Generating Reference Documentation for the Kubernetes API

This page shows how to update the Kubernetes API reference documentation.

The Kubernetes API reference documentation is built from the Kubernetes OpenAPI spec using the kubernetes-sigs/reference-docs generation code.

If you find bugs in the generated documentation, you need to fix them upstream.

If you need only to regenerate the reference documentation from the OpenAPI spec, continue reading this page.

Before you begin

Requirements:

- You need a machine that is running Linux or macOS.
- You need to have these tools installed:
  - Python v3.7.x+
  - Git
  - Golang version 1.13+
  - Pip used to install PyYAML
  - PyYAML v5.1.2
  - make
  - gcc compiler/linker
  - Docker (Required only for `kubectl` command reference)
- Your `PATH` environment variable must include the required build tools, such as the `Go` binary and `python`.
- You need to know how to create a pull request to a GitHub repository. This involves creating your own fork of the repository. For more information, see Work from a local clone.

Set up the local repositories

Create a local workspace and set your `GOPATH`:

```
mkdir -p $HOME/<workspace>
```

```
export GOPATH=$HOME/<workspace>
```

Get a local clone of the following repositories:

```
git clone github.com/kubernetes-sigs/reference-docs
```

Move into the `gen-apidocs` directory of the `reference-docs` repository and install the required Go packages:

```
go get -u github.com/go-openapi/loads
go get -u github.com/go-openapi/spec
```

If you don't already have the kubernetes/website repository, get it now:

```
git clone https://github.com/<your-username>/website
```

Get a clone of the kubernetes/kubernetes repository:

```
git clone https://github.com/kubernetes/kubernetes
```

- The base directory of your clone of the kubernetes/kubernetes repository is `<your-path-to>/kubernetes/kubernetes`. The remaining steps refer to your base directory as `<k8s-base>`.
- The base directory of your clone of the kubernetes/website repository is `<your-path-to>/website`. The remaining steps refer to your base directory as `<web-base>`.
- The base directory of your clone of the kubernetes-sigs/reference-docs repository is `<your-path-to>/reference-docs`. The remaining steps refer to your base directory as `<rdocs-base>`.

Generate the API reference docs

This section shows how to generate the published Kubernetes API reference documentation.

Set build variables

- Set `K8S_ROOT` to `<k8s-base>`.
- Set `K8S_WEBROOT` to `<web-base>`.
- Set `K8S_RELEASE` to the version of the docs you want to build. For example, if you want to build docs for Kubernetes 1.17.0, set `K8S_RELEASE` to 1.17.0.

For example:

```
export K8S_WEBROOT=<your-path-to>/website
export K8S_ROOT=<your-path-to>/kubernetes
export K8S_RELEASE=1.17.0
```

Create versioned directory and fetch Open API spec

The `updateapispec` build target creates the versioned build directory. After the directory is created, the Open API spec is fetched from the `<k8s-base>` repository. These steps ensure that the version of the configuration files and Kubernetes Open API spec match the release version. The versioned directory name follows the pattern of `v<major>_<minor>`.

In the `<rdocs-base>` directory, run the following build target:

```
cd <rdocs-base>
make updateapispec
```

Build the API reference docs

The `copyapi` target builds the API reference and copies the generated files to directories in `<web-base>`. Run the following command in `<rdocs-base>`:

```
cd <rdocs-base>
make copyapi
```

Verify that these two files have been generated:

```
[ -e "<rdocs-base>/gen-apidocs/build/index.html" ] && echo "index.html built" || echo "no index.html"
[ -e "<rdocs-base>/gen-apidocs/build/navData.js" ] && echo "navData.js built" || echo "no navData.js"
```

Go to the base of your local `<web-base>`, and view which files have been modified:

```
cd <web-base>
git status
```

