Kubernetes Web View

Dec 23, 2019

Contents:

Kubernetes Web View allows to list and view all Kubernetes resources (incl. CRDs) with permalink-friendly URLs in a plain-HTML frontend. This tool was mainly developed to provide a web-version of kubectl for troubleshooting and supporting colleagues, see also *Vision & Goals*.

Git repo: https://codeberg.org/hjacobs/kube-web-view (also mirrored to https://github.com/hjacobs/kube-web-view)

Live demo: https://kube-web-view.demo.j-serv.de/

CHAPTER 1

Getting Started

You can find example Kubernetes manifests for deployment in the deploy folder. You need a running Kubernetes cluster (version 1.10+) and `kubectl` correctly configured. A local test cluster with Minikube or kind will also work. It should be as simple as:

```
$ git clone https://codeberg.org/hjacobs/kube-web-view
$ kubectl apply -f kube-web-view/deploy
```

Afterwards you can open "kube-web-view" via kubectl port-forward (you might need to wait a bit for the pod to become ready):

```
$ kubectl port-forward service/kube-web-view 8080:80
```

Now direct your browser to http://localhost:8080/

Note that pod container logs and Kubernetes secrets are hidden by default for security reasons, you can enable them by uncommenting the respective CLI options in `kube-web-view/deploy/deployment.yaml`. See also *Security Considerations*.

For guidance on setting up Kubernetes Web View for your environment, read the *Setup* section.

CHAPTER 2

Vision & Goals

"kubectl for the web!"

Kubernetes Web View's goal is to provide a no-frills HTML frontend for listing and inspecting K8s objects in troubleshooting and incident response scenarios.

The main audience of Kubernetes Web View is experienced "power" users, on-call/SREs, and cluster operators. Understanding Kubernetes concepts and resources is expected.

The focus on troubleshooting and "kubectl on the web" led to the following design principles and goals:

- enable all (read-only) operations where people commonly use `kubectl` as their tool of choice
- all URLs should represent the full view state (permalinks) in order to make them shareable among colleagues and facilitate deep-linking from other tools
- all Kubernetes objects should be supported to be able to troubleshoot any kind of problem
- resource lists should be easily downloadable for further processing (spreadsheet, CLI tools like `grep`) and storage (e.g. for postmortems)
- selecting resources by label (similar to `kubectl get .. -l`) should be supported
- composing views of different resource types should be possible (similar to `kubectl get all`) to provide a common operational picture among colleagues (e.g. during incident response)
- adding custom "smart" deep links to other tools such as monitoring dashboards, logging providers, application registries, etc should be possible to facilitate troubleshooting and incident response
- keep the frontend as simple as possible (pure HTML) to avoid accidental problems, e.g. unresponsive JavaScript
- support multiple clusters to streamline discovery in on-call situations (only one entry URL to remember)
- facilitate ad-hoc analysis where possible (e.g. with download links for resources across clusters/namespaces)
- provide additional deep-linking and highlighting, e.g. to point colleagues to a certain part of a resource spec (line in YAML)
- allow customization for org-specific optimizations: e.g. custom view templates for CRDs, custom table views, custom CSS formatting

- provide means to continue investigation on the command line (e.g. by showing full `kubectl` command lines to copy)

Out-of-scope (non-goals) for Kubernetes Web View are:

- abstracting Kubernetes objects
- application management (e.g. managing deployments, Helm Charts, etc)
- write operations (this should be done via safe CI/CD tooling and/or GitOps)
- fancy UI (JavaScript, theming, etc)
- visualization (check out kube-ops-view)
- cost analysis (check out kube-resource-report)

CHAPTER 3

Features

3.1 Multiple Clusters

Kubernetes Web View can access one or more clusters via different methods:

- In-cluster authorization via ServiceAccount: this is the default mode when deploying kube-web-view to a single cluster
- Static list of cluster API URLs passed via the `--clusters` CLI option, e.g. `--clusters=myprodcluster=https://kube-prod.example.org; mytestcluster=https://kube-test.example.org`
- Clusters defined in kubeconfig file: kube-web-view will pick up all contexts defined in the kubeconfig file (`~/.kube/config` or path given via `--kubeconfig-path`). To only show some clusters, limit the kubeconfig contexts via the `--kubeconfig-contexts` command line option.
- Clusters defined in a cluster registry REST API: kube-web-view supports a custom REST API to discover clusters. Pass the URL via `--cluster-registry-url` and create a file with the OAuth2 Bearer token (`--cluster-registry-oauth2-bearer-token-path`). See the example Cluster Registry REST API.

