Blue Planet Orchestrate

Resource Adapter Development

Student Lab Guide

Blue Planet DevOps Toolkit 17.06.1

blueplanet

Contacting Ciena

Corporate headquarters	410-694-5700 or 800-921-1144	www.ciena.com
Customer technical support/warranty		
In North America	1-800-CIENA24 (243-6224) 410-865-4961	
In Europe, Middle East, and Africa	800-CIENA-24-7 (800-2436-2247) +44-207-012-5508	
In Asia-Pacific	800-CIENA-24-7 (800-2436-2247) +81-3-6367-3989 +91-124-4340-600	
In Caribbean and Latin America	800-CIENA-24-7 (800-2436-2247) 410-865-4944 (USA)	
Sales and General Information	410-694-5700	E-mail: sales@ciena.com
In North America	410-694-5700 or 800-207-3714	E-mail: sales@ciena.com
In Europe	+44-207-012-5500 (UK)	E-mail: sales@ciena.com
In Asia +81-3-3248-4680 (Japan)		E-mail: sales@ciena.com
In India	+91-124-434-0500	E-mail: sales@ciena.com
In Latin America	011-5255-1719-0220 (Mexico City)	E-mail: sales@ciena.com
Training	877-CIENA-TD (243-6283) or 410-865-8996	E-mail: techtng@ciena.com

For additional office locations and phone numbers, please visit the Ciena website at www.ciena.com.

Contents

Lab 1: Setup the Blue Planet DevOps Toolkit

Helpful hints:

Throughout these labs you can copy and paste commands and data from the `ra-training commands.txt` file found in the `ra_sample_files` folder in your Student Files.

File and directory names will be highlighted in the following font: `/home/vagrant`

Commands will be highlighted in **bold text**.

Important notes and warnings are in red text.

Links are in blue text.

The DevOps Toolkit provides a self-contained environment for developing and testing Blue Planet resource adapters. The toolkit is available on the Blue Planet Developers Exchange at http://community.ciena.com. The toolkit is deployed on your host computer using Vagrant and Virtualbox. The DevOps Vagrant box is a self-contained Blue Planet development environment that includes:

- Docker
- Git
- bpocore-dev docker image
- orchestrate-ui-dev docker image
- RASDK – Resource Adapter Development Kit

For details on system requirements see the information posted on the Blue Planet Developers Exchange.

When developing RAs and service templates you can use your IDE of choice, or a text editor.

If you are using Windows you must have an SSH utility. One solution is Cygwin. It can be downloaded from https://cygwin.com/install.html. When installing Cygwin be sure to install the **OpenSSH** package as this is not selected by default. Cygwin provides a unix-like terminal for Windows.

Note that Cygwin installs other Unix-based software. If you only want a SSH client utility, consider installing PUTTY from http://www.putty.org/.

Task 1: Install Vagrant and Virtualbox

To use the Vagrant box you must install Vagrant and Virtualbox on your host machine. Both are free downloads. Check the DevOps Exchange for details on the version of Vagrant and Virtualbox to use with the DevOps Toolkit.

- Vagrant version 1.9 available here: https://releases.hashicorp.com/vagrant/1.9.0/
- Oracle VirtualBox version 5.1.30 available here: https://www.virtualbox.org/wiki/Download_Old_Builds_5_1
  NOTE: Hardware virtualization **must** be enabled on your PC. See your PC documentation for instructions on enabling hardware virtualization through the BIOS.

Continued on next page…

Task 2: Setup the Vagrant Project

1. Open a command-line window on your computer.
2. Navigate to your home directory or another directory that you can easily get to. The DevOps Toolkit can be in any directory on your computer.
3. Create a directory for the DevOps toolkit, and make it the current directory:

```
$ mkdir ra_devops_toolkit

$ cd ra_devops_toolkit
```

   You can also use Explorer (Windows) or Finder (Mac) to create the directory.

   This is the directory where the DevOps Toolkit will be installed.

   This will be referred to as the *DevOps Toolkit directory* for the rest of the course.

4. Download the toolkit bundle and put it in the DevOps Toolkit directory.
5. Create a `Vagrantfile` in the current directory:

```
 $ vagrant init ra-17.06.1 devops-toolkit-orch-17.06.1.box
```

   `devops-toolkit-orch-17.06.1.box` is the name of the `toolkit .box` file, which is available on the Blue Planet community page. **You must use the exact name of the toolkit .box file**.