The output is similar to:

```
static/docs/reference/generated/kubernetes-api/v1.34/css/bootstrap.min.css
static/docs/reference/generated/kubernetes-api/v1.34/css/font-awesome.min.css
static/docs/reference/generated/kubernetes-api/v1.34/css/stylesheet.css
static/docs/reference/generated/kubernetes-api/v1.34/fonts/FontAwesome.otf
static/docs/reference/generated/kubernetes-api/v1.34/fonts/fontawesome-webfont.eot
static/docs/reference/generated/kubernetes-api/v1.34/fonts/fontawesome-webfont.svg
static/docs/reference/generated/kubernetes-api/v1.34/fonts/fontawesome-webfont.ttf
static/docs/reference/generated/kubernetes-api/v1.34/fonts/fontawesome-webfont.woff
static/docs/reference/generated/kubernetes-api/v1.34/fonts/fontawesome-webfont.woff2
static/docs/reference/generated/kubernetes-api/v1.34/index.html
static/docs/reference/generated/kubernetes-api/v1.34/js/jquery.scrollTo.min.js
static/docs/reference/generated/kubernetes-api/v1.34/js/navData.js
static/docs/reference/generated/kubernetes-api/v1.34/js/scroll.js
```

API reference location and versioning

The generated API reference files (HTML version) are copied to `<web-base>/static/docs/reference/generated/kubernetes-api/v1.34/`. This directory contains the standalone HTML API documentation.

Note:

The Markdown version of the API reference located at `<web-base>/content/en/docs/reference/kubernetes-api/` is generated separately using the gen-resourcesdocs generator.

Locally test the API reference

Publish a local version of the API reference. Verify the local preview.

```
cd <web-base>
git submodule update --init --recursive --depth 1 # if not already done
make container-serve
```

Commit the changes

In `<web-base>`, run `git add` and `git commit` to commit the change.

Submit your changes as a pull request to the kubernetes/website repository. Monitor your pull request, and respond to reviewer comments as needed. Continue to monitor your pull request until it has been merged.

What's next

- Generating Reference Documentation Quickstart
- Generating Reference Docs for Kubernetes Components and Tools
- Generating Reference Documentation for kubectl Commands

Generating Reference Documentation for kubectl Commands

This page shows how to generate the `kubectl` command reference.

Note:

This topic shows how to generate reference documentation for kubectl commands like kubectl apply and kubectl taint. This topic does not show how to generate the kubectl options reference page. For instructions on how to generate the kubectl options reference page, see Generating Reference Pages for Kubernetes Components and Tools.

Before you begin

Requirements:

- You need a machine that is running Linux or macOS.
- You need to have these tools installed:
  - Python v3.7.x+
  - Git
  - Golang version 1.13+
  - Pip used to install PyYAML
  - PyYAML v5.1.2
  - make
  - gcc compiler/linker
  - Docker (Required only for `kubectl` command reference)
- Your `PATH` environment variable must include the required build tools, such as the `Go` binary and `python`.
- You need to know how to create a pull request to a GitHub repository. This involves creating your own fork of the repository. For more information, see Work from a local clone.

Set up the local repositories

Create a local workspace and set your `GOPATH`:

```
mkdir -p $HOME/<workspace>
```

```
export GOPATH=$HOME/<workspace>
```

Get a local clone of the following repositories:

```
go get -u github.com/spf13/pflag
go get -u github.com/spf13/cobra
go get -u gopkg.in/yaml.v2
go get -u github.com/kubernetes-sigs/reference-docs
```

If you don't already have the kubernetes/website repository, get it now:

```
git clone https://github.com/<your-username>/website $GOPATH/src/github.com/<your-username>/website
```

Get a clone of the kubernetes/kubernetes repository as k8s.io/kubernetes:

```
git clone https://github.com/kubernetes/kubernetes $GOPATH/src/k8s.io/kubernetes
```

Remove the spf13 package from `$GOPATH/src/k8s.io/kubernetes/vendor/github.com`:

```
rm -rf $GOPATH/src/k8s.io/kubernetes/vendor/github.com/spf13
```

The kubernetes/kubernetes repository provides the `kubectl` and `kustomize` source code.

