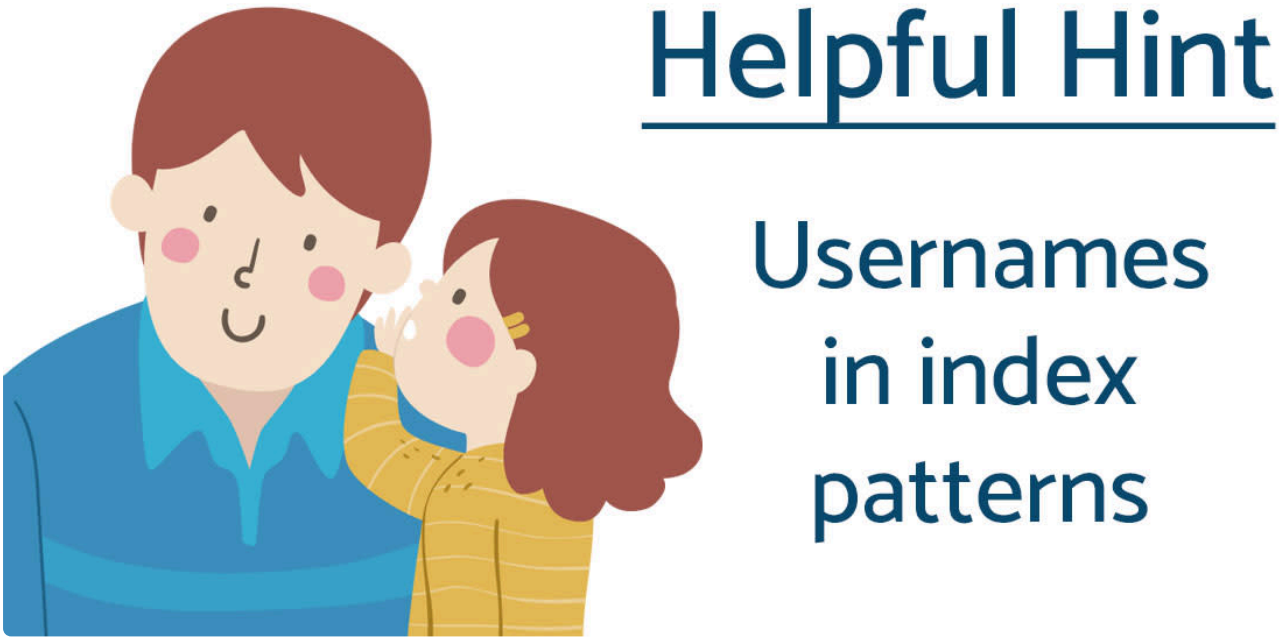
Helpful Hint: Username substitution in index patterns

Jochen Kressin · 2023-06-22



Welcome to another Helpful Hint post on mastering the Search Guard. Today, we're focusing on creating a Search Guard role that grants full access to an index that carries the same name as the user. This level of customization is highly beneficial for those environments where you want to restrict users to only their indexes in a multi-user Elasticsearch setup.

To fully comprehend this article, it is crucial to understand what variable substitution is, and how it applies to Search Guard. Let's start there.

Understanding Variable Substitution

Variable substitution, in the most general sense, is a process where specific placeholders are replaced by their corresponding values during the execution time. These placeholders are typically defined within the code, configuration files, or scripts and are intended to make these elements dynamic. This ability to change the value of a placeholder at runtime can add flexibility to your systems and software.

In the context of Search Guard, variable substitution allows for the dynamic configuration of various Search Guard features. For instance, you could use variables in index patterns to grant a user access to indices that match or begin with their username.

Search Guard ships with a powerful feature for implementing attribute-based security. This means you can use attributes of a user, like the username, and use it when defining roles and permission. This involves mapping these attributed to variables which you can then use in your configuration files.

However, we also ship with pre-defined variables to make your life easier. In this example we will use the variable *${user.name}*, which, unsurprisingly resolves to the current users name.

Configuring Search Guard Role

Let's put the theory into practice and create a role that permits users to access their indices. For example, user "John" will have access to the "john" index, and user "Alice" will have access to the "alice" index.

Step 1: Define the Role

In the *sg_roles.yml* file, create a new role. We'll name it "user_index_role" for this example. The structure of the role definition should be as follows:

```
user_index_role:
  cluster_permissions:
    - SGS_CLUSTER_COMPOSITE_OPS
  index_permissions:
    - index_patterns:
        - "${user.name}"
      allowed_actions:
        - SGS_ALL_ACCESS
```

Note that instead of defining a concrete index pattern, we used the ${user.name} variable. So, when John logs in, this will resolve to the corresponding username "john". For Alice, it resolves to "alice".

You can of course also use this variable in an index pattern. Say you have multiple indices for each user, like:

```
john-logs
john-data
john-tmp
```

A corresponding role definition would look like:

```
user_index_role:
  cluster_permissions:
    - SGS_CLUSTER_COMPOSITE_OPS
  index_permissions:
    - index_patterns:
        - "${user.name}-logs"
        - "${user.name}-data"
        - "${user.name}-tmp"
      allowed_actions:
        - SGS_ALL_ACCESS
```

And you can also combine variable substitutions with wildcards and regular expression. For example, to grant access to **any** index that starts with the username followed by a dash, use this role definition:

```
user_index_role:
  cluster_permissions:
    - SGS_CLUSTER_COMPOSITE_OPS
  index_permissions:
    - index_patterns:
        - "${user.name}-*"
      allowed_actions:
        - SGS_ALL_ACCESS
```

Here we have combined the username variable with a wildcard when defining the index pattern.

Step 2: Map Users to the Role

The next step involves mapping the relevant users to the role you've just defined. In our example, we want to assign any user to the user*index*role, so also here we use a wildcard for the role mapping in *sgrolesmapping.yml*:

```
user_index_role:
  users:
    - "*"
```

When applying this mapping, all users are automatically assigned to the user_index_role.

That's it! With these changes, each user will only have access to an index with their username. This configuration is especially useful for scenarios where you need to isolate user data while still maintaining flexibility and simplicity in role and permission management.

Conclusion

Search Guard offers an incredibly dynamic and flexible way of managing roles and permissions within Elasticsearch and Kibana. Leveraging the power of variable substitution, it's possible to build user-specific roles that align with your organization's security requirements.

Remember to always test your configuration changes before deploying them to production to ensure the security and integrity of your data. Stay tuned for more insightful posts on mastering Search Guard!

This article was published on the Search Guard blog: https://search-guard.com/blog/elasticsearch-variable-substitution-in-index-patterns/