Kubernetes test images

Overview

All the images found here are used in Kubernetes tests that ensure its features and functionality. The images are built and published as manifest lists, allowing multiarch and cross platform support.

This guide will provide information on how to: make changes to images, bump their version, build the new images, test the changes made, promote the newly built staging images.

Prerequisites

In order to build the docker test images, a Linux node is required. The node will require `make`, `docker (version 19.03.0 or newer)`, and `docker buildx`, which will be used to build multiarch images, as well as Windows images. In order to properly build multi-arch and Windows images, some initialization is required (in CI this is done in cloudbuild.yaml):

```
docker run --rm --privileged multiarch/qemu-user-static --reset -p yes
docker buildx create --name img-builder --use
docker buildx inspect --bootstrap
```

The node must be able to push the images to the desired container registry, make sure you are authenticated with the registry you're pushing to.

Updating the tests images

There are several thousands of tests in Kubernetes E2E testing. Not all of them are being run on new PRs, and thus, not all images are used, especially those that are not used by Conformance tests.

So, in order to prevent regressions in the images and failing jobs, any changes made to the image itself or its binaries will require the image's version to be bumped. In the case of a regression which cannot be immediately resolved, the image version used in E2E tests will be reverted to the last known stable version.

Most tests used in E2E testing suite use the `agnhost` image. It contains several subcommands with different functionalities used to validate different Kubernetes behaviors. If a new functionality needs testing, consider adding an `agnhost` subcommand for it first, before creating an entirely separate test image.

The general process of making updates to the images is as follows:

1. Making changes to an image
2. Building the image
3. Testing your changes
4. Promoting your changes

After going through these steps your image will be used in the e2e tests. There are some additional considerations for completely new images and Windows images.

Creating and promoting new images

If you intend to add an entirely different image and have it automatically built by the Image Builder and used in E2E tests, you will also have to define the postsubmit prow job for it. This can easily be done by running this script in `kubernetes/test-infra`.

Windows test images considerations

Ideally, the same `Dockerfile` can be used to build both Windows and Linux images. However, that isn't always possible. If a different `Dockerfile` is needed for an image, it should be named `Dockerfile_windows`. When building, `image-util.sh` will first check for this file name when building Windows images.

The building process uses `docker buildx` to build both Windows and Linux images, but there are a few limitations when it comes to the Windows images:

- The Dockerfile can have multiple stages, including Windows and Linux stages for the same image, but the Windows stage cannot have any `RUN` commands (see the agnhost's `Dockerfile_windows` as an example).
- The Windows stage cannot have any `WORKDIR` commands due to a bug (https://github.com/docker/buildx/issues/378)
- When copying Windows symlink files to a Windows image, `docker buildx` changes the symlink target, prepending `Files\` to them (https://github.com/docker/buildx/issues/373) (for example, the symlink target `C:\bin\busybox.exe` becomes `Files\C:\bin\busybox.exe`). This can be avoided by having symlink targets with relative paths and having the target duplicated (for example, the symlink target `busybox.exe` becomes `Files\busybox.exe` when copied, so the binary `C:\bin\Files\busybox.exe` should exist in order for the symlink to be used correctly). See the busybox's `Dockerfile_windows` as an example.
- `docker buildx` overwrites the image's PATH environment variable to a Linux PATH environment variable, which won't work properly on Windows. See https://github.com/moby/buildkit/issues/1560
- The base image for all the Windows images is nanoserver, which is ~10 times smaller than Windows Servercore. Most binaries added to the image will work out of the box, but some will not due to missing dependencies (**attention**: the image will still build successfully, even if the added binaries will not work). For example, `coredns.exe` requires `netapi32.dll`, which cannot be found on a nanoserver image, but we can copy it from a servercore image (see the agnhost image's `Dockerfile_windows` file as an example). A good rule of thumb is to use 64-bit applications instead of 32-bit as they have fewer dependencies. You can determine what dependencies are missing by running `procmon.exe` on the container's host (make sure that process isolation is used, not Hyper-V isolation). This is a useful guide on how to use `procmon.exe`.

Because of the docker buildx limitations regarding the `RUN` commands mentioned above, we need to use a Windows helper image in order to properly build the Windows images without requiring any Windows nodes in the regular image building process. This helper image can be found in at `e2eteam/powershell-helper:6.2.7`. It can be used by anyone, but if you need to build your own, you can read more about it here.

For Windows, in order to spawn process-isolated containers, the container OS version should closely match the host OS version. For this reason, we build test images for different Windows OS Versions: 1809 (Windows Server 2019), 20H2, ltsc2022. In order to add support for a new Windows OS version, a new entry for that OS version will have to be first added to the `windows-servercore-cache` and `busybox` images, followed by the rest of the images. These images are then used by the rest of the E2E test images as a cache / base image.