- Determine the base directory of your clone of the kubernetes/kubernetes repository. For example, if you followed the preceding step to get the repository, your base directory is `$GOPATH/src/k8s.io/kubernetes`. The remaining steps refer to your base directory as `<k8s-base>`.
- Determine the base directory of your clone of the kubernetes/website repository. For example, if you followed the preceding step to get the repository, your base directory is `$GOPATH/src/github.com/<your-username>/website`. The remaining steps refer to your base directory as `<web-base>`.
- Determine the base directory of your clone of the kubernetes-sigs/reference-docs repository. For example, if you followed the preceding step to get the repository, your base directory is `$GOPATH/src/github.com/kubernetes-sigs/reference-docs`. The remaining steps refer to your base directory as `<rdocs-base>`.

In your local k8s.io/kubernetes repository, check out the branch of interest, and make sure it is up to date. For example, if you want to generate docs for Kubernetes 1.33.0, you could use these commands:

```
cd <k8s-base>
git checkout v1.33.0
git pull https://github.com/kubernetes/kubernetes 1.33.0
```

If you do not need to edit the `kubectl` source code, follow the instructions for Setting build variables.

Edit the kubectl source code

The kubectl command reference documentation is automatically generated from the kubectl source code. If you want to change the reference documentation, the first step is to change one or more comments in the kubectl source code. Make the change in your local kubernetes/kubernetes repository, and then submit

a pull request to the master branch of github.com/kubernetes/kubernetes.

PR 56673 is an example of a pull request that fixes a typo in the kubectl source code.

Monitor your pull request, and respond to reviewer comments. Continue to monitor your pull request until it is merged into the target branch of the kubernetes/kubernetes repository.

Cherry pick your change into a release branch

Your change is now in the master branch, which is used for development of the next Kubernetes release. If you want your change to appear in the docs for a Kubernetes version that has already been released, you need to propose that your change be cherry picked into the release branch.

For example, suppose the master branch is being used to develop Kubernetes 1.34 and you want to backport your change to the release-1.33 branch. For instructions on how to do this, see Propose a Cherry Pick.

Monitor your cherry-pick pull request until it is merged into the release branch.

Note:

Proposing a cherry pick requires that you have permission to set a label and a milestone in your pull request. If you don't have those permissions, you will need to work with someone who can set the label and milestone for you.

Set build variables

Go to `<rdocs-base>`. On you command line, set the following environment variables.

- Set `K8S_ROOT` to `<k8s-base>`.
- Set `K8S_WEBROOT` to `<web-base>`.
- Set `K8S_RELEASE` to the version of the docs you want to build. For example, if you want to build docs for Kubernetes 1.33, set `K8S_RELEASE` to 1.33.

For example:

```
export K8S_WEBROOT=$GOPATH/src/github.com/<your-username>/website
export K8S_ROOT=$GOPATH/src/k8s.io/kubernetes
export K8S_RELEASE=1.33
```

Creating a versioned directory

The `createversiondirs` build target creates a versioned directory and copies the kubectl reference configuration files to the versioned directory. The versioned directory name follows the pattern of `v<major>_<minor>`.

In the `<rdocs-base>` directory, run the following build target:

```
cd <rdocs-base>
make createversiondirs
```

Check out a release tag in k8s.io/kubernetes

In your local `<k8s-base>` repository, check out the branch that has the version of Kubernetes that you want to document. For example, if you want to generate docs for Kubernetes 1.33.0, check out the `v1.33` tag. Make sure your local branch is up to date.

```
cd <k8s-base>
git checkout v1.33.0
git pull https://github.com/kubernetes/kubernetes v1.33.0
```

Run the doc generation code

In your local `<rdocs-base>`, run the `copycli` build target. The command runs as `root`:

```
cd <rdocs-base>
make copycli
```

The `copycli` command cleans the temporary build directory, generates the kubectl command files, and copies the collated kubectl command reference HTML page and assets to `<web-base>`.

