Starting a Windows Kubernetes cluster on GCE using kube-up

IMPORTANT PLEASE NOTE

Any time the file structure in the `windows` directory changes, `windows/BUILD` and `k8s.io/release/lib/releaselib.sh` must be manually updated with the changes. We HIGHLY recommend not changing the file structure, because consumers of Kubernetes releases depend on the release structure remaining stable.

Bring up the cluster

Prerequisites: a Google Cloud Platform project.

0. Prepare your environment

Clone this repository under your `$GOPATH/src` directory on a Linux machine. Then, optionally clean/prepare your environment using these commands:

```
# Remove files that interfere with get-kube / kube-up:
rm -rf ./kubernetes/; rm -f kubernetes.tar.gz; rm -f ~/.kube/config

# To run e2e test locally, make sure "Application Default Credentials" is set in any of the
# References: https://cloud.google.com/sdk/docs/authorizing#authorizing_with_a_service_acco
#             https://cloud.google.com/sdk/gcloud/reference/auth/application-default/
#    1. $HOME/.config/gcloud/application_default_credentials.json, if doesn't exist, run th
gcloud auth application-default login
# Or 2. Create a json format credential file as per http://cloud/docs/authentication/produc
#       then export to environment variable
export GOOGLE_APPLICATION_CREDENTIAL=[path_to_the_json_file]
```

1. Build Kubernetes

NOTE: this step is only needed if you want to test local changes you made to the codebase.

The most straightforward approach to build those binaries is to run `make release`. However, that builds binaries for all supported platforms, and can be slow. You can speed up the process by following the instructions below to only build the necessary binaries.

```
# Build binaries for both Linux and Windows:
KUBE_BUILD_PLATFORMS="linux/amd64 windows/amd64" make quick-release
```

2. Create a Kubernetes cluster

You can create a regular Kubernetes cluster or an end-to-end test cluster.

Only end-to-end test clusters support running the Kubernetes e2e tests (as both e2e cluster creation and e2e test scripts are setup based on `cluster/gce/config-test.sh`), also enables some debugging features such as SSH access on the Windows nodes.

Please make sure you set the environment variables properly following the instructions in the previous section.

First, set the following environment variables which are required for controlling the number of Linux and Windows nodes in the cluster and for enabling IP aliases (which are required for Windows pod routing). At least one Linux worker node is required and two are recommended because many default cluster-addons (e.g., `kube-dns`) need to run on Linux nodes. The master control plane only runs on Linux.

```
export NUM_NODES=2  # number of Linux nodes
export NUM_WINDOWS_NODES=2
export KUBE_GCE_ENABLE_IP_ALIASES=true
export KUBERNETES_NODE_PLATFORM=windows
export LOGGING_STACKDRIVER_RESOURCE_TYPES=new
```

Now bring up a cluster using one of the following two methods:

2a. Create a regular Kubernetes cluster Ensure your GCP authentication is current:

```
gcloud auth application-default login
gcloud auth login
```

Invoke kube-up.sh with these environment variables:

```
# WINDOWS_NODE_OS_DISTRIBUTION: the Windows version you want your nodes to
#   run, e.g. win2019 or win1909.
# KUBE_UP_AUTOMATIC_CLEANUP (optional): cleans up existing cluster without
#   prompting.
WINDOWS_NODE_OS_DISTRIBUTION=win2019 KUBE_UP_AUTOMATIC_CLEANUP=true ./cluster/kube-up.sh
```

If your GCP project is configured with two-factor authentication, you may need to tap your security key shortly after running `kube-up`.

To teardown the cluster run:

```
./cluster/kube-down.sh
```

If you want to run more than one cluster simultaneously, you can use two separate GCP projects and:

1. Use a separate shell for each project / cluster.
2. Set the `CLOUDSDK_CORE_PROJECT` environment variable to the GCP project you want to use in each shell. This variable will override your current gcloud config.

