Contents

1 Project Overview 3

2 Get The Code 5

3 Contents 7
  3.1 Quickstart ............................... 7
  3.2 Tracking Development ................... 11
  3.3 Requirements .......................... 11
  3.4 Deployment ............................ 11
  3.5 Overrides / Private Info ................ 15
  3.6 Custom Job Configuration ............... 16
TOAD (TripleO Automated Deployer) is a system that helps automate various OpenStack deployment scenarios using TripleO Quickstart.

In conjunction with Jenkins Job Builder and Jenkins, various scenarios and topologies can be scripted and then triggered via the Jenkins dashboard.

TOAD is used as a simple spin-up environment to bootstrap a testing infrastructure with the ability to run tests with TripleO Quickstart, and parse logs and write data into an ELK stack for data visualization.

Find below an image of how the general workflow happens within TOAD:
Chapter 1. Project Overview
CHAPTER 2

Get The Code

The source is available on GitHub.
Quickstart

If you're on a Fedora 25 (or later) or CentOS 7.3 system, and you're ok with running a bash script as root, you can bootstrap your system with the following command:

```
```

**Note:** You may have issues with Fedora 25 as not all roles in TripleO Quickstart are setup to handle Fedora. It is recommended that you use CentOS.

After bootstrapping your machine, you can perform an “all in one” Jenkins Master/Slave deployment with the following command:

```
su - toad
```

All In One Deployment

With the All In One (AIO) deployment, a Jenkins Master will be instantiated via Docker Compose and configured via Ansible. A Jenkins Slave will then be added to the Jenkins Master by logging into the virtual host that hosts the Docker containers (including the Jenkins Master).
For an AIO deployment, you first bootstrap the node and then deploy the contents of TOAD onto the virtual host (see *Quickstart*). After instantiating your Jenkins Master via Docker Compose, it is configured via Ansible.

During the Ansible run in the `deploy.sh` script, it will also deploy a Jenkins Slave from the Master. The host for the Slave is the virtual host itself, and this is done via the `toad_default` Docker network.
If we inspect this Docker network, we can see our Jenkins Master network address and the address of the gateway (our virtual host).

```
[toad@virthost toad]$ docker network inspect toad_default
[
  {
    "Name": "toad_default",
    "Id": "0e51b147a04426dbff8c27ff4205d6487c3fec8a1b1e764cac68d30ffdfd9104",
    "Scope": "local",
    "Driver": "bridge",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": [
        {
          "Subnet": "172.18.0.0/16",
          "Gateway": "172.18.0.1/16"
        }
      ]
    },
    "Internal": false,
    "Containers": {
      "7e8bfdc51266955b8efcb7909ed5e130206e3dd8c791b8660a90ef7bb25f8c0b": {
        "Name": "jenkins_master",
        "EndpointID": "abe24ce17a677b0f73bed5672ba96145c012236cda16459f2bebc3c429cb6283",
        "MacAddress": "02:42:ac:12:00:03",
        "IPv4Address": "172.18.0.3/16",
        "IPv6Address": ""
      }
    },
    "Options": {},
    "Labels": {}
  }
]
```

The Jenkins Master will then SSH from 172.18.0.3 into the virtual host via the toad_default bridge through the gateway, and configure it as a Jenkins Slave. The Jenkins Slave will be used to execute the Jenkins jobs that we’ve configured via JJB (Jenkins Job Builder), which will run the TripleO quickstart.sh script.

**Note:** The bridges created by Docker are dynamically named. You can link the bridge name in Linux to the Docker bridge by looking at the ID field in the `docker network inspect` output, by taking the first 12 characters, and comparing that to the bridge names output by running `brctl show` or `ip a`.

