
ConfigHub Documentation

Release v1.2.0

ConfigHub Support

Oct 25, 2017

Overview and Installation

1	ConfigHub overview	3
1.1	Context properties	3
1.2	Context resolution	4
2	Installation and setup	7
2.1	Installation	7
2.2	Create personal account	9
2.3	Create an organization	9
2.4	Create ConfigHub repository	10
3	PULL	11
3.1	Usage	11
3.2	Request Headers	12
4	PUSH	15
4.1	Usage	15
4.2	Request Headers	16
4.3	JSON File Format	17
5	RAW FILE	19
5.1	Usage	19
5.2	Request Headers	20
6	INFO	21
6.1	Usage	21
6.2	Request Headers	22
7	Repositories	25
7.1	Usage	25
7.2	Request Headers	26
8	System Status	27
8.1	Usage	27
8.2	Request Headers	28
9	Overview	29
9.1	Security groups	29
9.2	Securing files and properties	30

10 Properties editor	33
10.1 Editor toolbar	33
10.2 Create a new property	35
10.3 Add value to existing key	36
10.4 Editing keys and values	36

ConfigHub is a platform for software configuration. Configuration is managed programmatically or through the web-based user interface.

CHAPTER 1

ConfigHub overview

All configuration can be boiled down to key-value pairs (properties). Ignoring the format that surrounds various configuration components, configuration differences are always reduced to properties.

For example, in **Development** environment, a config file may contain a line like:

```
<Connector port="8080" redirectPort="8443"/>
```

While in **Production** the same line looks like this:

```
<Connector port="80" redirectPort="443"/>
```

Therefore, the merged configuration could be written as:

```
<Connector port="${ http.port }" redirectPort="${ http.redirect }"/>
```

And we have our two properties: `http.port` and `http.redirect`.

Context properties

In order to eliminate a mesh of configuration file and property duplication, ConfigHub changes the definition of a property. By assigning a context to a property value, a single property key can have multiple values, each with a unique context signature.

Note:

Context Property property = key: [context_1: value, context_2: value, ...]

When an application/client requests configuration, they only need to specify their context. Using a request context, the exact key-value pairing occurs, and the result is returned to the client.

Using example above, a request context `Production` would return:

```
http.port: 80
http.redirect: 443
```

While a request with context `Development` would return:

```
http.port: 8080
http.redirect: 8443
```

In this example, context is very simple - its composed with a single context hierarchy, `Environment`. However, context can be as complex as your environment demands - up to 10 context element hierarchy.

Context resolution

Context resolution is a process during which value-context of each key is compared to the request context in order to determine which properties should be returned.

Matching value to request context occurs in two steps:

1. Semantic Filter

For each context hierarchy, corresponding context-values from request and property are compared. For a match, corresponding context values have to satisfy following rules:

- If both are specified, they have to be the same;
- Either or both are a wildcard.

Example: Context-Request resolution

Assume a context property is defined with for a key `logger.level` with 4 values.

	Environment	Application	Instance	
Request-Context	Production	WebServer	Webserver-Jim	

Value-Context	*	*	Webserver-Jim	Match
Value-Context	Production	WebServer	*	Match
Value-Context	Production	*	*	Match
Value-Context	Development	*	*	No Match

The semantic filter has matched 3 values, and ignored a single value because **Environment** context hierarchy from Request-Context “Production” did not match “Development”.

2. Weight Filter

Weighted filter is only applied if Context-Request is fully-qualified (each context hierarchy is specified).

As repository’s context scope can vary in size (see Choosing Repository Context Scope), each of the context blocks is assigned specific weight. The widest scope specifications (left) carry less weight, while most specific parts (right) carry most weight.

For example, in a repository with 3 context hierarchies, weight is assigned as follows:

```
Environment [40] | Application [80] | Instance [160]
```


This repository might have a property defined with multiple values. Each value-context also has weight.

Example: Fully-Specified Request-Context resolution

	Environment	Application	Instance	Weight	
Value-Context	*	*	Webserver-Jim	160	Match
Value-Context	Production	WebServer	*	$40 + 80 = 120$	
Value-Context	Production	*	*	40	

The value with the highest *weight* is matched, as it is the most relevant value for the given context request.