3. Prefix your `kube-up.sh` and `kube-down.sh` commands with `PROJECT=${CLOUDSDK_CORE_PROJECT}`

2b. Create a Kubernetes end-to-end (E2E) test cluster If you have built your own release binaries following step 1, run the following command to bring up a cluster for running the K8s e2e tests. See the windows-gce e2e test configuration for the latest environment variables.

```
KUBE_GCE_ENABLE_IP_ALIASES=true KUBERNETES_NODE_PLATFORM=windows \
  KUBELET_TEST_ARGS=--feature-gates=KubeletPodResources=false \
  LOGGING_STACKDRIVER_RESOURCE_TYPES=new NUM_NODES=2 \
  NUM_WINDOWS_NODES=3 WINDOWS_NODE_OS_DISTRIBUTION=win2019 \
  ./hack/e2e-internal/e2e-up.sh
```

If any e2e cluster exists already, this command will prompt you to tear down and create a new one. To teardown existing e2e cluster only, run the command:

```
./hack/e2e-internal/e2e-down.sh
```

No matter what type of cluster you chose to create, the result should be a Kubernetes cluster with one Linux master node, `NUM_NODES` Linux worker nodes and `NUM_WINDOWS_NODES` Windows worker nodes.

Validating the cluster

Invoke this script to run a smoke test that verifies that the cluster has been brought up correctly:

```
cluster/gce/windows/smoke-test.sh
```

Sometimes the first run of the smoke test will fail because it took too long to pull the Windows test containers. The smoke test will usually pass on the next attempt.

Running e2e tests against the cluster

If you brought up an end-to-end test cluster using the steps above then you can use the steps below to run K8s e2e tests. These steps are based on kubernetes-sigs/windows-testing.

- Build the necessary test binaries. This must be done after every change to test code.

  ```
  make WHAT=test/e2e/e2e.test
  ```

- Set necessary environment variables and fetch the `run-e2e.sh` script:

  ```
  export KUBECONFIG=~/.kube/config
  export WORKSPACE=$(pwd)
  export ARTIFACTS=${WORKSPACE}/e2e-artifacts

  curl \
  ```

```
  https://raw.githubusercontent.com/kubernetes-sigs/windows-testing/master/gce/run-e2e.
  -o ${WORKSPACE}/run-e2e.sh
chmod u+x run-e2e.sh

# Fetch a prepull manifest for the k8s version you're using.
curl \
  https://raw.githubusercontent.com/kubernetes-sigs/windows-testing/master/gce/prepull-
  -o ${WORKSPACE}/prepull-head.yaml
```

The e2e test scripts make some annoying assumptions about the path to the k8s repository. If your `~/go/src/k8s.io/kubernetes` directory is actually a symlink to `~/go/src/github.com/<username>/kubernetes`, create this additional symlink:

```
cd ~/go/src/github.com; ln -s . github.com
```

Without this additional symlink you may receive this error when invoking the `run-e2e.sh` script:

```
chdir ../../github.com/<username>/kubernetes/_output/bin: no such file or directory
```

- The canonical arguments for running all Windows e2e tests against a cluster on GCE can be seen by searching for `--test-cmd-args` in the test configuration for the `ci-kubernetes-e2e-windows-gce` continuous test job. These arguments should be passed to the `run-e2e` script; escape the ginkgo arguments by adding quotes around them. For example:

  ```
  ./run-e2e.sh --node-os-distro=windows --minStartupPods=8 \
    --ginkgo.focus="\[Conformance\]|\[NodeConformance\]|\[sig-windows\]" \
    --ginkgo.skip="\[LinuxOnly\]|\[Serial\]|\[Feature:.+\]" \
    --ginkgo.parallel.total=8     # TODO: does this flag actually help?
  ```

  If you get auth errors, you may need to re-authenticate:

  ```
  gcloud auth application-default login
  gcloud auth login
  ```

- Run a single test by setting the ginkgo focus to match your test name; for example, the "DNS should provide DNS for the cluster" test can be run using:

  ```
  ./run-e2e.sh --node-os-distro=windows \
    --ginkgo.focus="provide\sDNS\sfor\sthe\scluster"
  ```

  Make sure to always include `--node-os-distro=windows` for testing against Windows nodes.

After the test run completes, log files can be found under the `${ARTIFACTS}` directory.

E2E Testing

Once you've created a pull request you can comment, `/test pull-kubernetes-e2e-windows-gce` to run the integration tests that cover the changes in this directory.