See also *Multiple Clusters*.

3.2 Listing Resources

Kubernetes Web View can list all Kubernetes resource types:

- non-namespaced cluster resources under `/clusters/{cluster}/{plural}`
- namespaced resources under `/clusters/{cluster}/namespaces/{namespace}/{plural}`

Multiple resource types can be listed on the same page by using their comma-separated plural resource names, e.g. to list deployments and ingresses on the same page: `/clusters/{cluster}/namespaces/{namespace}/deployments,ingresses`. Try out the live demo with deployments and ingresses on the same page.

To list resources across all namespaces, use `_all` for the namespace name in the URL.

Resources can be listed across all clusters by using `_all` for the cluster name in the URL.

Resources can be filtered by label: use the `selector` query parameter with label key=value pairs.

To facilitate processing in spreadsheets or command line tools (`grep`, `awk`, etc), all resource listings can be downloaded as tab-separated-values (TSV). Just append `download=tsv` to the URL.

Columns can be customized via the `labelcols` and `customcols` query parameters:

- `labelcols` is either a comma separated list of label names or "*" to show all labels
- `customcols` is a semicolon-separated list of Name=spec pairs, where "Name" is an arbitrary column name string and "spec" is a JMESPath expression: e.g. `Images=spec.containers[*].image` would show the container images in the "Images" column. Note that the semicolon to separate multiple custom columns must be urlencoded as `%3B`.
- `hidecols` is a comma separated list of column names to hide or "*" to hide all columns (label and custom columns will be added after the hide operation)

The `limit` query parameter can optionally limit the number of shown resources.

3.3 Searching

Any resource type can be searched by name and/or label value across clusters and namespaces. While Kubernetes Web View does not maintain its own search index, searches across clusters and resource types are done in parallel, so that results should be returned in a reasonable time. Please note that the search feature might produce (heavy) load on the queried Kubernetes API servers.

3.4 Viewing Resources

Object details are available via `/clusters/{cluster}/{resource-type}/{name}` for cluster resources and `/clusters/{cluster}/namespaces/{namespace}/{resource-type}/{name}` for namespaced resources. Object details are either rendered via HTML or can be viewed as their YAML source. Resources can also be downloaded as YAML.

To make it easier to point colleagues to a specific portion of a resource spec, the YAML view supports linking and highlighting individual lines. Just click on the respective line number.

3.5 Container Logs

Kubernetes Web View supports rendering pod container logs for individual pods and any resource spec with `matchLabels`, i.e. Deployments, ReplicaSets, DaemonSets, and StatefulSets. Just use the "Logs" tab or append `/logs` to the resource URL.

Note that container logs are disabled by default for security reasons, enable them via `--show-container-logs`.

3.6 Custom Resource Definitions (CRDs)

Kubernetes Web View automatically works for your CRDs. The list (table) view will render similar to the output of `kubectl get ..`, i.e. you can customize displayed table columns by modifying the

`additionalPrinterColumns` section of your CRD section. See the official Kubernetes docs on additional printer columns for details.

3.7 OAuth2

The web frontend can be secured via the builtin OAuth2 Authorization Grant flow support, see the *OAuth2 Support* section for details.

CHAPTER 4

Setup

This section guides through the various options of setting up Kubernetes Web View in your environment.

- Do you want to use kube-web-view as a local development/ops tool? See *Local Usage*
- Do you want to use it in a single cluster or access multiple clusters via kube-web-view? See *Single Cluster* or *Multiple Clusters*.
- How do you plan to secure your setup and authenticate users? See *Access Control*.
- Should users see everything in your cluster(s) or only some namespaces? See *Namespace Access*.
- Do you want to customize behavior and look & feel for your organization? See *Customization*.
- Please make sure to read the *Security Considerations*.

4.1 Local Usage

Kubernetes Web View was primarily built for a (central) deployment, but you can run it locally with your existing Kubeconfig file (default location is `~/.kube/config`). This will automatically pick up all contexts defined in Kubeconfig, i.e. works with single or multiple clusters:

```
docker run -it -p 8080:8080 -u $(id -u) -v $HOME/.kube:/.kube hjacobs/kube-web-view
```

Open http://localhost:8080/ in your browser to see the UI.