Here's the ConfigHub property editor view of the same property - with the values expanded.

Environment / new

Production *

Application / new

WebServer *

Instance / new

WebServer-Jim *

Q All

keys, comments and values

+ New

All

↓ 2

<

>

logger.level

> DEBUG

*

> *

> WebServer-jim

Resolved

> ERROR

Production

> WebServer

> *

Overridden

> INFO

Production

> *

> *

Overridden

> DEBUG

Development

> *

> *

Out of context

Installation and setup

The fastest way to evaluate ConfigHub is to use the [hosted demo version](#).

Installation

System requirements

The ConfigHub server application has the following prerequisites:

- Some modern Linux distribution (Debian Linux, Ubuntu Linux, or CentOS recommended)
- MySQL 5 or PostgreSQL 9 or later (latest stable version is recommended)
- Oracle Java SE 8 or later (latest stable update is recommended)

Download and install

- Download and install Java8 to your localhost. Set the `JAVA_HOME` environment variable to Java's bin directory:

```
export JAVA_HOME=/path/to/java8/bin
```

- [Download](#) the latest version of ConfigHub:
- Uncompress the downloaded file:

```
tar -xvzf confighub-<version>.tar.gz
```

ConfigHub service configuration

There are two configuration sections:

1. Database connections

2. Server configuration

- Database connections

Database connections are configured in `confighub-<version>/server/conf/tomee.xml` file. ConfigHub uses 2 databases, one for storage of all repository and user related data, and the other for the storage of all client API requests.

As of version v1.6, ConfigHub supports MySQL and PostgreSQL databases. Here's an example of a database configuration:

```
<?xml version="1.0" encoding="UTF-8"?>
<tomee>

  <Resource id="ConfigHubMainDS" type="DataSource">
    JdbcDriver = org.postgresql.Driver
    JdbcUrl = jdbc:postgresql://127.0.0.1:5432/ConfigHubMain
    UserName = username
    Password = password

    JtaManaged = false
    validationQuery="SELECT 1"
    maxWaitTime = 2 seconds
    maxActive = 200
  </Resource>

  <Resource id="ConfigHubApiRequestsDS" type="DataSource">
    JdbcDriver = com.mysql.jdbc.Driver
    JdbcUrl = jdbc:mysql://127.0.0.1:3306/ConfigHubClientRequests
    UserName = username
    Password = password

    JtaManaged = false
    validationQuery="SELECT 1"
    maxWaitTime = 2 seconds
    maxActive = 200
  </Resource>

</tomee>
```

The resource IDs `ConfigHubMainDS` and `ConfigHubApiRequestsDS` as well as parameter `JtaManaged = false` have to remain unchanged. The rest of the datasource definition can be modified to your specific needs.

Note: For the rest of the optional parameters, please consult [Tomee documentation](#)

- Server configuration

Edit the configuration file `confighub.sh` or `confighub.bat` for Windows installations in `confighub-<version>` directory. Each configuration parameter has to be specified.

```
# Memory assigned to the ConfigHub service. It is recommended to assign 4g or more.
export ALLOCATED_MEMORY=4g

# HTTP and HTTPS ports
export HTTP_PORT=80
export HTTPS_PORT=443

# Path to the location where all ConfigHub service logs are stored.
```

```
export LOG_PATH=/var/log/confighub

# Specify an override to the default self-signed certificate/keystore.
export KEYSTORE_FILE=cert/confighub_default.jks
export KEYSTORE_ALIAS=confighub
export KEYSTORE_PASSWORD=confighub
```

Starting and stopping ConfigHub service

- Start ConfigHub:

```
confighub-<version>/server/bin/startup.sh
```

- Stop ConfigHub:

```
confighub-<version>/server/bin/shutdown.sh
```

Note: If you are running ConfigHub on a reserved port (i.e. 80, and 443), use root access (or `sudo`).

Create personal account

To create a personal account click on the **Sign Up** button (top-right) of the browser.

Note: In the near future, we will support LDAP integration.

Create an organization

Organizations are a better way to share management responsibilities for your repositories. Some aspects of repository management are limited to repository owners, or admins - for example, deleting tokens, changing repository settings, adding/removing team members as well as access permissions.