The TripleO quickstart script will then setup an undercloud, controller, and compute node (by default, the `minimal.yml` configuration) via libvirt on the virtual host. The connection to the undercloud is made via the `brext` bridge, which is configured under libvirt as the external bridge. More information about the external bridge can be found in the TripleO documentation.
seen by running `virsh net-dumpxml external`:

```xml
[toad@virthost toad]$ sudo virsh net-dumpxml external
<network>
  <name>external</name>
  <uuid>c0adf892-1961-486c-b565-b26efafc3fe1</uuid>
  <forward mode='nat'>
    <nat>
      <port start='1024' end='65535'/>
    </nat>
  </forward>
  <bridge name='brext' stp='off' delay='0'/>
  <mac address='52:54:00:2e:81:c1'/>
  <ip address='192.168.23.1' netmask='255.255.255.0'>
    <dhcp>
      <range start='192.168.23.10' end='192.168.23.50'/>
    </dhcp>
  </ip>
</network>
```

Triple quickstart will also create another bridge called `brovc` for the communication between the undercloud and the overcloud. In libvirt it is configured as the `overcloud` network:

```xml
[toad@virthost toad]$ sudo virsh net-dumpxml overcloud
<network>
  <name>overcloud</name>
  <uuid>b21fa3cf-bd62-4d49-9856-768d8a8e4100</uuid>
  <bridge name='brovc' stp='off' delay='0'/>
  <mac address='52:54:00:db:6b:51'/>
</network>
```

That should give you an idea of how TOAD performs an AIO installation on a single network node. You’ll need a fairly robust machine for this type of setup though. It is recommended that you use a machine with at least 32GB of RAM, ideally 64GB of RAM.

**Logging Into Web Services**

Web services are deployed behind the traefik reverse proxy and can be accessed via hostnames. These hostnames are configured in `docker-compose.yml` via a traefik frontend rule. The hostname is one of `jenkins.${PROXY_DOMAIN}` or `kibana.${PROXY_DOMAIN}` where `${PROXY_DOMAIN}` is an environment variable defined in `toad/.env`. By default the domain is defined as `toad.tld`.

In order for the traefik proxy to know where to forward your requests, you’ll need to either

1. setup DNS services to point the subdomains to the primary address of your virtual host where Docker and Traefik are running
2. modify your `/etc/hosts` file to statically assign the hostname to the primary interface of your virtual host.

For example, in my network my virtual host primary network interface has the IP address of 192.168.5.151. I can then define the default hostname to that address in my `/etc/hosts` file.

```plaintext
192.168.5.151 kibana.toad.tld jenkins.toad.tld
```

Once you’ve completed the provisioning steps, you can login to your Jenkins master by browsing to `http://jenkins.toad.tld`.
Tracking Development

Development is tracked via Waffle.IO on the TOAD Waffle Board.

Requirements

TOAD is generally deployed in Docker containers. You can choose to deploy using Docker, or, together with an existing OpenStack deployment. Below you will find the list of requirements for each of the deployment scenarios.

For Ansible, several roles are required, and you can install them as follows:

```bash
ansible-galaxy install -r requirements.yml
```

Docker

TOAD primarily utilizes Docker containers. In order to use Docker, you need to install docker-compose.

At present, our docker-compose YAML file uses the version 2 specification, and should work with docker-compose version 1.6.0 or greater, and Docker engine 1.10.0 or later.

OpenStack

You’ll need to install the shade dependency so that you can interact with OpenStack (assuming you are deploying to an OpenStack cloud).

```bash
pip install --user shade
```

Setup OpenStack Connection

If you’re going to install to an OpenStack cloud, you’ll need to configure a cloud to connect to. You can do this by creating the `~/.config/openstack/` directory and placing the following contents into the `clouds.yml` file within that directory (adjust to your own cloud connection):

```yaml
clouds:
  mycloud:
    auth:
      auth_url: http://thecloud.com:5000/v2.0
      username: cloud_user
      password: cloud_pass
      project_name: "My Cloud Project"
```

Deployment

Deployment can be done via two methods: stand-alone Docker or OpenStack cloud.

Additionally, you can kick off the deployment with the `./scripts/deploy.sh` which bootstraps a simple deployment using the stand-alone Docker method.
Base Deployment

Start by creating `hosts/containers` (or similar) and add your baremetal machine with the following template:

```
jenkins_master
logstash
elasticsearch
kibana
```

These names (e.g. jenkins_master, logstash, etc) should match the names as defined in `./docker-compose.yml`.