Note that Kubernetes Web View does not support all different proprietary authentication mechanisms (like EKS, GCP), you can use "kubectl proxy" as a workaround:

```
kubectl proxy --port=8001 &  # start proxy in background
docker run -it --net=host -u $(id -u) hjacobs/kube-web-view --clusters=local=http://
↪localhost:8001
```

If you are using Docker for Mac, this needs to be slightly different in order to navigate the VM/container inception:

```
$ kubectl proxy --accept-hosts '.*' --port=8001 &
$ docker run -it -p 8080:8080 hjacobs/kube-web-view --clusters=local=http://docker.
↪for.mac.localhost:8001
```

Now direct your browser to http://localhost:8080

4.2 Single Cluster

Deploying Kubernetes Web View to a single cluster is straightforward as it will use RBAC and in-cluster ServiceAccount to talk with the Kubernetes API server:

```
kubectl apply -f deploy/
```

You can now use "kubectl port-forward service/kube-web-view 8080:80" to access the UI on http://localhost:8080/ or expose kube-web-view with a LB/Ingress. See *Access Control*.

4.3 Multiple Clusters

Kubernetes Web View can access multiple clusters via different methods:

- Static list of cluster API URLs passed via the `--clusters` CLI option, e.g. `--clusters=myprodcluster=https://kube-prod.example.org;mytestcluster=https://kube-test.example.org`
- Clusters defined in kubeconfig file: kube-web-view will pick up all contexts defined in the kubeconfig file (`~/.kube/config` or path given via `--kubeconfig-path`). To only show some clusters, limit the kubeconfig contexts via the `--kubeconfig-contexts` command line option. This behavior is the same as for *Local Usage*.
- Clusters defined in a cluster registry REST API: kube-web-view supports a custom REST API to discover clusters. Pass the URL via `--cluster-registry-url` and create a file with the OAuth2 Bearer token (`--cluster-registry-oauth2-bearer-token-path`). See the example Cluster Registry REST API.

Kubernetes Web View will access the Kubernetes API differently, depending on the configuration:

- when using `--clusters`: no authentication method (or token from `--cluster-auth-token-path`, or session token if `--cluster-auth-use-session-token` is set)
- when using `--kubeconfig-path`: try to use the authentication method defined in the Kubeconfig file (e.g. client certificate)
- when using `--cluster-registry-url`: use the Cluster Registry Bearer token from `--cluster-registry-oauth2-bearer-token-path`
- when using `--cluster-auth-token-path`: load the access token from the given file and use it as "Bearer" token for all Kubernetes API calls — this overwrites any of the above authentication methods
- when using `--cluster-auth-use-session-token`: use the OAuth session token as "Bearer" token for the Kubernetes API — this overwrites any other authentication method and only works when *OAuth2 Support* is enabled

You can also combine the `--clusters` option with `kubectl proxy` to access clusters which have an unsupported authentication method:

- start `kubectl proxy --port=8001` in a sidecar container

- run the kube-web-view container with the `--clusters=mycluster=http://localhost:8001` argument

You can use `--cluster-auth-token-path` to dynamically refresh the Bearer access token in the background. This is useful if you need to rotate the token regularly (e.g. every hour). Either run a sidecar process with a shared volume (e.g. "emptyDir") to write/refresh the token or mount a Kubernetes secret into kube-web-view's container at the given path.

4.4 Access Control

There are multiple options to secure your Kubernetes Web View deployment:

- Internal service without LoadBalancer/Ingress: this requires `kubectl port-forward service/kube-web-view 8080:80` to access the web UI. This is the easiest option to set up (no LB/Ingress/proxy/OAuth required), but inconvenient to use.
- Using a custom LB/proxy: you can expose the kube-web-view frontend through a custom proxy (e.g. nginx with ACLs, AWS ALB with authorization, etc). The setup highly depends on your environment and infrastructure.
- Using the built-in OAuth support: kube-web-view has support for the authorization grant OAuth redirect flow which works with common OAuth providers such as Google, GitHub, Cognito, and others. See *OAuth2 Support* on how to configure OAuth in Kubernetes Web View.

4.5 Namespace Access

Kubernetes Web View allows to limit namespace access with include and exclude patterns, examples:

- use `--include-namespaces=default,dev` to only allow access to the "default" and "dev" namespaces
- use `--exclude-namespaces=kube-.*` to deny access to all "kube-.*" (system) namespaces

Users can still access the "_all" namespaced resource lists and search across namespaces, but objects for non-allowed namespaces will be filtered out. You can use this feature to give users a more streamlined experience by hiding infrastructure namespaces (e.g. "kube-system") from them.