To distribute this workload, organization let's you assign repository owners and administrators.

To create a new organization, once logged in:

1. Click on the **New** link (top-right) and choose *Organization*
2. Enter the Organization name, and save.

Once the organization is created, you can create repositories and add users as owners and administrators.

Role	Access Description
Owner	Can manage every aspect of the organization and grant administrative access to other ConfigHub users. Only owners can delete repositories.
Adminis- trator	Can manage repositories, teams, access control and can add other ConfigHub users to the repositories.

Create ConfigHub repository

Location where all configuration is kept. Much like a repository for code, ConfigHub repository is a storage for configuration.

A repository can be owned by either a personal account, or an organization.

To create a new repository, once logged in:

1. Click on the **New** link (top-right) and choose *Repository*
2. Choose the owner account and enter the repository name. *Description is optional*
3. Choose and label your context, and save.

Choosing repository context

When deciding on a repository context, you are essentially modeling your environments.

There are several factors you will need to consider when choosing the context scope for your repository. Are you running multiple independent projects, do you have different environments, multiple applications and do they run several instances?

Even though context scope can always be changed, getting it right the first time will save you some time in the long run.

To help you determine the right scope, answering these questions might help:

- Do you have different environments (i.e. Development, Production, Test, etc.)?
- Do you have multiple applications?
- Are you running multiple instances (with different configuration) of your applications?

Each time you answered yes, your context scope grew by factor of one. And if you did answer yes to all of them, your labeled context should look like this: `Environment | Application | Instance`.

Context hierarchy order

Context is defined in the order of precedence. From the widest scope (left), to the narrowest scope (right) - similar to the way you would narrow in on a general specification. For example, to explain a person Jim, you could say: `Human > Male > Jim`, and therefore your repository context would be set up as: `Species | Sex | Name`.

CHAPTER 3

PULL

With fully specified context, pull configuration from ConfigHub service. The JSON response may contain key-value pairs, as well as resolved files (as per request).

- API URL (with token): `https://confighub-api/rest/pull`
- API URL (no token): `https://confighub-api/rest/pull/<account>/<repository>`

Note:

- All data returned is in JSON format.
 - All dates are expected and returned in ISO 8601 format (UTC): `YYYY-MM-DDTHH:MM:SSZ`.
 - All parameters are passed through HTTP header fields.
 - Returned data will contain resolved properties and files, unless limited by the `No-Properties` or `No-Files` flags.
 - Method: GET
-

Usage

```
curl -i https://api.confighub.com/rest/pull \
  -H "Client-Token: <token>" \
  -H "Context: <context>" \
  -H "Application-Name: myApp" \
  -H "Client-Version: <optional version>" \
  -H "Tag: <optional tag label>" \
  -H "Repository-Date: <optional repo date>"
```

```
HTTP/1.1 200 OK
Date: Fri, 10 Jun 2016 22:38:13 GMT
Content-Type: application/json
```

```
Content-Length: 2167
Server: TomEE
{
  "generatedOn": "06/10/2016 22:38:13",
  "account": "ConfigHub",
  "repo": "Demo",
  "context": "Production;TimeKeeper",
  "files": {
    "demo.props": {
      "content": " ... ",
      "content-type": "text/plain"
    },
    "server.xml": {
      "content": " ... ",
      "content-type": "application/xml"
    }
  },
  "properties": {
    "db.host": {
      "val": "prod.mydomain.com"
    },
    "server.http.port": {
      "val": "80"
    },
    "db.name": {
      "val": "ProdDatabase"
    },
    ...
    "db.user": {
      "val": "admin"
    }
  }
}
```

Request Headers

Client-Token

Client token identifies a specific repository. This field is not required if the account and repository are specified as part of the URL.

Context

Context for the pull request has to be a fully-qualified-context (each context rank has to be specified - no wildcards). Context items are semi-colon delimited, and are ordered in order of have to be in context rank order. For example, a repository with context size of 3 levels Environment > Application > Instance could be defined as:

```
-H "Context: Production;MyApp;MyAppInstance "
```

Repository-Date

ISO 8601 date format (UTC) YYYY-MM-DDTHH:MM:SSZ lets you specify a point in time for which to pull configuration. If not specified, latest configuration is returned.