   Vagrant creates a `Vagrantfile` in the directory. For details on Vagrantfiles, see Vagrantfiles in the Vagrant documentation.

6. Install Virtual Box Guest Additions:

```
$ vagrant plugin install vagrant-vbguest
```

Task 3: Edit your Vagrantfile

You need to customize your environment by editing the `Vagrantfile`. For more information and an example of customizing VirtualBox, see VBoxManage in the VirtualBox documentation. These instructions provide the minimum customizations needed to run the DevOps Toolkit.

1. Open the `Vagrantfile` in a text editor. Be sure the text editor will not use "smart quotes". **Do not use Wordpad or other Rich Text Format editors.**
2. Around line 28 uncomment the line below to create a private network that allows host-only access. Remove the # at the start of the line to uncomment the line:

```
config.vm.network "private_network", ip: "192.168.33.10"
```

   NOTE: Be sure the IP address parameter is included. On some Windows systems this may not be present.

Continued on next page…

3. Around line 45-51 is a block that is commented out that starts with:

```
# config.vm.provider "virtualbox" do |vb|
```

and ends with:

```
#end
```

Uncomment both of those lines.

4. Between those lines add the following:

```
vb.customize ["modifyvm", :id, "--memory", "4096"]
```

When you're done the block should look like this:

```
config.vm.provider "virtualbox" do |vb|
               vb.customize ["modifyvm", :id, "--memory", "4096"]
  #   # Display the VirtualBox GUI when booting the machine
  #   vb.gui = true
  #
  #   # Customize the amount of memory on the VM:
  #   vb.memory = "1024"
end
```

5. Just before the block of code starting around line 45, add the follow lines to create a shared directory for your host machine and the virtual machine.

```
  config.vm.synced_folder "./shared", "/home/vagrant/shared", type: "nfs"
  config.vm.synced_folder ".", "/vagrant", disabled: true
```

6. Save the file.

Continued on next page…

Task 4: Create a 'shared' Directory

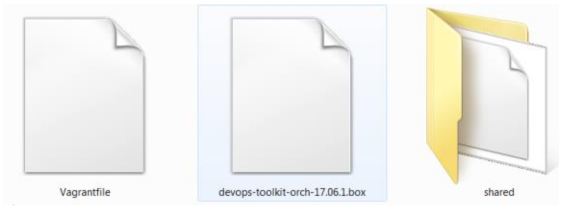
1. In a terminal window on your host computer navigate to DevOps Toolkit directory. (For Windows users, make sure you open a DOS prompt with Administrative rights.)
2. Create a shared directory:

```
$ mkdir shared
```

   You can also use Explorer (Windows) or Finder (Mac) to create the directory.
3. Copy the course sample files directory into the **shared** directory.

When you're done the DevOps Toolkit directory should have:

- **Vagrantfile**
- DevOps **Toolkit.box** file (file name depends on the version)
- **shared** directory with the **bpo_sample_files** directory inside



Continued on next page…

Task 5: Starting the VM

1. Once you have modified the **`Vagrantfile`** and have created the **`shared`** directory you can start the virtual machine and login.
2. In a terminal window on your host computer go to the DevOps Toolkit directory. (For Windows users, make sure you open a DOS prompt with Administrative rights.)
3. Start the VM:

```
$ vagrant up
```

4. Once the VM has booted, connect to the machine.
   For Mac and Linux systems use the Vagrant SSH command:

```
$ vagrant ssh
```

For Windows the above command will work if you are using Cygwin. If not, open an SSH application and connect to the virtual machine using the IP address 192.168.33.10. Use the following credentials:

- Username: *vagrant*
- Password: *vagrant*

Once you have login you should see the default Ubuntu prompt:

```
vagrant@vagrant:~$
```

For the rest of this course the virtual machine is referred to as the *Vagrant development environment*. In future labs, you will be directed to go to the "Vagrant bash prompt". Note that future examples will typically only show a prompt of $ in place of the full prompt (vagrang@vagrant:~$) in order to highlight the commands being executed.

