Divingbell

Release 0.1.0

May 10, 2018

Contents

Divingbell is a lightweight solution for: 1. Bare metal configuration management for a few very targeted use cases 2. Bare metal package manager orchestration

CHAPTER 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 run as privileged containers which mount the host filesystem and chroot into that filesystem to enforce configuration and package state. (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.

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 will only re-run when Armada spawns/respawns the container, or if information relevant to the host changes in the configmap.

CHAPTER 4

Daemonset configs

4.1 sysctl

Used to manage host level sysctl tunables. Ex:

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

4.2 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'
```

4.3 ethtool

Used to manage host level NIC tunables. Ex:

```
conf:
  ethtool:
    ens3:
```

```
    tx-tcp-segmentation: off
    tx-checksum-ip-generic: on
```

4.4 packages

Not implemented

4.5 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
```

4.5.1 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.

4.5.2 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.

4.5.3 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.

4.5.4 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 5

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:
            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 6

Recorded Demo

A recorded demo of using Divingbell can be found here.