Tag

Name of the defined tag. Returned configuration is for a point in time as specified by the tag. If both Tag and *Repository-Date* headers are specified, *Repository-Date* is only used if the tag is no longer available.

Security-Profile-Auth

If a repository is enabled for and uses Security-Profiles (SP) with encryption, choose any of several ways to decrypt resolved property values.

1. Server-Side decryption by providing SP name(s) and password(s): - Token is created that specifies SP name/password pairs; - SP name/password pairs are specified using this request parameter.
2. Client-Side decryption is also available by: - Use of ConfigHub API in a selected language come functionality for local decryption; - A client can implement its own decryption;

Security-Profile-Auth uses JSON format: `{'Security-Profile_1': 'password', 'Security-Profile_2': 'password', ...}`

Client-Version

Version of the client API. If not specified, ConfigHub assumes the latest version. Even through this is not a required parameter, you are encouraged to specify a version.

Application-Name

This field helps you identify application or a client pulling configuration. Visible in Pull Request tab.

Include-Comments

If value is `true` response includes comments for property keys.

Include-Value-Context

If value is `true` response includes context of resolved property values.

Pretty

If value is `true`, returned JSON is 'pretty' - formatted.

No-Properties

If value is `true` key-value pairs are not returned. This is useful if you are only interested in pulling files, and want to make transaction more efficient.

No-Files

If value is `true` resolved files are not returned. This is useful if you are only interested in pulling properties, and want to make transaction more efficient.

Push API allows clients to update or create properties, context values and tags.

- API URL (with token): `https://confighub-api/rest/push`
- API URL (no token): `https://confighub-api/rest/push/<account>/<repository>`

Note:

- All data returned is in JSON format.
- No data is returned.
- Response code: 200 (Success); 304 (Not modified).
- Method: POST

Usage

```
curl -i https://api.confighub.com/rest/push \
-H "Content-Type: application/json" \
-H "Client-Token: <token>" \
-H "Client-Version: v1.5" \
-H "Application-Name: myApp" \
-X POST -d '
{
  "changeComment": "Adding a new key and value",
  "enableKeyCreation": true,
  "data": [
    {
      "key": "propertyKey",
      "readme": "",
      "deprecated": false,
      "vdt": "Text",
```

```
"push": true,
"securityGroup": "GroupName",
"password": "",
"values": [
  {
    "context": "el;*;el2",
    "value": "",
    "active": true
  },
  {
    "context": "el;*;*",
    "value": "",
    "active": true
  }
]
},
{
  "file": "/path/to/filename.ext",
  "context": "el;*;*",
  "content": "some file content"
}
...
]
```

Successful Response:

```
HTTP/1.1 200 OK
Date: Tue, 15 Nov 2016 17:15:43 GMT
Content-Length: 0
Server: TomEE
```

Error Response:

```
HTTP/1.1 304 Not Modified
Date: Tue, 15 Nov 2016 02:49:23 GMT
ETag: "Invalid password specified."
Server: TomEE
```

Request Headers

Content-Type Required

Content-type header attribute must be set to `application/json`.

Client-Token

Client token identifies a specific repository. This field is not required if the account and repository are specified as part of the URL.

Client-Version

Version of the client API. If not specified, ConfigHub assumes the latest version. Even though this is not a required parameter, you are encouraged to specify a version.

Application-Name

This field helps you identify application or a client pushing configuration. Visible in Pull Request tab.

JSON File Format

Json file you are uploading is a Json Object.

As the push transaction is atomic, a top level *changeComment* parameter will apply for all changes.

The format allows for addition, modification and deletion of any specified element. To delete any element (i.e. key and all values, or a specific value, or a specific file), add parameter *“opp”*: *“delete”* to the element.

