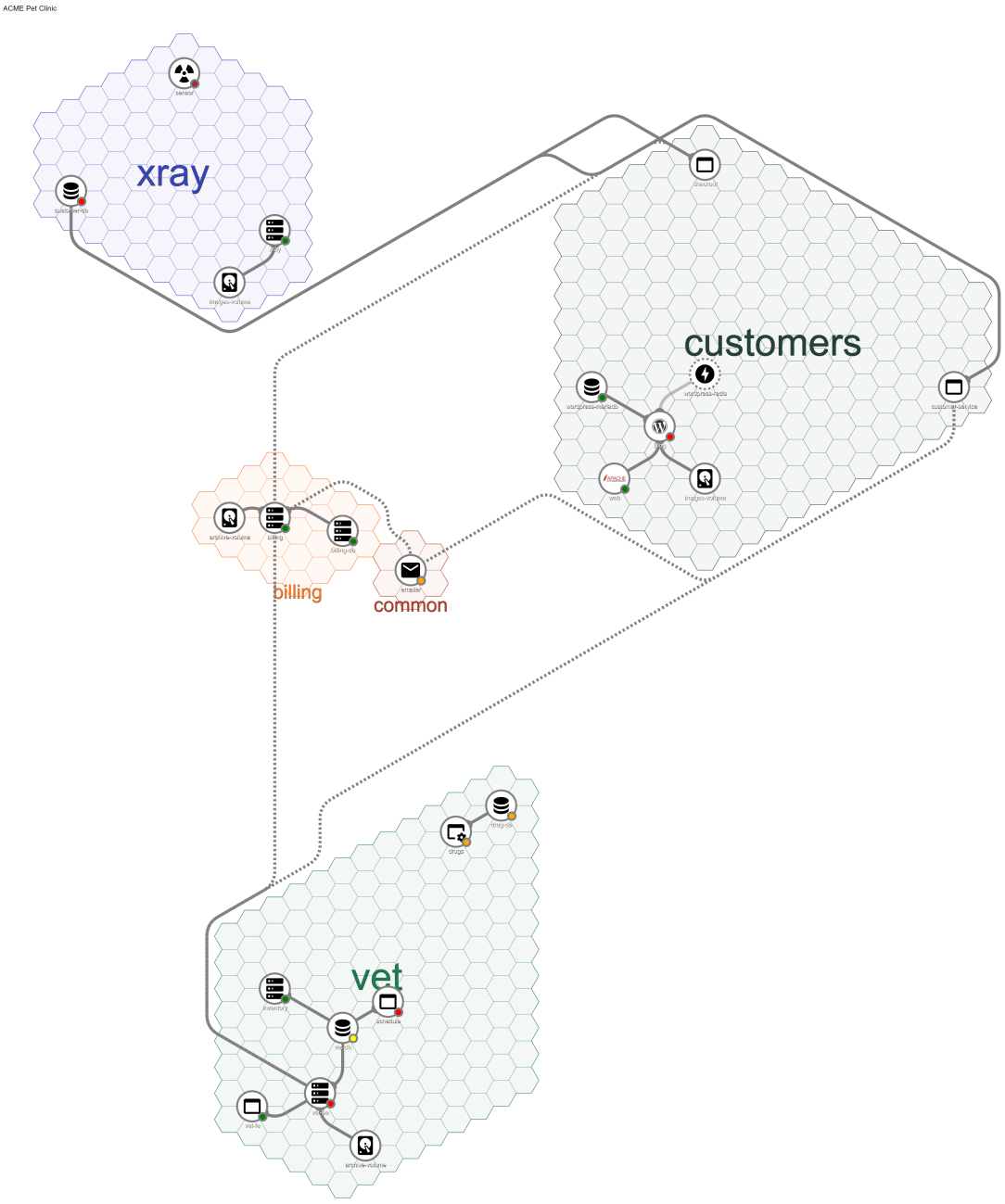
nivio Documentation

Daniel Pozzi

Oct 21, 2022

Contents:

Nivio is a tool for application landscape management targeted at teams (developers, operators, managers). It follows a no-op approach, i.e. there is no interface for manual data maintenance. Instead, Nivio pulls all its information from data sources, like files and web APIs (e.g. monitoring items) or allows pushing information via its API.



- **It is easy to install and to maintain.** Runs dockerized on a single server with moderate to low hardware requirements. It stores the items, so it can be discarded at any time and be refilled with the next start.
- **No-op usage** Besides its initial configuration, it is designed to gather the application landscape information from configurable items, preferably code repositories.
- **Renders the landscapes as a graph** See above.
- **Multiple configuration sources** While Nivio has its proprietary YAML format, you can also use docker-compose files, or use them as basis and enrich them using further files

- **PULL and PUSH** Basic indexing of landscapes driven by observed configuration files, or send data to the API.
- **Aggregation of item state** Using sources, like Prometheus, and marking items accordingly.

CHAPTER 1

Getting Started

1.1 Installation

The easiest way to get started is by bringing up Nivio in Docker:

```
export NIVIO_BASE_URL=http://localhost:8080
docker run -it --rm -e NIVIO_BASE_URL -p 8080:8080 dedica/nivio:latest
```

Take a look at http://localhost:8080/

A friendly reminder on the application starting screen tells you that there are no landscapes loaded, yet. To do so, follow the instructions below.

1.2 Creating the first landscape

The simplest possible landscape definition consists of:

- an identifier, used e.g. when updating the landscape
- a name, that is shown in the UI
- at least 1 item (things get weird if there are no items at all)

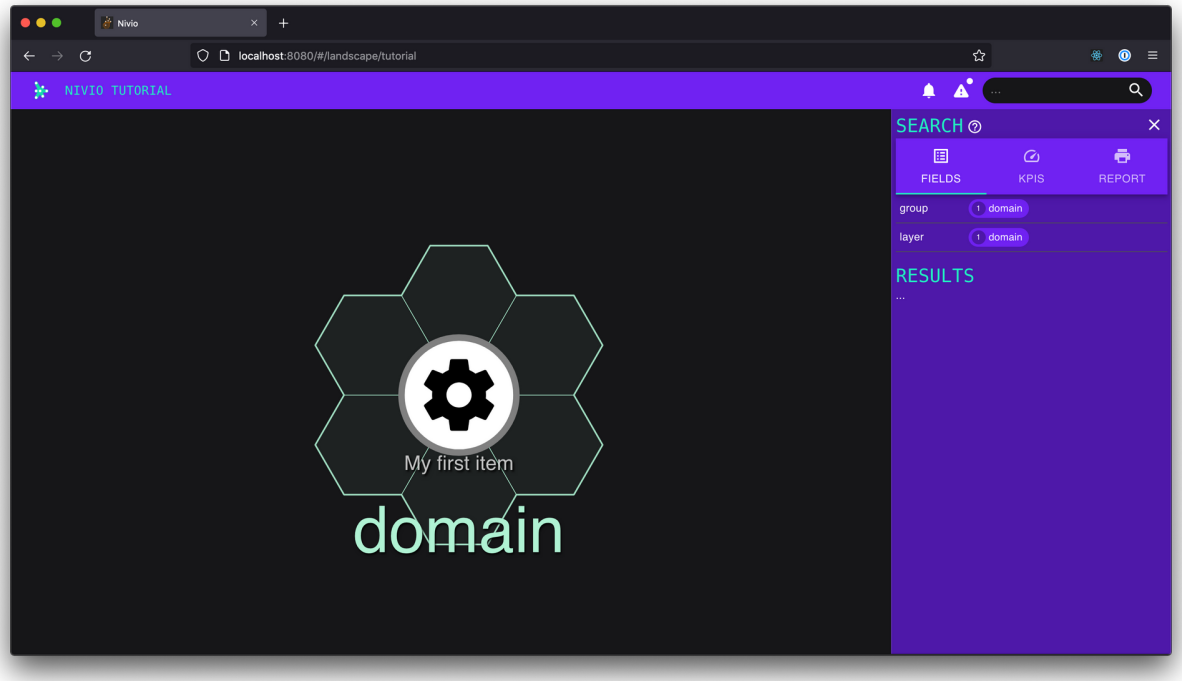
In YAML, it looks like this:

```
identifier: tutorial
name: Nivio Tutorial
items:
  - identifier: item-1
    name: My first item
```

