# Kolla-ansible 源码分析

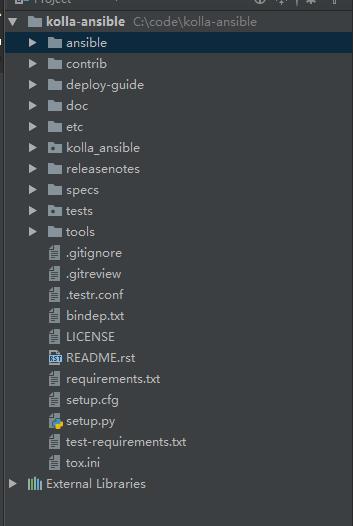
# 简介

Kolla-ansible项目提供一个完整的Ansible Playbook，来部署Docker的镜像，再完成openstack组件的自动化部署。并提供all-in-one和multihost的环境。

源码地址：<https://github.com/openstack/kolla-ansible.git>

# 源码目录概要

1. 一级目录



Ansible: ansible的整个playbook代码,包括部署docker容器和openstack组件。源码主要集中在这个目录下。

Contrib:包括用heat和magnum的部署环境和vagrant的部署环境。

Deploy-guide: 部署指南，主要包括all-in-one和mulihosts两种部署方式的指南。

Doc:文档。

Etc: 一些配置文件，安装完了引用到了/etc目录下，all-in-one只要修改很少量的配置。

Kolla-ansible: 记录版本信息，cmd子目录下有生成密码和合并密码的两个脚本。pbr打包的时候被封装成为可执行命令。

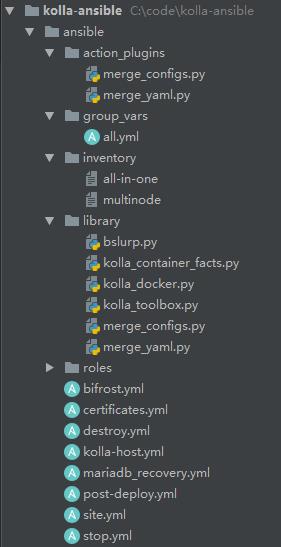
Releasenodes: 发布特性说明。

Specs: 包含有Kolla社区关键参数代码库的变化。

Tests: 包括一些功能测试工具，这里还包括两个自定义的ansible plugin（merge\_config）和module（kolla\_docker）的测试。

Tools: 一些和kolla交换的脚本工具，大部分是可手动调用，主要完成一些安装前后的一些操作。有些会被ansible目录下的task调用到。

1. 二级目录



Ansible/action\_plugins: 自定义ansible插件，两个脚本，用于合并yml和conifg的配置文件。

Ansible/group\_vars: ansible脚本的全局变量定义。

Ansible/inventory: 包含all-in-one和mulitnode的样板hosts清单。

Ansible/library: 包括一些自定义的ansible模块，bslurp.py和kolla\_docker.py用到比较多。

Ansible/role: 所有的openstack的组件，几乎包含了说有开源项目，当前版本有60个组件。

Ansible:除了文件夹之外的ansible脚本，主要用户安装前后的环境准备和清理，数据库恢复等特殊系统级的操作。

# 关键代码解读

## Setup.cfg安装配置入口文件， 见中文注释

|  |
| --- |
| [metadata] name = kolla-ansible //项目名称 summary = Ansible Deployment of Kolla containers description-file =  README.rst author = OpenStack author-email = openstack-dev@lists.openstack.org home-page = http://docs.openstack.org/developer/kolla-ansible/ license = Apache License, Version 2.0 classifier =  Environment :: OpenStack  Intended Audience :: Information Technology  Intended Audience :: System Administrators  License :: OSI Approved :: Apache Software License  Operating System :: POSIX :: Linux  Programming Language :: Python  Programming Language :: Python :: 2  Programming Language :: Python :: 2.7  Programming Language :: Python :: 3  Programming Language :: Python :: 3.5  [files] packages =  kolla\_ansible //包名称 data\_files = //pbr方式打包对应的文件映射  share/kolla-ansible/ansible = ansible/\*  share/kolla-ansible/tools = tools/validate-docker-execute.sh  share/kolla-ansible/tools = tools/cleanup-containers  share/kolla-ansible/tools = tools/cleanup-host  share/kolla-ansible/tools = tools/cleanup-images  share/kolla-ansible/tools = tools/stop-containers  share/kolla-ansible/doc = doc/\*  share/kolla-ansible/etc\_examples = etc/\*  share/kolla-ansible = tools/init-runonce  share/kolla-ansible = tools/init-vpn  share/kolla-ansible = tools/openrc-example  share/kolla-ansible = setup.cfg  scripts = //可执行脚本  tools/kolla-ansible  [entry\_points] console\_scripts = //控制台可执行脚本，执行两个Python文件的main函数  kolla-genpwd = kolla\_ansible.cmd.genpwd:main  kolla-mergepwd = kolla\_ansible.cmd.mergepwd:main  [global] setup-hooks =  pbr.hooks.setup\_hook  [pbr] //打包方式  [build\_sphinx] all\_files = 1 build-dir = doc/build source-dir = doc  [build\_releasenotes] all\_files = 1 build-dir = releasenotes/build source-dir = releasenotes/source |

## Setup.py

安装执行脚本，通过pbr打包，执行过程会读取setup.cfg配置，还会安装同父目录下requirements.txt中的依赖。更多参考

<https://julien.danjou.info/blog/2017/packaging-python-with-pbr>

|  |
| --- |
| **import** setuptools  # In python < 2.7.4, a lazy loading of package `pbr` will break # setuptools if some other modules registered functions in `atexit`. # solution from: http://bugs.python.org/issue15881#msg170215 **try**:  **import** multiprocessing # noqa **except** ImportError:  **pass** setuptools.setup(  setup\_requires=['pbr>=2.0.0'],  pbr=**True**) |