Locate the generated files

Verify that these two files have been generated:

```
[ -e "<rdocs-base>/gen-kubectldocs/generators/build/index.html" ] && echo "index.html built" || echo "no index.html"
[ -e "<rdocs-base>/gen-kubectldocs/generators/build/navData.js" ] && echo "navData.js built" || echo "no navData.js"
```

Locate the copied files

Verify that all generated files have been copied to your `<web-base>`:

```
cd <web-base>
git status
```

The output should include the modified files:

```
static/docs/reference/generated/kubectl/kubectl-commands.html
static/docs/reference/generated/kubectl/navData.js
```

The output may also include:

```
static/docs/reference/generated/kubectl/scroll.js
static/docs/reference/generated/kubectl/stylesheet.css
static/docs/reference/generated/kubectl/tabvisibility.js
static/docs/reference/generated/kubectl/node_modules/bootstrap/dist/css/bootstrap.min.css
static/docs/reference/generated/kubectl/node_modules/highlight.js/styles/default.css
static/docs/reference/generated/kubectl/node_modules/jquery.scrollto/jquery.scrollTo.min.js
static/docs/reference/generated/kubectl/node_modules/jquery/dist/jquery.min.js
static/docs/reference/generated/kubectl/node_modules/font-awesome/css/font-awesome.min.css
```

Locally test the documentation

Build the Kubernetes documentation in your local `<web-base>`.

```
cd <web-base>
git submodule update --init --recursive --depth 1 # if not already done
make container-serve
```

View the local preview.

Add and commit changes in kubernetes/website

Run `git add` and `git commit` to commit the files.

Create a pull request

Create a pull request to the `kubernetes/website` repository. Monitor your pull request, and respond to review comments as needed. Continue to monitor your pull request until it is merged.

A few minutes after your pull request is merged, your updated reference topics will be visible in the published documentation.

What's next

- Generating Reference Documentation Quickstart
- Generating Reference Documentation for Kubernetes Components and Tools
- Generating Reference Documentation for the Kubernetes API

Contributing to the Upstream Kubernetes Code

This page shows how to contribute to the upstream `kubernetes/kubernetes` project. You can fix bugs found in the Kubernetes API documentation or the content of the Kubernetes components such as `kubeadm`, `kube-apiserver`, and `kube-controller-manager`.

If you instead want to regenerate the reference documentation for the Kubernetes API or the `kube-*` components from the upstream code, see the following instructions:

- Generating Reference Documentation for the Kubernetes API
- Generating Reference Documentation for the Kubernetes Components and Tools

Before you begin

- You need to have these tools installed:
  - Git
  - Golang version 1.13+
  - Docker
  - etcd
  - make
  - gcc compiler/linker
- Your `GOPATH` environment variable must be set, and the location of `etcd` must be in your `PATH` environment variable.
- You need to know how to create a pull request to a GitHub repository. Typically, this involves creating a fork of the repository. For more information, see Creating a Pull Request and GitHub Standard Fork & Pull Request Workflow.

The big picture

The reference documentation for the Kubernetes API and the `kube-*` components such as `kube-apiserver`, `kube-controller-manager` are automatically generated from the source code in the upstream Kubernetes.

When you see bugs in the generated documentation, you may want to consider creating a patch to fix it in the upstream project.

Clone the Kubernetes repository

If you don't already have the kubernetes/kubernetes repository, get it now:

```
mkdir $GOPATH/src
cd $GOPATH/src
go get github.com/kubernetes/kubernetes
```

Determine the base directory of your clone of the kubernetes/kubernetes repository. For example, if you followed the preceding step to get the repository, your base directory is `$GOPATH/src/github.com/kubernetes/kubernetes`. The remaining steps refer to your base directory as `<k8s-base>`.

Determine the base directory of your clone of the kubernetes-sigs/reference-docs repository. For example, if you followed the preceding step to get the repository, your base directory is `$GOPATH/src/github.com/kubernetes-sigs/reference-docs`. The remaining steps refer to your base directory as `<rdocs-base>`.

Edit the Kubernetes source code

The Kubernetes API reference documentation is automatically generated from an OpenAPI spec, which is generated from the Kubernetes source code. If you want to change the API reference documentation, the first step is to change one or more comments in the Kubernetes source code.

The documentation for the `kube-*` components is also generated from the upstream source code. You must change the code related to the component you want to fix in order to fix the generated documentation.

Make changes to the upstream source code

Note:

The following steps are an example, not a general procedure. Details will be different in your situation.

Here's an example of editing a comment in the Kubernetes source code.

