aicli Documentation

Release latest

Dec 06, 2022

CONTENTS

1 Deploying **3**
- 1.1 Offline token . . . 3
- 1.2 using rpm . . . 3
- 1.3 using pip . . . 3
- 1.4 Container mode . . . 3

2 How to use **5**
- 2.1 Configuration . . . 5
- 2.2 Basic usage . . . 6
- 2.3 Typical workflow . . . 6
- 2.4 Deployment workflow . . . 13
- 2.5 Ansible Integration . . . 14
- 2.6 Billi/ZTP Integration . . . 14

WARNING: The tool described in this repository is not supported in any way by Red Hat!!!

This is a sample python client on top of the generated assisted-installer python library to ease working with assisted installer API

Available features:

- create/delete cluster (autoinjecting pull secret file and ssh public key, static networking)
- create/delete manifests
- download discovery iso
- wait for hosts
- list cluster/hosts
- update hostnames and host roles
- update cluster attributes such as api vip and ingress vip
- launch cluster install
- wait for cluster
- create day2 cluster for adding hosts

CHAPTER

ONE

DEPLOYING

1.1 Offline token

When using SAAS mode, an offline token is needed in order to interact with the api. This token can be retrieved at https://cloud.redhat.com/openshift/token

1.2 using rpm

```
sudo dnf -y copr enable karmab/aicli ; sudo dnf -y install aicli
```

1.3 using pip

Install the package with

```
pip3 install aicli
```

To upgrade it later on, use the following

```
pip3 install -U aicli assisted-service-client
```

1.4 Container mode

```
alias aicli='podman run --security-opt label=disable --net host -it --rm -e AI_
↪OFFLINETOKEN=$AI_OFFLINETOKEN -v $HOME/.aicli:/root/.aicli -v $PWD:/workdir quay.io/
↪karmab/aicli'
```

Where AI_OFFLINETOKEN is an environment variable pointing to your saas offline token

With onprem mode, you can instead use `-e AI_URL=$AI_URL`

The container engine can also be docker.

CHAPTER

TWO

HOW TO USE

2.1 Configuration

2.1.1 Setting target url

by default, the tool targets the SAAS (which means providing an offline token)

Alternatively, You can indicate a target AI_URL, using the flag *–url*

The url needs to be provided in this format `http://$AI_IP:8090`

The target url can also be specified through environment variable AI_URL

the flag `--staging` can be set to target the internal staging environment

Alternatively, set the env variable STAGING to true

In container mode, that would be `-e STAGING=true`

2.1.2 TLS settings

You can use the flag ca, cert and key to provide ssl ca content, an ssl cert and key path to use for tls purpose

Alternatively, set the env variable AI_CA, AI_CERT and AI_KEY

2.1.3 Creating an onprem environment

You can run the following command to deploy AI locally using podman:

```
aicli create onprem
```

The ip to use for the configuration is detected automatically using a socket connection to 8.8.8.8

If this is blocked, you can use pass `-P ip=$your_ip` instead

You can also provide `-P keep=true` to keep both configmap.yml and pod.yml so you can tweak them yourself

When no longer needed, the onprem deployment can be deleted using:

```
aicli delete onprem
```

Note that deploying AI this way is not the recommended approach for production/supported usage.

2.2 Basic usage

the main objects you can interact with are cluster and hosts. For all of them, you can use create/delete/info/list/update subcommand

For most of the objects, you can provide either an id or a name.

2.2.1 Parameters

For most of the commands, you can pass parameters either on the command line by repeating `-P key=value` or by putting them in a parameter file, in yaml format.

If present, the file `aicli_parameters.yml` is also parsed.

2.3 Typical workflow

Interacting with the api should be straightforward but let's walk through the typical steps you can use to deploy a cluster to completion

Note: consider using `aicli create deployment` to deploy with a single step

2.3.1 Create cluster

The basic way to create a cluster is to run:

```
aicli create cluster mycluster
```

When creating a cluster, two types of parameters can be provided, *cluster keywords* and *extra keywords*.

The cluster keywords are parameters exposed by AI api, while the extra keywords are goodies that provide shortcuts to set things such as sno, network_type,...