End of Lab

Lab 2: Starting the Orchestrate Docker Containers

In this lab, you will start the docker containers needed for the development environment and then navigate to your Orchestrate web UI to verify the Vagrant development environment is running.

You will need three SSH sessions connected the Vagrant development environment:

- One for bpocore-dev
- One for orchestrate-ui-dev
- One for normal development work

Task 1: Verify Image Versions

1. From the Vagrant bash prompt verify the version of the Docker containers:

```
$ docker images
```

This will display all available images with the version. The version is listed in the Tag column.

```
$ docker images
REPOSITORY                                                   TAG
   IMAGE ID            CREATED             SIZE
artifactory.ciena.com/blueplanet/frost-orchestrate-ui-dev    1706.4.19
   0e7d4a84467d        3 months ago        237.2 MB
artifactory.ciena.com/blueplanet/script-dev                  2.1.4
   f2b6959de885        3 months ago        825.7 MB
artifactory.ciena.com/blueplanet/base-image-devops-toolkit   20170714
   51de460ae6eb        4 months ago        613.6 MB
artifactory.ciena.com/blueplanet/bpocore-dev
   1.7.0-8289-bcff141   884ee878bdeb        4 months ago        725.2 MB
```

Note the version for frost-orchestrate-ui-dev and bpocore-dev. In the above output the versions are:

frost-orchestrate-ui-dev: 1706.4.19

bpocore-dev: 1.7.0-8289-bcff141

Continued on next page…

Task 2: Start the Docker Containers

2. At the Vagrant bash prompt start bpocore-dev using the following command. Enter the command on a single line:

```
$ docker run --name bpocore-dev -it --rm --publish 8181:8181 -v□
   /bp2:/var/log/orchestrate/ artifactory.ciena.com/blueplanet/bpocore-dev:
   1.7.0-8289-bcff141
```

   Notes:

   - The tag (in bold in the command above) must match the tag displayed for the image when you ran the **docker images** command.
   - The □ represents a space character. There should be no space character between "artifactory.ciena.com/blueplanet/bpocore-dev:" and "1.7.0-8289-bcff141".

   Wait for bpocore-dev to start. A series of log messages will display. Once they stop you can continue. No return prompt will display.

   Leave this terminal window open.

3. Open a new SSH session to the Vagrant environment.
4. From the Vagrant bash prompt start the Orchestrate UI using the **docker run** command. Enter the command on a single line:

```
$ docker run --rm -it -p 9980:80 --name orchestrate-ui-dev --link□
   bpocore-dev:bpocore-dev□
   artifactory.ciena.com/blueplanet/frost-orchestrate-ui-dev:1706.4.19
```

   Notes:

   - The tag (in bold in the command above) must match the tag displayed for the image when you ran the **docker images** command.
   - The □ represents a space character.

   No log messages or return prompt will display.

   Leave this terminal window open.

5. You can now navigate to the orchestrate web UI using the IP address of the Vagrant virtual machine. In a web browser go to **http://192.168.33.10:9980/orchestrate/**. You should see the Blue Planet interface.

Once you have verified that you can access the Orchestrate web UI, leave the containers running and the SSH session windows open. You will need them for the next lab exercise.

End of Lab

Lab 3: Test RA Onboarding

In this lab you will test the RA onboarding process to verify your Vagrant development environment is working properly.

At this point you should have three SSH sessions to the Vagrant environment open:

- One for bpocore-dev – this should be showing log messages
- One for orchestrate-ui-dev – this should be showing an empty screen below the command
- One for development work – this should be at the Ubuntu prompt

Task 1: Generate SSH Keys

1. From the Vagrant bash prompt return to the home directory, if you are not already there.
2. Generate new SSH keys using the **ssh-keygen** command, but use your email address (or make one up). Accept the default for all prompts:

```
$ ssh-keygen -t rsa -C "your_email@yourcompany.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/home/vagrant/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/vagrant/.ssh/id_rsa.
Your public key has been saved in /home/vagrant/.ssh/id_rsa.pub.
```

3. Verify your key was created:

```
$ cat ~/.ssh/id_rsa.pub
```

You will see a long, random alphanumeric string that ends with your email address.