For example, to delete a key “aKey” and all its values, specify:

```
{
  "data": [
    {
      "key": "aKey",
      "opp": "delete"
    }
  ]
}
```

To delete a specific key value:

```
{
  "data": [
    {
      "key": "aKey",
      "values": [
        "context": "el;*;*",
        "opp": "delete"
      ]
    }
  ]
}
```


CHAPTER 5

RAW FILE

With fully specified context, pull a specific file from ConfigHub service. The response contains raw, resolved configuration file

- API URL (with token): `https://confighub-api/rest/rawFile`
- API URL (no token): `https://confighub-api/rest/rawFile/<account>/<repository>`

Note:

- All data returned is in JSON format.
- All dates are expected and returned in ISO 8601 format (UTC): `YYYY-MM-DDTHH:MM:SSZ`.
 - All parameters are passed through HTTP header fields.
 - Returned data is the content of the resolved file.
 - Method: GET

Usage

```
curl -i https://api.confighub.com/rest/rawFile \
-H "Client-Token: <token>" \
-H "Context: <context>" \
-H "File: <absolute path>" \
-H "Application-Name: myApp" \
-H "Client-Version: <optional version>" \
-H "Tag: <optional tag label>" \
-H "Repository-Date: <optional repo date>"
```

Request Headers

Client-Token

Client token identifies a specific repository. This field is not required if the account and repository are specified as part of the URL.

Context

Context for the pull request has to be a fully-qualified-context (each context rank has to be specified - no wildcards). Context items are semi-colon delimited, and are ordered in order of have to be in context rank order. For example, a repository with context size of 3 levels Environment > Application > Instance could be defined as:

```
-H "Context: Production;MyApp;MyAppInstance "
```

Repository-Date

ISO 8601 date format (UTC) YYYY-MM-DDTHH:MM:SSZ lets you specify a point in time for which to pull configuration. If not specified, latest configuration is returned.

Tag

Name of the defined tag. Returned configuration is for a point in time as specified by the tag. If both Tag and *Repository-Date* headers are specified, *Repository-Date* is only used if the tag is no longer available.

Security-Profile-Auth

If a repository is enabled for and uses Security-Profiles (SP) with encryption, choose any of several ways to decrypt resolved property values.

1. Server-Side decryption by providing SP name(s) and password(s): - Token is created that specifies SP name/password pairs; - SP name/password pairs are specified using this request parameter.
2. Client-Side decryption is also available by: - Use of ConfigHub API in a selected language come functionality for local decryption; - A client can implement its own decryption;

Security-Profile-Auth uses JSON format: `{'Security-Profile_1':'password', 'Security-Profile_2':'password', ...}`

Client-Version

Version of the client API. If not specified, ConfigHub assumes the latest version. Even through this is not a required parameter, you are encouraged to specify a version.

Application-Name

This field helps you identify application or a client pulling configuration. Visible in Pull Request tab.

This API provides information about a specific repository. It allows for glob syntax search of configuration files, and context definition.

- API URL (with token): `https://confighub-api/rest/info`
- API URL (no token): `https://confighub-api/rest/info/<account>/<repository>`

Note:

- All data returned is in JSON format.
- All dates are expected and returned in ISO 8601 format (UTC): `YYYY-MM-DDTHH:MM:SSZ`.
- All parameters are passed through HTTP header fields.
- Method: GET

Usage

```
curl -i https://api.confighub.com/rest/info \
-H "Client-Token: <token>" \
-H "Repository-Date: <ISO 8601 date (UTC)>" \
-H "Tag: <repo tag>" \
-H "Client-Version: v1.5" \
-H "Files: true/false" \
-H "Files-Glob: <glob expression>" \
-H "Context-Elements: true/false" \
-H "Context-Labels: <comma delimited list>" \
-H "Pretty: true/false"
```

```
HTTP/1.1 200 OK
Date: Wed, 16 Nov 2016 18:12:33 GMT
```

```
Content-Type: application/json
Content-Length: 483
Server: TomEE
{
  "account": "ConfigHub",
  "repository": "HowItWorks",
  "generatedOn": "11/16/2016 18:12:33",
  "context": [ "Environment", "Application" ],
  "contextElements":
  {
    "Environment": [ "Production", "Development" ],
    "Application": [ "Analytics", "Collector", "WebDashboard" ]
  },
  "files": [
    {
      "name": "nginx2.conf",
      "path": "nginx/nginx2.conf",
      "lastModified": 1479260284272
    }
  ]
}
```

Request Headers

Client-Token

Client token identifies a specific repository. This field is not required if the account and repository are specified as part of the URL.