## tools\kolla-ansible

该脚本是封装了ansible-playbook，对kolla进行了ansible的定制。主要根据action的类型，传递不同的配置文件。

中间基础变量定义：

|  |
| --- |
| *find\_base\_dir* INVENTORY="**$**{BASEDIR}/ansible/inventory/all-in-one" PLAYBOOK="**$**{BASEDIR}/ansible/site.yml" VERBOSITY= EXTRA\_OPTS=**$**{EXTRA\_OPTS} CONFIG\_DIR="/etc/kolla" PASSWORDS\_FILE="**$**{CONFIG\_DIR}/passwords.yml" DANGER\_CONFIRM= INCLUDE\_IMAGES=  Find\_base\_dir是一个脚本开始时候的一个函数（不展开解释），用于找到kolla-ansible脚本所在的路径。 |

脚本传参解释：

|  |
| --- |
| **while [** "$#" **-gt** 0 **]**; **do  case** "$1" **in** (--inventory|-i)  INVENTORY="$2"  *shift* 2  **;;** (--playbook|-p)  PLAYBOOK="$2"  *shift* 2  **;;** (--tags|-t)  EXTRA\_OPTS="$EXTRA\_OPTS --tags $2"  *shift* 2  **;;** (--verbose|-v)  VERBOSITY="$VERBOSITY --verbose"  *shift* 1  **;;** (--configdir)  CONFIG\_DIR="$2"  *shift* 2  **;;** (--yes-i-really-really-mean-it)  DANGER\_CONFIRM="$1"  *shift* 1  **;;** (--include-images)  INCLUDE\_IMAGES="$1"  *shift* 1  **;;** (--key|-k)  VAULT\_PASS\_FILE="$2"  EXTRA\_OPTS="$EXTRA\_OPTS --vault-password-file=$VAULT\_PASS\_FILE"  *shift* 2  **;;** (--extra|-e)  EXTRA\_OPTS="$EXTRA\_OPTS -e $2"  *shift* 2  **;;** (--passwords)  PASSWORDS\_FILE="$2"  *shift* 2  **;;** (--help|-h)  *usage  shift  exit* 0  **;;** (--)  *shift  break* **;;** (\*)  *echo* "error"  *exit* 3  **;; esac done  case** "$1" **in** (prechecks)  ACTION="Pre-deployment checking"  EXTRA\_OPTS="$EXTRA\_OPTS -e action=precheck"  **;;** (check)  ACTION="Post-deployment checking"  EXTRA\_OPTS="$EXTRA\_OPTS -e action=check"  **;;** (mariadb\_recovery)  ACTION="Attempting to restart mariadb cluster"  EXTRA\_OPTS="$EXTRA\_OPTS -e action=deploy -e common\_run=true"  PLAYBOOK="**$**{BASEDIR}/ansible/mariadb\_recovery.yml"  **;;** (destroy)  ACTION="Destroy Kolla containers, volumes and host configuration"  PLAYBOOK="**$**{BASEDIR}/ansible/destroy.yml"  **if [[** "**$**{INCLUDE\_IMAGES}" **==** "--include-images" **]]**; **then** EXTRA\_OPTS="$EXTRA\_OPTS -e destroy\_include\_images=yes"  **fi  if [[** "**$**{DANGER\_CONFIRM}" **!=** "--yes-i-really-really-mean-it" **]]**; **then** *cat <<* **EOF** *WARNING:  This will PERMANENTLY DESTROY all deployed kolla containers, volumes and host configuration.  There is no way to recover from this action. To confirm, please add the following option:  --yes-i-really-really-mean-it* **EOF** *exit* 1  **fi  ;;** (bootstrap-servers)  ACTION="Bootstraping servers"  PLAYBOOK="**$**{BASEDIR}/ansible/kolla-host.yml"  EXTRA\_OPTS="$EXTRA\_OPTS -e action=bootstrap-servers"  **;;** (deploy)  ACTION="Deploying Playbooks"  EXTRA\_OPTS="$EXTRA\_OPTS -e action=deploy"  **;;** (deploy-bifrost)  ACTION="Deploying Bifrost"  PLAYBOOK="**$**{BASEDIR}/ansible/bifrost.yml"  EXTRA\_OPTS="$EXTRA\_OPTS -e action=deploy"  **;;** (deploy-servers)  ACTION="Deploying servers with bifrost"  PLAYBOOK="**$**{BASEDIR}/ansible/bifrost.yml"  EXTRA\_OPTS="$EXTRA\_OPTS -e action=deploy-servers"  **;;** (post-deploy)  ACTION="Post-Deploying Playbooks"  PLAYBOOK="**$**{BASEDIR}/ansible/post-deploy.yml"  **;;** (pull)  ACTION="Pulling Docker images"  EXTRA\_OPTS="$EXTRA\_OPTS -e action=pull"  **;;** (upgrade)  ACTION="Upgrading OpenStack Environment"  EXTRA\_OPTS="$EXTRA\_OPTS -e action=upgrade -e serial=**$**{ANSIBLE\_SERIAL}"  **;;** (reconfigure)  ACTION="Reconfigure OpenStack service"  EXTRA\_OPTS="$EXTRA\_OPTS -e action=reconfigure -e serial=**$**{ANSIBLE\_SERIAL}"  **;;** (stop)  ACTION="Stop Kolla containers"  PLAYBOOK="**$**{BASEDIR}/ansible/stop.yml"  **;;** (certificates)  ACTION="Generate TLS Certificates"  PLAYBOOK="**$**{BASEDIR}/ansible/certificates.yml"  **;;** (genconfig)  ACTION="Generate configuration files for enabled OpenStack services"  EXTRA\_OPTS="$EXTRA\_OPTS -e action=config"  **;;** (\*) *usage  exit* 0  **;; esac**  这段是根据传递的参数，不同的参数针对不同的配置文件等额外属性。这里第一个参数有好多action，如deploy，post-deploy，stop等等。 |

最后三行，组合命令，执行：

|  |
| --- |
| CONFIG\_OPTS="-e @**$**{CONFIG\_DIR}/globals.yml -e @**$**{PASSWORDS\_FILE} -e CONFIG\_DIR=**$**{CONFIG\_DIR}" CMD="ansible-playbook -i $INVENTORY $CONFIG\_OPTS $EXTRA\_OPTS $PLAYBOOK $VERBOSITY" *process\_cmd*  传递进来的参数组合成ansible-playbook的CMD命令后，调用process\_cmd函数执行。 |

例子1：初始命令

kolla-ansible deploy -i /home/all-in-one

封装后命令

ansible-playbook -i /home/all-in-one -e @/etc/kolla/globals.yml -e @/etc/kolla/passwords.yml -e CONFIG\_DIR=/etc/kolla -e action=deploy /usr/share/kolla-ansible/ansible/site.yml