Adding baremetal slaves to a Docker deployment

If you need to add jenkins slaves (baremetal), add slave information in `./hosts/containers` as the following (be sure to add `ansible_connection=ssh` as well).

```
[jenkins_slave]
slave01 ansible_connection=ssh ansible_host=10.10.1.1 ansible_user=ansible

[jenkins_slave:vars]
slave_description=TOAD Testing Node
slave_remoteFS=/home/stack
slave_port=22
slave_credentialsId=stack-credential
slave_label=toad
```

Running containers and start provisioning

Then, you can run the following commands to setup containers and to setup the TOAD environment.

```
$ docker-compose up -d
$ ansible-playbook site.yml -vvvv -i hosts/containers \
   -e use_openstack_deploy=false -e deploy_type='docker' -c docker
```

After you finish, you can stop these containers and restart them.

```
$ docker-compose stop
```

Or, to restart the containers:

```
$ docker-compose restart
```

The following command deletes the containers:

```
$ docker-compose down
```

Base Deployment (OpenStack)

**NOTE**: Deploying directly to OpenStack virtual machines is deprecated. It is recommended that you perform a deployment using the Docker method (even if that is hosted in a cloud instance on OpenStack). In a future version this method may be removed.
You may need to modify the `hostvars/localhost` file to adjust the `security_group` names, as the playbook does not currently create security groups and rules for you. It is assumed you’ve created the following sets of security groups, and opened the corresponding ports:

- default
- TCP: 22
- elasticsearch
- TCP: 9200
- filebeat-input
- TCP: 5044
- web_ports
- TCP: 80, 443

**NOTE:** The security groups are only relevant for OpenStack cloud deployments.

The base set of four VMs created for the CI components in OpenStack are listed as follows (as defined in `hostvars/localhost`):

```yaml
instance_list:
- { name: elasticsearch, security_groups: "default,elasticsearch" }
- { name: logstash, security_groups: "default,filebeat-input" }
- { name: kibana, security_groups: "default,web_ports" }
- { name: jenkins_master, security_groups: "default,web_ports" }
```

After configuration, you can run the following command which will connect to localhost to run the `shade` applications, authenticate to the OpenStack API you’ve supplied in `clouds.yml` and then deploy the stack.

```bash
ansible-playbook site.yml
```

### Configure Jenkins plugins

In order to configure `scp` plugin, you’ll need to use the `jenkins_scp_sites` var. It expects a list of sites where Jenkins will copy the artifacts from the jobs. The hostname / IP address should be relative to the Jenkins master server, as that is where the SCP module will be executed.

Format is the following (see *Example Variable Override File* for an example):

```yaml
jenkins_scp_sites:
- hostname: testHostname
  user: jenkins1
  password: abc
  path: /test/path
- hostname: testHostname
  port: 23
  user: jenkins1
  keyfile: abc
  path: /test/path
```

### Jenkins Slave Installation

If you wish to automate the deployment of your Jenkins baremetal slave machine, you can use Kickstart (or other similar methods). A base minimal installation of a CentOS node, as booted from a cdrom (we’re using CentOS as

3.4. Deployment
booted from the vFlash partition on a DRAC) can be configured during boot by pressing tab at the “Install CentOS” screen.

Add the following after the word `quiet` to statically configure a network and boot from the `ks.cfg` file (as supplied in the `samples/` directory). You’ll need to host the `ks.cfg` file from a web server accessible from your Jenkins baremetal slave node.

```
...quiet inst.ks=http://10.10.0.10/ks.cfg ksdevice=em1 ip=10.10.0.100::10.10.0.1:255.255.255.0:nfv-slave-01:em1:none nameserver=10.10.10.1
```

- `inst.ks`: Network path to the Kickstart file
- `ksdevice`: Device name to apply the network configuration to
- `ip`: Format: `[my_ip_address]::[gateway]::[netmask]::[hostname]::[device_name]::[boot_options]`
- `nameserver`: IP address of DNS nameserver

After booting, your machine should automatically deploy to a base minimum.