Note that `--exclude-namespaces` always takes precedence over `--include-namespaces`, i.e. you can include all "foo-.*" namespaces (`--include-namespaces=foo-.*`) and exclude only "foo-bar" via (`--exclude-namespaces=foo-bar`).

Please use Kubernetes RBAC roles for proper access control, kube-web-view's namespace filtering is just another layer of protection. Please also read the *Security Considerations*.

CHAPTER 5

OAuth2 Support

Kubernetes Web View support OAuth2 for protecting its web frontend. Use the following environment variables to enable it:

OAUTH2_AUTHORIZE_URL OAuth 2 authorization endpoint URL, e.g. https://oauth2.example.org/authorize

OAUTH2_ACCESS_TOKEN_URL Token endpoint URL for the OAuth 2 Authorization Code Grant flow, e.g. https://oauth2.example.org/token

OAUTH2_CLIENT_ID OAuth 2 client ID

OAUTH2_CLIENT_ID_FILE Path to file containing the client ID. Use this instead of `OAUTH2_CLIENT_ID` to read the client ID dynamically from file.

OAUTH2_CLIENT_SECRET OAuth 2 client secret

OAUTH2_CLIENT_SECRET_FILE Path to file containing the client secret. Use this instead of `OAUTH2_CLIENT_SECRET` to read the client secret dynamically from file.

SESSION_SECRET_KEY Secret to encrypt the session cookie. Must be 32 bytes base64-encoded. Use `cryptography.fernet.Fernet.generate_key()` to generate such a key.

The OAuth2 login flow will (by default) just protect the web frontend, the configured credentials (in-cluster Service Account, Kubeconfig, or Cluster Registry) will be used to access the cluster(s). This behavior can be changed and the session's OAuth2 access token can be used for cluster authentication instead of using configured credentials. Enable this operation mode via `--cluster-auth-use-session-token`.

The OAuth redirect flow will not do any extra authorization by default, i.e. everybody who can login with your OAuth provider can use Kubernetes Web View! You can plug in a custom Python hook function (coroutine) via `--oauth2-authorized-hook` to validate the login or do any extra work (store extra info in the session, deny access, log user ID, etc). Note that the hook needs to be a coroutine function with signature like `async def authorized(data, session)`. The result should be boolean true if the login is successful, and false otherwise. Examples of such hooks are provided in the examples directory. A minimal `hooks.py` would look like:

```
import logging

async def oauth2_authorized(data: dict, session):
```

(continues on next page)

(continued from previous page)

```
    access_token = data["access_token"]
    # TODO: do something with the access token, e.g. look up user info
    logging.info("New OAuth login!")
    # TODO: validate whether login is allowed or not
    return True  # allow all OAuth logins
```

This file would need to be in the Python search path, e.g. as `hooks.py` in the root ("/") of the Docker image. Pass the hook function as `--oauth2-authorized-hook=hooks.oauth2_authorized` to Kubernetes Web View.

5.1 Google OAuth Provider

This section explains how to use the Google OAuth 2.0 provider with Kubernetes Web View:

- follow the instructions on https://developers.google.com/identity/protocols/OAuth2 to obtain OAuth 2.0 credentials such as client ID and client secret
- use `https://{my-kube-web-view-host}/oauth2/callback` as one of the **Authorized redirect URIs** in the Google API Console
- use "https://accounts.google.com/o/oauth2/v2/auth?scope=email" for `OAUTH2_AUTHORIZE_URL`
- use "https://oauth2.googleapis.com/token" for `OAUTH2_ACCESS_TOKEN_URL`
- pass the obtained client ID in the `OAUTH2_CLIENT_ID` environment variable
- pass the obtained client secret in the `OAUTH2_CLIENT_SECRET` environment variable

5.2 GitHub OAuth Provider

How to use GitHub as the OAuth provider with Kubernetes Web View:

- create a new OAuth app in the GitHub UI
- use `https://{my-kube-web-view-host}/oauth2/callback` as the **Authorization callback URL** in the GitHub UI
- use "https://github.com/login/oauth/authorize" for `OAUTH2_AUTHORIZE_URL`
- use "https://github.com/login/oauth/access_token" for the `OAUTH2_ACCESS_TOKEN_URL`
- pass the obtained client ID in the `OAUTH2_CLIENT_ID` environment variable
- pass the obtained client secret in the `OAUTH2_CLIENT_SECRET` environment variable