例子2：初始命令

kolla-ansible post-deploy

封装后命令

ansible-playbook -i /usr/share/kolla-ansible/ansible/inventory/all-in-one -e @/etc/kolla/globals.yml -e @/etc/kolla/passwords.yml -e CONFIG\_DIR=/etc/kolla /usr/share/kolla-ansible/ansible/post-deploy.yml

## ansible剧本代码解读

由于openstack的组件比较多，且大多数处于并列关系。这里不再一一展开，以neutron为例进行解读。其他没有涉及的有重要的点，会进行内容穿插。

### Ansible/library/

该目录下是一些自定义的一些模块，这些module在目标节点上运行，包括bslurp.py，kolla\_container\_facts.py，kolla\_docker.py，kolla-toolbox.py，merge\_configs.py，merge\_yaml.py，前四个我会一一介绍，后两个是空文件，代码其实在action\_plugins目录中（那两个空文件是映射同名的action,我们可以向module一样使用action）。

**Bslurp.py**，好像做了文件分发的事情，通过copy\_from\_host（我没查到调用它的地方）和copy\_to\_host（在ceph角色部署的时候，下发keyring到osd节点是通过这个模块函数下发的）两个函数实现，。判断依据是模块参数dest。见代码中文注释。

**def copy\_from\_host**(module):  
 #此处省略不少代码  
 module.exit\_json(content=base64.b64encode(data), sha1=sha1, mode=mode,  
 source=src)  
  
**def copy\_to\_host**(module):  
 compress = module.params.get('compress')  
 dest = module.params.get('dest')  
 mode = int(module.params.get('mode'), 0)  
 sha1 = module.params.get('sha1')

# src是加密后的数据  
 src = module.params.get('src')  
 # decode已经加密的数据  
 data = base64.b64decode(src)

#解压数据  
 raw\_data = zlib.decompress(data) **if** compress **else** data

#sha1安全算法数据校验  
 **if** sha1:  
 **if** os.path.exists(dest):  
 **if** os.access(dest, os.R\_OK):  
 **with** open(dest, 'rb') **as** f:  
 **if** hashlib.sha1(f.read()).hexdigest() == sha1:  
 module.exit\_json(changed=**False**)  
 **else**:  
 module.exit\_json(failed=**True**, changed=**False**,  
 msg='file is not accessible: {}'.format(dest))  
  
 **if** sha1 != hashlib.sha1(raw\_data).hexdigest():  
 module.exit\_json(failed=**True**, changed=**False**,  
 msg='sha1 sum does not match data')

# 保存数据到dest值。这段代码有健壮性问题，没有考虑到磁盘写满的场景，保险的做法创建一个tmp文件，把数据拷贝到tmp文件，再把tmp文件重命名为dest值。否则容易把文件写空。  
 **with** os.fdopen(os.open(dest, os.O\_WRONLY | os.O\_CREAT, mode), 'wb') **as** f:  
 f.write(raw\_data)  
 #调用module要求的exit\_json接口退出。  
 module.exit\_json(changed=**True**)  
  
  
**def main**():

# 定义dict类型的参数，ansible module的接口要求  
 argument\_spec = dict(  
 compress=dict(default=**True**, type='bool'),  
 dest=dict(type='str'),  
 mode=dict(default='0644', type='str'),  
 sha1=dict(default=**None**, type='str'),  
 src=dict(required=**True**, type='str')  
 )

# 创建ansible模块对象  
 module = AnsibleModule(argument\_spec)  
 # 获取模块dest参数值  
 dest = module.params.get('dest')  
  
 **try**:  
 **if** dest:

# 如果dest参数存在，则推送操作，push下发到相应的host  
 copy\_to\_host(module)  
 **else**:

# 如果dest参数不存在，则进行pull操作。  
 copy\_from\_host(module)  
 **except** Exception:

# 异常场景下退出，ansible自定义模块语法规范。   
 module.exit\_json(failed=**True**, changed=**True**,  
 msg=repr(traceback.format\_exc()))  
  
  
# import module snippets  
**from** ansible.module\_utils.basic **import** \* # noqa  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 main()

kolla\_docker.py，容器相关的操作，openstack的组件都通过容器部署，每个组件role的部署都会用到，非常重要。

**...**

**#创建docker的client函数**

**def get\_docker\_client**():  
 **try**:  
 **return** docker.Client  
 **except** AttributeError:  
 **return** docker.APIClient  
  
  
**class DockerWorker**(object):  
  
 **def** \_\_init\_\_(self, module):

# 构造函数，传入参数是AnsibleModule类型的对象  
 self.module = module

# params参数续传  
 self.params = self.module.params  
 self.changed = **False** # TLS not fully implemented  
 # tls\_config = self.generate\_tls()

# 创建一个docker.client对象  
 options = {  
 'version': self.params.get('api\_version')  
 }  
 self.dc = get\_docker\_client()(\*\*options)

# ....

# ....

# 启动容器的函数，是AnsibleModule的其中一种action

**def start\_container**(self):

#检查镜像是否存在，不存在pull  
 **if not** self.check\_image():  
 self.pull\_image()

#检查容器  
 container = self.check\_container()

#容器异样，则删除，再回调  
 **if** container **and** self.check\_container\_differs():  
 self.stop\_container()  
 self.remove\_container()  
 container = self.check\_container()

#容器不存在，创建，再回调  
 **if not** container:  
 self.create\_container()  
 container = self.check\_container()

#容器状态非启动，则启动  
 **if not** container['Status'].startswith('Up '):  
 self.changed = **True** self.dc.start(container=self.params.get('name'))  
  
 # We do not want to detach so we wait around for container to exit

#如果container没有detach断开，那么进入wait状态,调用fail\_json方法，传递fail的参数

**if not** self.params.get('detach'):  
 rc = self.dc.wait(self.params.get('name'))  
 **if** rc != 0:  
 self.module.fail\_json(  
 failed=**True**,  
 changed=**True**,  
 msg="Container exited with non-zero return code"  
 )

#如果返回参数remove\_on\_exit，那么删除该container  
 **if** self.params.get('remove\_on\_exit'):  
 self.stop\_container()  
 self.remove\_container()

