Divingbell

Release 0.1.0

Sep 23, 2020

Contents

CHAPTER 1

Introduction

Divingbell is a lightweight solution for:

1. Bare metal configuration management for a few very targeted use cases via the following modules:
   - *apparmor*
   - *ethtool*
   - *exec* (run arbitrary scripts)
   - system *limits*
   - *mounts*
   - permissions (*perm*)
   - *sysctl* values
   - basic user account management (*uamlite*)
2. Bare metal package manager orchestration using *apt* module

1.1 What problems does it solve?

The needs identified for Divingbell were:

1. To plug gaps in day 1 tools (e.g., Drydock) for node configuration
2. To provide a day 2 solution for managing these configurations going forward
3. [Future] To provide a day 2 solution for system level host patching

CHAPTER 2

Design and Implementation

Divingbell DaemonSets mostly run as unprivileged containers which mount the host filesystem and chroot into that filesystem to enforce configuration and package state, or executes scripts in a namespace of `systemd` (PID=1). (The diving bell analogue can be thought of as something that descends into the deeps to facilitate work done down below the surface).

We use the DaemonSet construct as a way of getting a copy of each pod on every node, but the work done by this chart's pods behaves like an event-driven job. In practice this means that the chart internals run once on pod startup, followed by an infinite sleep such that the pods always report a "Running" status that k8s recognizes as the healthy (expected) result for a DaemonSet.

In order to keep configuration as isolated as possible from other systems that manage common files like `/etc/fstab` and `/etc/sysctl.conf`, Divingbell DaemonSets manage all of their configuration in separate files (e.g. by writing unique files to `/etc/sysctl.d` or defining unique Systemd units) to avoid potential conflicts. Another example is `limits` management, where Divingbell DaemonSets write separate files to `/etc/security/limits.d`.

To maximize robustness and utility, the DaemonSets in this chart are made to be idempotent. In addition, they are designed to implicitly restore the original system state after previously defined states are undefined. (e.g., removing a previously defined mount from the yaml manifest, with no record of the original mount in the updated manifest).

CHAPTER 3

Lifecycle management

This chart's DaemonSets will be spawned by Armada. They run in an event-driven fashion: the idempotent automation for each DaemonSet only re-runs when Armada spawns/respawns the container, or if information relevant to the host changes in the configmap.

CHAPTER 4

DaemonSet configs

4.1 apparmor

Used to manage host level apparmor profiles/rules. Ex.:

```
conf:
  apparmor:
    complain_mode: "true"
    profiles:
      profile-1: |
        #include <tunables/global>
          /usr/sbin/profile-1 {
            #include <abstractions/apache2-common>
            #include <abstractions/base>
            #include <abstractions/nis>

            capability dac_override,
            capability dac_read_search,
            capability net_bind_service,
            capability setgid,
            capability setuid,

            /data/www/safe/* r,
            deny /data/www/unsafe/* r,
          }
      profile-2: |
        #include <tunables/global>
          /usr/sbin/profile-2 {
            #include <abstractions/apache2-common>
            #include <abstractions/base>
            #include <abstractions/nis>

            capability dac_override,
            capability dac_read_search,
```

(continues on next page)

(continued from previous page)

```
            capability net_bind_service,
            capability setgid,
            capability setuid,

            /data/www/safe/* r,
            deny /data/www/unsafe/* r,
          }
```

4.2 apt

`apt` DaemonSet does package management. It is able to install a package of a specific version, upgrade an existing one to requested version, and perform a full-system upgrade. Version is optional, and if not provided, the latest available package is installed. It can also remove packages that were previously installed by divingbell (it is done by excluding the packages you want to remove from the configuration).

Note: When `conf.apt.upgrade` is `true`, packages are upgraded *after* the requested packages are installed.

Note: When `conf.apt.allow_downgrade` is `true`, the `--allow-downgrades` flag is passed to `apt-get install`, allowing it to downgrade a package if so specified in your packages list.

