# Contributing to krane

:+1::tada: First off, thanks for taking the time to contribute! :tada::+1:

The following is a set of guidelines for contributing to krane. Please take a moment to read through them before submitting your first PR.

#### Table Of Contents

[Code of Conduct](#code-of-conduct)

[What should I know before I get started?](#what-should-i-know-before-i-get-started)

\* [High-level design decisions](#high-level-design-decisions)

\* [Feature acceptance policies](#feature-acceptance-policies)

\* [Adding a new resource type](#contributing-a-new-resource-type)

\* [Contributor License Agreement](#contributor-license-agreement)

[Development](#development)

\* [Setup](#setup)

\* [Running the test suite locally](#running-the-test-suite-locally)

\* [Releasing a new version (Shopify employees)](#releasing-a-new-version-shopify-employees)

\* [CI (External contributors)](#ci-external-contributors)

## Code of Conduct

This project and everyone participating in it are governed by the [Code of Conduct](https://github.com/Shopify/krane/blob/master/CODE\_OF\_CONDUCT.md).

By participating, you are expected to uphold this code. Please report unacceptable

behavior to [krane@shopify.com](mailto:krane@shopify.com).

## Maintainers

This project is currently under the stewardship of the Production Platform group at Shopify.

The two primary maintainers are @knverey and @dturn. Approval from at least one primary maintainer is

required for all significant feature proposals and code architecture changes. In general,

two people must approve all non-trivial PRs.

## What should I know before I get started?

### High-level design decisions

\*\*Logging\*\*

Since we are primarily a CLI tool, logging is our entire user interface. Please think carefully about the information you log. Is it clear? Is it logged at the right time? Is it helpful?

In particular, we need to ensure that every mutation we’ve done to the cluster is clearly described (and is something the operator wanted us to do!).

We handle Kubernetes secrets, so it is critical that changes do not cause the contents of any secrets in the template set to be logged.

\*\*Project architecture\*\*

The main interface of this project is our five tasks: `DeployTask`, `GlobalDeployTask`, `RestartTask`, `RunnerTask`, and `RenderTask`. The code in these classes should be high-level abstractions, with implementation details encapsulated in other classes. The public interface of these tasks is a `run` method (and a `run!` equivalent), the body of which should read like a set of phases and steps. Note that non-task classes are considered internal and we reserve the right to change their API at any time.

An important design principle of the tasks is that they should try to fail fast before touching the cluster if they will not succeed overall. Part of how we achieve this is by separating each task into phases, where the first phase simply gathers information and runs validations to determine what needs to be done and whether that will be able to succeed. In practice, this is the “Initializing <task>” phase for all tasks, plus the “Checking initial resource statuses” phase for DeployTask. Our users should be able to assume that these initial phases never modify their clusters.

\*\*Thread safety\*\*

This tool must be able to run concurrent deploys to different targets safely, including when used as a library. Each of those deploys will internally also use parallelism via ruby threads, as do our integration tests. This means all of our code must be thread-safe. Notably, our code must not modify the global namespace (e.g. environment variables, classes, class variables or constants), and all gems we depend on must also be thread-safe.

\_Note\_: Local tests do not run in parallel by default. To enable it, use `PARALLELIZE\_ME=1 PARALLELISM=$NUM\_THREADS`. Unit tests never run in parallel because they use mocha, which is not thread-safe (mocha cannot be used in integration tests).

\*\*Performance and the sync cycle\*\*

DeployTask must remain performant when given several hundred resources at a time, generating 1000+ pods. This means only `sync` methods can make calls to the Kubernetes API server during result verification. This both limits the number of API calls made and ensures a consistent view of the world within each polling cycle.

### Feature acceptance policies

\*\*Our mission\*\*

This project's mission is to make it easy to ship changes to a Kubernetes namespace and understand the result. Features that introduce new classes of responsibility to the tool are not usually accepted.

Deploys can be a very tempting place to cram features. Imagine a proposed feature actually fits better elsewhere—where might that be? (Examples: validator in CI, custom controller, initializer, pre-processing step in the CD pipeline, or even Kubernetes core)

\*\*Template rendering\*\*

The basic ERB renderer included with the tool is intended as a convenience feature for a better out-of-the box experience. Providing complex rendering capabilities is outside the scope of this project's mission, and enhancements in this area may be rejected.

\*\*Composability\*\*

This project strives to be composable with other tools in the ecosystem, such as renderers and validators. The deploy task must work with any Kubernetes templates provided to it, no matter how they were generated.

\*\*Universality\*\*

This project is open-source. Features tied to any specific organization (including Shopify) will be rejected.

### Contributing a new resource type

The list of fully supported types is effectively the list of classes found in `lib/krane/kubernetes\_resource/`.

