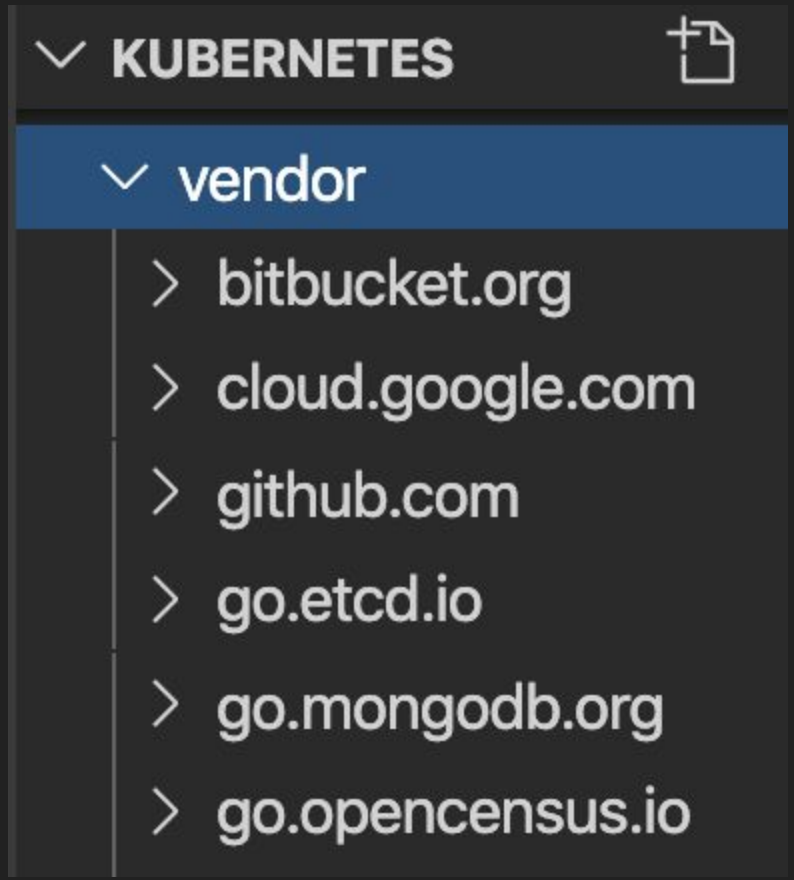
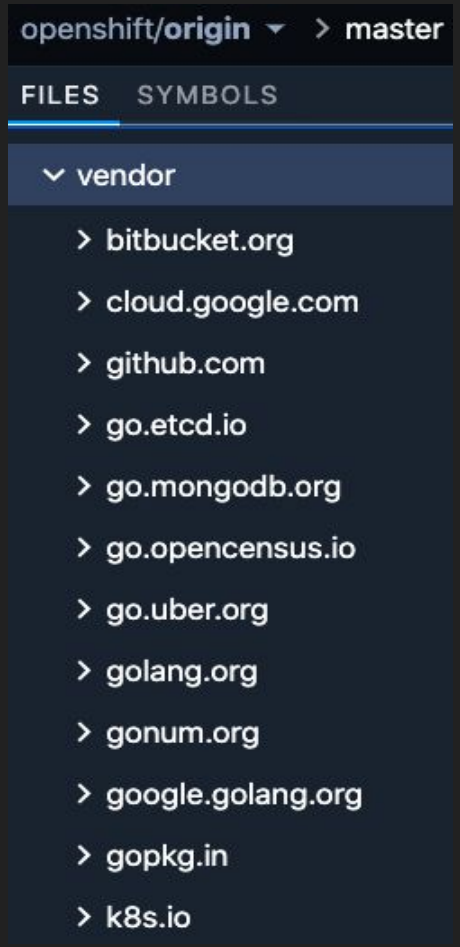
E2E Testing



upstream github.com/kubernetes/kubernetes

Contains all the dependencies to build kubernetes.