In your local kubernetes/kubernetes repository, check out the default branch, and make sure it is up to date:

```
cd <k8s-base>
git checkout master
git pull https://github.com/kubernetes/kubernetes master
```

Suppose this source file in that default branch has the typo "atmost":

kubernetes/kubernetes/staging/src/k8s.io/api/apps/v1/types.go

In your local environment, open `types.go`, and change "atmost" to "at most".

Verify that you have changed the file:

```
git status
```

The output shows that you are on the master branch, and that the `types.go` source file has been modified:

```
On branch master
...
    modified:   staging/src/k8s.io/api/apps/v1/types.go
```

Commit your edited file

Run `git add` and `git commit` to commit the changes you have made so far. In the next step, you will do a second commit. It is important to keep your changes separated into two commits.

Generate the OpenAPI spec and related files

Go to `<k8s-base>` and run these scripts:

```
./hack/update-codegen.sh
./hack/update-openapi-spec.sh
```

Run `git status` to see what was generated.

```
On branch master
...
    modified:   api/openapi-spec/swagger.json
    modified:   api/openapi-spec/v3/apis__apps__v1_openapi.json
    modified:   pkg/generated/openapi/zz_generated.openapi.go
    modified:   staging/src/k8s.io/api/apps/v1/generated.proto
    modified:   staging/src/k8s.io/api/apps/v1/types_swagger_doc_generated.go
```

View the contents of `api/openapi-spec/swagger.json` to make sure the typo is fixed. For example, you could run `git diff -a api/openapi-spec/swagger.json`. This is important, because `swagger.json` is the input to the second stage of the doc generation process.

Run `git add` and `git commit` to commit your changes. Now you have two commits: one that contains the edited `types.go` file, and one that contains the generated OpenAPI spec and related files. Keep these two commits separate. That is, do not squash your commits.

Submit your changes as a pull request to the master branch of the kubernetes/kubernetes repository. Monitor your pull request, and respond to reviewer comments as needed. Continue to monitor your pull request until it is merged.

PR 57758 is an example of a pull request that fixes a typo in the Kubernetes source code.

Note:

It can be tricky to determine the correct source file to be changed. In the preceding example, the authoritative source file is in the `staging` directory in the `kubernetes/kubernetes` repository. But in your situation,the `staging` directory might not be the place to find the authoritative source. For guidance, check the `README` files in kubernetes/kubernetes repository and in related repositories, such as kubernetes/apiserver.

Cherry pick your commit into a release branch

In the preceding section, you edited a file in the master branch and then ran scripts to generate an OpenAPI spec and related files. Then you submitted your changes in a pull request to the master branch of the kubernetes/kubernetes repository. Now suppose you want to backport your change into a release branch. For example, suppose the master branch is being used to develop Kubernetes version 1.34, and you want to backport your change into the release-1.33 branch.

Recall that your pull request has two commits: one for editing `types.go` and one for the files generated by scripts. The next step is to propose a cherry pick of your first commit into the release-1.33 branch. The idea is to cherry pick the commit that edited `types.go`, but not the commit that has the results of running the scripts. For instructions, see Propose a Cherry Pick.

Note:

Proposing a cherry pick requires that you have permission to set a label and a milestone in your pull request. If you don't have those permissions, you will need to work with someone who can set the label and milestone for you.

When you have a pull request in place for cherry picking your one commit into the release-1.33 branch, the next step is to run these scripts in the release-1.33 branch of your local environment.

```
./hack/update-codegen.sh
./hack/update-openapi-spec.sh
```

Now add a commit to your cherry-pick pull request that has the recently generated OpenAPI spec and related files. Monitor your pull request until it gets merged into the release-1.33 branch.

At this point, both the master branch and the release-1.33 branch have your updated `types.go` file and a set of generated files that reflect the change you made to `types.go`. Note that the generated OpenAPI spec and other generated files in the release-1.33 branch are not necessarily the same as the generated files in the master branch. The generated files in the release-1.33 branch contain API elements only from Kubernetes 1.33. The generated files in the master branch might contain API elements that are not in 1.33, but are under development for 1.34.