**def generate\_module**():  
 # NOTE(jeffrey4l): add empty string '' to choices let us use  
 # pid\_mode: "{{ service.pid\_mode | default ('') }}" in yaml

#定义参数字典，ansible module的api规范  
 argument\_spec = dict(  
 common\_options=dict(required=**False**, type='dict', default=dict()),

#action参数，必须传递，类型为str,value值必须在choices的列表  
 action=dict(required=**True**, type='str',  
 choices=['compare\_container', 'compare\_image',  
 'create\_volume', 'get\_container\_env',  
 'get\_container\_state', 'pull\_image',  
 'recreate\_or\_restart\_container',  
 'remove\_container', 'remove\_volume',  
 'restart\_container', 'start\_container',  
 'stop\_container']),  
 api\_version=dict(required=**False**, type='str', default='auto'),  
 auth\_email=dict(required=**False**, type='str'),  
 auth\_password=dict(required=**False**, type='str'),  
 auth\_registry=dict(required=**False**, type='str'),  
 auth\_username=dict(required=**False**, type='str'),  
 detach=dict(required=**False**, type='bool', default=**True**),  
 labels=dict(required=**False**, type='dict', default=dict()),  
 name=dict(required=**False**, type='str'),  
 environment=dict(required=**False**, type='dict'),  
 image=dict(required=**False**, type='str'),  
 ipc\_mode=dict(required=**False**, type='str', choices=['host', '']),  
 cap\_add=dict(required=**False**, type='list', default=list()),  
 security\_opt=dict(required=**False**, type='list', default=list()),  
 pid\_mode=dict(required=**False**, type='str', choices=['host', '']),  
 privileged=dict(required=**False**, type='bool', default=**False**),  
 graceful\_timeout=dict(required=**False**, type='int', default=10),  
 remove\_on\_exit=dict(required=**False**, type='bool', default=**True**),  
 restart\_policy=dict(required=**False**, type='str', choices=[  
 'no',  
 'never',  
 'on-failure',  
 'always',  
 'unless-stopped']),  
 restart\_retries=dict(required=**False**, type='int', default=10),  
 tls\_verify=dict(required=**False**, type='bool', default=**False**),  
 tls\_cert=dict(required=**False**, type='str'),  
 tls\_key=dict(required=**False**, type='str'),  
 tls\_cacert=dict(required=**False**, type='str'),  
 volumes=dict(required=**False**, type='list'),  
 volumes\_from=dict(required=**False**, type='list')  
 )

# 属性依赖ansible module的api规范，如start\_container这个action, #必须要image和name这个两个属性。  
 required\_if = [  
 ['action', 'pull\_image', ['image']],  
 ['action', 'start\_container', ['image', 'name']],  
 ['action', 'compare\_container', ['name']],  
 ['action', 'compare\_image', ['name']],  
 ['action', 'create\_volume', ['name']],  
 ['action', 'get\_container\_env', ['name']],  
 ['action', 'get\_container\_state', ['name']],  
 ['action', 'recreate\_or\_restart\_container', ['name']],  
 ['action', 'remove\_container', ['name']],  
 ['action', 'remove\_volume', ['name']],  
 ['action', 'restart\_container', ['name']],  
 ['action', 'stop\_container', ['name']]  
 ]  
 #实例化

module = AnsibleModule(  
 argument\_spec=argument\_spec,  
 required\_if=required\_if,  
 bypass\_checks=**False** )

#以下部分主要做环境变量和通用参数以及特殊参数的更新。  
 new\_args = module.params.pop('common\_options', dict())  
  
 # NOTE(jeffrey4l): merge the environment  
 env = module.params.pop('environment', dict())  
 **if** env:  
 new\_args['environment'].update(env)  
  
 **for** key, value **in** module.params.items():  
 **if** key **in** new\_args **and** value **is None**:  
 **continue** new\_args[key] = value  
  
 # if pid\_mode = ""/None/False, remove it  
 **if not** new\_args.get('pid\_mode', **False**):  
 new\_args.pop('pid\_mode', **None**)  
 # if ipc\_mode = ""/None/False, remove it  
 **if not** new\_args.get('ipc\_mode', **False**):  
 new\_args.pop('ipc\_mode', **None**)  
  
 module.params = new\_args

# 返回为AnsibleModule实例  
 **return** module  
  
  
**def main**():  
 module = generate\_module()  
  
 **try**:  
 dw = DockerWorker(module)  
 # *TODO(inc0): We keep it bool to have ansible deal with consistent* # types. If we ever add method that will have to return some  
 # meaningful data, we need to refactor all methods to return dicts.  
 #返回值 result是action传递的函数名的运行成功与否的结果，意义#不大

result = bool(getattr(dw, module.params.get('action'))())  
 module.exit\_json(changed=dw.changed, result=result)  
 **except** Exception:  
 module.exit\_json(failed=**True**, changed=**True**,  
 msg=repr(traceback.format\_exc()))  
  
# import module snippets  
**from** ansible.module\_utils.basic **import** \* # noqa  
**if** \_\_name\_\_ == '\_\_main\_\_':  
 main()

**Kolla\_toolbox.py**，在toolbox容器中运行ansible命令。这个我就不展开了，只贴关键代码。

**#生成commandline函数，只包含ansible的命令**

**def gen\_commandline**(params):  
 command = ['ansible', 'localhost']  
 **....**

**....**  
 **return** command

**#主函数**

**def main**():  
 ....

....  
 client = get\_docker\_client()(  
 version=module.params.get('api\_version'))

#调用函数，获取命令（dict类型）  
 command\_line = gen\_commandline(module.params)

#过滤名称为kolla\_toolbox的容器列表  
 kolla\_toolbox = client.containers(filters=dict(name='kolla\_toolbox',  
 status='running'))  
 **if not** kolla\_toolbox:  
 module.fail\_json(msg='kolla\_toolbox container is not running.')  
 #默认只有一个，所有选了数组的第一个。kolla\_toolbox变量名不建议重复使用，开源代码就是坑。  
 kolla\_toolbox = kolla\_toolbox[0]

#在容器中执行命令  
 job = client.exec\_create(kolla\_toolbox, command\_line)  
 output = client.exec\_start(job)

....