Task 2: Set up Git

Git is used to onboard resource type definitions into the orchestrator. Before using Git you need configure your name and email address:

1. From the Vagrant bash prompt configure your email address using the following command:

```
$ git config --global user.email "your_email@yourcompany.com"
```

2. Configure your name using the following command:

```
$ git config --global user.name "RA Developer"
```

Continued on next page…

Task 3: Test RA Onboarding

1. From Vagrant development environment bash prompt move to the `examples/example-ra-cisco` directory.

```
$ cd examples/example-ra-cisco
```

2. Run the **orch-env** script:

```
$ source scripts/orch-env
```

   You will see messages related to hosts. You should not see any error messages.

3. Prepare the RA environment:

```
$ make prepare-venv
```

   You will see a series of messages as the environment is prepared.

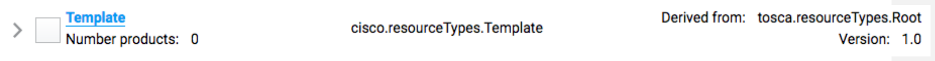
4. Run the RA:

```
$ env/bin/racisco --declare-rp --dev --no-clustering
```

   You will see a series of messages ending with one like this:

```
INFO rpsdk.utils.ra market at http://172.17.0.2:8181/bpocore/market/api/v1
   READY
```

5. In the Orchestrate web UI go to the **Orchestration** → **Resource Types** screen. http://192.168.33.10:9980/orchestrate/#/list/resource-types
6. In the filter field on the top of the page, enter the word *Template*.
   You should see a line item for the **CISCO : Template**:



Template
Number products: 0
cisco.resourceTypes.Template
Derived from: tosca.resourceTypes.Root
Version: 1.0

7. Once you have verified that **Cisco : Template** is available in the Orchestrator, you can stop the RA:
   a. Go to the terminal window running the RA.
   b. Press **Ctrl+C** to stop the RA. This will return you to the Ubuntu prompt:

```
vagrant@vagrant:~/examples/example-ra-cisco$
```

8. Return to the vagrant home directory.

Leave this terminal window and the other terminal windows open.

Data Persistence in the Development Environment

Data added to the bpocore-dev database does not persist when the docker container is stopped. This means that anything you add to the orchestrator, including RAs you onboard, will not be there the next time you start bpocore-dev. This is true only in the development environment, not in a production environment. Anytime you stop and restart bpocore-dev you will need to add your SSH keys, onboard your RAs, and onboard any other resource types and service templates you added.

Continued on next page…

This doesn't impact files stored in the Vagrant virtual machine. However, you should always have a copy of your development work saved outside the Vagrant development environment. During training, you will use the **shared** folder you created previously because it's available in your host machine and the development environment. This ensures the work you create is stored outside the development environment.

The next lab shows you how to properly suspend your Vagrant development environment to maintain the bpocore-dev database.

End of Lab

Lab 4: Suspending and Restarting the Development Environment

If you reboot or shut down your host machine, you could stop the docker containers running bpocore-dev and orchestrate-ui-dev. This means you will have to restart both docker containers and will lose any changes made to the bpocore-dev database. This lab shows you how to properly suspend your development environment *before* shutting down your host OS.

You should have three terminal windows open:

- One for bpocore-dev – this should be showing log messages
- One for orchestrate-ui-dev – this should be showing log messages
- One at the Ubuntu prompt

Task 1: Suspend the Development Environment

1. If you are in the Vagrant environment you will need to exit the vagrant environment. At the Vagrant bash prompt, type **exit** and press enter to exit the Vagrant development environment. This will return you to your host machine, for example:

```
$ exit
logout
Connection to 127.0.0.1 closed.
PEM-TRAINING-01:devops1.0.1 trainer$
```

   When you exit the development environment you will see the logout message and a message indicating the SSH connection was closed.

   Note: If you are using Putty or other similar tools to connect to the development environment the connection will be closed, but you will need to open a command prompt window and navigate to the directory containing the `Vagrantfile` and `.box` file.

2. At the prompt for your host machine suspend the Vagrant:

```
vagrant suspend
```

   You will see message indicating the VM is suspending. Once the VM is finished suspending you will see the normal prompt for your host machine.

   The terminal windows running bpocore-dev and orchestrate-ui-dev will also be returned to the prompt for your host machine. (For Windows systems, the terminal windows may automatically close.)

3. Go to the web browser window running the Orchestrate web UI and reload the page. The connection should timeout, indicating the development environment has been suspended.

