Agnhost

Overview

There are significant differences between Linux and Windows, especially in the way something can be obtained or tested. For example, the DNS suffix list can be found in `/etc/resolv.conf` on Linux, but on Windows, such file does not exist, the same information could retrieved through other means. To combat those differences, `agnhost` was created.

`agnhost` is an extendable CLI that behaves and outputs the same expected content, no matter the underlying OS. The name itself reflects this idea, being a portmanteau word of the words agnostic and host.

The image was created for testing purposes, reducing the need for having different test cases for the same tested behaviour.

Usage

The `agnhost` binary has several subcommands which are can be used to test different Kubernetes features; their behaviour and output is not affected by the underlying OS.

For example, let's consider the following `pod.yaml` file:

```
apiVersion: v1
kind: Pod
metadata:
  name: test-agnhost
spec:
  containers:
  - args:
    - dns-suffix
    image: k8s.gcr.io/e2e-test-images/agnhost:2.14
    name: agnhost
  dnsConfig:
    nameservers:
    - 1.1.1.1
    searches:
    - resolv.conf.local
  dnsPolicy: None
```

After we've used it to create a pod:

```
kubectl create -f pod.yaml
```

We can then check the container's output to see what is DNS suffix list the Pod was configured with:

```
kubectl logs pod/test-agnhost
```

The output will be `resolv.conf.local`, as expected. Alternatively, the Pod could be created with the `pause` argument instead, allowing us execute multiple commands:

```
    kubectl exec test-agnhost -- /agnhost dns-suffix
    kubectl exec test-agnhost -- /agnhost dns-server-list
```

The `agnhost` binary is a CLI with the following subcommands:

audit-proxy

The audit proxy is used to test dynamic auditing. It listens on port 8080 for incoming audit events and writes them in a uniform manner to stdout.

Usage:

```
    kubectl exec test-agnhost -- /agnhost audit-proxy
```

connect

Tries to open a TCP or SCTP connection to the given host and port. On error it prints an error message prefixed with a specific fixed string that test cases can check for:

- `UNKNOWN` - Generic/unknown (non-network) error (eg, bad arguments)
- `TIMEOUT` - The connection attempt timed out
- `DNS` - An error in DNS resolution
- `REFUSED` - Connection refused
- `OTHER` - Other networking error (eg, "no route to host")

(Theoretically it would be nicer for it to distinguish these by exit code, but it's much easier for test programs to compare strings in the output than to check the exit code.)

Usage:

```
    kubectl exec test-agnhost -- /agnhost connect [--timeout=<duration>] [--
protocol=<protocol>] <host>:<port>
```

The optional `--protocol` parameter can be set to `sctp` to test SCTP connections. The default value is `tcp`.

crd-conversion-webhook

The subcommand tests `CustomResourceConversionWebhook`. After deploying it to Kubernetes cluster, the administrator needs to create a `CustomResourceConversion.Webhook` in Kubernetes cluster to use remote webhook for conversions.

The subcommand starts a HTTP server, listening on port 443, and creating the `/crdconvert` endpoint.

Usage

```
    kubectl exec test-agnhost -- /agnhost crd-conversion-webhook \
        [--tls-cert-file <tls-cert-file>] [--tls-private-key-file <tls-private-key-
file>]
```

dns-server-list

It will output the host's configured DNS servers, separated by commas.

Usage:

```
kubectl exec test-agnhost -- /agnhost dns-server-list
```

dns-suffix

It will output the host's configured DNS suffix list, separated by commas.

Usage:

```
kubectl exec test-agnhost -- /agnhost dns-suffix
```

entrypoint-tester

This subcommand will print the arguments it's passed and exists.

Usage:

```
kubectl exec test-agnhost -- /agnhost entrypoint-tester foo lish args
```

etc-hosts

It will output the contents of host's `hosts` file. This file's location is `/etc/hosts` on Linux, while on Windows it is `C:/Windows/System32/drivers/etc/hosts`.