Generate the published reference docs

The preceding section showed how to edit a source file and then generate several files, including `api/openapi-spec/swagger.json` in the `kubernetes/kubernetes` repository. The `swagger.json` file is the OpenAPI definition file to use for generating the API reference documentation.

You are now ready to follow the Generating Reference Documentation for the Kubernetes API guide to generate the published Kubernetes API reference documentation.

What's next

- Generating Reference Documentation for the Kubernetes API
- Generating Reference Docs for Kubernetes Components and Tools
- Generating Reference Documentation for kubectl Commands

Generating Reference Documentation for Metrics

This page demonstrates the generation of metrics reference documentation.

Before you begin

Requirements:

- You need a machine that is running Linux or macOS.
- You need to have these tools installed:
  - Python v3.7.x+
  - Git
  - Golang version 1.13+
  - Pip used to install PyYAML
  - PyYAML v5.1.2
  - make
  - gcc compiler/linker
  - Docker (Required only for `kubectl` command reference)
- Your `PATH` environment variable must include the required build tools, such as the `Go` binary and `python`.
- You need to know how to create a pull request to a GitHub repository. This involves creating your own fork of the repository. For more information, see Work from a local clone.

Clone the Kubernetes repository

The metric generation happens in the Kubernetes repository. To clone the repository, change directories to where you want the clone to exist.

Then, execute the following command:

```
git clone https://www.github.com/kubernetes/kubernetes
```

This creates a `kubernetes` folder in your current working directory.

Generate the metrics

Inside the cloned Kubernetes repository, locate the `test/instrumentation/documentation` directory. The metrics documentation is generated in this directory.

With each release, new metrics are added. After you run the metrics documentation generator script, copy the metrics documentation to the Kubernetes website and publish the updated metrics documentation.

To generate the latest metrics, make sure you are in the root of the cloned Kubernetes directory. Then, execute the following command:

```
./test/instrumentation/update-documentation.sh
```

To check for changes, execute:

```
git status
```

The output is similar to:

```
./test/instrumentation/documentation/documentation.md
./test/instrumentation/documentation/documentation-list.yaml
```

Copy the generated metrics documentation file to the Kubernetes website repository

1. Set the Kubernetes website root environment variable.

   Execute the following command to set the website root:

   ```
   export WEBSITE_ROOT=<path to website root>
   ```

2. Copy the generated metrics file to the Kubernetes website repository.

   ```
   cp ./test/instrumentation/documentation/documentation.md "${WEBSITE_ROOT}/content/en/docs/reference/instrumentation/metrics.m
   ```

   Note:

   If you get an error, check that you have permission to copy the file. You can use `chown` to change the file ownership back to your own user.

Create a pull request

To create a pull request, follow the instructions in Opening a pull request.

What's next

- Contribute-upstream
- Generating Reference Docs for Kubernetes Components and Tools
- Generating Reference Documentation for kubectl Commands

Reference Documentation Quickstart

This page shows how to use the `update-imported-docs.py` script to generate the Kubernetes reference documentation. The script automates the build setup and generates the reference documentation for a release.

Before you begin

Requirements:

- You need a machine that is running Linux or macOS.
- You need to have these tools installed:
  - Python v3.7.x+
  - Git
  - Golang version 1.13+
  - Pip used to install PyYAML
  - PyYAML v5.1.2
  - make
  - gcc compiler/linker

    - Docker (Required only for `kubectl` command reference)

- Your `PATH` environment variable must include the required build tools, such as the `Go` binary and `python`.

- You need to know how to create a pull request to a GitHub repository. This involves creating your own fork of the repository. For more information, see Work from a local clone.

Getting the docs repository

Make sure your `website` fork is up-to-date with the `kubernetes/website` remote on GitHub (`main` branch), and clone your `website` fork.

```
mkdir github.com
cd github.com
git clone git@github.com:<your_github_username>/website.git
```

Determine the base directory of your clone. For example, if you followed the preceding step to get the repository, your base directory is `github.com/website`. The remaining steps refer to your base directory as `<web-base>`.

Note:

If you want to change the content of the component tools and API reference, see the contributing upstream guide.