Making changes to an image

Make updates to the functionality of the images required for your test case and update the version number.

The version can easily be bumped by modifying the file `test/images/${IMAGE_NAME}/VERSION`, which will be used when building the image.

Some test images (`agnhost`) are used as bases for other images (`kitten` , `nautilus`). If the parent image's `VERSION` has been bumped, also bump the version in the children's `BASEIMAGE` files in order for base image changes to be reflected in the child images as well.

Keep in mind that the Kubernetes CI will not run with the image changes you've made until promoted. It is a good idea to build the image and push it to your own registry first, and run some tests that are using that image. Continue with the steps below to build, test and promote the changes.

Building images

The images are built through `make` . Since some images (`agnhost`) are used as a base for other images, it is recommended to build them first, if needed.

An image can be built by simply running the command:

```
make all WHAT=agnhost
```

To build AND push an image, the following command can be used:

```
make all-push WHAT=agnhost
```

By default, the images will be tagged and pushed under the `k8s.gcr.io/e2e-test-images` registry. That can changed by running this command instead:

```
REGISTRY=foo_registry make all-push WHAT=agnhost
```

NOTE (for test `gcr.io` image publishers): Some tests (e.g.: `should serve a basic image on each replica with a private image`) require the `agnhost` image to be published in an authenticated repo as well:

```
REGISTRY=k8s.gcr.io/e2e-test-images make all-push WHAT=agnhost
REGISTRY=gcr.io/k8s-authenticated-test make all-push WHAT=agnhost
```

Additionally, `WHAT=all-conformance` can be used to build / push the images most commonly used in E2E Conformance tests.

Testing images

Once the image has been built and pushed to an accesible registry, you can run the tests using that image by having the environment variable `KUBE_TEST_REPO_LIST` set before running the tests that are using the image:

```
export KUBE_TEST_REPO_LIST=/path/to/repo_list.yaml
```

`repo_list.yaml` is a configuration file used by the E2E tests, in which you can set alternative registries to pull the images from. Sample file:

```
promoterE2eRegistry: your-awesome-registry
gcRegistry: your-awesome-registry
```

```
sampleRegistry: your-awesome-registry
```

Keep in mind that some tests are using multiple images, so it is a good idea to also build and push those images.

Finally, make sure to bump the image version used in E2E testing by modifying the file `test/utils/image/manifest.go`, and recompile afterwards:

```
./build/run.sh make WHAT=test/e2e/e2e.test
```

After all the above has been done, run the desired tests to make sure your changes work.

Promoting Images

Now that you have made the changes and tested locally, you are ready to share those changes. This is a multi step process:

1. In the same pull request containing your proposed changes, bump the version of the image in question. Each test image has a VERSION file in its directory. For example, the agnhost image's VERSION file is in `test/images/agnhost/VERSION`.

2. After the pull request has been approved and merged, an **automatic** postsubmit job will then be triggered which will build the images that were changed. For example, if a change was made in `test/images/agnhost`, then the job post-kubernetes-push-e2e-agnhost-test-images will be triggered. The postsubmit job will push the images to the `gcr.io/k8s-staging-e2e-test-images` registry. You can use the image from the staging registry to do more testing if required. All the postsubmit jobs and their logs for all the images can be seen in testgrid which is helpful for troubleshooting. Note that these images are not the same as used by the e2e jobs and still need to be promoted to the final registry.

3. The next step is to promote the image to the `k8s.gcr.io/e2e-test-images` registry by adding a line in kubernetes/k8s.io. See this pull request for an example You will need the image manifest list's digest, which can be obtained by using manifest-tool:

   ```
   manifest-tool inspect --raw gcr.io/k8s-staging-e2e-test-
   images/${IMAGE_NAME}:${VERSION} | jq '.[0].Digest'
   ```

4. Finally you can open a pull request to update the e2e tests to use the newly promoted image by updating the image tag in test/utils/image/manifest.go file. See this pull request for an example.

You have now gone through the entire process and your changes will be used by the e2e tests.

Known issues and workarounds

`docker manifest create` fails due to permission denied on `/etc/docker/certs.d/gcr.io` (https://github.com/docker/for-linux/issues/396). This issue can be resolved by running:

```
sudo chmod o+x /etc/docker
```

A few images have been mirrored from dockerhub into the `gcr.io/k8s-staging-e2e-test-images` registry (`busybox`, `glusterdynamic-provisioner`, `httpd`, `httpd-new`, `nginx`, `nginx-new`, `perl`), and

they only have a noop Dockerfile. However, due to an issue, the same SHA cannot be pushed twice. A small change to them is required in order to generate a new SHA, which can then be pushed and promoted.