Repository-Date

ISO 8601 date format (UTC) YYYY-MM-DDTHH:MM:SSZ lets you specify a point in time for which to pull repository information. If not specified, latest information is returned.

Tag

Name of the defined tag. Returned information is for a point in time as specified by the tag. If both Tag and *Repository-Date* headers are specified, *Repository-Date* is only used if the tag is no longer available.

Client-Version

Version of the client API. If not specified, ConfigHub assumes the latest version. Even though this is not a required parameter, you are encouraged to specify a version.

Files

Boolean flag to indicate if all files should be returned. If *Files-Glob* header is specified, this flag is ignored and treated true by default.

Files-Glob

Enables glob expressions while searching for files over their path and name.

Context-Elements

Boolean flag to indicate if all context elements should be returned. If *Context-Labels* header is specified, this flag is ignored and treated true by default.

Context-Labels

Limit context elements returned by the list of context labels. Comma delimited list of context labels.

Pretty

If value is `true`, returned JSON is 'pretty' - formatted.

CHAPTER 7

Repositories

Use this API to get full details of settings for all defined repositories.

- API URL: `https://confighub-api/rest/info/all`

Note:

- All data returned is in JSON format.
 - All dates are expected and returned in ISO 8601 format (UTC): YYYY-MM-DDTHH:MM:SSZ.
 - All parameters are passed through HTTP header fields.
 - Method: GET
-

Usage

```
curl -i https://api.confighub.com/rest/info/all \
-H "Client-Version: v1.5" \
-H "Pretty: true/false"
```

```
HTTP/1.1 200 OK
Date: Fri, 25 Nov 2016 19:47:39 GMT
Content-Type: application/json
Content-Length: 2776
Server: TomEE
[
  {
    "account": "ConfigHub",
    "name": "Demo",
    "isPrivate": false,
    "isPersonal": false,
    "description": "This is a demo repository. Saving changes is disabled for all_
    ↪options, however all options are available for you to explore.",
```

```
"created": "2016-05-05T16:26Z",
"accessControlsEnabled": false,
"vdtEnabled": true,
"securityEnabled": true,
"contextGroupsEnabled": true,
"keyCount": 27,
"valueCount": 42,
"userCount": 1,
"context": [
  "Environment",
  "Application"
]
},
{
  "account": "ConfigHub",
  "name": "HowItWorks",
  "isPrivate": false,
  "isPersonal": false,
  "created": "2016-10-04T21:19Z",
  "accessControlsEnabled": false,
  "vdtEnabled": true,
  "securityEnabled": true,
  "contextGroupsEnabled": false,
  "keyCount": 13,
  "valueCount": 23,
  "userCount": 0,
  "context": [
    "Environment",
    "Application"
  ]
}
]
```

Request Headers

Client-Version

Version of the client API. If not specified, ConfigHub assumes the latest version. Even though this is not a required parameter, you are encouraged to specify a version.

Pretty

If value is `true`, returned JSON is 'pretty' - formatted.

CHAPTER 8

System Status

API returns details of ConfigHub License and version.

- API URL: `https://confighub-api/rest/info/system`

Note:

- All data returned is in JSON format.
 - All parameters are passed through HTTP header fields.
 - Method: GET
-

Usage

```
curl -i https://api.confighub.com/rest/info/system \
-H "Client-Version: v1.5" \
-H "Pretty: true/false"
```

```
HTTP/1.1 200 OK
Date: Fri, 25 Nov 2016 19:55:01 GMT
Content-Type: application/json
Content-Length: 635
Server: TomEE
{
  "version": {
    "version": "v1.2.0"
  },
  "license": {
    "First Name": "John",
    "Last Name": "Doe",
    "Email": "john.doe@acme.com",
    "Company": "Acme Inc.",
  }
}
```

```
    "Title": "CTO",
    "Type": "Trial",
    "Expires": "Wed, Mar 1, 2017",
    "LicenseKey": "...",
  }
}
```

Request Headers

Client-Version

Version of the client API. If not specified, ConfigHub assumes the latest version. Even though this is not a required parameter, you are encouraged to specify a version.