Note: When `conf.apt.strict` is `true`, any packages not in conf.apt.packages will be removed regardless of whether or not divingbell previously installed them. (The default behavior is for only packages previously installed by divingbell to be removed.) USE THIS OPTION WITH EXTREME CAUTION.

Here is an example configuration for it:

```
conf:
  apt:
    upgrade: false
    allow_downgrade: false
    strict: false
    packages:
    - name: <PACKAGE1>
      version: <VERSION1>
    - name: <PACKAGE2>
```

It is also permissible to use `conf.apt.packages` as a map, in which case all the packages from the different groups will be installed. This is primary useful for logical organization. The keys of the map are ignored, and the values are a list of the same format as the packages list above. No deduplication or other processing is performed, just a simple concatenation (without any ordering guarantees). For example:

```
conf:
  apt:
    packages:
      group1:
      - name: <PACKAGE1>
        version: <VERSION1>
      - name: <PACKAGE2>
```

(continues on next page)

(continued from previous page)

```
      group2:
      - name: <PACKAGE3>
      - name: <PACKAGE4>
```

Is equivalent to:

```
conf:
  apt:
    packages:
      - name: <PACKAGE1>
        version: <VERSION1>
      - name: <PACKAGE2>
      - name: <PACKAGE3>
      - name: <PACKAGE4>
```

There is a possibility to blacklist packages, e.g. `telnetd` and `nis`:

```
conf:
  apt:
    blacklistpkgs:
    - telnetd
    - nis
```

It is also possible to provide `debconf` settings for packages the following way:

```
conf:
  apt:
    packages:
    - name: openssh-server
      debconf:
      - question: openssh-server/permit-root-login
        question_type: boolean
        answer: false
```

4.3 ethtool

Used to manage host level NIC tunables. Ex.:

```
conf:
  ethtool:
    ens3:
      tx-tcp-segmentation: off
      tx-checksum-ip-generic: on
```

4.4 exec

Used to execute scripts on nodes in `systemd` (PID=1) namespace, for ex.:

```
exec:
  002-script2.sh:
    data: |
```

(continues on next page)

(continued from previous page)

```
    #!/bin/bash
    echo ${BASH_SOURCE[0]}
001-script1.sh:
  blocking_policy: foreground_halt_pod_on_failure
  env:
    env1: env1-val
    env2: env2-val
  args:
  - arg1
  - arg2
  data: |
    #!/bin/bash
    echo script name: ${BASH_SOURCE[0]}
    echo args: $@
    echo env: $env1 $env2 $env3
```

Scripts are executed in alphanumeric order with the key names used. Therefore in this example, `001-script1.sh` runs first, followed by `002-script2.sh`. Targeting of directives to specific nodes by hostname or node label is achievable by the use of the overrides capability described below.

The following set of options is fully implemeneted:

- `rerun_policy` may optionally be set to `always`, `never`, or `once_successfully` for a given script. That script would always be rerun, never be rerun, or rerun until the first successful execution respectively. Default value is `always`. This is tracked via a hash of the dict object for the script (i.e. script name, script data, script args, script env, etc). If any of that info changes, so will the hash, and it will be seen as a new object which will be executed regardless of this setting.
- `script_timeout` may optionally be set to the number of seconds to wait for script completion before termination. Default value is `1800` (30 min).
- `rerun_interval` may optionally be set to the number of seconds to wait between rerunning a given script which ran successfully the previous time. Default value is `infinite`.
- `retry_interval` may optionally be set to the number of seconds to wait between rerunning a given script which did not run successfully the previous time. Default behavior is to match the `rerun_interval`.

The following set of options is partially implemeneted:

- `blocking_policy` may optionally be set to `background`, `foreground`, or `foreground_halt_pod_on_failure` for a given script. This may be used to run a script in the background (running in parallel, i.e. non-blocking) or in the foreground (blocking). In either case, a failure of the script does not cause a failure (CrashLoop) of the pod. The third option may be used where the reverse behavior is desired (i.e., it would not proceed with running the next script in the sequence until the current script ran successfully). `background` option is not yet implemeneted. Default value is `foreground`.