### Jenkins Slave Deployment

To deploy a Jenkins slave, you need to have a baremetal machine to connect to. You can tell Ansible about this machine by creating a new inventory file in the `hosts/` directory. You won’t pollute the repository since all inventory files except the `hosts/localhost` file as ignored.

Start by creating `hosts/slaves` (or similar) and add your baremetal machine with the following template:

```
[jenkins_slave]
slave01 ansible_host=10.10.1.1 ansible_user=ansible

[jenkins_slave:vars]
slave_description=TOAD Testing Node
slave_remoteFS=/home/stack
slave_port=22
slave_credentialsId=stack-credential
slave_label=toad
```

Add additional fields if necessary. It is assumed that the `ansible` user has been previously created, and that you can login either via SSH keys, or provide the `--ask-pass` flag to your Ansible run. The `ansible` user is also assumed to have been setup with passwordless sudo (unless you add `--ask-become-pass` during your Ansible run).

For OSP deployments, the build slaves need to be registered under RHN, and repositories and guest images need to be synced locally. In order to enable repository sync, you need to set the `slave_mirror_sync` var to `true`.

**NOTE:** By default, the system relies on the slave hostname and public IP to generate a valid repository address. Please ensure that slave hostname is set properly, and that is resolving to a public ip, reachable by all the VMs or baremetal servers involved in the deployments.

### Baremetal deployment

In order to perform baremetal deployments, an additional repository to host the hardware environment configuration is needed. A sample repository is provided: `https://github.com/redhat-nfvpe/toad_envs`

You can customize the repositories using the following settings: `- jenkins_job_baremetal_env_git_src`: path to the repository where to host the environments `- jenkins_job_baremetal_env_path`: if the environment is in a subfolder of the repo, please specify the relative path here.
The environment repo needs to have a folder for each environment that wants to be tested. Each environment needs to have the following content: - deploy_config.yml: it contains extra_args var, containing the parameters needed to deploy the overcloud. It specifies flavors, nodes to scale and templates to be used. - env_settings.yml: TripleO quickstart environment settings for the baremetal deployment. It defines the network settings, undercloud configuration parameters and any additional settings needed. - instackenv.json: Data file where all the baremetal nodes are specified. For each node, the IPMI address/user/password is required, as well as the provisioning MAC addresses. - net_environment.yml: TripleO environment file that will be used. You can specify here all the typical TripleO settings that need to be customized.

**RHN subscription**

On a Red Hat system, subscription of slaves can be managed automatically if you pass the right credentials:

- rhn_subscription_username
- rhn_subscription_password
- rhn_subscription_pool_id

Subscription can be managed automatically either on master or slaves, with the flags:

- master_subscribe_rhn
- slave_subscribe_rhn

**Overrides / Private Info**

There may be some variables you don’t want to expose into a Git repo. You can store those in the ~/.ansible/vars/toad_vars.yml file.

**Warning:** You must create a ~/.ansible/vars/toad_vars.yml file, even if it is blank. This file is loaded via var_files directives in Ansible and your deployment will fail if the file doesn’t exist.

The following list of options are required when deploying to an OpenStack cloud:

**Cloud Configuration**

- cloud_name_prefix
- cloud_name
- cloud_region_name
- cloud_availability_zone
- cloud_image
- cloud_flavor
- cloud_key_name

The jenkins_scp_sites variable is required when you need to copy configuration files off the slave to the master. Note that the hostname is relative to the master (in this case, files are copied off the slave into the master node, since that’s where the SCP command is run).