Overview of update-imported-docs

The `update-imported-docs.py` script is located in the `<web-base>/update-imported-docs/` directory.

The script builds the following references:

- Component and tool reference pages
- The `kubectl` command reference
- The Kubernetes API reference

Note:

The `kubelet` reference page is not generated by this script and is maintained manually. To update the kubelet reference, follow the standard contribution process described in Opening a pull request.

The `update-imported-docs.py` script generates the Kubernetes reference documentation from the Kubernetes source code. The script creates a temporary directory under `/tmp` on your machine and clones the required repositories: `kubernetes/kubernetes` and `kubernetes-sigs/reference-docs` into this directory. The script sets your `GOPATH` to this temporary directory. Three additional environment variables are set:

- `K8S_RELEASE`
- `K8S_ROOT`
- `K8S_WEBROOT`

The script requires two arguments to run successfully:

- A YAML configuration file (`reference.yml`)
- A release version, for example:`1.17`

The configuration file contains a `generate-command` field. The `generate-command` field defines a series of build instructions from `kubernetes-sigs/reference-docs/Makefile`. The `K8S_RELEASE` variable determines the version of the release.

The `update-imported-docs.py` script performs the following steps:

1. Clones the related repositories specified in a configuration file. For the purpose of generating reference docs, the repository that is cloned by default is `kubernetes-sigs/reference-docs`.
2. Runs commands under the cloned repositories to prepare the docs generator and then generates the HTML and Markdown files.
3. Copies the generated HTML and Markdown files to a local clone of the `<web-base>` repository under locations specified in the configuration file.
4. Updates `kubectl` command links from `kubectl`.md to the refer to the sections in the `kubectl` command reference.

When the generated files are in your local clone of the `<web-base>` repository, you can submit them in a pull request to `<web-base>`.

Configuration file format

Each configuration file may contain multiple repos that will be imported together. When necessary, you can customize the configuration file by manually editing it. You may create new config files for importing other groups of documents. The following is an example of the YAML configuration file:

```
repos:
- name: community  remote: https://github.com/kubernetes/community.git  branch: master  files:  - src: contributors/devel/README.m
```

Single page Markdown documents, imported by the tool, must adhere to the Documentation Style Guide.

Customizing reference.yml

Open `<web-base>/update-imported-docs/reference.yml` for editing. Do not change the content for the `generate-command` field unless you understand how the command is used to build the references. You should not need to update `reference.yml`. At times, changes in the upstream source code, may require changes to the configuration file (for example: golang version dependencies and third-party library changes). If you encounter build issues, contact the SIG-Docs team on the #sig-docs Kubernetes Slack channel.

Note:

The `generate-command` is an optional entry, which can be used to run a given command or a short script to generate the docs from within a repository.

In `reference.yml`, `files` contains a list of `src` and `dst` fields. The `src` field contains the location of a generated Markdown file in the cloned `kubernetes-sigs/reference-docs` build directory, and the `dst` field specifies where to copy this file in the cloned `kubernetes/website` repository. For example:

```
repos:
- name: reference-docs  remote: https://github.com/kubernetes-sigs/reference-docs.git  files:  - src: gen-compdocs/build/kube-apis
```

Note that when there are many files to be copied from the same source directory to the same destination directory, you can use wildcards in the value given to `src`. You must provide the directory name as the value for `dst`. For example:

```
  files:
  - src: gen-compdocs/build/kubeadm*.md
    dst: content/en/docs/reference/setup-tools/kubeadm/generated/
```

Running the update-imported-docs tool

You can run the `update-imported-docs.py` tool as follows:

```
cd <web-base>/update-imported-docs
./update-imported-docs.py <configuration-file.yml> <release-version>
```

For example:

```
./update-imported-docs.py reference.yml 1.17
```

Fixing Links

The `release.yml` configuration file contains instructions to fix relative links. To fix relative links within your imported files, set the`gen-absolute-links` property to `true`. You can find an example of this in `release.yml`.

Adding and committing changes in kubernetes/website

List the files that were generated and copied to `<web-base>`:

```
cd <web-base>
git status
```

The output shows the new and modified files. The generated output varies depending upon changes made to the upstream source code.