Create the file tutorial.yaml and upload (POST) this file using curl:

```
curl -X POST -H "Content-Type: application/yaml" --data-binary @tutorial.yaml ${NIVIO_
↪BASE_URL}/api/landscape
```

Take another look at http://localhost:8080/



You should find your landscape with the item in it. The item might be assigned to a default group called *domain*.

The OpenAPI (aka Swagger) documentation is located at `/v3/api-docs` (JSON) or `/swagger-ui.html` (HTML GUI).

1.3 Updating the landscape

For example, simply change the name of the item and run the same curl command again.

The landscape diagram should update immediately in your browser.

As long as the landscape's *identifier* is the same, it will be updated in place. If you change the landscape identifier in the YAML file, then Nivio will create a new, separate landscape.

```
curl -X PUT -H "Content-Type: application/yaml" --data-binary @tutorial.yaml ${NIVIO_
↪BASE_URL}/api/landscape/tutorial
```

If the update doesn't seem to be happening, make sure the curl request didn't fail and you used `PUT` as method. If the reason is not obvious, you can also investigate in the Nivio logs.

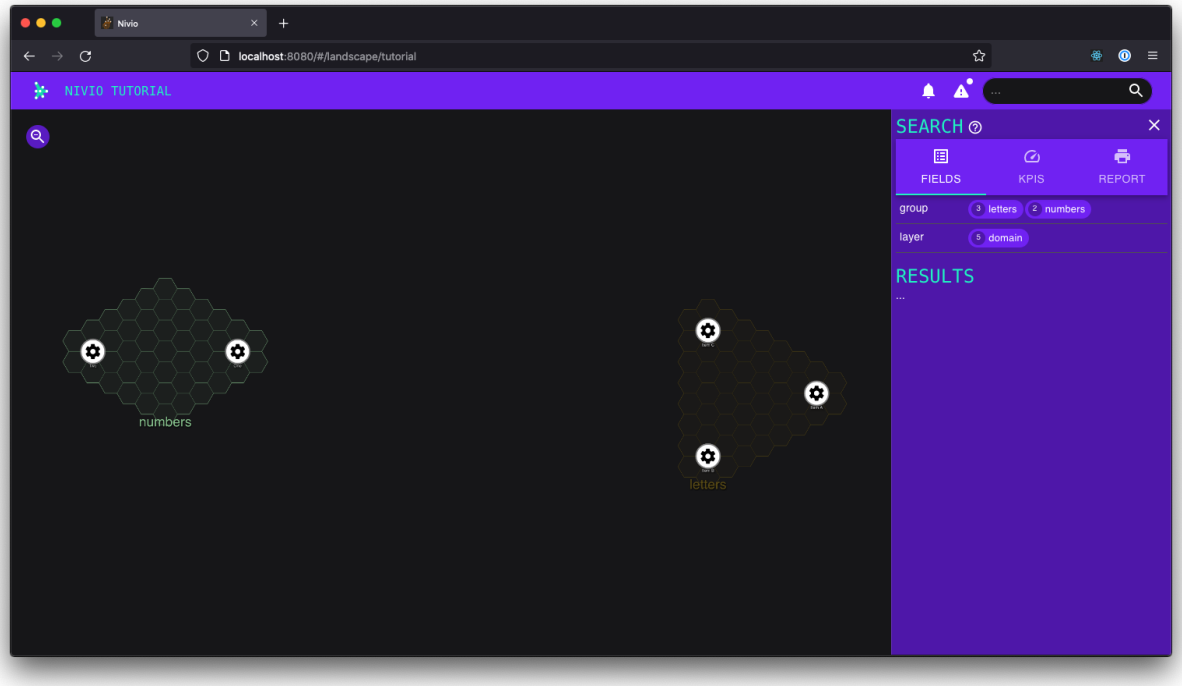
For some changes, unfortunately though, you will have to restart Nivio and submit the landscape afresh.

1.4 Exploring the Nivio model

1.4.1 Adding your own groups

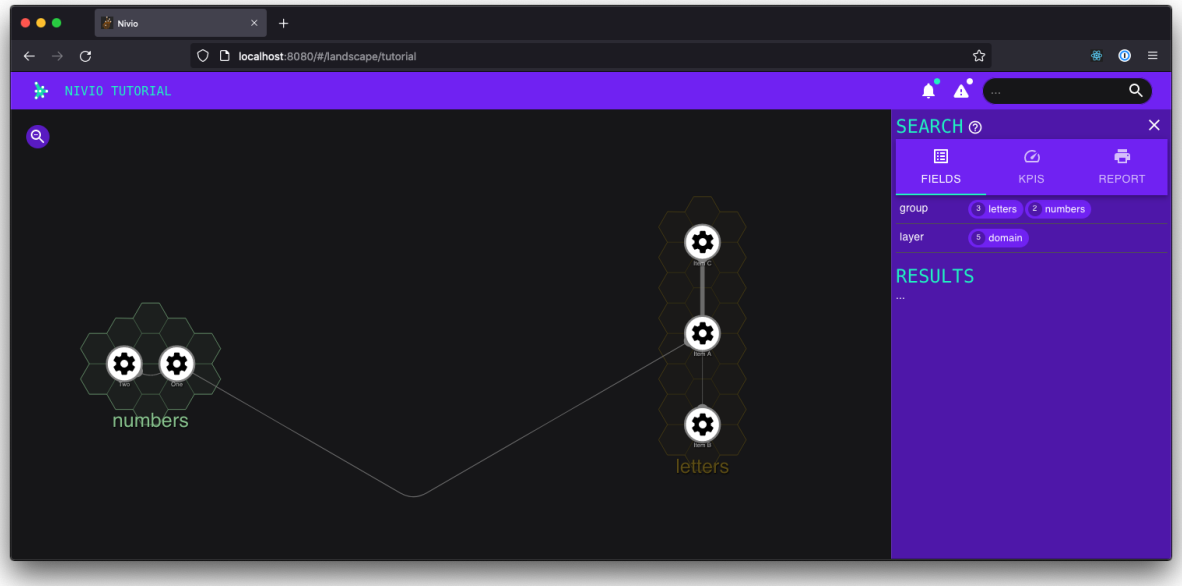
- Without groups, all items will be assigned to a default group. Items can be assigned to groups using the `group` attribute.
- If a group does not exist, nivio creates it. To customize a group, add it to the configuration.
- However, you can't create a group with no attributes. Nivio will not accept the landscape. So at least add an `owner`.
- If intended to delete already set group attributes such as `description` or `contact`, you can do so. But please pay attention that this is not true for the attributes of items or of the landscape. The reason is that group attributes for the groups are all manually set, and the user should have full control about it. For items, instead, best practice is that attributes stick to the old values if they are missing in one configuration file. This is because items are mainly set by several config files. Also, landscapes are not intended to have missing attributes. Therefore, the landscape's attributes stick to formerly set ones if they are missing.

```yaml
groups:
  letters:
    owner: Myself
    description: All the letters
  numbers:
    owner: Myself
    description: All the numbers
items:
  - identifier: a
    name: Item A
    group: letters
    ...
```