Those keywords can be listed with `aicli list cluster-keywords` and `aicli list extra-keywords`

For instance, we can use the following command to create a cluster with a specific version and forcing the domain

```
aicli create cluster -P openshift_version=4.9 -P base_dns_domain=karmatron.local -P pull_
↪secret=openshift_pull.json mycluster
```

This command will also use your default ssh public key so you don't need to specify any (using ssh_public_key variable for instance)

To create a sno cluster

```
aicli create cluster -P sno=true mycluster
```

Within the extra parameters, the most common ones to use are listed below:

Parameter	Meaning
api_ip	Api ip
ingress_ip	Ingress ip
domain	Ingress ip
minimal	Whether to use minimal iso
static_network_config	the nmstate data to inject
manifests	a directory from where pick manifests to inject
network_type	Which sdn to use
sno	Whether to deploy a SNO
sno_disk	Which disk to use for SNO install
hosts	An array of hosts to automatically update data from
pull_secret	The path to your pull_secret (openshift_pull.json by default

Note: there are DNS requirements associated to the name of the cluster and the domain for an install to be available without /etc/hosts hacks

When a cluster gets created, an underlying infraenv named *$cluster_infraenv* also gets created under the hood.

In general, you shouldn't have to care about this object, but notice it is actually where the iso information lives. The purpose of this object is to be able to boot nodes with a discovery iso without deciding initially on which cluster they belong (this is called late binding).

The nomenclature we use for the infraenv is consistent with what happens in AI UI, which means you can create a cluster and follow in the UI or use aicli to interact with a cluster created through the UI.

You can set the parameter *infraenv* to false to prevent an infraenv to get created for the cluster. You will then have to use the bind subcommand to associate hosts discovered through a given infraenv to some specific cluster. This same parameter can be provided when updating a cluster so that no action on infraenvs happen.

2.3.2 Custom ignition

You can inject custom ignition in the discovery iso by either:

- specifying `ignition_config_override` parameter with an ignition string

For instance,

```
ignition_config_override: "{\"ignition\":{\"version\":\"3.1.0\"},\"systemd\":{\"units\":[
↪{\"name\":\"ca-patch.service\",\"enabled\":true,\"contents\":\"[Service]\\
↪nType=oneshot\\nExecStart=/usr/local/bin/ca-patch.sh\\n\\n[Install]\\nWantedBy=multi-
↪user.target\"}]},\"storage\":{\"files\":[{\"path\":\"/usr/local/bin/ca-patch.sh\",\
↪"mode\":720,\"contents\":{\"source\":\"data:text/plain;charset=utf-8;base64,
↪IyEvYmluL2Jhc2gKc3VjY2Vzcz0wCnVudGlsIFsgJHN1Y2Nlc3MgLWd0IDEgXTsgZG8KICB0bXA9JChta3RlbXAp CiAgY2F0IDw8RU
↪"}}]},\"kernelArguments\":{\"shouldExist\":[\"ipv6.disable=1\"]}}"
```

- using the extra keyword `discovery_ignition_file` and providing a path where to store the ignition directly (that is, without specifying ignition_config_override)

Note that for such ignition to be applied to the nodes once they really get installed, a machineconfig is needed (which can be injected as custom manifest)

2.3.3 Custom kernel arguments

You can specify kernel arguments that only apply to the discovery ISO by using the keyword `kernel_arguments`.

The data can be provided either as a string (say `a=xxx b=yyy`) in which case the arguments are considered to be appended or as a dict such as

```
kernel_arguments:
- operation: append
  value: 'a=xxx'
- operation: append
  value: 'b=yyy'
```

2.3.4 Custom networking

Network type

Default CNI when creating a cluster is `OVNKubernetes`

the network_type extra keyword can be used to specify another one such as OpenshiftSDN, Calico, Contrail

Note that non standard CNIs typically have extra requirements of injecting custom manifests.

Tuning network of the nodes

In order to use custom/static networking for your hosts, you need to provide nmstate information in the parameter file using the field *static_network_config*

You can also customize things such as cluster_networks, machine_networks and service_networks, for instance when trying to do a dual stack installation

You can find different samples here covering how to do:

- static networking
- bonding
- dual stack

Registry url

A non standard registry can be specified using the keyword `registry_url` (or `disconnected_url`) which is useful for disconnected install or testing okd.

Specify the corresponding ca content using the `ca` variable or let it undefined to let aicli fetch this cert using openssl. This approach requires that there exists connectivity between aicli and your custom registry.

2.3.5 Adding extra manifests

You can inject extra manifests (for instance if you are using a non standard CNI), for instance from the mydir directory, using the following commands

```
aicli create manifests --dir mydir mycluster
```

A flag allows you to have them stored in the openshift folder.