....  
 module.exit\_json(\*\*ret)

**Kolla\_container\_facts.py** 调用dockerclient的python接口获取指定容器的facts信息，只传递一个name值即可。result类型是dict(changed=xxx, \_containers=[]),代码不展开了。

### Ansible/action\_plugins/

该目录下记录了自定义的action的plugins，这些plugins在master上运行。但也可以在library目录下定义同名空文件，可以当作module使用。这里有两个代码文件，merge\_configs.py和merge\_yaml.py，用于conf和yml配置文件的合并。这里就分下下merge\_yaml.py这个action plugin.

Merge\_yaml.py，在task的参数sources中传递多个yml文件，合并之后输出到目标节点的dest中。期间在合并的同时，进行了参数模拟变量的渲染工作，最后调用copy模块把渲染后的数据文件复制过去。分析代码如下：

**from** ansible.plugins **import** action  
  
#继承父类action.ActionBase  
**class ActionModule**(action.ActionBase):  
  
 TRANSFERS\_FILES = **True  
  
 def read\_config**(self, source):  
 result = **None** # Only use config if present  
 **if** os.access(source, os.R\_OK):  
 **with** open(source, 'r') **as** f:  
 template\_data = f.read()  
 # 渲染template模板数据，因为最终执行copy模块的时候，  
 # 变量被重新还原了，所以这里要先template变量先渲染，  
 # 因为有些变量对可能在copy模块中会消失  
 template\_data = self.\_templar.template(template\_data)  
 #把YAML数据，转化为dict对象  
 result = safe\_load(template\_data)  
 **return** result **or** {}  
  
 # 自定义action plugin必须实现的方法  
 **def run**(self, tmp=**None**, task\_vars=**None**):  
 #task\_vars是这个task的一些外传入的变量,  
 # 如host vars, group vars, config vars,etc  
 **if** task\_vars **is None**:  
 task\_vars = dict()  
 #自定义action plugin必须调用父类的run方法  
 result = super(ActionModule, self).run(tmp, task\_vars)  
  
 # NOTE(jeffrey4l): Ansible 2.1 add a remote\_user param to the  
 # \_make\_tmp\_path function. inspect the number of the args here. In  
 # this way, ansible 2.0 and ansible 2.1 are both supported  
 #创建tmp临时目录，兼容2.0以后的版本  
 make\_tmp\_path\_args = inspect.getargspec(self.\_make\_tmp\_path)[0]  
 **if not** tmp **and** len(make\_tmp\_path\_args) == 1:  
 tmp = self.\_make\_tmp\_path()  
 **if not** tmp **and** len(make\_tmp\_path\_args) == 2:  
 remote\_user = (task\_vars.get('ansible\_user')  
 **or** self.\_play\_context.remote\_user)  
 tmp = self.\_make\_tmp\_path(remote\_user)  
 **# save template args.  
 # \_task.args是这个task的参数，这里把参数中key为vars的对应值  
 # 保存到extra\_vars变量中** extra\_vars = self.\_task.args.get('vars', list())  
 **# 备份template的可用变量**  
 old\_vars = self.\_templar.\_available\_variables  
 **# 将task\_vars和extra\_vars的所有变量merge到一起，赋值到temp\_vars** temp\_vars = task\_vars.copy()  
 temp\_vars.update(extra\_vars)  
 **#把最新的变量数据设置到templar对象（模板对象）**  
 self.\_templar.set\_available\_variables(temp\_vars)  
  
 output = {}  
 # 获取task的参数为sources的values值,可能是单个文件，  
 # 也有可能是多个文件组成的list  
 sources = self.\_task.args.get('sources', **None**)  
 #非数组，转化为只有一个item的数组  
 **if not** isinstance(sources, list):  
 sources = [sources]  
 #便历sources数组，读取文件中内容，并合并更新  
 #dict.update方式有去重效果，相当于merge  
 **for** source **in** sources:  
 output.update(self.read\_config(source))  
  
 # restore original vars  
 #还原templar对象的变量  
 self.\_templar.set\_available\_variables(old\_vars)  
 #把最新的合并好的数据output传递到远端的target host。复制给xfered变量  
 remote\_path = self.\_connection.\_shell.join\_path(tmp, 'src')  
 xfered = self.\_transfer\_data(remote\_path,  
 dump(output,  
 default\_flow\_style=**False**))  
 #把本task的参数拷贝，作为新模块的参数new\_module\_args  
 new\_module\_args = self.\_task.args.copy()  
 #更新new\_module\_args的src的值,后面copy模块的参数要求  
 new\_module\_args.update(  
 dict(  
 src=xfered  
 )  
 )  
 #删除new\_module\_args的sources参数，后面copy模块的参数要求  
 **del** new\_module\_args['sources']  
 #传入最新的参数new\_module\_args，task\_vars执行copy的module  
 result.update(self.\_execute\_module(module\_name='copy',  
 module\_args=new\_module\_args,  
 task\_vars=task\_vars,  
 tmp=tmp))  
 #返回result， action plugin 接口的要求  
 **return** result

### Ansible/inventory/all-in-one

#control主机组包含本地localhost节点，连接方式为local

[control]  
localhost ansible\_connection=local

[network]  
localhost ansible\_connection=local

#neutron主机组包含network组下的所有节点

[neutron:children]  
network

# Neutron

#neutron-server主机组包含control组下的所有节点  
[neutron-server:children]  
control

#neutron-dhcp-agent主机组包含neutron组下的所有节点  
[neutron-dhcp-agent:children]  
neutron  
  
[neutron-l3-agent:children]  
neutron  
  
[neutron-lbaas-agent:children]  
neutron  
  
[neutron-metadata-agent:children]  
neutron  
  
[neutron-vpnaas-agent:children]  
neutron  
  
[neutron-bgp-dragent:children]  
neutron

### Ansible/site.yml

#调用ansible的setup获取节点的facts。gather\_facts被设置为false是为了避免ansible再次去gathering facts.

- name: Gather facts for all hosts  
 hosts: all  
 serial: '{{ serial|default("0") }}'  
 gather\_facts: **false** tasks:  
 - setup:  
 tags: always  
  
# NOTE(pbourke): This case covers deploying subsets of hosts using --limit. The  
# limit arg will cause the first play to gather facts only about that node,  
# meaning facts such as IP addresses for rabbitmq nodes etc. will be undefined  
# in the case of adding a single compute node.  
# We don't want to add the delegate parameters to the above play as it will  
# result in ((num\_nodes-1)^2) number of SSHs when running for all nodes  
# which can be very inefficient.