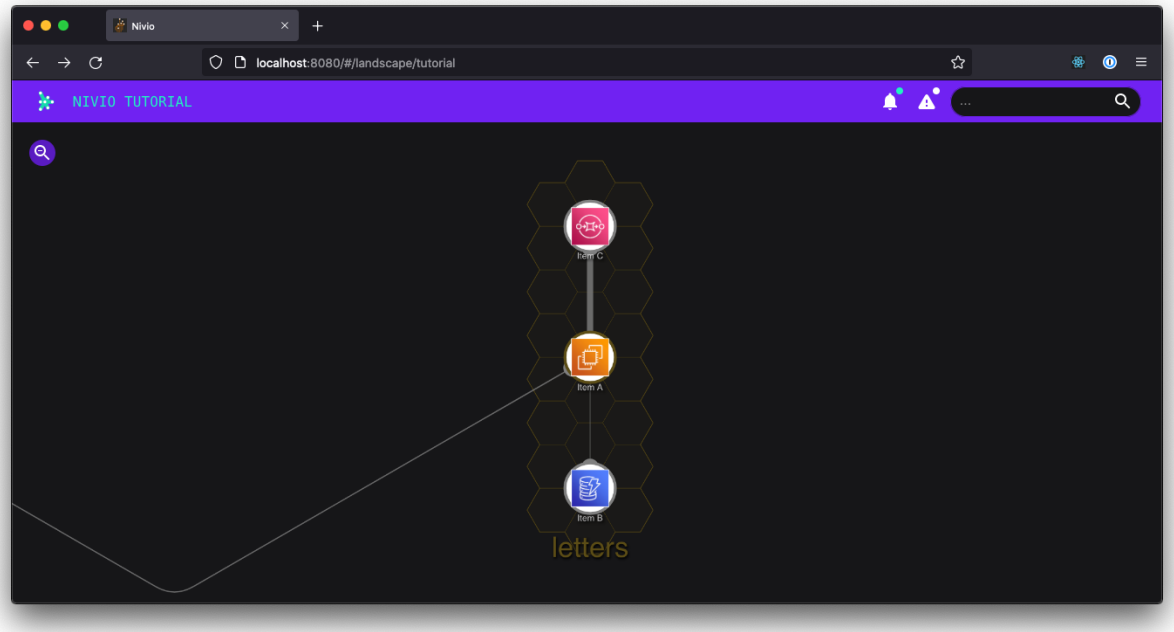
1.5 Adding relations between items

- Relations are a key element of every graph. Note that relations are directional in Nivio.
- A relation can have a *weight* attribute between 0-5 to control the width of the line between the two items.



1.6 Adding icons

```yaml
- identifier: a
  name: Item A
  group: letters
  icon: https://visioguy.github.io/IconSets/aws/icons/amazon_ec2.png
- identifier: b
  name: Item B
  group: letters
  icon: https://visioguy.github.io/IconSets/aws/icons/amazon_dynamodb.png
- identifier: c
  name: Item C
  group: letters
  icon: https://visioguy.github.io/IconSets/aws/icons/amazon_simple_queue_service_
↪(sqs).png
```



1.6.1 Observations

- We are not using them here, but don't forget that Nivio supports all the [Material Design Icons](https://materialdesignicons.com/) out of the box!
- If you do use custom icons intensively, you should find a place for hosting them. You can define an environment variable *NIVIO_ICON_FOLDER* which contains a path reachable for the app.

1.7 Introducing KPIs

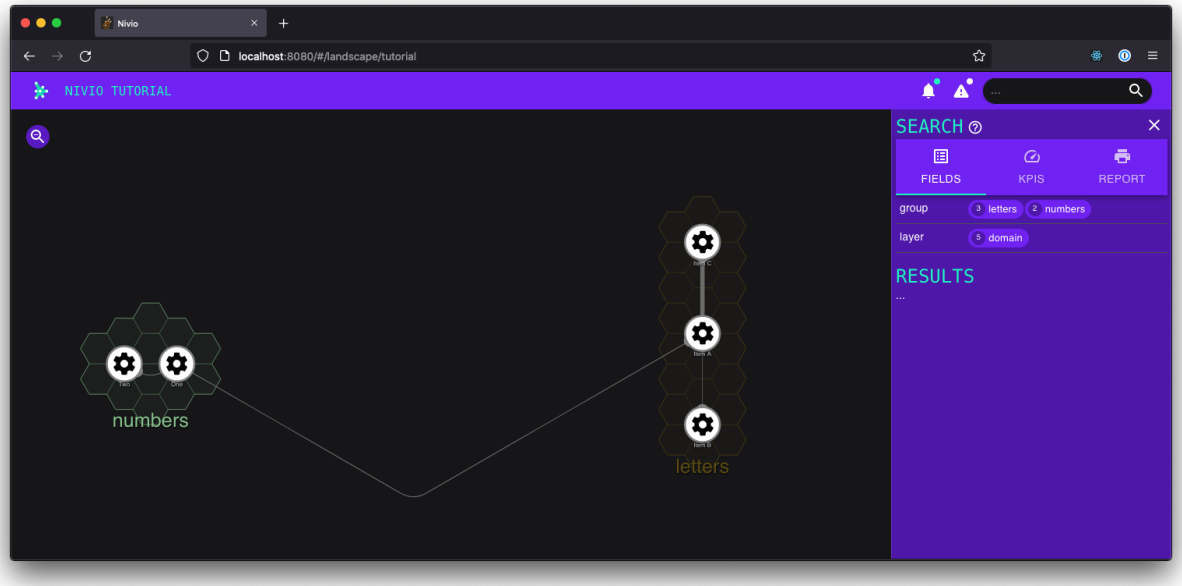
Before you can set a KPI on an item, you need to define the KPI itself.