The following set of options is not yet implemeneted:

- `rerun_interval_persist` may optionally be set to `false` for a given script. This makes the script execute on pod/node startup regardless of the interval since the last successful execution. Default value is `true`.
- `rerun_max_count` may optionally be set to the maximum number of times a succeeding script should be retried. Successful exec count does not persist through pod/node restart. Default value is `infinite`.
- `retry_interval_persist` may optionally be set to `false` for a given script. This makes the script execute on pod/node startup, regardless of the time since the last execution. Default value is `true`.
- `retry_max_count` may optionally be set to the maximum number of times a failing script should be retried. Failed exec count does not persist through pod/node restart. Default value is `infinite`.

4.5 limits

Used to manage host level limits. Ex.:

```
conf:
  limits:
    nofile:
      domain: 'root'
      type: 'soft'
      item: 'nofile'
      value: '101'
    core_dump:
      domain: '0:'
      type: 'hard'
      item: 'core'
      value: 0
```

Previous values of newly set limits are backed up to `/var/divingbell/limits`.

4.6 mounts

Used to manage host level mounts (outside of those in `/etc/fstab`). Ex.:

```
conf:
  mounts:
    mnt:
      mnt_tgt: /mnt
      device: tmpfs
      type: tmpfs
      options: 'defaults,noatime,nosuid,nodev,noexec,mode=1777,size=1024M'
```

4.7 perm

Used to manage permissions. Ex.:

```
conf:
  perms:
    -
      path: '/etc/shadow'
      owner: 'root'
      group: 'shadow'
      permissions: '0640'
    -
      path: '/etc/passwd'
      owner: 'root'
      group: 'root'
      permissions: '0644'
```

Module supports `rerun_policy` and `rerun_interval` options (like in `exec` module). Previous values of newly set permissions are backed up to `/var/divingbell/perm`.

4.8 sysctl

Used to manage host level sysctl tunables. Ex.:

```
conf:
  sysctl:
    net/ipv4/ip_forward: 1
    net/ipv6/conf/all/forwarding: 1
```

4.9 uamlite

Used to manage host level local user accounts, their SSH keys, and their sudo access. Ex.:

```
conf:
  uamlite:
    purge_expired_users: false
    users:
    - user_name: testuser
      user_crypt_passwd: $6$...
      user_sudo: true
      user_sshkeys:
      - ssh-rsa AAAAB3N... key1-comment
      - ssh-rsa AAAAVY6... key2-comment
```

CHAPTER 5

Operations

5.1 Setting apparmor profiles

The way apparmor loading/unloading implemented is through saving settings to a file and than running `apparmor_parser` command. The DaemonSet supports both enforcement and complain mode, enforcement being the default. To request complain mode for the profiles, add `complain_mode: "true"` nested under apparmor entry.

It's easy to mess up host with rules, if profile names would distinguish from file content. Ex:

```
conf:
  apparmor:
    profiles:
      profile-1: |
        #include <tunables/global>
          /usr/sbin/profile-1 {
            #include <abstractions/base>
            capability setgid,
          }
      profile-2: |
        #include <tunables/global>
          /usr/sbin/profile-1 {
            #include <abstractions/base>
            capability net_bind_service,
          }
```

Even when profiles are different (profile-1 vs profile-2) - filenames are the same (profile-1), that means that only one set of rules in memory would be active for particular profile (either setgid or net_bind_service), but not both. Such problems are hard to debug, so caution needed while setting configs up.

5.2 Setting user passwords

Including `user_crypt_passwd` to set a user password is optional.

If setting a password for the user, the chart expects the password to be encrypted with SHA-512 and formatted in the way that `crypt` library expects. Run the following command to generate the needed encrypted password from the plaintext password:

```
python3 -c "from getpass import getpass; from crypt import *; p=getpass(); print('\n
↪'+crypt(p, METHOD_SHA512)) if p==getpass('Please repeat: ') else print('\nPassword␣
↪mismatch.')"
```

Use the output of the above command as the `user_crypt_passwd` for the user. (Credit to unix.stackexchange.com.) If the password is not formatted how crypt expects, the chart will throw an error and fail to render.