Usage:

```
kubectl exec test-agnhost -- /agnhost etc-hosts
```

fake-gitserver

Fakes a git server. When doing `git clone http://localhost:8000`, you will clone an empty git repo named `localhost` on local. You can also use `git clone http://localhost:8000 my-repo-name` to rename that repo. Access to the service with the backing pod will show you below information.

```
curl -w "\n" http://localhost:8000
I am a fake git server
```

Usage:

```
kubectl exec test-agnhost -- /agnhost fake-gitserver
```

guestbook

Starts a HTTP server on the given `--http-port` (default: 80), serving various endpoints representing a guestbook app. The endpoints and their purpose are:

- `/register` : A guestbook replica will subscribe to a primary, to its given `--replicaof` endpoint. The primary will then push any updates it receives to its registered replicas through the `--backend-port` (default: 6379).
- `/get` : Returns `{"data": value}` , where the `value` is the stored value for the given `key` if non-empty, or the entire store.
- `/set` : Will set the given key-value pair in its own store and propagate it to its replicas, if any. Will return `{"data": "Updated"}` to the caller on success.
- `/guestbook` : Will proxy the request to `agnhost-primary` if the given `cmd` is `set` , or `agnhost-replica` if the given `cmd` is `get` .

Usage:

```
guestbook="test/e2e/testing-manifests/guestbook"
sed_expr="s|{{.AgnhostImage}}|k8s.gcr.io/e2e-test-images/agnhost:2.14|"

# create the services.
kubectl create -f ${guestbook}/frontend-service.yaml
kubectl create -f ${guestbook}/agnhost-primary-service.yaml
kubectl create -f ${guestbook}/agnhost-replica-service.yaml

# create the deployments.
cat ${guestbook}/frontend-deployment.yaml.in | sed ${sed_expr} | kubectl create -f -
cat ${guestbook}/agnhost-primary-deployment.yaml.in | sed ${sed_expr} | kubectl
create -f -
cat ${guestbook}/agnhost-replica-deployment.yaml.in | sed ${sed_expr} | kubectl
create -f -
```

help

Prints the binary's help menu. Additionally, it can be followed by another subcommand in order to get more information about that subcommand, including its possible arguments.

Usage:

```
    kubectl exec test-agnhost -- /agnhost help
```

inclusterclient

The subcommand will periodically poll the Kubernetes `/healthz` endpoint using the in-cluster config. Because of this, the subcommand is meant to be run inside of a Kubernetes pod. It can also be used to validate token rotation.

The given `--poll-interval` flag (default is 30 seconds) represents the poll interval in seconds of the call to `/healhz` .

Usage:

```
    kubectl exec test-agnhost -- /agnhost inclusterclient [--poll-interval <poll-
interval>]
```

liveness

Starts a simple server that is alive for 10 seconds, then reports unhealthy for the rest of its (hopefully) short existence.

Usage:

```
kubectl exec test-agnhost -- /agnhost liveness
```

grpc-health-checking

Started the gRPC health checking server. The health checking response can be controlled with the time delay or via http control server.

- `--delay-unhealthy-sec` - the delay to change status to NOT_SERVING. Endpoint reporting SERVING for `delay-unhealthy-sec` (`-1` by default) seconds and then NOT_SERVING. Negative value indicates always SERVING. Use `0` to start endpoint as NOT_SERVING.
- `--port` (default: `5000`) can be used to override the gRPC port number.
- `--http-port` (default: `8080`) can be used to override the http control server port number.
- `--service` (default: ``) can be used used to specify which service this endpoint will respond to.

Usage:

```
kubectl exec test-agnhost -- /agnhost grpc-health-checking \
  [--delay-unhealthy-sec 5] [--service ""] \
  [--port 5000] [--http-port 8080]

kubectl exec test-agnhost -- curl http://localhost:8080/make-not-serving
```

logs-generator

The `logs-generator` subcommand is a tool to create predictable load on the logs delivery system. It generates random lines with predictable format and predictable average length. Each line can be later uniquely identified to ensure logs delivery.

Tool is parametrized with the total number of number that should be generated and the duration of the generation process. For example, if you want to create a throughput of 100 lines per second for a minute, you set total number of lines to 6000 and duration to 1 minute.

Parameters are passed through CLI flags. There are no defaults, you should always pass the flags to the container. Total number of line is parametrized through the flag `--log-lines-total` and duration in go format is parametrized through the flag `--run-duration` .

Inside the container all log lines are written to the stdout.

Each line is on average 100 bytes long and follows this pattern:

```
2000-12-31T12:59:59Z <id> <method>
/api/v1/namespaces/<namespace>/endpoints/<random_string> <random_number>
```

Where `<id>` refers to the number from 0 to `total_lines - 1` , which is unique for each line in a given run of the container.

Examples:

```
docker run -i \
  k8s.gcr.io/e2e-test-images/agnhost:2.29 \
  logs-generator --log-lines-total 10 --run-duration 1s
```

```
kubectl run logs-generator \
  --generator=run-pod/v1 \
  --image=k8s.gcr.io/e2e-test-images/agnhost:2.29 \
  --restart=Never \
  -- logs-generator -t 10 -d 1s
```

mounttest

The `mounttest` subcommand can be used to create files with various permissions, read files, and output file system type, mode, owner, and permissions for any given file.