Here we create one called `capacity`. It requires a value of `90-100` in order to be considered healthy (green), `80-90` for semi-healthy (yellow), and everything else will be considered unhealthy (red):

```
config:
  kpis:
    capacity:
      label: capacity
      ranges:
        GREEN: 90;100
        YELLOW: 80;90
        RED: 0;80
```

Now the KPI can be used simply by adding the corresponding attribute to any item:

```
items:
  - identifier: a
    name: Item A
    capacity: 87
```



1.7.1 Observations

- The status of a KPI (i.e. its color) is automatically assessed based on the attribute value - awesome!
- There are two other colors that you could use, orange and brown.
- You only need to define a KPI once, then you can use the respective attribute on as many items as you need to. And of course, an item can use multiple KPIs.
- Nivio ships with a number of default KPIs (for more information see *Data Assessment using KPIs* section)

1.8 Summary

1.8.1 Conclusion

Once you have figured out its configuration language, Nivio is a great tool for quickly creating a cool visualization of your graph and of the dependencies between the entities you are managing, whether it's software architecture, an application landscape, your organization, or any other domain. The addition of dynamic KPIs and the status aggregation make it really easy to spot issues quickly!

1.8.2 Best practices

- Use `groups` and `items`. Use `relations` between items. They also can have weights.
- A group requires at least 1 attribute in order to work. Use `owner`, because Nivio always shows something on the owner anyways, and "No owner" doesn't look great.
- The following attributes should be on every item: a `name`, an `icon`, a `group` assignment, KPI attributes for the info tab, and any `random word` with a `value` for the details tab. This way, the item's details will be nicely filled out across the info, relations (if applicable), and details tabs.
- Define your own KPIs that make the most sense for your use case.

- Find a place to host your custom icons. Don't abuse and don't rely on external hosts!
- Make sure to read *Using Templates to dynamically assign data* before putting too much effort into item configuration.

1.8.3 Caveats

- The colors that are automatically assigned to groups tend to be dark and lack contrast, because they are also used for printing with white background. While you can assign colors manually, it's very tedious to do so.
- You can experiment a lot with the structure of the landscape, but better figure out the attributes once and then stick to them. For example, icons, or color of groups, etc. - in order to see them updated everywhere (or at all!) you'll often have to restart Nivio and submit the landscape again.

1.9 Bonus: Having Nivio pull your data

In order to have data automatically observed to changes Nivio expects a seed configuration at start time. You need to set the environment variable *SEED*. The configuration file contains basic data, references to item descriptions `sources`, which can be local paths or URLs. The descriptions can be gathered by HTTP, i.e. it is possible to fetch files from protected sources via authentication headers. Think of GitLab or GitHub and the related tokens. To use secrets etc., please refer to *Environment variables*.

```
identifier: nivio:example
name: Landscape example
contact: mail@acme.org
description: This is an example landscape.
sources:
  - "./items/wordpress.yml"
  - url: "http://some.server/docker-compose.yml"
    format: docker-compose-v2
  - url: https://gitlab.com/bonndan/nivio-private-demo/raw/docker-compose.yml
    headerTokenName: PRIVATE_TOKEN
    headerTokenValue: ${MY_SECRET_TOKEN_ENV_VAR}
```

1.10 Deleting items

Items not referenced anymore in the descriptions will be deleted automatically on a complete and successful re-index run. If an error occurs fetching the source while indexing, the behaviour of the indexer changes to treat the available data as partial input. This means only inserts and updates will happen and no deletion.

1.11 Environment variables

The following environment variables can be set to configure nivio:

DEMO

A non-empty value causes Nivio to start in demo mode with prepared data. Use the value 'all' to load more landscapes.

GITHUB_JWT

GitHub JSON Web Token (JWT) to connect to GitHub as a GitHub App.

GITHUB_LOGIN

GitHub user name. Can also be used to connect as organization with OAuth.

GITHUB_OAUTH

GitHUb OAuth Token to connect to GitHub via personal access token.

GITHUB_PASSWORD

GitHub password (for username/password login).

GITLAB_HOST_URL

The full URL to the GitLab API, e.g. http://your.gitlab.server.com/api/v4.

GITLAB_PASSWORD

GitLab OAuth login password (optional).

GITLAB_PERSONAL_ACCESS_TOKEN

Personal token to access the GitLab API at *GITLAB_HOST_URL* (optional).

GITLAB_USERNAME

GitLab OAuth login username (optional). If used, *GITLAB_PASSWORD* is also required).

KUBERNETES_MASTER

K8s master URL (optional). All variables from https://github.com/fabric8io/kubernetes-client#configuring-the-client can be used.

NIVIO_AUTH_ALLOWED_ORIGINS

Patterns for allowed origins when the app requires authentication

NIVIO_AUTH_GITHUB_ALIAS_ATTRIBUTE

GitHub user attribute to use as alias

NIVIO_AUTH_GITHUB_CLIENT_ID

GitHub app OAuth2 client id

NIVIO_AUTH_GITHUB_CLIENT_SECRET

GitHub app OAuth2 client secret

NIVIO_AUTH_GITHUB_NAME_ATTRIBUTE

GitHub user attribute to use as name

NIVIO_AUTH_LOGIN_MODE

Authentication mode: none, optional, required

NIVIO_BASE_URL

The base URL of Nivio to be used for frontends if running behind a proxy.

NIVIO_BRANDING_BACKGROUND

Branding background color (hexadecimal only).

NIVIO_BRANDING_FOREGROUND

Branding foreground color (hexadecimal only).

NIVIO_BRANDING_LOGO_URL

A URL pointing to a logo.

NIVIO_BRANDING_MESSAGE

A welcome message on the front page.

NIVIO_BRANDING_SECONDARY

Accent color used for active elements (hexadecimal only).

NIVIO_ICON_FOLDER

A folder containing icons named similar to material design icons

NIVIO_MAIL_HOST

SMTP mail host.

NIVIO_MAIL_PASSWORD

SMTP mail password.

NIVIO_MAIL_PORT

SMTP mail port.

NIVIO_MAIL_USERNAME

SMTP mail username.

PORT

The port Nivio runs on.

SEED

A semicolon-separated list of file paths containing landscape configurations.

SONAR_LOGIN

SonarQube login (username).

SONAR_PASSWORD

SonarQube password.

SONAR_PROXY_HOST

SonarQube proxy host (optional).

SONAR_PROXY_PORT

SonarQube proxy port (optional).

SONAR_SERVER_URL

SonarQube server URL.