- name: Gather facts for all hosts (if using --limit)  
 hosts: all  
 serial: '{{ serial|default("0") }}'  
 gather\_facts: **false** tasks:  
 - setup:  
 delegate\_facts: **True** delegate\_to: "{{ item }}"  
 with\_items: "{{ groups['all'] }}"  
 when:  
 - (play\_hosts | length) != (groups['all'] | length)

#检测openstack\_release全局变量信息，默认在globals.yml是不配置的，而

#在ansible/group\_vars/all.yml中配置的默认值是auto。这里的两个tasks

#就是如果在auto的场景下，通过python的pbr包去检测安装好的#kolla-ansible版本，再将该版本号赋值给openstack\_release变量。

#这里用到了ansible自带的local\_action模块和register中间信息存储模块。  
- name: Detect openstack\_release variable  
 hosts: all  
 gather\_facts: **false** tasks:  
 - name: Get current kolla-ansible version number  
 local\_action: command python -c "import pbr.version; print(pbr.version.VersionInfo('kolla-ansible'))"  
 register: kolla\_ansible\_version  
 changed\_when: **false** when: openstack\_release == "auto"  
  
 - name: Set openstack\_release variable  
 set\_fact:  
 openstack\_release: "{{ kolla\_ansible\_version.stdout }}"  
 when: openstack\_release == "auto"  
 tags: always

#对所有节点进行recheck检查，前提条件是ansible-playbook命令传递的#action的值是precheck。  
- name: Apply role prechecks  
 gather\_facts: **false** hosts:  
 - all  
 roles:  
 - role: prechecks  
 when: action == "precheck"

# 基于ntp时间同步角色的部署，hosts组为chrony-server和chrony

#前提条件是enable\_chrony变量是否是yes，该值可在etc/kolla/globals.yml

#中配置，默认是no。  
- name: Apply role chrony  
 gather\_facts: **false** hosts:  
 - chrony-server  
 - chrony  
 serial: '{{ serial|default("0") }}'  
 roles:  
 - { role: chrony,  
 tags: chrony,  
 when: enable\_chrony | bool }

#部署neutron角色，这里部署的节点除了neutron相关的host组之外，还包括#compute和manila-share（openstack的一个文件共享组件）组。

- name: Apply role neutron  
 gather\_facts: **false** hosts:  
 - neutron-server  
 - neutron-dhcp-agent  
 - neutron-l3-agent  
 - neutron-lbaas-agent  
 - neutron-metadata-agent  
 - neutron-vpnaas-agent  
 - compute  
 - manila-share  
 serial: '{{ serial|default("0") }}'  
 roles:  
 - { role: neutron,  
 tags: neutron,  
 when: enable\_neutron | bool }

l

### Ansible/role/neutron/task 检查场景

该场景的action是precheck。由tasks/main.yml引用precheck.yml。

**precheck.yml**

---  
# kolla\_container\_facts是自定义的library，上文已经分析过代码，  
# 用于获取容器名为neutron\_server的一些容器属性数据，注册到中间变量container\_facts  
- name: Get container facts  
 kolla\_container\_facts:  
 name:  
 - neutron\_server  
 register: container\_facts  
  
# 中间变量container\_facts没有找到neutron\_server关键字且该主机在neutron-server主机组中，  
# 判断neutron\_server\_port 端口是否已经stopped  
- name: Checking free port for Neutron Server  
 wait\_for:  
 host: "{{ hostvars[inventory\_hostname]['ansible\_' + api\_interface]['ipv4']['address'] }}"  
 port: "{{ neutron\_server\_port }}"  
 connect\_timeout: 1  
 timeout: 1  
 state: stopped  
 when:  
 - container\_facts['neutron\_server'] is not defined  
 - inventory\_hostname in groups['neutron-server']  
  
# enable\_neutron\_agent\_ha是true,且只规划了多个一个dhcp和l3服务节点，给出fail提示  
- name: Checking number of network agents  
 local\_action: fail msg="Number of network agents are less than two when enabling agent ha"  
 changed\_when: **false** when:  
 - enable\_neutron\_agent\_ha | bool  
 - groups['neutron-dhcp-agent'] | length < 2  
 or groups['neutron-l3-agent'] | length < 2  
  
# When MountFlags is set to shared, a signal bit configured on 20th bit of a number  
# We need to check the 20th bit. 2^20 = 1048576. So we are validating against it.  
# 检查docker服务的MountFlags是否设置为了shared  
- name: Checking if 'MountFlags' for docker service is set to 'shared'  
 command: systemctl show docker  
 register: result  
 changed\_when: **false** failed\_when: result.stdout.find('MountFlags=1048576') == -1  
 when:  
 - (inventory\_hostname in groups['neutron-dhcp-agent']  
 or inventory\_hostname in groups['neutron-l3-agent']  
 or inventory\_hostname in groups['neutron-metadata-agent'])  
 - ansible\_os\_family == 'RedHat' or ansible\_distribution == 'Ubuntu'

### Ansible/role/neutron/task 部署场景

该场景的action是deploy。由tasks/main.yml引用deploy.yml

**deploy.yml**

# enforce ironic usage only with openvswitch  
# 裸机部署检查，检查ironic服务必须启动，neutron的plugin必须使用OpenvSwitch

- include: ironic-check.yml  
  
#在neutron-server的节点执行注册  
- include: register.yml  
 when: inventory\_hostname in groups['neutron-server']  
  
#执行配置，拷贝配置文件，启动组件容器主要都在这里实现  
- include: config.yml  
  
#在nova fake driver模拟场景下，计算节点执行config-neutron-fake.yml，不详细分析  
#nova fake driver可以在单个计算节点中创建多个docker容器运行novc-compute，  
#Nova fake driver can not work with all-in-one deployment. This is because the fake  
#neutron-openvswitch-agent for the fake nova-compute container conflicts with  
#neutron-openvswitch-agent on the compute nodes. Therefore, in the inventory  
#the network node must be different than the compute node.  
- include: config-neutron-fake.yml  
 when:  
 - enable\_nova\_fake | bool  
 - inventory\_hostname in groups['compute']  
  