Note that any GitHub user can now login to your deployment of Kubernetes Web View! You have to configure a `--oauth2-authorized-hook` function to validate the GitHub login and only allow certain usernames:

- copy `hooks.py` from `examples/oauth2-validate-github-token/hooks.py` (see examples dir) to a new folder
- customize the username in `hooks.py` to match your allowed GitHub user logins
- create a new `Dockerfile` in the same folder
- edit the `Dockerfile` to have two lines: 1) `FROM hjacobs/kube-web-view:{version}` (replace "{version}"!) as the first line, and 2) `COPY hooks.py /` to copy our OAuth validation function
- build the Docker image

- configure your kube-web-view deployment and add `--oauth2-authorized-hook=hooks.oauth2_authorized` as argument
- deploy kube-web-view with the new Docker image and CLI option

5.3 AWS Cognito Provider

5.3.1 Setting up Cognito

A number of steps need to be taken to setup Amazon Cognito for OAuth2. These instructions are correct as of August 2019.

Create User Pool

1. Create a User Pool
2. Choose how you want End Users to sign in (for example via Email, Username or otherwise)
3. Once you have gone through all the settings (customise to your liking) for creating a user pool, add an App Client

Create an App Client

1. Choose a Name that is relevant to the application (eg kube-web-view)
2. Make sure the **Generate client secret** option is selected, and set your **Refresh token expiration** time to whatever you are comfortable with.

The App Client will then generate a Client ID and Client Secret, wich will be used later

App Client Settings

1. Select the previously created client
2. Fill in the **Callback URL(s)** section with `https://{my-kube-web-view-host}/oauth2/callback`
3. Under **OAuth 2.0**, choose the relevant **Allowed OAuth Flows** (eg *Authorization Code Grant*, *Implicit Grant*)
4. Choose the **Allowed OAuth Scopes** you want to include. *email* is the minimum you will need

IMPORTANT: Domain Name

You must create a domain name for OAuth to function against AWS Cognito, otherwise the required Authorization and Token URLs will not be exposed.

You can choose whether to use an AWS-hosted Cognito Domain (eg `https://{your-chosen-domain}.auth.us-east-1.amazoncognito.com`), or to use your own domain.

Update Deployment

You can now update your Deployment with the relevant Environment variables. If you have chosen to use an AWS Cognito Domain, then the `{FQDN}` variable in the below section will be `https://{your-chosen-domain}.auth.{aws-region}.amazoncognito.com`. Otherwise, replace it with your domain

- use "https://{FQDN}/oauth2/authorize" for `OAUTH2_AUTHORIZE_URL`
- use "https://{FQDN}/oauth2/token" for `OAUTH2_ACCESS_TOKEN_URL`
- Use the App Client ID generated during "Create an App Client" in the `OAUTH2_CLIENT_ID` environment variable
- Use the App Client secret in the `OAUTH2_CLIENT_SECRET` environment variable. If you cannot see the secret, press "Show Details" in the AWS Console

5.3.2 Terraform

An example Terraform deployment of the above is below:

```
# Create the User Pool
resource "aws_cognito_user_pool" "kube-web-view" {
  name = "userpool-kube-web-view"
  alias_attributes = [
    "email",
    "preferred_username"
  ]

  auto_verified_attributes = [
    "email"
  ]

  schema {
    attribute_data_type      = "String"
    developer_only_attribute = false
    mutable                  = true
    name                     = "name"
    required                 = true

    string_attribute_constraints {
      min_length = 3
      max_length = 70
    }
  }

  admin_create_user_config {
    allow_admin_create_user_only = true
  }

  tags = {
    "Name" = "userpool-kube-web-view"
  }
}

# Create the oauth2 Domain

resource "aws_cognito_user_pool_domain" "kube-web-view" {
  domain = "oauth-kube-web-view"
```

(continues on next page)

(continued from previous page)

```
  user_pool_id = aws_cognito_user_pool.kube-web-view.id
}

# kube-web-view Client

resource "aws_cognito_user_pool_client" "kube-web-view" {
  name = "kube-web-view"
  user_pool_id = aws_cognito_user_pool.kube-web-view.id

  allowed_oauth_flows = [
    "code",
    "implicit"
  ]

  allowed_oauth_scopes = [
    "email",
    "openid",
    "profile",
  ]

  supported_identity_providers = [
    "COGNITO"
  ]

  generate_secret = true

  allowed_oauth_flows_user_pool_client = true

  callback_urls = [
    "https://{my-kube-web-view-host}/oauth2/callback"
  ]
}


# Outputs

output "kube-web-view-id" {
  description = "Kube Web View App ID"
  value = aws_cognito_user_pool_client.kube-web-view.id
}

output "kube-web-view-secret" {
  description = "Kube Web View App Secret"
  value = aws_cognito_user_pool_client.kube-web-view.client_secret
```

