CICD shell

Table of Contents

Setup

To use the `cicd` command, you need to fill in the configuration file located in `/vagrant/config/shell`.

The cicd shell is installed with the devbox. Please read the section *Install_outside_the_devbox* if you need to install it on a different system.

Usage

The name of the command line utility is cicd. The first mandatory position argument is the zone (dev, testing, staging or prod). The general scheme of the command line is:

```
cicd ZONE facts        [ROLE] [-n NODE] [-g GROUP] [-s STACK] [-a --all] [-d --down]
cicd ZONE runpuppet    [ROLE] [-n NODE] [-g GROUP] [-s STACK]
cicd ZONE runpuppet    [ROLE] [-n NODE] [-g GROUP] [-s STACK]
cicd ZONE ping         [ROLE] [-n NODE] [-g GROUP] [-s STACK]
cicd ZONE data [-k KEY][ROLE] [-n NODE] [-g GROUP] [-s STACK]
cicd ZONE orch         CMD                         [-s STACK]
cicd ZONE service      ACTION SERVICE [ROLE] [-n NODE] [-g GROUP] [-s STACK]
cicd ZONE console
```

Here is the help as display by invoking cicd alone:

```
→ cicd
CICD command line utility (v1.1.1)

Usage: cicd ZONE (console | stats | data | orch | facts | ping | du | service
              runpuppet | sync | result | gentags)

Available options:
  -h,--help                Show this help text
  ZONE                     ZONE such as dev, staging, testing or prod

Available commands:
  console                  Open the specialized salt console
  stats                    Stats (special permission required)
  data                     Return configuration data for a specific property
  orch                     Run an orchestration command on the infrastructure
  facts                    Return essential facts about nodes
  ping                     Ping nodes
  du                       Return disk usage
  service                  Service management for a specific node
  runpuppet                Apply puppet configuration
  sync                     Sync data from master to nodes
  result                   Display the results of jobs executed by the user
  gentags                  Generate node completion file
```

You can request the help at each level. For instance:

```
→ cicd staging facts -h
```

- Commands are executed remotely through an API. Behind the scene they call either the `puppetdb`, the `saltmaster` or the `pgserver`.
- Commands to the saltmaster together with their results are recorded in a centralized database included the date and name of the person that executes them.
- By default, all commands target a specific default hostgroup/stack defined in `/vagrant/conf/shell`

>_ facts

The command displays a subset of important facts (static information) about your nodes such as the `fqdn`, `ip`, `os`, `role`, ...

You can toggle the `facts` query to target all hostgroups/stacks with the `-a/ --all` flag. Here is how to get all facts for all slaves in every stack:

```
λ ~ → cicd prod facts jenkins.slave --all
{
  "fqdn": "SVAPPCAVL595.cirb.lan",
  "ip": "192.168.34.153",
  "os": "CentOS 6.6",
  "hostgroup": "irisbox",
  "subgroup": "jenkins",
  "role": "slave"
}
{
  "fqdn": "SVAPPCAVL649.prd.srv.cirb.lan",
  "ip": "192.168.34.9",
  "os": "RedHat 6.7",
  "hostgroup": "iam",
  "subgroup": "jenkins",
  "role": "slave"
}
...
```

As usual, use `-n` to target a single node:

```
→ cicd prod facts -n svappcavl771.prd.srv.cirb.lan
{
  "hostgroup": "fmx",
  "subgroup": "jenkins",
  "role": "slave",
  "os": "RedHat 7.2",
  "ip": "192.168.34.81",
  "puppet run": "Wed Dec  7 15:23:00 CET 2016",
  "jenkins job": "22"
}
```

Use the `--down` flag to gather `facts` on a disconnected minion.