You can then use `aicli list manifests mycluster` to confirm they were properly uploaded, or use `aicli delete manifests` for deletion

Such manifests can be specified directly in your parameter file so that they get injected at cluster creation. For this, include the keyword manifests and point it to the directory where your manifests are stored.

2.3.6 Gather iso

Once the cluster (and the corresponding infraenv) are created, we can get the discovery iso url using the following command

```
aicli info iso mycluster
```

or download it locally with

```
aicli download iso mycluster
```

Note that when AI api was in v1, a specific call `create iso` was needed to trigger the creation of the iso, but it's no longer needed (the command is maintained for retrocompatibility but does the same as info iso`

When using this call, the expiration time of the token associated to the iso is checked and if necessary, it gets refreshed (and as such so does the url)

2.3.7 Use a debugging user in the iso

For troubleshooting purposes, typically when the host cant be accessed through ssh, you can add a user named aicli with password aicli in the discovery iso by running

```
aicli update cluster -P password=true mycluster
```

password boolean can be specified at cluster's creation

2.3.8 Wait for hosts

After booting some nodes with the iso, we normally wait for them to show up in the UI or in `aicli list hosts` output.

Alternatively, we can use the following command to wait for 3 hosts to appear in mycluster

```
aicli wait hosts mycluster -n 3
```

2.3.9 Optionally Update hosts

Once we have enough nodes, we need them show as `known` in list hosts output in order to start the cluster deployment.

It might be necessary to update some specific information of the nodes, such as the requested hostname (localhost name is forbidden) or to assign a specific role to the nodes

Updating hostnames

To change a specific host name, we can use the following

```
aicli update host $host -P requested_hostname=new_name
```

or simply

```
aicli update host $host -P name=new_name
```

If there are several matching hosts belonging to a same cluster, then the name is instead used as a prefix and the host names are sequentially assigned to name-0, name-1, ... That makes it easy to change all the localhost fqdns of your cluster with a single call

Updating roles

To change the role of a given host to worker, you can run

```
aicli update $host -P role=worker
```

Updating extra args

To specify extra args for a given host, you can run

```
aicli update $host -P extra_args="xxxx"
```

For instance, you can run the following to append kargs

```
aicli update host $host -P extra_args="--append-karg=rd.multipath=default --append-
↪karg=root=/dev/disk/by-label/dm-mpath-root"
```

Specifying installation disk

To specify installation disk for a given host, you can run

```
aicli update $host -P disk="xxxx"
```

For instance, To force installation disk to /dev/sdb, use

```
aicli update host $host -P disk=sdb"
```

Skipping formatting specific disks

To avoid formatting of given disks, you can run

```
aicli update $host -P skip_disks=[disk1,disk2]
```

For instance, To skip formatting of disks /dev/sdb and /dev/sdc, use

```
aicli update host $host -P skip_disk=[sdb,sdc]"
```

Note: the parameter `skip_formatting_disks` can be used instead but you will need to provide a dict with disk_id and skip_formatting

Specifying node labels

To declare specific labels for a host, you can run

```
aicli update $host -P labels=[label1,label2,key3=value3]
```

For instance, To specify the label `productionready` for a host, use

```
aicli update host $host -P labels=[productionready]"
```

Note: the parameter `node_labels` can be used instead

Applying custom ignition on first boot of a host

We can use the extra keyword `ignition_file` and provide a path where to store the ignition directly and call

```
aicli update host $host -P ignition_file=your_ignition_path"
```

Updating from a parameter file

You can specify in your parameter file a hosts array so that the information for updating hosts is gathered from there.

For instance, if you have the following information in your parameter file

```
hosts:
- name: xxx.fantastic.com
  role: master
- name: yyy.fantastic.com
  role: role
  extra_args: "ip=dhcp6"
```

Running `aicli update hosts --paramfile my_params.yml` will change the roles of the hosts with the corresponding name, if found, and add the specified extra_args for the second host.

You can either use the name to match a host, or specify id (to match system uuid) or mac (to match any mac address from the host inventory) so that the name of the host can be forced.

2.3.10 Updating cluster

At this step, you might need to update cluster data so that the cluster is ready to install.

For instance, you might want to specify api vip and ingress vip now that hosts cidrs have been discovered.