CHAPTER 2

Input Sources

2.1 Kubernetes cluster inspection

Kubernetes clusters are inspected using Fabric8.io's Java client. See https://github.com/fabric8io/kubernetes-client#configuring-the-client for configuration. Parsing can be configured via a URL, i.e. the examined namespace can be given (otherwise all namespaces are scanned) and a label for building groups can be named. Both parameters and even the whole URL are optional.

```
identifier: k8s:example
name: Kubernetes example
sources:
  - url: http://192.168.99.100?namespace=mynamespace&groupLabel=labelToUseForGrouping
    format: kubernetes
```

2.2 Rancher 1 Cluster Inspection

Rancher clusters can be indexed one project (aka environment in the GUI speak) at a time. Access credentials can be read from environment variables. To exclude internal stacks, like those responsible for internal networking, blacklist them.

```
identifier: rancher:example
name: Rancher 1.6 API example
config:
  groupBlacklist: [".*infra.*"]

sources:
  - url: "http://rancher-server/v2-beta/"
    projectName: Default
    apiAccessKey: ${API_ACCESS_KEY}
    apiSecretKey: ${API_SECRET_KEY}
    format: rancher1
```

2.3 Nivio proprietary format

Nivio provides its own format which allows to set all model properties manually (see *Model and Syntax* section).

2.4 Reading from CSV

Nivio can parse CSV files regarding rows as landscape items. The order of the columns in the file is important because headers are ignored and not mapped automatically. Instead, each column number, starting at zero, can be assigned to an item property in the `mapping` configuration. Additionally, the CSV separator char and the number of lines to skip (usually 1 for the header row) can be set.

```
sources:
 - url: "./services/test.csv"
   format: csv
   mapping:
     identifier: 1
     name: 0
     description: 2
     providedBy: 3
   separator: ";"
   skipLines: 1
```

2.5 Reading and Mapping from JSON

Any JSON file or URL can be parsed and mapped into a landscape description. Existing structures are mapped to landscape components using JSONPath, a query language to traverse JSON objects and select specific properties.

```
identifier: example:customjson
name: Custom JSON example
sources:
  - format: customJSON
    url: /mnt/items.json
    mapping:
      items: "$.items"
      item:
        identifier: "$.id"
        endOfLife: "$.end_of_life.date"
        name: '$.moreThanAName|find "([\\w\\s]*),"'
        nivio.link.homepage: "$.a_named_link"
        nivio.relations.inbound: "$.@dependencies.@upstream|fetch|$.items[*].id"
        nivio.relations.providers: "$.infra|fetch|$.items[*].id"
```

The folling mapping steps are supported:

- **JsonPath**. Simply add the path expression. Note that the root for items is the root of the assigned JSON subnode. For more info on JsonPath, see https://support.smartbear.com/alertsite/docs/monitors/api/endpoint/jsonpath.html.
- **fetch** an URL, starting with the keyword _fetch_
- **find** a subtext using regular expressions. starting with the keyword _find_. Make sure to put lines containing regexes in **single quotes** and surround the pattern with **double quotes** (a CSV parser is used in combination with a YAML parser, so this is the combination that works).

2.6 Reading from GraphViz dot files

https://www.graphviz.org/ is a graph visualisation software which uses the dot language https://graphviz.org/doc/info/lang.html to describe graphs. It is possible to add arbitrary attributes to nodes and edges, so nivio can use these attributes to enhance items and relations. However, it is necessary to prefix attributes that should be taken into account using the string "**nivio_**".

```
digraph G {
        main [
            nivio_owner = Marketing,
            nivio_software="Wordpress 2.0",
            nivio_group=FooBar,
            nivio_contact="foo@bar.com"
        ]
        main -> parse -> execute
        main -> init [nivio_format = json, nivio_type=PROVIDER, nivio_description=
↪"init the procedure", nivio_frameworks="PHP:7.2,Angular:9"]
        main -> cleanup
        execute -> make_string
        execute -> printf
        init -> make_string
        main -> printf
        execute -> compare
        }
```

Also remember to put non-ascii words (like email addresses) or sentences into double quotes.

To configure this as input source, add:

```
sources:
 - url: "./test/foo.dot"
   format: dot
```

2.7 External data

Nivio can load external data that cannot be used directly to build landscapes, but is still valuable. For example, the number of GitHub issues might be interesting to see on a landscape item that is an open source component. To attach such data to landscape components, use links having special known identifiers like "*github*" or "*sonar*".

This is work in progress. Currently supported link identifiers are:

- `github` for GitHub repositories
- `gitlab` for GitLab repositories
- `spring.health` for Spring Boot health actuators https://docs.spring.io/spring-boot/docs/current/actuator-api/htmlsingle/#health

```
items:
  - identifier: nivio
    links:
      github: https://github.com/dedica-team/nivio
      spring.health: http://localhost:8090/actuator/health
      # sonar: http://hihi.huhu not implemented yet
```

CHAPTER 3

Model and Syntax

3.1 Landscape

A landscape is defined as a collection of items which somehow belong together, be it for technical or business reasons. For example, a company department might model ALL its applications in production as one landscape and use grouping or tagging to further separate the applications. A second landscape could be used to model a future layout with a different infrastructure. Both landscapes could have items in common (like a database, load balancer, etc.), so their configuration can be reused.

3.2 LandscapeDescription

Name	Type	Description	Remarks	Example
assignTemplates	Map<array>		optional, defaults to null	null
color	String		optional, defaults to null	null
config	LandscapeConfig<LandscapeConfig>		optional, defaults to null	null
contact	String	Primary contact method, preferably an email address.	optional, defaults to null	null
description	String	A brief description of the landscape.	optional, defaults to null	null
groups	Map<GroupDescription>	Description of item groups (optional, can also be given in sources).	optional, defaults to null	null
icon	String	An icon or logo url	optional, defaults to null	null
identifier	String	Immutable unique identifier. Maybe use an URN.	**required**, defaults to null	null
isPartial	Boolean		optional, defaults to null	null
items	List<ItemDescription>	List of configuration sources. Handled in the given order, latter extend/overwrite earlier values like items etc.	optional, defaults to null	null
labels	Map		optional, defaults to null	null
links	Map<Link>	Key-value pairs of related links. Some keys like 'github' cause that the endpoint data is parsed and added to to corresponding landscape component.	optional, defaults to null	github: https://github.com/dedica-team/nivio
name	String	Human readable name.	**required**, defaults to null	null
owner	String	The business owner (person or team), preferably an email address.	optional, defaults to null	null
partial	Boolean	marks that the landscape is not complete, but an update	optional, defaults to null	null
templates	Map<ItemDescription>	Item descriptions to be used as templates. All values except identifier and name will be applied to the assigned items.	optional, defaults to null	null

3.3 SourceReference

This is a reference to a configuration file.

Name	Type	Description	Remarks	Example
assignTemplates	Map<array>	A map with template identifier as key and item identifier matchers as value	optional, defaults to null	endOfLife: [web, "java6*"]
basicAuthPassword	String		optional, defaults to null	null
basicAuthUsername	String		optional, defaults to null	null
deprecation	String	deprecation info (typically used in OpenAPI specs)	optional, defaults to null	null
format	String	The input format.	optional, defaults to null	null
headerTokenName	String		optional, defaults to null	null
headerTokenValue	String		optional, defaults to null	null
href	String	The link target.	**required**, defaults to null	null
hreflang	String	hateoas language	optional, defaults to null	null
media	String	hateoas media type	optional, defaults to null	null
name	String	HateOAS / OpenAPI name	optional, defaults to null	null
rel	String	hateoas relation type	optional, defaults to null	null
title	String	hateoas title	optional, defaults to null	null
type	String		optional, defaults to null	null
url	String	A URL, but can also be a relative path.	optional, defaults to null	./a/items.yaml

3.4 LandscapeConfig

Configuration of key performance indicators (i.e. status indicators) and layouting tweaks.

