External Repository Staging Area

This directory is the staging area for packages that have been split to their own repository. The content here will be periodically published to respective top-level k8s.io repositories.

Repositories currently staged here:

- `k8s.io/api`
- `k8s.io/apiextensions-apiserver`
- `k8s.io/apimachinery`
- `k8s.io/apiserver`
- `k8s.io/cli-runtime`
- `k8s.io/client-go`
- `k8s.io/cloud-provider`
- `k8s.io/cluster-bootstrap`
- `k8s.io/code-generator`
- `k8s.io/component-base`
- `k8s.io/controller-manager`
- `k8s.io/cri-api`
- `k8s.io/csi-api`
- `k8s.io/csi-translation-lib`
- `k8s.io/kube-aggregator`
- `k8s.io/kube-controller-manager`
- `k8s.io/kube-proxy`
- `k8s.io/kube-scheduler`
- `k8s.io/kubectl`
- `k8s.io/kubelet`
- `k8s.io/legacy-cloud-providers`
- `k8s.io/metrics`
- `k8s.io/mount-utils`
- `k8s.io/pod-security-admission`
- `k8s.io/sample-apiserver`
- `k8s.io/sample-cli-plugin`
- `k8s.io/sample-controller`

The code in the staging/ directory is authoritative, i.e. the only copy of the code. You can directly modify such code.

Using staged repositories from Kubernetes code

Kubernetes code uses the repositories in this directory via symlinks in the `vendor/k8s.io` directory into this staging area. For example, when Kubernetes code imports a package from the `k8s.io/client-go` repository, that import is resolved to `staging/src/k8s.io/client-go` relative to the project root:

```
// pkg/example/some_code.go
package example
```

```
import (
  "k8s.io/client-go/dynamic" // resolves to staging/src/k8s.io/client-go/dynamic
)
```

Once the change-over to external repositories is complete, these repositories will actually be vendored from `k8s.io/<package-name>`.

Creating a new repository in staging

Adding the staging repository in `kubernetes/kubernetes`:

1. Send an email to the SIG Architecture mailing list and the mailing list of the SIG which would own the repo requesting approval for creating the staging repository.
2. Once approval has been granted, create the new staging repository.
3. Add a symlink to the staging repo in `vendor/k8s.io`.
4. Update `import-restrictions.yaml` to add the list of other staging repos that this new repo can import.
5. Add all mandatory template files to the staging repo as mentioned in https://github.com/kubernetes/kubernetes-template-project.
6. Make sure that the `.github/PULL_REQUEST_TEMPLATE.md` and `CONTRIBUTING.md` files mention that PRs are not directly accepted to the repo.

Creating the published repository

1. Create an issue in the `kubernetes/org` repo to request creation of the respective published repository in the Kubernetes org. The published repository **must** have an initial empty commit. It also needs specific access rules and branch settings. See #kubernetes/org#58 for an example.
2. Setup branch protection and enable access to the `stage-bots` team by adding the repo in `prow/config.yaml`. See #kubernetes/test-infra#9292 for an example.
3. Once the repository has been created in the Kubernetes org, update the publishing-bot to publish the staging repository by updating:
   - `rules.yaml`: Make sure that the list of dependencies reflects the staging repos in the `Godeps.json` file.
   - `fetch-all-latest-and-push.sh`: Add the staging repo in the list of repos to be published.
4. Add the staging and published repositories as a subproject for the SIG that owns the repos in `sigs.yaml`.

5. Add the repo to the list of staging repos in this `README.md` file.