CHAPTER 6

Customization

Kubernetes Web View's behavior and appearance can be customized for the needs of your organization:

- resource type links shown in the *Sidebar* can be customized to add CRDs, or to optimize for frequent access
- default *Label & Custom Columns* can be defined to show values of standardized object labels (e.g. "app", "version", etc)
- *External Links* can be added to link objects to monitoring tools, internal application registries, custom UIs, etc
- the *Search* can be customized to search in CRDs, or to cover frequent search cases
- setting *Preferred API Versions* allows forcing the use of specific/newer Kubernetes API versions
- one of the *Themes* can be selected as default, or you can create your own CSS theme
- *HTML Templates* can be customized to match your branding, to add static links, and to inject custom JS/CSS
- *Static Assets* can be included to add images, JS, or CSS files
- *Prerender Hooks* can be used to add or transform view data

6.1 Sidebar

The resource types linked in the left sidebar can be customized, e.g. to include CRDs or to remove resource types which are not frequently accessed.

Example command line argument to show the "StackSet" CRD in the "Controllers" section and to add secrets to "Pod Management":

```
--sidebar-resource-types=Controllers=stacksets,deployments,cronjobs;Pod␣
↪Management=ingresses,services,pods,secrets
```

You can use *HTML Templates* for further customization of the sidebar (e.g. to add non-resource links).

6.2 Label & Custom Columns

Most organizations have a standard set of labels for Kubernetes resources, e.g. all pods might have "app" and "version" labels. You can instruct Kubernetes Web View to show these labels as columns for the respective resource types via the `--default-label-columns` command line option.

Example command line argument to show the "application" and "version" labels for pods and the "team" label for deployments:

```
--default-label-columns=pods=application,version;deployments=team
```

Note that the label names are separated by comma (","), whereas multiple different entries for different resource types are separated by semicolon (";").

Users of the web UI can remove the pre-configured label columns by passing a single comma as the `labelcols` query parameter: `/clusters/../namespaces/_all/pods?labelcols=,`.

You can hide existing columns via the `--default-hidden-columns` command line option, e.g. to remove the "Nominated Node" and "Readiness Gates" columns from pod tables:

```
--default-hidden-columns=pods=Nominated Node,Readiness Gates
```

Arbitrary custom columns can be defined with JMESPath expressions, e.g. add a column "Images" for pods and the column "Strategy" for deployments:

```
--default-custom-columns=pods=Images=spec.containers[*].image;;
↪deployments=Strategy=spec.strategy
```

Multiple column definitions are separated by a single semicolon (";") whereas multiple different entries for different resource types are separated by two semicolons (";;"). Please be aware that custom columns require one additional Kubernetes API call per listing.

6.3 External Links

You can configure external links per resource type or based on certain labels with these two command line options:

`--object-links` Define URL templates per resource type (e.g. to link all pods to a monitoring dashboard per pod)

`--label-links` Define URL templates per label, e.g. to link to an application registry for the "app" label, team overview for a "team" label, etc

The URL templates are Python string format strings and receive the following variables for replacement:

`{cluster}` The cluster name.

`{namespace}` The namespace name of the object.

`{name}` The object name.

`{label}` Only for label links: the label name.

`{label_value}` Only for label links: the label value.

Example command line argument to add links to a monitoring dashboard per pod:

```
--object-links=pods=https://mymonitoringsystem/pod-dashboard?cluster={cluster};
↪namespace={namespace};name={name}
```

Example command line argument to link resources with an "application" label to Kubernetes Resource Report:

```
--label-links=application=https://myresourcereport/application-{label_value}.html
```

Links can optionally specify the icon and link title (tooltip) by appending icon name and title text separated by pipe ("|"):

```
--label-links=application=https://myresourcereport/application-{label_value}.
↪html|file-invoice-dollar|Kubernetes Resource Report
```

Check the Font Awesome Gallery for available icon names (some ideas: "external-link-alt", "eye", "th-large", "search", "tools").