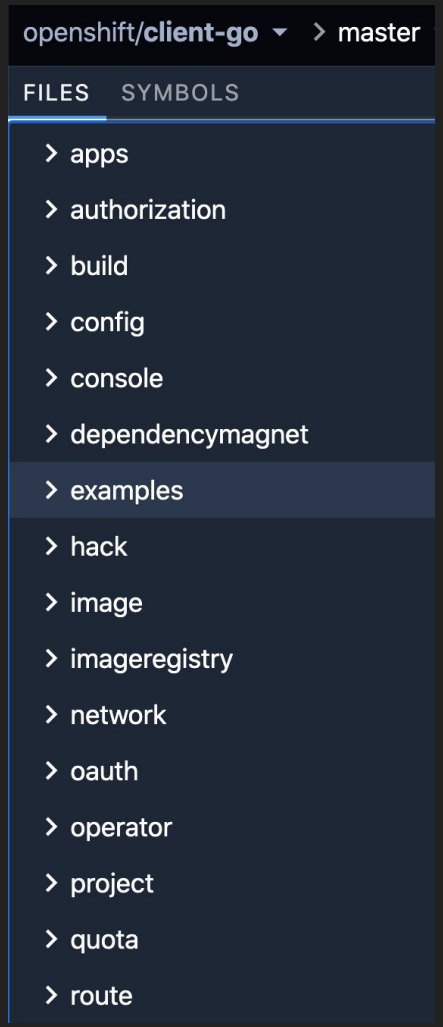


origin/okd: github.com/openshift/origin

Contains all the dependencies to build OpenShift.

OpenShift is split into three major repositories

1. https://github.com/openshift/api/ - all the external API objects definitions.
2. https://github.com/openshift/client-go/ - OpenShift magic
3. https://github.com/openshift/origin/ - the actual code behind OpenShift.



client-go: github.com/openshift/client-go

Interfaces, clientset, informers for openshift specific objects.

E.g.: builds, route, imagestream.

```
origin$ make build-extended-test
hack/build-go.sh cmd/openshift-tests
++ Building go targets for darwin/amd64: cmd/openshift-tests
[INFO] hack/build-go.sh exited with code 0 after 00h 04m 06s
```

```
origin$ find ./ -name openshift-tests
.//_output/local/bin/darwin/amd64/openshift-tests

# optionally add openshift-tests to path

origin$ openshift-tests help run
```

```
➜  origin$ openshift-tests help run
Run a test suite against an OpenShift server

 This command will run one of the following suites against a cluster identified by the current KUBECONFIG file. See the suite description for more on what actions the suite will take.

 If you specify the --dry-run argument, the names of each individual test that is part of the suite will be printed, one per line.
Available test suites:

openshift/conformance
  Tests that ensure an OpenShift cluster and components are working properly.


openshift/disruptive
  The disruptive test suite.



Example usage:

$ openshift-tests run openshift/image-registry --dry-run
```

https://docs.openshift.com/container-platform/3.11/scaling_performance/using_cluster_loader.html

This can also be used at the customer.

Tested on OCP v3.11

$ yum install atomic-openshift-tests

$ cd /usr/libexec/atomic-openshift/

$ ./extended.test --**ginkgo.focus**="Load cluster"

Ginkgo

A Golang BDD Testing Framework

- Ginkgo manages tests and Gomega is used for assertions.

- These tools support “behavior driven development”, which describes expected behavior in “specs”.

- Ginkgo test suites can be run with the ginkgo tool or as a normal Go test with go test.

spec examples

```
ginkgo.BeforeEach(func() {
  c = f.ClientSet
  ns = f.Namespace.Name

  // this test wants powerful permissions.  Since the namespace names are unique, we can leave this
  // lying around so we don't have to race any caches
  err := auth.BindClusterRoleInNamespace(c.RbacV1(), "edit", f.Namespace.Name,
    rbacv1.Subject{Kind: rbacv1.ServiceAccountKind, Namespace: f.Namespace.Name, Name: "default"})
  framework.ExpectNoError(err)

  err = auth.WaitForAuthorizationUpdate(c.AuthorizationV1(),
    serviceaccount.MakeUsername(f.Namespace.Name, "default"),
    f.Namespace.Name, "create", schema.GroupResource{Resource: "pods"}, true)
  framework.ExpectNoError(err)
})
```

```
framework.KubeDescribe("Secret", func() {
  ginkgo.It("should create a pod that reads a secret", func() {
    test := "test/fixtures/doc-yaml/user-guide/secrets"
    secretYaml := readFile(test, "secret.yaml")
    podYaml := readFile(test, "secret-pod.yaml.in")

    nsFlag := fmt.Sprintf("--namespace=%v", ns)
    podName := "secret-test-pod"

    ginkgo.By("creating secret and pod")
    framework.RunKubectlOrDieInput(ns, secretYaml, "create", "-f", "-", nsFlag)
    framework.RunKubectlOrDieInput(ns, podYaml, "create", "-f", "-", nsFlag)
    err := e2epod.WaitForPodNoLongerRunningInNamespace(c, podName, ns)
    framework.ExpectNoError(err)

    ginkgo.By("checking if secret was read correctly")
    _, err = framework.LookForStringInLog(ns, "secret-test-pod", "test-container", "value-1", serverStartTimeout)
    framework.ExpectNoError(err)
  })
})
```

#sig-testing in kubernetes slack

https://k8s-testgrid.appspot.com/

https://prow.k8s.io/

https://prow.istio.io/

https://deck-ci.apps.ci.l2s4.p1.openshiftapps.com/