The subcommand can accept the following flags:

- `fs_type` : Path to print the FS type for.
- `file_mode` : Path to print the mode bits of.
- `file_perm` : Path to print the perms of.
- `file_owner` : Path to print the owning UID and GID of.
- `new_file_0644` : Path to write to and read from with perm 0644.
- `new_file_0666` : Path to write to and read from with perm 0666.
- `new_file_0660` : Path to write to and read from with perm 0660.
- `new_file_0777` : Path to write to and read from with perm 0777.
- `file_content` : Path to read the file content from.
- `file_content_in_loop` : Path to read the file content in loop from.
- `retry_time` (default: 180): Retry time during the loop.
- `break_on_expected_content` (default: true): Break out of loop on expected content (use with `--file_content_in_loop` flag only).

Usage:

```
    kubectl exec test-agnhost -- /agnhost mounttest \
        [--fs_type <path>] [--file_mode <path>] [--file_perm <path>] [--file_owner
<path>] \
        [--new_file_0644 <path>] [--new_file_0666 <path>] [--new_file_0660 <path>]
[--new_file_0777 <path>] \
        [--file_content <path>] [--file_content_in_loop <path>] \
        [--retry_time <seconds>] [--break_on_expected_content <true_or_false>]
```

net

The goal of this Go project is to consolidate all low-level network testing "daemons" into one place. In network testing we frequently have need of simple daemons (common/Runner) that perform some "trivial" set of actions on a socket.

Usage:

- A package for each general area that is being tested, for example `nat/` will contain Runners that test various NAT features.
- Every runner should be registered via `main.go:makeRunnerMap()` .
- Runners receive a JSON options structure as to their configuration. `Run()` should return the disposition of the test.

Runners can be executed into two different ways, either through the command-line or via an HTTP request:

Command-line:

```
    kubectl exec test-agnhost -- /agnhost net --runner <runner> --options <json>
    kubectl exec test-agnhost -- /agnhost net \
        --runner nat-closewait-client \
        --options '{"RemoteAddr":"127.0.0.1:9999"}'
```

HTTP server:

```
    kubectl exec test-agnhost -- /agnhost net --serve :8889
    kubectl exec test-agnhost -- curl -v -X POST localhost:8889/run/nat-closewait-
server \
        -d '{"LocalAddr":"127.0.0.1:9999"}'
```

netexec

Starts a HTTP(S) server on given port with the following endpoints:

- `/` : Returns the request's timestamp.
- `/clientip` : Returns the request's IP address.
- `/dial` : Creates a given number of requests to the given host and port using the given protocol, and returns a JSON with the fields `responses` (successful request responses) and `errors` (failed request responses). Returns `200 OK` status code if the last request succeeded, `417 Expectation Failed` if it did not, or `400 Bad Request` if any of the endpoint's parameters is invalid. The endpoint's parameters are:
  - `host` : The host that will be dialed.
  - `port` : The port that will be dialed.
  - `request` : The HTTP endpoint or data to be sent through UDP. If not specified, it will result in a `400 Bad Request` status code being returned.
  - `protocol` : The protocol which will be used when making the request. Default value: `http` . Acceptable values: `http` , `udp` , `sctp` .
  - `tries` : The number of times the request will be performed. Default value: `1` .
- `/echo` : Returns the given `msg` (`/echo?msg=echoed_msg`), with the optional status `code` .
- `/exit` : Closes the server with the given code and graceful shutdown. The endpoint's parameters are:
  - `code` : The exit code for the process. Default value: 0. Allows an integer [0-127].
  - `timeout` : The amount of time to wait for connections to close before shutting down. Acceptable values are golang durations. If 0 the process will exit immediately without shutdown.
  - `wait` : The amount of time to wait before starting shutdown. Acceptable values are golang durations. If 0 the process will start shutdown immediately.
- `/healthz` : Returns `200 OK` if the server is ready, `412 Status Precondition Failed` otherwise. The server is considered not ready if the UDP server did not start yet or it exited.