6.4 Search

The default search resource types can be customized, e.g. to include Custom Resource Definitions (CRDs) or to optimize for frequent search patterns. Pass comma-separated lists of resource types (plural name) to the following two command line options:

`--search-default-resource-types` Set the resource types to search by default (when using the navbar search box). Must be a comma-separated list of resource types, e.g. "deployments,pods".

`--search-offered-resource-types` Customize the list of resource types shown on the search page (`/search`). Must be a comma-separated list of resource types, e.g. "deployments,pods,nodes".

Note that all resource types can be searched by using a deep-link, i.e. these options will only restrict what is shown in the HTML UI, but they will not prohibit searching for other resource types.

6.5 Preferred API Versions

You might want to change the default preferred API version returned by the Kubernetes API server. This is useful to force using a later/newer API version for some resources, e.g. the Kubernetes HorizontalPodAutoscaler has a different spec for later versions.

Here the example CLI option to force using new API versions for Deployment and HPA (the default is `autoscaling/v1` as of Kubernetes 1.14):

```
--preferred-api-versions=horizontalpodautoscalers=autoscaling/v2beta2;
↪deployments=apps/v1
```

6.6 Themes

Kubernetes Web View ships with a number of color (CSS) themes. You can choose a default theme for your users via `--default-theme` and/or limit the selection via `--theme-options`. Available themes are:

darkly Flatly in night mode: dark background, blue and green as primary colors, see darkly demo

default Kubernetes Web View default theme: white background, blue as primary color, see default demo

flatly Flat and thick: white background, blue and green as primary colors, see flatly demo

slate Shades of gunmetal grey: dark grey background, grey colors, see slate demo

superhero The brave and the blue: dark background, orange navbar, see superhero demo

You can use one of the Bulmaswatch themes to create your own.

6.7 HTML Templates

Custom Jinja2 HTML templates can override any of the default templates. Mount your custom templates into kube-web-view's pod and point the `--templates-path` to it.

Here some of the common templates you might want to customize:

`base.html` The main HTML layout (contains `<head>` and `<body>` tags).

`partials/extrahead.html` Optional extra content for the `<head>` HTML part. Use this template to add any custom JS/CSS.

`partials/navbar.html` The top navigation bar.

`partials/sidebar.html` Template for the left sidebar, customize this to add your own links. Note that you can change the list of resource types without touching HTML via `--sidebar-resource-types`, see *the sidebar section*.

`partials/footer.html` Footer element at the end of the HTML `<body>`.

You can find all the standard templates in the official git repo: https://codeberg.org/hjacobs/kube-web-view/src/branch/master/kube_web/templates

You can build your own Docker image containing the templates or you can use a volume of type `emptyDir` and some InitContainer to inject your templates. Example pod spec with a custom footer:

```
spec:
  initContainers:
  - name: generate-templates
    image: busybox
    command: ["sh", "-c", "mkdir /templates/partials && echo '<footer class=\"footer\
↪">YOUR CUSTOM CONTENT HERE</footer>' > /templates/partials/footer.html"]
    volumeMounts:
    - mountPath: /templates
      name: templates

  containers:
  - name: kube-web-view
    # see https://codeberg.org/hjacobs/kube-web-view/releases
    image: hjacobs/kube-web-view:latest
    args:
    - --port=8080
    - --templates-path=/templates
    ports:
    - containerPort: 8080
    readinessProbe:
      httpGet:
        path: /health
        port: 8080
    volumeMounts:
    - mountPath: /templates
      name: templates
      readOnly: true
    resources:
      limits:
        memory: 100Mi
```

(continues on next page)

(continued from previous page)

```
      requests:
        cpu: 5m
        memory: 100Mi
    securityContext:
      readOnlyRootFilesystem: true
      runAsNonRoot: true
      runAsUser: 1000
  volumes:
  - name: templates
    emptyDir:
      sizeLimit: 50Mi
```

6.8 Static Assets

As you might want to add or change static assets (e.g. JS, CSS, images), you can point Kubernetes Web View to a folder containing your custom assets. Use the `--static-assets-path` command line option for this and either build a custom Docker image or mount your asset directory into the pod.

6.9 Prerender Hooks

The view data (`context` for Jinja2 template) can be modified by custom prerender hooks to allow advanced customization.