Continued on next page…

Task 2: Restart the Development Environment

1. In the terminal on your host machine navigate to the directory containing the **`Vagrantfile`** and the **`.box`** file.
2. Restart the development environment:

```
vagrant up
```

3. After the Vagrant VM starts up, go to the web browser window running the Orchestrate web UI and reload the page. The UI should display.
4. Connect to the Vagrant virtual machine.

   For Mac and Linux systems use the Vagrant SSH command:

```
vagrant ssh
```

   For Windows open an SSH application and connect to the virtual machine using the IP address 192.168.33.10. Use the following credentials:

- Username: *vagrant*
- Password: *vagrant*

   Once you login you should see the default Ubuntu prompt:

```
vagrant@vagrant:~$
```

5. Verify the docker containers are running:

```
$ docker ps
```

   This will list the current containers. The bpocore-dev and orchestrate-ui-dev containers should be listed and the STATUS column show an UP time.

Task 3: View bpocore-dev Logs

After restarting the Vagrant development environment, the bpocore log window will no longer display. You can view the logs using the **tail -f** command.

1. Open a new SSH session to the Vagrant development environment.
2. At the Vagrant bash prompt enter the `tail -f` command to view the bpocore-dev logs:

```
$ tail -f /bp2/bpocore.log
```

This will display the current log and new messages as they are added to the log. You can leave this window open to assist with troubleshooting.

If you want to close the log press **Ctrl+C**.

Continued on next page…

Task 4: Stopping Docker Containers

This procedure shows you how to stop and restart docker containers.

1. Verify the container is running using the following command:

```
$ docker ps –a
```

This displays the running containers:

```
CONTAINER
   ID         IMAGE
          COMMAND                CREATED              STATUS               PORTS
                                  NAMES

f2afa2438477        artifactory.ciena.com/blueplanet/frost-orchestrate-ui-d
   ev:1702.2.3    "nginx"             10 minutes ago      Up 10
   minutes        0.0.0.0:9980->80/tcp             orchestrate-ui-dev

630a27979194        artifactory.ciena.com/blueplanet/bpocore-dev:1.6.1-7983
   -d0ba3e1       "/main.py"          12 minutes ago      Up 12
   minutes        22/tcp, 0.0.0.0:8181->8181/tcp   bpocore-dev
```

2. Stop the running container using the container ID (highlighted in yellow in the example above):

```
$ docker stop f2afa2438477
```

3. Repeat for the other container.

Later when you need to start the containers use the **docker start** command.

Troubleshooting Docker

This section is included for your reference. If the docker containers do not shut down properly you may get an error when trying to start the container:

```
docker: Error response from daemon: Conflict. The name "/orchestrate-ui-dev"
   is already in use by container
   d881c733c283835c206dabcdced1d170c5733b2e3fc4553a865ea6f47381854e. You
   have to remove (or rename) that container to be able to reuse that name.
```

This means the container is already running. If bpocore-dev or orchestrate-ui-dev are not working properly you may need to restart the container:

1. Use the **docker ps -a** command to view all containers.
2. Locate the container ID for the container you need restart.
3. Use the **docker stop <container ID>** command to stop the container.
4. Use the **docker start <container ID>** command to start the container.

Continued on next page…

If you are still experiencing issues starting the containers, you may need to remove the current containers. Use the **docker rm** command to remove the container, then use the **docker run** command to start a new container. For example:

```
$ docker rm ddbae0bfd050


$ docker run --rm -it -p 9980:80 --name orchestrate-ui-dev --link
   bpocore-dev:bpocore-dev
   artifactory.ciena.com/blueplanet/frost-orchestrate-ui-dev:1702.2.3
```

End of Lab

Lab 5: Create an RA Project

The goal of this lab is to create your first RA Project. Note: The RA created in this lab will be used in Lab 6 and Lab 14. Please follow the directions carefully as mistakes may have an impact on future labs.

Task 1: Render a New RA Project

1. Open an SSH session to the Vagrant development environment.
1. At the Vagrant bash prompt navigate to the **`shared`** directory.

```
$ cd shared
```

The **`shared`** directory is used for RA projects because it is available in both the Vagrant development environment and your host machine. This allows you to use any IDE or text editor on your host machine to create your RA.