- `/hostname` : Returns the server's hostname.
- `/hostName` : Returns the server's hostname.
- `/redirect` : Returns a redirect response to the given `location` , with the optional status `code` (`/redirect?location=/echo%3Fmsg=foobar&code=307`).
- `/shell` : Executes the given `shellCommand` or `cmd` (`/shell?cmd=some-command`) and returns a JSON containing the fields `output` (command's output) and `error` (command's error message). Returns `200 OK` if the command succeeded, `417 Expectation Failed` if not.
- `/shutdown` : Closes the server with the exit code 0.
- `/upload` : Accepts a file to be uploaded, writing it in the `/uploads` folder on the host. Returns a JSON with the fields `output` (containing the file's name on the server) and `error` containing any potential server side errors.

If `--tls-cert-file` is added (ideally in conjunction with `--tls-private-key-file` , the HTTP server will be upgraded to HTTPS. The image has default, `localhost` -based cert/privkey files at `/localhost.crt` and `/localhost.key` (see: `porter` subcommand)

If `--http-override` is set, the HTTP(S) server will always serve the override path & options, ignoring the request URL.

It will also start a UDP server on the indicated UDP port that responds to the following commands:

- `hostname` : Returns the server's hostname
- `echo <msg>` : Returns the given `<msg>`
- `clientip` : Returns the request's IP address

The UDP server can be disabled by setting `--udp-port -1` .

Additionally, if (and only if) `--sctp-port` is passed, it will start an SCTP server on that port, responding to the same commands as the UDP server.

Usage:

```
    kubectl exec test-agnhost -- /agnhost netexec [--http-port <http-port>] [--udp-port <udp-port>] [--sctp-port <sctp-port>] [--tls-cert-file <cert-file>] [--tls-private-key-file <privkey-file>]
```

nettest

A tiny web server for checking networking connectivity.

Will dial out to, and expect to hear from, every pod that is a member of the service passed in the flag `--service` .

Will serve a webserver on given `--port` , and will create the following endpoints:

- `/read` : to see the current state, or `/quit` to shut down.
- `/status` : to see `pass/running/fail` determination. (literally, it will return one of those words.)
- `/write` : is used by other network test pods to register connectivity.

Usage:

```
    kubectl exec test-agnhost -- /agnhost nettest [--port <port>] [--peers <peers>]
[--service <service>] [--namespace <namespace>] [--delay-shutdown <delay>]
```

no-snat-test

The subcommand requires the following environment variables to be set, and they should be valid IP addresses:

- `POD_IP`
- `NODE_IP`

Serves the following endpoints on the given port (defaults to `8080`).

- `/whoami` - returns the request's IP address.
- `/checknosnat` - queries `ip/whoami` for each provided IP (`/checknosnat?ips=ip1,ip2`), and if all the response bodies match the `POD_IP`, it will return a 200 response, 500 otherwise.

Usage:

```
    kubectl run test-agnhost \
      --generator=run-pod/v1 \
      --image=k8s.gcr.io/e2e-test-images/agnhost:2.14 \
      --restart=Never \
      --env "POD_IP=<POD_IP>" \
      --env "NODE_IP=<NODE_IP>" \
      -- no-snat-test [--port <port>]
```

no-snat-test-proxy

Serves the `/checknosnat` endpoint on the given port (defaults to `31235`). The endpoint proxies the request to the given `target` (`/checknosnat?target=target_ip&ips=ip1,ip2` -> `target_ip/checknosnat?ips=ip1,ip2`) and will return the same status as the status as the proxied request, or 500 on error.

Usage:

```
    kubectl exec test-agnhost -- /agnhost no-snat-test-proxy [--port <port>]
```

pause

It will pause the execution of the binary. This can be used for containers which have to be kept in a `Running` state for various purposes, including executing other `agnhost` commands.

Usage:

```
    kubectl exec test-agnhost -- /agnhost pause
```

port-forward-tester

Listens for TCP connections on a given address and port, optionally checks the data received, and sends a configurable number of data chunks, with a configurable interval between chunks.

The subcommand is using the following environment variables:

- `BIND_ADDRESS` (optional): The address on which it will start listening for TCP connections (default value: `localhost`)
- `BIND_PORT`: The port on which it will start listening for TCP connections.
- `EXPECTED_CLIENT_DATA` (optional): If set, it will check that the request sends the same exact data.
- `CHUNKS`: How many chunks of data to write in the response.
- `CHUNK_SIZE`: The expected size of each written chunk of data. If it does not match the actual size of the written data, it will exit with the exit code `4`.
- `CHUNK_INTERVAL`: The amount of time to wait in between chunks.