Pretty

If value is `true`, returned JSON is ‘pretty’ - formatted.








APIs on GitHub: [Java](#), [Python](#)

All API transactions between the client and the ConfigHub service are made using HTTPs secure protocol. Private installation can be configured to also use HTTP (non-secure) protocol, but then risk in-flight security breach.

Security groups

Security Group is an abstract wrapper to which properties and/or files may be assigned. Security Group is password protected, so any modification of the data contained in them requires authentication before the change is allowed.

In addition to a password challenge, security group may also be defined with **Encryption** enabled using one of several provided cyphers. If enabled, content assigned to a security group will, in addition to being password protected, be encrypted, using the selected cypher and the security group's password.

 Config  Revisions  Security  Tokens  Pull requests  Teams  Settings

Security Group Name:

☒ Encrypt with

AES/CBC/PKCS5Padding ▲

AES/CBC/PKCS5Padding
AES/ECB/PKCS5Padding
DES/CBC/PKCS5Padding
DES/ECB/PKCS5Padding
DESEde/CBC/PKCS5Padding
DESEde/ECB/PKCS5Padding

cipher

Password:

Re-enter Password:

Create security group

Cancel

Securing files and properties

Once the security group is created, you can add any number of properties and files to it. To assign an existing property to a security group, you need to add a security group to the key.

Assignment to a property

Readme:


Type: Text ▼ Security: Off ▼ Push: Disabled ▼

Key:

Off
DBPasswords

Save key change

Cancel



2 values assigned ▼

Assignment to a file

server / conf /

File:



Active: ☒

Context:

Environment Application

Security: Off ▼

Off
DBPasswords

 Save file Cancel Preview 

CHAPTER 10

Properties editor

Properties are traditionally defined as::

```
key: value
```

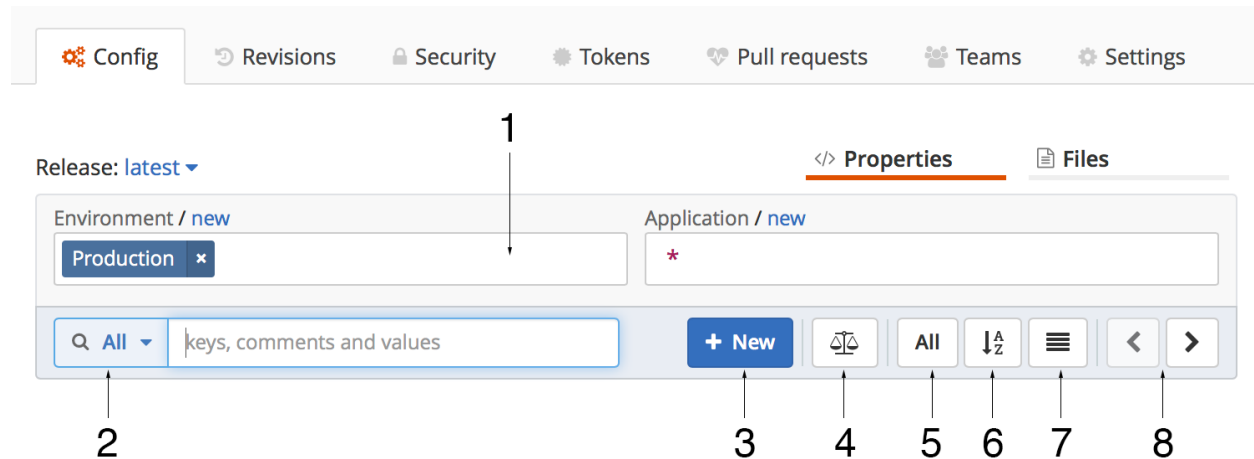
In ConfigHub, a property is defined with a context attached to the value. This schema allows many values to be assigned to a single key, with the requirement that each value's context is unique among other values assigned to the same key.:

```
key: [  
  context: value,  
  context: value,  
  ...  
]
```

Following sections show how context properties and context elements are created, edited and deleted.

Editor toolbar

Properties editor is the default view of ConfigHub UI where all/resolved properties are managed. The properties toolbar allows provides following functionality:



1. Context selection

These fields allow you to set the working context, and view configuration that is “resolved” by that context. Unlike the context requested by the client, editor context can have multiple context elements specified in a single context hierarchy.