Name	Type	Description	Remarks	Example
branding	Branding<Branding>		optional, defaults to null	null
greedy	Boolean	Flag that enables instant creation items based relation targets that cannot be found in the sources.	optional, defaults to null	null
groupBlacklist	List	Names or patterns of groups that should be excluded from the landscape. Used to improve automatic scanning results.	optional, defaults to null	.*infra.*
kpis	Map<KPIConfig>	Key performance indicator configs. Each KPI must have a unique identifier.	optional, defaults to null	null
labelBlacklist	List	Names or patterns of labels that should be ignored. Used to improve automatic scanning results.	optional, defaults to null	.*COMPOSITION.*
layoutConfig	LayoutConfig<LayoutConfig>		**required**, defaults to null	null

3.5 KPIConfig

The configuration of landscape specific key performance indicators that derive status information from landscape components. Usually the KPIs work on labels

Name	Type	Description	Remarks	Example
description	String	Description of the purpose of the KPI	optional, defaults to null	null
enabled	Boolean	A flag indicating that the KPI is active. Can be used to disable default KPIs.	optional, defaults to null	null
label	String	Key of the label to evaluate	**required**, defaults to null	costs
matches	Map	A map of string based matchers that determine the resulting status (GREEN|YELLOW|ORANGE|RED|BROWN). Use a semicolon to separate matchers.	optional, defaults to null	RED: BAD;err.*
messageTemplate	String	Template for the displayed message, containing a placeholder for the assessed value '%s	optional, defaults to null	The current value is: %s
ranges	Map	A map of number based ranges that determine the resulting status (GREEN|YELLOW|ORANGE|RED|BROWN). Use a semicolon to separate upper and lower bounds. Tries to evaluate label values as numbers.	optional, defaults to null	GREEN: 0;99.999999

3.6 LayoutConfig

Layout configuration for landscapes with unusual number or ratios of items, groups and relations.

Name	Type	Description	Remarks	Example
groupLayoutInitialTemp	Integer	The initial temperature for layouts of groups.	**required**, defaults to 900	900
groupMaxDistanceLimit	Integer	A maximum distance between groups up to where forces are applied.	**required**, defaults to 1000	1000
groupMinDistanceLimit	Integer	The minimum distance between groups.	**required**, defaults to 50	50
itemLayoutInitialTemp	Integer	The initial temperature for layouts of items within groups.	**required**, defaults to 300	300
itemMaxDistanceLimit	Integer	A maximum distance between items up to where forces are applied.	**required**, defaults to 350	350
itemMinDistanceLimit	Integer	The minimum distance between items.	**required**, defaults to 100	100

3.7 Branding

Map branding (tweaks visuals)

Name	Type	Description	Remarks	Example
mapStylesheet	String	A resolvable URL pointing to a CSS stylesheet. This stylesheet is included in the generated SVG map. Use is to style the appearance of the map.	optional, defaults to null	https://acme.com/css/acme.css

3.8 GroupDescription

A group of items. Could be used as bounded context, for instance.

Name	Type	Description	Remarks	Example
color	String	The HTML (hexcode only!) color used to draw the group and its items. If no color is given, one is computed.	optional, defaults to null	05ffaa
contact	String	A contact method, preferably email.	optional, defaults to null	null
contains	List	A list of item identifiers or SQL-like queries to easily assign items to this group.	optional, defaults to null	identifier LIKE 'DB1'
description	String	A brief description.	optional, defaults to null	null
environment	String		optional, defaults to null	null
identifier	String	A unique identifier for the group (also used as name). Descriptions are merged based on the identifier.	**required**, defaults to null	shipping
labels	Map		optional, defaults to null	null
links	Map<Link>	Key-value pairs of related links. Some keys like 'github' cause that the endpoint data is parsed and added to to corresponding landscape component.	optional, defaults to null	github: https://github.com/dedica-team/nivio
name	String		optional, defaults to null	null
owner	String	The business owner of the group.	optional, defaults to null	null

3.9 ItemDescription

List of configuration sources. Handled in the given order, latter extend/overwrite earlier values like items etc.

Name	Type	Description	Remarks	Example
address	String	The technical address of the item (should be an URI). Taken into account when matching relation endpoints.	optional, defaults to null	null
color	String	Overrides the group color. Use an HTML hex color code without the leading hash.	optional, defaults to null	4400FF
contact	String	The primary way to contact a responsible person or team. Preferably use an email address.	optional, defaults to null	johnson@acme.com
description	String	A brief description.	optional, defaults to null	null
frameworks	Map	The parts used to create the item. Usually refers to technical frameworks.	optional, defaults to null	java: 8
group	String	The identifier of the group this item belongs in. Every item requires to be member of a group internally, so if nothing is given, the value is set to its layer.	optional, defaults to null	shipping
icon	String	An icon name or URL to set the displayed map icon. The default icon set is https://materialdesignicons.com/ and all names can be used (aliases do not work).	optional, defaults to null	null
identifier	String	Immutable unique identifier (maybe use an URN). Primary means to identify items in searches.	**required**, defaults to null	null
interfaces	Set<InterfaceDescription>	A collection of low level interfaces. Can be used to describe HTTP API endpoints for instance.	optional, defaults to null	null
labels	Map		optional, defaults to null	null
layer	String	The technical layer	optional, defaults to null	infrastructure
lifecycle	String	The lifecycle state of an item.	optional, defaults to null	null
links	Map<Link>	Key-value pairs of related links. Some keys like 'github' cause that the endpoint data is parsed and added to to corresponding landscape component.	optional, defaults to null	github: https://github.com/dedica-team/nivio
name	String	A human readable name/title. The name is considered when items are searched.	optional, defaults to null	my beautiful service
owner	String	The business owner of the item. Preferably use an email address.	optional, defaults to null	johnson@acme.com
providedBy	List	A collection of identifiers which are providers for this item (i.e. hard dependencies that are required). This is a convenience field to build relations.	optional, defaults to null	shipping-mysqldb
status	List<map>	A list of statuses that works like hardcoded KPIs.	optional, defaults to null	null
statuses	List<map>	A list of statuses that works like hardcoded KPIs.	optional, defaults to null	null
tags	List		optional,	null

3.10 InterfaceDescription

Describes a low-level interface of an item.