**SCP Site Configuration**

```yaml
jenkins_scp_sites:
  - hostname: 127.0.0.1
    path: "{{ jenkins_master_results_directory }}"
```

When adding slaves, you would do so by creating a new file in the hosts/ directory. For example you would create a hosts/slaves file and add your Jenkins slaves via the [jenkins_slave] and [jenkins_slave:vars] headers.
Jenkins Slave Configuration

- slave_name
- slave_description
- slave_remoteFS
- slave_host
- slave_port
- slave_credentialsId
- slave_label

Example Override Variable File

Many of the values can be found in your OpenStack RC file, which can typically be found in the Access & Security section of the Horizon dashboard.

cloud_name_prefix: redhat  # virtual machine name prefix
cloud_name: mycloud       # same as specified in clouds.yml
cloud_region_name: mycloud_region  # OS_REGION_NAME
cloud_availability_zone: nova   # availability zone
cloud_image: c0a97bbd-0cdd-4ed1-b6c1-052123456789  # unique image ID
cloud_flavor: m1.medium
cloud_key_name: my_pub_key     # name of your keypair

jenkins_job_builder_git_jobs_src: https://github.com/redhat-nfvpe/jenkins-jobs.git
  # branched from upstream for customization purposes
jenkins_job_config_git_src: https://github.com/redhat-nfvpe/job-configs.git
jenkins_job_builder_config_jenkins_user: admin  # default username
jenkins_job_builder_config_jenkins_password: admin  # default password

# Can only specify a single site to SCP files to at the end of the run.
jenkins_scp_sites:
  - hostname: 127.0.0.1
    path: "{{ jenkins_master_results_directory }}"  # defined in vars/main.yml

Custom Job Configuration

With TOAD it is possible to create your own jobs, job configuration files, and environment files in order to define your cloud networking configuration. These configurations are defined by default in order to provide you a sample set of data to get things started quickly.

It’s highly likely you’ll want to build your own jobs, and therefore fork the standard set of configuration repositories, and pass that to the job deployment.

Job Repository Workflow

In the following diagram, you can see how three main Git repositories are used in the deployment of Jenkins jobs, and then the job configuration and environment files used in the job run.
There are three main repositories in use within TOAD:

<table>
<thead>
<tr>
<th>Repository</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jenkins-jobs.git</td>
<td>Definition of the Jenkins Jobs, parsed via Jenkins Job Builder</td>
</tr>
<tr>
<td>job-configs.git</td>
<td>Extra configuration files used during job deployment on the slave</td>
</tr>
<tr>
<td>toad_envs.git</td>
<td>Environment files for undercloud and overcloud configuration</td>
</tr>
</tbody>
</table>

**jenkins-jobs.git**

The jenkins-jobs.git repository contains the macros and job definitions that are parsed by Jenkins Job Builder (JJB). During deployment of the Jenkins master the ansible-jenkins-job-builder role executes and takes information in the jenkins-jobs.git repo and builds the job files and configures the Jenkins master.

**job-configs.git**

During job runs, files in job-configs.git are copied from the master to the slave, and used during the TripleO Quickstart (ooqoq) runs. The types of files copied over would include:

- requirements files for extra python packages or Ansible roles used during the run
- yaml files containing variables used during Ansible execution (ooqoq, roles)
- scripts used during image build runs (jinja2)

**toad_envs.git**

The toad_envs repository holds various hardware environment configuration files used for the deployment of the network during a TripleO run. These files are used by the undercloud during the OpenStack stack deployment in order to know how to configure the networking. These files are generally only used in baremetal overcloud deployments.
Update Jenkins Jobs

TOAD enables a developer to iterate through job configuration and push those changes into the local development environment. The deployment of the jobs is done via JJB, and the TOAD Ansible playbooks allows a simple job update in your Jenkins environment.

To pull the latest changes from your upstream working repository (which you can override in ~/.ansible/vars/toad_vars.yml) and update your job definitions, run the following Ansible command from your toad working directory:

```
ansible-playbook site.yml --limit jenkins_master --tags jenkins_jobs
```

**Note:** You will need to add `-c docker` if you’re deploying into a Docker environment.

If you want to make local changes to the jobs prior to submitting changes to the upstream jenkins-jobs repository, you can make use of the `sync_jobs` tag instead.

```
ansible-playbook site.yml --limit jenkins_master --tags sync_jobs
```

With the use of the `sync_jobs` tags you can build up local jobs and test them prior to submitting them back upstream for review.