For example, if you wanted to see configuration returned for both *Production* and *Development* environments, you could select both in the Environment context hierarchy. The returned configuration would show all keys and values that can be returned for the combination of the specified context.

We call this type of context - where any number of context elements can be selected per context hierarchy, a **non-qualified context**;

2. Search type

Searching keys, comments or values, may return results either among the resolved values (as indicated by specified context #1), or among all configuration properties in this repository.

The **All** selection, searches all properties in the repository, where **Resolved** searches only among properties resolved by the specified context.

3. New property

This button toggles the new property form, where a new or existing key can be specified as well as a property value.

4. Comparison View

Side-by-side comparison of resolved properties from any combination of contexts, times or tags.

5. All key toggle

Let's you see all property keys. If some keys did not resolve values as per the specified context #1, these keys will not be shown in the properties editor. Clicking this key, will include them in the display, but their values are still left out of the view.

6. Key sort order

Keys are sorted alphabetically. Sort order toggles direction.

7. Value context alignment

To see all value contexts aligned as a table view, toggle this button.

8. Pagination navigation

Move between pages of results.

Create a new property

1. Readme

Comment for the property key. Visible and searchable in property editor.

2. Key Attributes

- **Type:** Refers to the data type of values. When a non-Text value is selected, this value type is enforced in the UI, and the type information is included in the JSON API response.
- **Security:** Chose pre created *Security Group*. All values assigned to the key will adhere to access controls set by the Security Group, and will be encrypted as per Security Group settings.
- **Push:** If enabled, this key and its values may be modified via PUSH API.

3. Key

Property key is unique per repository. Specifying a key that is already defined, will just add new value to the existing key. **Deprecated** flag can be enabled for a key. If enabled, client PULL response JSON will include a deprecated flag on the key object. Use this flag to log all deprecated key usages.

4. Value

Value for this key. If *Type* (key attribute) flag is specified, value input field is changed in consideration of the value type.

5. Context

Each key's value has to have a unique context. Having a unique context guarantees that a fully-specified-context, as requested by the client applications, can only receive back a single value per key.

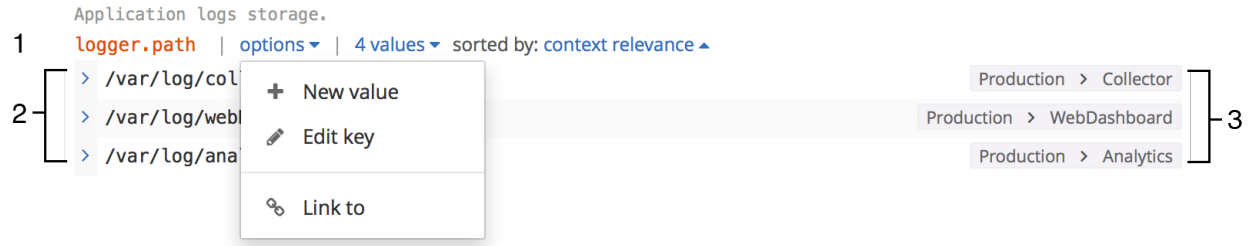
Each context’s hierarchy may contain a wildcard or a single context element.

6. Save or Save with change comments

Clicking on a talk-bubble button next to the “Save property” will pop-open a “Change comment” text box. Comments entered here will be visible next to the change in the *Revisions* tab.

Add value to existing key

Mousing over the existing key, attributes or values, shows additional options for the key.



Choosing *New value* will open a new value form.

Application logs storage.
logger.path

Active: ☒

Value:

Context:

Environment Application

> /var/log/collector Production > Collector

> /var/log/webDash Production > WebDashboard

> /var/log/analytics Production > Analytics

The value’s form elements are the same as specified in [Create a new property](#) section. Additional option is the **Active** toggle. When a value is *Disabled*, it is treated as if it is deleted. A *Disabled* value is never returned to the client.

Editing keys and values

Double clicking on a key or a value, will open the editing form for either key or a value. You can also click on *Edit key* from the key’s *options* menu. Clicking the right array next to the value will also trigger the value form.