Usage:

```
    kubectl run test-agnhost \
      --generator=run-pod/v1 \
      --image=k8s.gcr.io/e2e-test-images/agnhost:2.21 \
      --restart=Never \
      --env "BIND_ADDRESS=localhost" \
      --env "BIND_PORT=8080" \
      --env "EXPECTED_CLIENT_DATA='Hello there!'" \
      --env "CHUNKS=1" \
      --env "CHUNK_SIZE=10" \
      --env "CHUNK_INTERVAL=1" \
      -- port-forward-tester
```

porter

Serves requested data on ports specified in environment variables of the form `SERVE_{PORT,TLS_PORT,SCTP_PORT}_[NNNN]`. eg: - `SERVE_PORT_9001` - serve TCP connections on port 9001 - `SERVE_TLS_PORT_9002` - serve TLS-encrypted TCP connections on port 9002 - `SERVE_SCTP_PORT_9003` - serve SCTP connections on port 9003

The included `localhost.crt` is a PEM-encoded TLS cert with SAN IPs `127.0.0.1` and `[::1]`, expiring in January 2084, generated from `src/crypto/tls`:

```
    go run generate_cert.go  --rsa-bits 2048 --host 127.0.0.1,::1,example.com --ca -
-start-date "Jan 1 00:00:00 1970" --duration=1000000h
```

To use a different cert/key, mount them into the pod and set the `CERT_FILE` and `KEY_FILE` environment variables to the desired paths.

Usage:

```
    kubectl exec test-agnhost -- /agnhost porter
```

resource-consumer-controller

This subcommand starts an HTTP server that spreads requests around resource consumers. The HTTP server has the same endpoints and usage as the one spawned by the `resource-consumer` subcommand.

The subcommand can accept the following flags:

- `port` (default: 8080): The port number to listen to.
- `consumer-port` (default: 8080): Port number of consumers.
- `consumer-service-name` (default: `resource-consumer`): Name of service containing resource consumers.
- `consumer-service-namespace` (default: `default`): Namespace of service containing resource consumers.

Usage:

```
    kubectl exec test-agnhost -- /agnhost resource-consumer-controller \
        [--port <port>] [--consumer-port <port>] [--consumer-service-name <service-
name>] [--consumer-service-namespace <namespace>]
```

serve-hostname

This is a small util app to serve your hostname on TCP and/or UDP. Useful for testing.

The subcommand can accept the following flags:

- `tcp` (default: `false`): Serve raw over TCP.
- `udp` (default: `false`): Serve raw over UDP.
- `http` (default: `true`): Serve HTTP.
- `close` (default: `false`): Close connection per each HTTP request.
- `port` (default: `9376`): The port number to listen to.

Keep in mind that `--http` cannot be given at the same time as `--tcp` or `--udp`.

Usage:

```
    kubectl exec test-agnhost -- /agnhost serve-hostname [--tcp] [--udp] [--http] [-
-close] [--port <port>]
```

test-webserver

Starts a simple HTTP fileserver which serves any file specified in the URL path, if it exists.

The subcommand can accept the following flags:

- `port` (default: `80`): The port number to listen to.

Usage:

```
    kubectl exec test-agnhost -- /agnhost test-webserver [--port <port>]
```

webhook (Kubernetes External Admission Webhook)

The subcommand tests MutatingAdmissionWebhook and ValidatingAdmissionWebhook. After deploying it to kubernetes cluster, administrator needs to create a MutatingWebhookConfiguration or ValidatingWebhookConfiguration in kubernetes cluster to register remote webhook admission controllers.

More details on the configuration can be found from here Dynamic Admission Control.

Check the MutatingAdmissionWebhook and ValidatingAdmissionWebhook documentations for more information about them.

Usage:

```
    kubectl exec test-agnhost -- /agnhost webhook [--tls-cert-file <key-file>] [--
tls-private-key-file <cert-file>]
```

Other tools

The image contains `iperf`, `curl`, `dns-tools` (including `dig`), CoreDNS, for both Windows and Linux.

For Windows, the image is based on `busybox`, meaning that most of the Linux common tools are also available on it, making it possible to run most Linux commands in the `agnhost` Windows container as well. Keep in mind that there might still be some differences though (e.g.: `wget` does not have the `-T` argument on Windows).

The Windows `agnhost` image includes a `nc` binary that is 100% compliant with its Linux equivalent.

Image

The image can be found at `k8s.gcr.io/e2e-test-images/agnhost:2.35` for both Linux and Windows containers (based on `mcr.microsoft.com/windows/nanoserver:1809`, `mcr.microsoft.com/windows/nanoserver:20H2`, and `mcr.microsoft.com/windows/nanoserver:ltsc2022`).