Generated component tool files

```
content/en/docs/reference/command-line-tools-reference/kube-apiserver.md
content/en/docs/reference/command-line-tools-reference/kube-controller-manager.md
content/en/docs/reference/command-line-tools-reference/kube-proxy.md
content/en/docs/reference/command-line-tools-reference/kube-scheduler.md
content/en/docs/reference/setup-tools/kubeadm/generated/kubeadm.md
content/en/docs/reference/kubectl/kubectl.md
```

Generated kubectl command reference files

```
static/docs/reference/generated/kubectl/kubectl-commands.html
static/docs/reference/generated/kubectl/navData.js
static/docs/reference/generated/kubectl/scroll.js
static/docs/reference/generated/kubectl/stylesheet.css
static/docs/reference/generated/kubectl/tabvisibility.js
static/docs/reference/generated/kubectl/node_modules/bootstrap/dist/css/bootstrap.min.css
static/docs/reference/generated/kubectl/node_modules/highlight.js/styles/default.css
static/docs/reference/generated/kubectl/node_modules/jquery.scrollto/jquery.scrollTo.min.js
static/docs/reference/generated/kubectl/node_modules/jquery/dist/jquery.min.js
static/docs/reference/generated/kubectl/css/font-awesome.min.css
```

Generated Kubernetes API reference directories and files

```
static/docs/reference/generated/kubernetes-api/v1.34/index.html
static/docs/reference/generated/kubernetes-api/v1.34/js/navData.js
static/docs/reference/generated/kubernetes-api/v1.34/js/scroll.js
static/docs/reference/generated/kubernetes-api/v1.34/js/query.scrollTo.min.js
static/docs/reference/generated/kubernetes-api/v1.34/css/font-awesome.min.css
static/docs/reference/generated/kubernetes-api/v1.34/css/bootstrap.min.css
static/docs/reference/generated/kubernetes-api/v1.34/css/stylesheet.css
static/docs/reference/generated/kubernetes-api/v1.34/fonts/FontAwesome.otf
static/docs/reference/generated/kubernetes-api/v1.34/fonts/fontawesome-webfont.eot
static/docs/reference/generated/kubernetes-api/v1.34/fonts/fontawesome-webfont.svg
static/docs/reference/generated/kubernetes-api/v1.34/fonts/fontawesome-webfont.ttf
static/docs/reference/generated/kubernetes-api/v1.34/fonts/fontawesome-webfont.woff
static/docs/reference/generated/kubernetes-api/v1.34/fonts/fontawesome-webfont.woff2
```

Run `git add` and `git commit` to commit the files.

Creating a pull request

Create a pull request to the `kubernetes/website` repository. Monitor your pull request, and respond to review comments as needed. Continue to monitor your pull request until it is merged.

A few minutes after your pull request is merged, your updated reference topics will be visible in the published documentation.

What's next

To generate the individual reference documentation by manually setting up the required build repositories and running the build targets, see the following guides:

- Generating Reference Documentation for Kubernetes Components and Tools
- Generating Reference Documentation for kubectl Commands
- Generating Reference Documentation for the Kubernetes API

Generating Reference Pages for Kubernetes Components and Tools

This page shows how to build the Kubernetes component and tool reference pages.

Before you begin

Start with the Prerequisites section in the Reference Documentation Quickstart guide.

Follow the Reference Documentation Quickstart to generate the Kubernetes component and tool reference pages.

What's next

- Generating Reference Documentation Quickstart
- Generating Reference Documentation for kubectl Commands
- Generating Reference Documentation for the Kubernetes API
- Contributing to the Upstream Kubernetes Project for Documentation

---

Updating Reference Documentation

The topics in this section document how to generate the Kubernetes reference guides.

To build the reference documentation, see the following guide:

- Generating Reference Documentation Quickstart

Contributing to the Upstream Kubernetes Code

Generating Reference Documentation for the Kubernetes API

Generating Reference Documentation for kubectl Commands

Generating Reference Documentation for Metrics

Generating Reference Pages for Kubernetes Components and Tools