For this, you can run

```
aicli update cluster -P api_ip=$api_ip -P ingress_ip=$ingress_ip mycluster
```

2.3.11 Launch cluster deployment

Once your hosts all show as known, the cluster status should appear as ready in `aicli info cluster mycluster`

At this point, you can trigger the deployment using the following command

```
aicli start mycluster
```

2.3.12 Monitor deployment

Wait for cluster

When the cluster is installing, you can wait for it to complete using the following command

```
aicli wait mycluster
```

Monitor events

You can also see all events associated to your cluster using

```
aicli get events cluster
```

2.3.13 Gather assets

Once installation has started, you can gather relevant assets for your cluster such as

- kubeconfig
- kubeadmin-password
- installconfig

For instance, to gather the kubeconfig, you can use the following to get it downloaded to your current directory as `kubeconfig.mycluster`

```
aicli download kubeconfig mycluster
```

2.3.14 Add extra workers

For this purpose, we assume we already have the cluster installed.

If we create a new cluster with the same original name and the '-day2' extension, the api code will create a dedicated cluster for adding host purposes.

```
aicli create cluster mycluster-day2
```

Alternatively, we can convert the original cluster into one to use for adding hosts by using the following call but note this is not the recommended approach

```
aicli update cluster mycluster -P day2=true
```

Note that when creating the day2 cluster, a DNS check on api_vip_dnsname is done. If it doesn't succeed and the base cluster is HA, then api vip is used instead of fqdn to guarantee functionality

You can also update manually this data using the following command

```
aicli update cluster mycluster-day2 -P api_vip_dnsname=$api_ip
```

In both cases, once we have the day2 cluster in the proper state, the same workflow is to be used:

- gather the iso associated to this cluster/infraenv with `aicli info iso mycluster-day2`.
- boot nodes with this iso.
- wait for them to show in `aicli list hosts` output as *known*
- launch `aicli start cluster mycluster-day2`

Individual hosts installation can also be triggered by calling:

```
aicli start hosts $host1 $host2 ...
```

2.4 Deployment workflow

Instead of deploying the cluster step by step, you can put all the relevant information in your parameter file and then have all the steps run for you

You can use a command such as the following one

```
aicli create deployment --paramfile my_params.yml myclu
```

The parameter file could be similar to the following one

```
base_dns_domain: karmalabs.com
api_vip: 192.168.122.253
ingress_vip: 192.168.122.252
download_iso_path: /var/www/html
download_iso_cmd: "chown apache.apache /var/www/html/ci-ai.iso"
iso_url: http://192.168.122.1/ci-ai.iso
bmc_user: admin
bmc_password: password
hosts:
- name: ci-ai-master-0
```

(continues on next page)

(continued from previous page)

```
  bmc_url: http://192.168.122.1:8000/redfish/v1/Systems/11111111-1111-1111-1111-
↪111111111181
- name: ci-ai-master-1
  bmc_url: http://192.168.122.1:8000/redfish/v1/Systems/11111111-1111-1111-1111-
↪111111111182
- name: ci-ai-master-2
  bmc_url: http://192.168.122.1:8000/redfish/v1/Systems/11111111-1111-1111-1111-
↪111111111183
```

Note that in this case, we are providing bmc information for our hosts so that they get booted with the discovery iso automatically.

We also have the iso downloaded automatically to a path corresponding to a web server

If you omit this kind of information, you can still have the deployment done semi automatically by just waiting for the iso url to be displayed and plug it manually to your target nodes.

2.5 Ansible Integration

aicli modules are available at https://github.com/karmab/ansible-aicli-modules and provide the following primitives:

- ai_cluster
- ai_cluster_info
- ai_infraenv
- ai_infraenv_info
- ai_host_info

2.6 Billi/ZTP Integration

ZTP Workflow is a deployment methodology that relies on Assisted Installer run as an operator to drive the deployment in a kubernetes native way.

Billi is another way to deploy which involves running Assisted Installer in an ephemeral way on one of the target nodes.

Although you shouldn't need aicli at all for those use cases, a subcommand `create cluster-manifests` is available to generate the YAML manifests that are typically used and taking as input an aicli parameter file

For instance, using the sample bili.yml, the command `aicli create agent-manifests --pf samples/bili.yml` will generate the following objects ready to be used with Billi - install-config.yaml - agent-config.yaml

With the flag `--ztp`, the following objects will be generated that could be used as input in ZTP workflow

- agent-cluster-install.yaml
- cluster-deployment.yaml
- cluster-image-set.yaml
- infraenv.yaml
- nmstateconfig.yaml
- pull-secret.yaml