At least one user must be defined with a password and sudo in order for the built-in `ubuntu` account to be disabled. This is because in a situation where network access is unavailable, console username/password access will be the only login option.

5.3 Setting user sudo

Including `user_sudo` to set user sudo access is optional. The default value is `false`.

At least one user must be defined with sudo access in order for the built-in `ubuntu` account to be disabled.

5.4 SSH keys

Including `user_sshkeys` for defining one or more user SSH keys is optional.

The chart will throw an error and fail to render if the SSH key is not one of the following formats:

- dsa (ssh-dss . . .)
- ecdsa (ecdsa-. . .)
- ed25519 (ssh-ed25519 . . .)
- rsa (ssh-rsa . . .)

Setting `user_sshkeys` to `[ Unmanaged ]` will instruct divingbell not to manage the user's authorized_keys file.

At least one user must be defined with an SSH key and sudo in order for the built-in `ubuntu` account to be disabled.

5.5 Purging expired users

Including the `purge_expired_users` key-value pair is optional. The default value is `false`.

This option must be set to `true` if it is desired to purge expired accounts and remove their home directories. Otherwise, removed accounts are expired (so users cannot login) but their home directories remain intact, in order to maintain UID consistency (in the event the same accounts gets re-added later, they regain access to their home directory files without UID mismatching).

CHAPTER 6

Node specific configurations

Although we expect these DaemonSets to run indiscriminately on all nodes in the infrastructure, we also expect that different nodes will need to be given a different set of data depending on the node role/function. This chart supports establishing value overrides for nodes with specific label value pairs and for targeting nodes with specific hostnames. The overridden configuration is merged with the normal config data, with the override data taking precedence.

The chart will then generate one DaemonSet for each host and label override, in addition to a default DaemonSet for which no overrides are applied. Each DaemonSet generated will also exclude from its scheduling criteria all other hosts and labels defined in other overrides for the same DaemonSet, to ensure that there is no overlap of DaemonSets (i.e., one and only one DaemonSet of a given type for each node).

Overrides example with sysctl DaemonSet:

```
conf:
  sysctl:
    net.ipv4.ip_forward: 1
    net.ipv6.conf.all.forwarding: 1
    fs.file-max: 9999
  overrides:
    divingbell_sysctl:
      labels:
      - label:
          key: compute_type
          values:
          - "dpdk"
          - "sriov"
        conf:
          sysctl:
            net.ipv4.ip_forward: 0
      - label:
          key: another_label
          values:
          - "another_value"
        conf:
          sysctl:
```

(continues on next page)

(continued from previous page)

```
        net.ipv6.conf.all.forwarding: 0
  hosts:
  - name: superhost
    conf:
      sysctl:
        net.ipv4.ip_forward: 0
        fs.file-max: 12345
  - name: superhost2
    conf:
      sysctl:
        fs.file-max: 23456
```

Caveats:

1. For a given node, at most one override operation applies. If a node meets override criteria for both a label and a host, then the host overrides take precedence and are used for that node. The label overrides are not used in this case. This is especially important to note if you are defining new host overrides for a node that is already consuming matching label overrides, as defining a host override would make those label overrides no longer apply.
2. In the event of label conflicts, the last applicable label override defined takes precedence. In this example, overrides defined for "another_label" would take precedence and be applied to nodes that contained both of the defined labels.

CHAPTER 7

Dev Environment with Vagrant

The point of Dev env to prepare working environment for development.

Vagrantfile allows to run on working copy with modifications e.g. to 020-test script. The approach is to setup Gate test but do not delete the pods and other stuff. You have:

1. test run of previous tests and their results
2. your changes from working tree are applied smoothly
3. your not committed test runs in prepared env

CHAPTER 8

Recorded Demo

Here are a few demo recording of Divingbell in action:

- Divingbell limits module: link, link (with overrides)
- Divingbell sysctl module: link, link (with overrides)
- Divingbell perms module: link, link (with overrides)

CHAPTER 9

Further Reading

Airship.