Name	Type	Description	Remarks	Example
deprecated	Boolean		optional, defaults to null	null
description	String	A brief description.	optional, defaults to null	null
format	String	The payload format.	optional, defaults to null	null
name	String		optional, defaults to null	null
parameters	String		optional, defaults to null	null
path	String		optional, defaults to null	null
payload	String		optional, defaults to null	null
protection	String	A description of the interface protection method.	optional, defaults to null	null
summary	String		optional, defaults to null	null
url	String	A URL describing the endpoint.	optional, defaults to null	null

3.11 Link

A link to an external resource. Contains a href (URL) plus various attributes for authentication and/or hateoas.

Name	Type	Description	Remarks	Example
basicAuthPassword	String		optional, defaults to null	null
basicAuthUsername	String		optional, defaults to null	null
deprecation	String	deprecation info (typically used in OpenAPI specs)	optional, defaults to null	null
headerTokenName	String		optional, defaults to null	null
headerTokenValue	String		optional, defaults to null	null
href	String	The link target.	**required**, defaults to null	null
hreflang	String	hateoas language	optional, defaults to null	null
media	String	hateoas media type	optional, defaults to null	null
name	String	HateOAS / OpenAPI name	optional, defaults to null	null
rel	String	hateoas relation type	optional, defaults to null	null
title	String	hateoas title	optional, defaults to null	null
type	String		optional, defaults to null	null

Plus, there are labels having a special meaning:

- `capability` The capability the service provides for the business or, in case of infrastructure, the technical capability like enabling service discovery, configuration, secrets, or persistence.
- `color` A hex color code (items inherit group colors as default)
- `costs` Running costs of the item.
- `fill` Background image (for displaying purposes).
- `frameworks` A comma-separated list of frameworks as key-value pairs (key is name, value is version).
- `health` Description of the item's health status.
- `icon` Icon/image (for displaying purposes).
- `label` A custom label (like a note, but very short).
- `lifecycle` A lifecycle phase (`PLANNED|plan`, `INTEGRATION|int`, `PRODUCTION|prod`, `END_OF_LIFE|eol|end`).
- `note` A custom note.
- `scale` Number of instances.
- `security` Description of the item's security status.
- `shortname` Abbreviated name.
- `software` Software/OS name.
- `stability` Description of the item's stability.
- `team` Name of the responsible team (e.g. technical owner).
- `version` The version (e.g. software version or protocol version).
- `visibility` Visibility to other items.
- `weight` Importance or relations. Used as factor for drawn width if numbers between 0 and 5 are given.

You can also store **custom properties** as labels, but keep in mind that

- label keys are converted to lowercase and
- label values are stored as string.

Item configuration

```yaml
items:
  - identifier: blog-server
    shortName: blog1
    group: content
    mycustomlabel1: foo
    mycustomlabel_2: bar
    any: entry is stored as label
    frameworks:
      php: 7.1

  - identifier: auth-gateway
    shortName: blog1
    layer: ingress
    group: content

  - identifier: DB1
    software: MariaDB
    version: 10.3.11
```

(continues on next page)

(continued from previous page)

```
    type: database
    layer: infrastructure
```

3.12 Item Groups

Groups can have the following attributes:

- `identifier`: A unique identifier in the landscape. Provided automatically via the dictionary key, so do not set it.
- `contains` Array of references to other items (identifiers and CQN queries).
- `owner` Owning party (e.g. marketing).
- `description` A short description.
- `team` Technical owner.
- `contact` Support/notification contact (email). May be addressed in case of errors.
- `color` A hex color code for rendering.
- `links` A map/dictionary of URLs to more information.

Group configuration

```
groups:
  content:
    description: All services responsible to provide information on the web.
    owner: Joe Armstrong
    team: Team Content
    contact: joe@acme.org
    color: "#345345"
    links:
      wiki: http://wiki.acme.org/teamContent

  infrastructure:
    team: Admins
```

3.13 Item Identification and Referencing

An item can be uniquely identified by its landscape, its group, and its identifier. A fully qualified identifier is composed of these three: `mylandscape`, `agroup`, and `theitem`. Since the group is optional, items with unique identifier can also be addressed using `mylandscape` and `theitem`, or just `theitem`. Nivio tries to resolve the correct item and raises an error if it cannot be found or the result is ambiguous.

Service references are required to describe a `provider` relation or `dataflow`.

```
items:
  - identifier: theservice
    group: agroup
    relations:
      - target: anothergroup/anotherservice
        format: json
```

(continues on next page)

(continued from previous page)

```
7           type: dataflow
8           label: Data Sync
```

CHAPTER 4

Data Assessment using KPIs

KPIs (Key Performance Indicators) can be used to evaluate landscape components (typically items, but also groups) based on their properties. The result is a status represented by colors (ordinal):

- UNKNOWN (order 0): status could not be determined
- GREEN (order 1): everything OK
- YELLOW (order 2): ignorable warning
- ORANGE (order 3): warning
- RED (order 4): error
- BROWN (order 5): fubar

4.1 Built in KPIs

4.1.1 Scaling

This KPI evaluates the scale label and tries to find bottlenecks where providers for many items are down or not scaled.

- red if 0 as provider for other items
- yellow if scaled to 0 without relations
- orange of scaled to 0 as data sink
- unknown if no label or not a number
- green if scaled higher than 1
- yellow if a bottleneck (more than 1 item depends on it)

4.1.2 Lifecycle

This KPI evaluates the lifecycle label for "official" values.

- PRODUCTION turns the KPI value to GREEN
- END_OF_LIFE turns it to ORANGE

4.1.3 Other

- health (examines the health label on items)
- condition (K8s condition true/false evaluation)

By default all shipped *KPIs (Key Performance Indicators)* are disabled. Set `enabled` to true in the config to enable them.

```
identifier: kpi_example

config:
  kpis:
    lifecycle:
      enabled: true
    scaling:
      enabled: true
```

4.2 Custom KPIs

Custom KPIs can be configured in the landscape config using ranges and/or matchers (regular expressions) and applied to everything having labels. In the example below a KPI `monthlyCosts` is defined, using ranges on the label `costs`, and the KPI `myEval` evaluates a label `foo`.

- Both ranges (inclusive lower and upper limits) and matchers are separated by semicolon.
- The displayed message can be customized by a template. The placeholder for the value is '%s'.

```
identifier: kpi_example
name: Using KPIs for data assessment

config:
  kpis:
    monthlyCosts:
      description: Evaluates the monthly maintenance costs
      label: costs
      messageTemplate: "Monthly costs: $%s"
      ranges:
        GREEN: 0;99.999999
        YELLOW: 100;199.999999
        RED: 200;499.999999
        BROWN: 500;1000000
    myEval:
      description: evaluate the label "foo"
      label: foo
      matches:
        GREEN: "OK;good;nice"
        RED: "BAD;err.*"
```

(continues on next page)

(continued from previous page)

```
    health:
      description: can be overridden
```

The pet clinic demo uses a custom KPI which evaluates radiation levels. In this simulation a sensor item (see xray group) collects a made up radiation (in mrem) in a label also named `radiation`. This label is then examined by the custom KPI. See https://github.com/dedica-team/nivio/blob/develop/src/test/resources/example/pet_clinic.yml

CHAPTER 5

Shortcuts and convenience functions

5.1 Assigning items to groups

Often lots of items can be read from input data sources, but no information on logical grouping is available. To mitigate that, you can describe groups and use the `contains` field:

- To pick items by their identifier, add single strings which are treated as identifiers.
- Furthermore you can use SQL-like WHERE conditions to assign items to groups. In the following example `identifier LIKE 'DB1%'` is the query which would match both items.

```yaml
items:
  - identifier: DB1-gateway
    shortName: blog1
    layer: ingress

  - identifier: DB1
    software: MariaDB
    version: 10.3.11
    type: database
    layer: infrastructure

groups:
  infrastructure:
    team: Admins
    contains:
      - DB1
      - "identifier LIKE 'DB1%'"
```

5.2 Using Templates to dynamically assign data

To prevent repetitive configuration of items, i.e. entering the same owner again and again, templates can be used to prefill values. Templates are just item descriptions, except that the identifier is used for referencing and that names are

ignored. A template value is only applied if the target value is null.