For example, to add generated custom links for deployments to the resource detail view, create a coroutine function with signature like `async def resource_view_prerender(cluster, namespace, resource, context)` in a file `hooks.py`:

```
async def resource_view_prerender(cluster, namespace: str, resource, context: dict):
    if resource.kind == "Deployment":
        link = {
            "href": f"https://cd.example.org/pipelines/{resource.labels['pipeline-id
↪']}/{resource.labels['deployment-id']}",
            "class": "is-link",
            "title": "Pipeline link",
            "icon": "external-link-alt",
        }
        context["links"].append(link)
```

This file would need to be in the Python search path, e.g. as `hooks.py` in the root ("/") of the Docker image. Pass the hook function as `--resource-view-prerender-hook=hooks.resource_view_prerender` to Kubernetes Web View.

Note that you can also do more advanced stuff in the prerender hook, e.g. call out to external systems to look up additional information.

CHAPTER 7

Security Considerations

Kubernetes Web View exposes all Kubernetes object details via its web frontend. There are a number of security precautions to make:

- Do not expose Kubernetes Web View to the public without **authorization** (e.g. OAuth2 redirect flow or some authorizing web proxy).
- The default **RBAC** role for kube-web-view (provided in the `deploy` folder) provides **full read-only access** to the cluster — modify it accordingly to limit the scope.
- Design and **understand your access control**: decide whether you use kube-web-view only locally (with personal credentials), have a central deployment (with service credentials) to multiple clusters, use *OAuth2 Support* for cluster access via `--cluster-auth-use-session-token`, or have it deployed per cluster with limited access.
- Understand the security risks of exposing your cluster details to Kubernetes Web View users — you should only **trust users** who you would also give full read-only access to the Kubernetes API.
- Check your Kubernetes objects for **potential sensitive information**, e.g. application developers might have used container environment variables (`env`) to contain passwords (instead of using secrets or other methods), mitigate accordingly!

Kubernetes Web View tries to have some sane defaults to prevent information leakage:

- Pod container logs are not shown by default as they might contain sensitive information (e.g. access logs, personal data, etc). You have to enable them via the `--show-container-logs` command line flag.
- Contents of Kubernetes secrets are masked out (hidden) by default. If you are sure that you want to show secrets (e.g. because you only run kube-web-view on your local computer (`localhost`)), you can disable this feature via the `--show-secrets` command line flag.

Note that these are just additional features to prevent accidental security issues — **you are responsible for securing Kubernetes Web View appropriately!**

CHAPTER 8

Alternative UIs

Tip: Also check out the blog post about Kubernetes web UIs in 2019 for a look at some different web UIs and why Kubernetes Web View was created.

This page lists a number of alternative, open source UIs for Kubernetes.

8.1 K8Dash

https://github.com/herbrandson/k8dash, web, node.js

> "K8Dash is the easiest way to manage your Kubernetes cluster."

8.2 Konstellate

https://github.com/containership/konstellate, web, Clojure

> "Visualize Kubernetes Applications"

8.3 Kubernetator

https://github.com/smpio/kubernator, web, node.js

> "Kubernator is an alternative Kubernetes UI. In contrast to high-level Kubernetes Dashboard, it provides low-level control and clean view on all objects in a cluster with the ability to create new ones, edit and resolve conflicts. As an entirely client-side app (like kubectl), it doesn't require any backend except Kubernetes API server itself, and also respects cluster's access control."

8.4 Kubernetes Dashboard

https://github.com/kubernetes/dashboard, web

> "Kubernetes Dashboard is a general purpose, web-based UI for Kubernetes clusters. It allows users to manage applications running in the cluster and troubleshoot them, as well as manage the cluster itself."

8.5 Kubernetes Operational View

https://github.com/hjacobs/kube-ops-view, web

> "Read-only system dashboard for multiple K8s clusters"

Uses WebGL to render nodes and pods.

8.6 Kubernetes Resource Report

https://github.com/hjacobs/kube-resource-report/, web

> "Report Kubernetes cluster and pod resource requests vs usage and generate static HTML"

Generates static HTML files for cost reporting.

8.7 Kubricks

https://github.com/kubricksllc/Kubricks, desktop app

> "Visualizer/troubleshooting tool for single Kubernetes clusters"

8.8 Octant

https://github.com/vmware/octant, web, Go

> "A web-based, highly extensible platform for developers to better understand the complexity of Kubernetes clusters."

8.9 Weave Scope

https://github.com/weaveworks/scope, web

> "Monitoring, visualisation & management for Docker & Kubernetes"

CHAPTER 9

Indices and tables

- genindex
- modindex
- search