>_ data

The command displays configuration data about your node. For instance you might display the docker version of your jenkins slave:

```
→ cicd prod data jenkins.slave -k docker::version
{
  "fqdn": "svappcavl736.cirb.lan",
  "subgroup": "jenkins",
  "role": "slave",
  "docker::version": "1.9.1-25.el7"
}
```

To display ALL known configurations for a specific node:

```
→ cicd prod data -n svappcavl771.prd.srv.cirb.lan
```

›_ runpuppet

The command runs the puppet agent on one or multiple nodes. When a node is specified with `-n`, the command will wait back for a result.

```
→ cicd dev runpuppet -n svappcavl000.dev.srv.cirb.lan
```

On all other cases, the command first asks for confirmation, then returns quickly with a `jobid`. The process is asynchronous because it might take quite a while to complete.

Here are some examples:

```
→ cicd dev runpuppet ①
→ cicd dev runpuppet -g jenkins ②
→ cicd dev runpuppet jenkins.slave ③
```

① run puppet on all the dev nodes of your stack

② run on a subgroup of machines

③ target a role

In a second step, you use ›_ `result` to retrieve from the database the result of your call [1: polling is currently the sole supported workflow, server push notification could be implemented in the future].

›_ result

You can view the result of a `runpuppet` by using the provided job id (`jid`)

```
→ cicd testing result -j 20160621104434055991
```

In case the result is not yet available the command will automatically be retry 12 times (3 min).

You can also ask for the last n executed commands:

```
→ cicd testing result -n 2
```

>_ du

The command displays disk usage. Try:

```
→ cicd staging du -n svappcavl703.sta.srv.cirb.lan
```

>_ console

For longer session within a specific zone, you can save some typing by opening a `console` for that zone. Inside the console, you would omit the zone from the command line. Here is an example:

```
→ cicd staging console

[cicd prod]$ facts
```

Another usage of the console is to run specific `salt` commands that are not exposed by the `cicd` command line. This is done via the pep shortcut. For instance:

```
$ pep -G 'hostgroup:iam' file.replace '/etc/resolv.conf' pattern='192.168.34.250' repl='192.168.34.244' ①

$ pep -L fqdn1,fqdn2 --client=local_async puppetutils.run_agent ②
```

① -G means `grain` target (*grains* is the salt terminology for facts).

② -L means `list` target
local_asyn means the :autofit-option:command is asynchronous and does not display its result (just a jid)

- Have a look at the saltstack documentation to learn more about targeting minions.
- Take a look here for a list of possible commands.

>_ orch

Salt can run multiple commands as well using the orchestrate runner. The orchestration is executed on the salt master to allow inter minion requisites, like ordering the application of states on different minions that must not happen simultaneously, or for halting the state run on all minions if a minion fails one of its states (more about this topic can be found in the saltstack website).

The orchestration should be defined in the orch folder. You will find some examples here.

Orchestrate commands can be started using:

```
→ cicd testing orch CMD
```

Authentication

The permissions to target machines and perform actions are realized through our Active directory. As an example to access the machines of the `middleware` hostgroup, you will need to be part of the `GP_APP_SALT_MIDDLEWARE` group.

These permissions should have been set for you already. If they don't, please contact the `cicd` team.

Install outside the devbox

Before installing the `cicd-shell` on any linux system [2: `macos` might also work], you will need:

1. the nix package manager installed and active for your user.
2. the cirb nixpkgs config

You can then proceed to install with:

```
nix-env -f ~/.config/nixpkgs/pin.nix -i cicd-shell ①
```

① the `-f` flag ensures that we point to the same nixpkgs version but can be omitted

If you haven't installed `nix` already, here is the quick how to:

```
bash <(curl https://nixos.org/nix/install)
```

This will perform a single-user installation of Nix, meaning that /nix is owned by the invoking user. The script will only invoke `sudo` to create /nix if it doesn't already exist. At that point, the script will prompt you for a password.

To activate `nix` in your shell, add the following line in your `.bash_profile`:

```
source ~/.nix-profile/etc/profile.d/nix.sh'
```