Multiple templates can be assigned to items too. In this case the first assigned value "wins" and will not be overwritten by templates applied later.

```
identifier: nivio:example
name: Landscape example

sources:
  - url: "./items/docker-compose.yml"
    format: docker-compose-v2
    assignTemplates:
      endOfLife: [web]
      myGroupTemplate: ["*"]

templates:

  myGroupTemplate:
    group: billing

  endOfLife:
    tags: [eol]
```

For CQ queries read https://github.com/npgall/cqengine#string-based-queries-sql-and-cqn-dialects.

5.3 Using Labels to assign data

You can set labels (string:string) to items which are evaluated as model fields if

- the key contains `nivio.` **AND**
- the rest of the key equals a field name.

Labels can be set using docker-compose files too. However, docker labels do not allow arrays, so use comma separated strings:

```
services:
  foo:
    labels:
      nivio.name: A nice name
      nivio.providedBy: "bar, baz"
      nivio.relations: "atarget, anotherTarget"
      nivio.link.repo: "https://github.com/foo/bar"
```

Remember to escape URLs with double quotes.

5.4 Relations between landscape items

Usually environments such as Docker or K8s provide few to none information on the relation between landscape items (e.g. which database a service uses). However, in 12-factor apps there is configuration through environment variables (https://12factor.net/config) and these can be parsed. Nivio provides an experimental feature which regards these variables as DSL (???). They are read and assigned as item labels, then examined:

- The key is split using the underscore character.

- If it contains parts like `url`, `uri`, `host` etc., the label is taken into account as **identifier**, i.e. Nivio looks for a target having the identifier, name, or address equal to the value.

Labels are examined as follows:

- In the case of being an URI, the host and name path components are extracted and used as names or identifiers.

To prevent false positives, certain labels can be omitted:

```
identifier: some-landscape

items:
  - identifier: foo
    labels:
      HOST: bar
      SOME_LABEL: mysql://ahost/foobar

  - identifier: bar
    type: database
```

CHAPTER 6

Output

6.1 Searching

Nivio indexes all landscape items in an in-memory search engine called Lucene. You can build sophisticated queries on various item fields (see *Model and Syntax*). For further information see https://www.lucenetutorial.com/lucene-query-syntax.html

6.2 Modifying Item Appearance

6.2.1 Icons by Type

The icon of an item is determined by its item type (e.g. server, database, . . .) and defaults to a cog symbol.

```
items:
  - identifier: bar
    type: database
```

As type values all items from https://materialdesignicons.com/ can be chosen. Just add the icon name without the "SVG" suffix, like "account".

```
items:
  - identifier: bar
    type: account
```

Alternatively you can use any icon name on the icon field.

```
items:
  - identifier: bar
    icon: flash-circle
```

6.2.2 Vendor Logos

The *icon* property can also work with a predefined vendor name, like "redis", prefixed with `vendor://` as scheme.

Vendor icons are work in progress.

```
items:
  - identifier: bar
    icon: vendor://redis
```

To change the appearance of an item to a vendor logo the `icon` or `fill` properties can be set. Both properties take a valid URL.

6.2.3 External Images

To include external images in the map, just set the `icon` property (or *fill*) to a valid URL.

```
items:
  - identifier: foo
    icon: http://my.custom/icon.png
```

6.2.4 Background fill

While icon (see *External Images* above) is rendered as centered image on the node, fill is used to paint the entire background and is more suitable to be used with images, photos, and so on.

```
items:
  - identifier: bar
    fill: http://my.custom/background.png
```

6.2.5 UTF-8 Symbols and shortname as Icons

If **NO** icon, type, or fill value is set, but a shortname value is given, the value is displayed on the icon. The first example would display `FOOBAR` on the item and the second an enlarged unicorn symbol (shortnames less than three characters are enlarged).

```
items:
  - identifier: bar
    shortname: FOOBAR
  - identifier: pony
    shortname:
```

CHAPTER 7

Custom(er) Branding

The appearance of rendered maps can be altered to match corporate identities. When an SVG map is created, Nivio tries to load and include custom CSS from a URL which can be configured in the landscape configuration. Furthermore, a logo can be included. A logo is configured in the landscape config and must be a URL pointing to an includable file.

```
identifier: branded_landscape
name: branded

config:
  branding:
    mapStylesheet: https://acme.com/css/acme.css
    mapLogo: https://acme.com/images/logo.png

items:
  ...
```

You can also apply custom colors to the user interface. Set the following environment variables to hex values (e.g. `#234234`):

- *NIVIO_BRANDING_FOREGROUND* to set the primary color for interface elements
- *NIVIO_BRANDING_BACKGROUND* for the background color (dark grey is default)
- *NIVIO_BRANDING_SECONDARY* to set the accent color used for active elements

CHAPTER 8

Troubleshooting

8.1 Behind a proxy

If you deploy Nivio to run under a different path than root (`/`), make sure to set the environment variables `SERVER_SERVLET_CONTEXT_PATH` and *`NIVIO_BASE_URL`* to the path.

```
SERVER_SERVLET_CONTEXT_PATH: /my-landscape
NIVIO_BASE_URL: https://foo.com/my-landscape/
```

8.2 Graph Layout Tweaking

In rare cases the layout needs some manual improvements. Internally Nivio uses a forced directed layout, which can be influenced by tweaking some parameters (although mxgraph is not used anymore, for further explanation see https://jgraph.github.io/mxgraph/java/docs/com/mxgraph/layout/mxFastOrganicLayout.html). In order to change the default setting of the LayoutConfig, add a section to the landscape description as follows:

```
identifier: nivio:example
name: Landscape example
config:
  layoutConfig:
    itemMinDistanceLimit: 60
    itemMaxDistanceLimit: 360
    groupMinDistanceLimit: 140
    groupMaxDistanceLimit: 300
    itemLayoutInitialTemp: 380
    groupLayoutInitialTemp: 1000
```

CHAPTER 9

References

Similar approaches can be found at our system graph collection.

Nivio has been inspired by pivio and uses similar semantics, but has a different focus.

Index