This gem uses subclasses of `KubernetesResource` to implement custom success/failure detection logic for each resource type. If no subclass exists for a type you're deploying, the gem simply assumes `kubectl apply` succeeded (and prints a warning about this assumption). We're always looking to support more types! Here are the basic steps for contributing a new one:

1. Create a file for your type in `lib/krane/kubernetes\_resource/`

2. Create a new class that inherits from `KubernetesResource`. Minimally, it should implement the following methods:

\* `sync` -- Gather the data you'll need to determine `deploy\_succeeded?` and `deploy\_failed?`. The superclass's implementation fetches the corresponding resource, parses it and stores it in `@instance\_data`. You can define your own implementation if you need something else.

\* `deploy\_succeeded?`

\* `deploy\_failed?`

3. Adjust the `TIMEOUT` constant to an appropriate value for this type.

4. Add the new class to list of resources in

[`deploy\_task.rb`](https://github.com/Shopify/krane/blob/master/lib/krane/deploy\_task.rb#L8)

5. Add the new resource to the [prune whitelist](https://github.com/Shopify/krane/blob/master/lib/krane/deploy\_task.rb#L81)

6. Add a basic example of the type to the hello-cloud [fixture set](https://github.com/Shopify/krane/tree/master/test/fixtures/hello-cloud) and appropriate assertions to `#assert\_all\_up` in [`hello\_cloud.rb`](https://github.com/Shopify/krane/blob/master/test/helpers/fixture\_sets/hello\_cloud.rb). This will get you coverage in several existing tests, such as `test\_full\_hello\_cloud\_set\_deploy\_succeeds`.

7. Add tests for any edge cases you foresee.

### Contributor License Agreement

New contributors will be required to sign [Shopify's Contributor License Agreement (CLA)](https://cla.shopify.com/).

There are two versions of the CLA: one for individuals and one for organizations.

# Development

## Setup

If you work for Shopify, just run `dev up`, but otherwise:

1. [Install kubectl version 1.10.0 or higher](https://kubernetes.io/docs/user-guide/prereqs/) and make sure it is in your path

2. [Install minikube](https://kubernetes.io/docs/getting-started-guides/minikube/#installation) (required to run the test suite)

3. [Install any required minikube drivers](https://github.com/kubernetes/minikube/blob/master/docs/drivers.md) (on OS X, you may need the [hyperkit driver](https://github.com/kubernetes/minikube/blob/master/docs/drivers.md#hyperkit-driver)

4. Check out the repo

5. Run `bin/setup` to install dependencies

To install this gem onto your local machine, run `bundle exec rake install`.

## Running the test suite locally

Using minikube:

1. Start [minikube](https://kubernetes.io/docs/getting-started-guides/minikube/#installation) (`minikube start [options]`).

2. Make sure you have a context named "minikube" in your kubeconfig. Minikube adds this context for you when you run `minikube start`. You can check for it using `kubectl config get-contexts`.

3. Run `bundle exec rake test` (or `dev test` if you work for Shopify).

Using another local cluster:

1. Start your cluster.

2. Put the name of the context you want to use in a file named `.local-context` in the root of this project. For example: `echo "dind" > .local-context`.

3. Run `bundle exec rake test` (or `dev test` if you work for Shopify).

To make StatsD log what it would have emitted, run a test with `STATSD\_ENV=development`.

To see the full-color output of a specific integration test, you can use `PRINT\_LOGS=1`. For example: `PRINT\_LOGS=1 bundle exec ruby -I test test/integration/krane\_deploy\_test.rb -n/test\_name/`.

![test-output](screenshots/test-output.png)

## Releasing a new version (Shopify employees)

1. On a new branch, create a new heading in CHANGELOG.md for your version and move the entries from "Next" under it. Leave the "Next" heading in the file (this helps with the diff for rebases after the release).

1. Make sure CHANGELOG.md includes all user-facing changes since the last release. Things like test changes or refactors do not need to be included.

1. Update the version number in `version.rb`.

1. Commit your changes with message "Version x.y.z" and open a PR.

1. After merging your PR, deploy via [Shipit](https://shipit.shopify.io/shopify/krane/rubygems). Shipit will automatically tag the release and upload the gem to [rubygems.org](https://rubygems.org/gems/krane).

## CI (External contributors)

Please make sure you run the tests locally before submitting your PR (see [Running the test suite locally](#running-the-test-suite-locally)). After reviewing your PR, a Shopify employee will trigger CI for you.

#### Employees: Triggering CI for a contributed PR

Go to the [krane pipeline](https://buildkite.com/shopify/krane) and click "New Build". Use branch `external\_contrib\_ci` and the specific sha of the commit you want to build. Add `BUILDKITE\_REFSPEC="refs/pull/${PR\_NUM}/head"` in the Environment Variables section. Since CI is only visible to Shopify employees, you will need to provide any failing tests and output to the the contributor.

<img width="350" alt="build external contrib PR" src="https://screenshot.click/2017-11-07--163728\_7ovek-wrpwq.png">