#在neutron-server的节点执行创建数据库，创建容器  
#bootstrap.yml会去创建数据库相关信息，结束后会去调用#bootstrap\_servcie.yml该文件是用于在server节点上创建容器。

- include: bootstrap.yml  
 when: inventory\_hostname in groups['neutron-server']  
  
#执行handlers目录下的task任务  
- name: Flush Handlers  
 meta: flush\_handlers

**Register.yml,** 往keystone中注册neutron服务的鉴权相关信息。

---  
# 在keystone创建neutron的service 和endpoint  
# kolla\_toolbox见library分析，用于在toolbox容器中执行ansible命令  
# kolla\_keystone\_service模块是kolla-ansible的父项目kolla中的代码，已经是一个可调用的ansible模块  
#service名称为neutron,对应的endpoint分为内部，管理员，公共三个。  
# 变量主要在ansible/role/neutron/defauts/main.yml和ansible/group\_vars/all.yml中  
- name: Creating the Neutron service and endpoint  
 kolla\_toolbox:  
 module\_name: "kolla\_keystone\_service"  
 module\_args:  
 service\_name: "neutron"  
 service\_type: "network"  
 description: "Openstack Networking"  
 endpoint\_region: "{{ openstack\_region\_name }}"  
 url: "{{ item.url }}"  
 interface: "{{ item.interface }}"  
 region\_name: "{{ openstack\_region\_name }}"  
 auth: "{{ '{{ openstack\_neutron\_auth }}' }}"  
 module\_extra\_vars:  
 openstack\_neutron\_auth: "{{ openstack\_neutron\_auth }}"  
 run\_once: **True** with\_items:  
 - {'interface': 'admin', 'url': '{{ neutron\_admin\_endpoint }}'}  
 - {'interface': 'internal', 'url': '{{ neutron\_internal\_endpoint }}'}  
 - {'interface': 'public', 'url': '{{ neutron\_public\_endpoint }}'}  
   
# 同上，创建项目，用户，角色。经分析openstack\_neutron\_auth变量实际为#openstack的admin的auth。  
- name: Creating the Neutron project, user, and role  
 kolla\_toolbox:  
 module\_name: "kolla\_keystone\_user"  
 module\_args:  
 project: "service"  
 user: "{{ neutron\_keystone\_user }}"  
 password: "{{ neutron\_keystone\_password }}"  
 role: "admin"  
 region\_name: "{{ openstack\_region\_name }}"  
 auth: "{{ '{{ openstack\_neutron\_auth }}' }}"  
 module\_extra\_vars:  
 openstack\_neutron\_auth: "{{ openstack\_neutron\_auth }}"  
 run\_once: **True**

**Config.yml**，配置文件合并下发，创建或重启容器。

#调用sysctl模块，配置ip转发相关配置  
- name: Setting sysctl values  
 vars:  
 neutron\_l3\_agent: "{{ neutron\_services['neutron-l3-agent'] }}"  
 neutron\_vpnaas\_agent: "{{ neutron\_services['neutron-vpnaas-agent'] }}"  
 sysctl: name={{ item.name }} value={{ item.value }} sysctl\_set=**yes** with\_items:  
 - { name: "net.ipv4.ip\_forward", value: 1}  
 - { name: "net.ipv4.conf.all.rp\_filter", value: 0}  
 - { name: "net.ipv4.conf.default.rp\_filter", value: 0}  
 when:  
 - set\_sysctl | bool  
 - (neutron\_l3\_agent.enabled | bool and neutron\_l3\_agent.host\_in\_groups | bool)  
 or (neutron\_vpnaas\_agent.enabled | bool and neutron\_vpnaas\_agent.host\_in\_groups | bool)  
  
# 创建neutron各服务的配置文件目录，前提条件主要看host\_in\_groups变量，这个在  
# ansible/role/neutron/defauts/main.yml文件中进行了详细的定义  
- name: Ensuring config directories exist  
 file:  
 path: "{{ node\_config\_directory }}/{{ item.key }}"  
 state: "directory"  
 recurse: **yes** when:  
 - item.value.enabled | bool  
 - item.value.host\_in\_groups | bool  
 with\_dict: "{{ neutron\_services }}"

....

....  
  
#下发配置文件到指定目录，三份陪配置文件合一，merge\_conifgs模块在action plugin中已经分析过了  
#文件下发完了之后，通知相应组件的容器重启。在handlers目录下。  
# 重启容器这个操作会调用recreate\_or\_restart\_container这个action，第一次会创建容器。  
- name: Copying over neutron\_lbaas.conf  
 vars:  
 service\_name: "{{ item.key }}"  
 services\_need\_neutron\_lbaas\_conf:  
 - "neutron-server"  
 - "neutron-lbaas-agent"  
 merge\_configs:  
 sources:  
 - "{{ role\_path }}/templates/neutron\_lbaas.conf.j2"  
 - "{{ node\_custom\_config }}/neutron/neutron\_lbaas.conf"  
 - "{{ node\_custom\_config }}/neutron/{{ inventory\_hostname }}/neutron\_lbaas.conf"  
 dest: "{{ node\_config\_directory }}/{{ item.key }}/neutron\_lbaas.conf"  
 register: neutron\_lbaas\_confs  
 when:  
 - item.value.enabled | bool  
 - item.value.host\_in\_groups | bool  
 - item.key in services\_need\_neutron\_lbaas\_conf  
 with\_dict: "{{ neutron\_services }}"  
 notify:  
 - "Restart {{ item.key }} container"

....

....

*#kolla\_docker是自定义的模块，通过调用compare\_container查找该节点的所有neutron service容器*- name: Check neutron containers  
 kolla\_docker:  
 action: "compare\_container"  
 common\_options: "{{ docker\_common\_options }}"  
 name: "{{ item.value.container\_name }}"  
 image: "{{ item.value.image }}"  
 privileged: "{{ item.value.privileged | default(False) }}"  
 volumes: "{{ item.value.volumes }}"  
 register: check\_neutron\_containers  
 when:  
 - action != "config"  
 - item.value.enabled | bool  
 - item.value.host\_in\_groups | bool  
 with\_dict: "{{ neutron\_services }}"  
 notify:  
 - "Restart {{ item.key }} container"

**Bootstrap.yml**，创建neutron数据库对象。

---  
# kolla\_toolbox自定义模块，在toolbox容器中调用mysql\_db的ansible模块，创建db  
# delegate\_to指定在第一个neutron-server上执行，run\_onece只运行一次  
- name: Creating Neutron database  
 kolla\_toolbox:  
 module\_name: mysql\_db  
 module\_args:  
 login\_host: "{{ database\_address }}"  
 login\_port: "{{ database\_port }}"  
 login\_user: "{{ database\_user }}"  
 login\_password: "{{ database\_password }}"  
 name: "{{ neutron\_database\_name }}"  
 register: database  
 run\_once: **True** delegate\_to: "{{ groups['neutron-server'][0] }}"  
  
# 创建neuron数据库的用户并设置权限  
- name: Creating Neutron database user and setting permissions  
 kolla\_toolbox:  
 module\_name: mysql\_user  
 module\_args:  
 login\_host: "{{ database\_address }}"  
 login\_port: "{{ database\_port }}"  
 login\_user: "{{ database\_user }}"  
 login\_password: "{{ database\_password }}"  
 name: "{{ neutron\_database\_name }}"  
 password: "{{ neutron\_database\_password }}"  
 host: "%"  
 priv: "{{ neutron\_database\_name }}.\*:ALL"  
 append\_privs: "yes"  
 run\_once: **True** delegate\_to: "{{ groups['neutron-server'][0] }}"  
  
#数据库改变之后，调用bootstrap\_service.yml  
- include: bootstrap\_service.yml  
 when: database.changed

**Bootstrap\_service.yml**，创建bootrsap\_neutron，bootrsap\_neutron\_lbassd-agent，bootrsap\_neutron\_vpnaas\_agent容器。

**Ansible/role/neutron/handlers/main.yml**，重建或重启neutron相关容器。列出一个分析下。

# 举例分析：neutron\_lbaas\_confs变量是之前执行的时候注册的变量，这里的task变量  
# neutron\_lbaas\_conf从neutron\_lbaas\_confs的结果中取值分析作为when的条件判断  
# kolla\_docker模块的volumes参数是从role/neutron/defaults/main.yml中获取，如下，  
#volumes:  
# - "{{ node\_config\_directory }}/neutron-lbaas-agent/:{{  
# container\_config\_directory }}/:ro"  
# - "/etc/localtime:/etc/localtime:ro"  
# - "/run:/run:shared"  
# - "kolla\_logs:/var/log/kolla/"  
#且在config.yml已经把neutron\_lbaas的配置文件下发{{ node\_config\_directory }}/  
# neutron-lbaas-agent/这个目录系了。 容器中的路径container\_config\_directory变量  
# 可以在groups\_var中找到,值为 /var/lib/kolla/config\_files。  
- name: Restart neutron-server container  
 vars:  
 service\_name: "neutron-server"  
 service: "{{ neutron\_services[service\_name] }}"  
 config\_json: "{{ neutron\_config\_jsons.results|selectattr('item.key', 'equalto', service\_name)|first }}"  
 neutron\_conf: "{{ neutron\_confs.results|selectattr('item.key', 'equalto', service\_name)|first }}"  
 neutron\_lbaas\_conf: "{{ neutron\_lbaas\_confs.results|selectattr('item.key', 'equalto', service\_name)|first }}"  
 neutron\_ml2\_conf: "{{ neutron\_ml2\_confs.results|selectattr('item.key', 'equalto', service\_name)|first }}"  
 policy\_json: "{{ policy\_jsons.results|selectattr('item.key', 'equalto', service\_name)|first }}"  
 neutron\_server\_container: "{{ check\_neutron\_containers.results|selectattr('item.key', 'equalto', service\_name)|first }}"  
 kolla\_docker:  
 action: "recreate\_or\_restart\_container"  
 common\_options: "{{ docker\_common\_options }}"  
 name: "{{ service.container\_name }}"  
 image: "{{ service.image }}"  
 volumes: "{{ service.volumes }}"  
 privileged: "{{ service.privileged | default(False) }}"  
 when:  
 - action != "config"  
 - service.enabled | bool  
 - service.host\_in\_groups | bool  
 - config\_json | changed  
 or neutron\_conf | changed  
 or neutron\_lbaas\_conf | changed  
 or neutron\_vpnaas\_conf | changed  
 or neutron\_ml2\_conf | changed  
 or policy\_json | changed  
 or neutron\_server\_container | changed

### Ansible/role/neutron/task 下拉镜像

**Pull.yml** 下拉镜像

#调用kolla\_docker的pull\_image 下拉镜像  
- name: Pulling neutron images  
 kolla\_docker:  
 action: "pull\_image"  
 common\_options: "{{ docker\_common\_options }}"  
 image: "{{ item.value.image }}"  
 when:  
 - item.value.enabled | bool  
 - item.value.host\_in\_groups | bool  
 with\_dict: "{{ neutron\_services }}"

# 其他流程介绍

## 镜像路径

kolla-ansible\ansible\roles\chrony\defaults\main.yml文件中的一段，如下

#docker\_registry是本地注册的registry地址，在global.yml中会配置。Registry虚拟机为会从定向5000端口带宿主机的4000端口。

docker\_namespace也在global.yml定义，如果从官网下载源码镜像的话，配置成lokolla。

kolla\_base\_distro默认centos，也可以配置成ubuntu

kolla\_install\_type 默认为binary, 我们配置在global.yml配置成了source

例子：

docker\_registry：192.168.102.15:4000

docker\_namespace：lokolla

kolla\_base\_distro：“centos”

kolla\_install\_type: source

openstack\_release： auto (自发现，前文有讲到)

所以最后的chrony\_image\_full为192.168.102.15:4000/lokolla/centos-source-chrony:4.0.2

chrony\_image: "{{ docker\_registry ~ '/' if docker\_registry else '' }}{{ docker\_namespace }}/{{ kolla\_base\_distro }}-{{ kolla\_install\_type }}-chrony"  
chrony\_tag: "{{ openstack\_release }}"  
chrony\_image\_full: "{{ chrony\_image }}:{{ chrony\_tag }}"