2. Run the **paster** command to create a new RA project.

```
$ paster
```

You will be prompted to enter information about your RA. Use the information provided below in *`italics`*.

- **The name of the RA [MyRA]:** *`radocker`*
- **Python package name for your RA [radocker]:** *`radocker`*
- **The author of the RA [Ciena Engineer]:** *`your_name`*
- **The vendor of the RA [Ciena]:** *`your_company`*
- **List the endpoints supported by your RA. [['cli', 'snmp']]:** *`['cli']`*
- **General type or class of things the RA connects to, e.g. Ciena_Optical. [Demo]:** *`NixServer`*
- **List the subgroup or family of devices/domains that your RA manages, e.g. Ciena6500 [['Demo-A']]:** *`NixDockerServer`*
- **List the specific types of devices/domains supported by this RA, e.g. CN6500 [['DEMO-A-1']]:** `docker`
- **Select your RA's configuration optimization settings:** *`1`*
- **Select if you'd like the virtualenv directory initialized for you:** *`1`*
- **Will you be running the RA as an Orchestrate resource provider? [Y/n]:** *`Y`*
- **Organization which maintains the RA. (URL is recommended) [http://ciena.com]:** `http://ciena.com`
- **The vendor's license for the RA. (URL is recommended). [http://ciena.com/license]:** `http://training/license`
- **RP ID (must be a valid UUID or urn such as urn:ciena:bp:ra:ciena6500) [urn:ciena:bp:ra:radocker]:** `urn:ciena:bp:ra:radocker`
- **ID for the RP's first domain (must be a valid UUID or urn such as urn:ciena:bp:domain:ciena6500) [urn:ciena:bp:domain:radocker]:** `urn:ciena:bp:domain:radocker`
- **Maintainer/owner of this RP [Ciena Engineer]:** `Ciena Student`
- **Email for the maintainer [ciena.engineer@email.com]:** `ciena.student@email.com`

Continued on next page…

- **Human-readable title of this RP [radocker]:** `RA Docker`
- **Describe the RP []:** `A manager for docker images`
- **RP version [0.0.1]:** *0.0.1*
- **Select your resource provider type**: `1 - domain manager`

  Once you complete all the prompts, the **paster** command will create the RA project in the **`radocker`** directory.

Continued on next page…

Task 2: Prepare the RA

1. From the Vagrant bash prompt, move to the `radocker` directory:

```
$ cd radocker
```

2. Use the **make** command to prepare the RA:

```
$ make prepare-venv
```

The **make** command downloads a set of python libraries therefore an internet connection is required to successfully execute this command. If this command fails, it may be necessary for you to make a change to your Makefile. Ask your instructor about the **--always-copy** option in this situation.

3. Run the RA:

```
$ env/bin/radocker
```

Note: You may see some Python errors. These are normal at this point and can be ignored.

4. Watch the output in the RA window. Once you see the message "twisted manhole started", open a new terminal window and login to the Vagrant development environment.
5. Use the **curl** command to get information about the RA:

```
$ curl -s http://localhost:8080/api/v1/raInfo | python -m json.tool
```

You should see response similar to this:

```
{
    "items": [
        {
            "author": "Ciena Trainer",
            "id": "UKQX34E2FJJBR5C3V75AEQ43EGA",
            "lastStartTime": "Thu, 03 Aug 2017 18:13:41 +0000",
            "name": "radocker",
            "sessionLimit": 1000000,
            "vendor": "Ciena Training"
        }
    ],
    "itemsAvailable": 1,
    "kind": "#raInfo",
    "nextPageToken": null
}
```

Continued on next page...

6. In a web browser on your host computer go to http://192.168.33.10:8080/api/v1/swagger-ui/ This displays the Swagger UI for the RA.
7. In the SSH session window running the RA press **Ctrl+C** on your keyboard to stop the RA. This will return you to the bash prompt.

Please notify your instructor when you have completed the lab.

End of Lab

Lab 6: Customizing Your RA

Task 1: Create a Git Repo

1. At the Vagrant bash prompt, verify that you are in the RA project directory. The prompt should be:

```
vagrant@vagrant:~/shared/radocker$
```

2. Create a Git repository:

```
$ git init
$ git add -A
$ git commit -m "initial RA"
```

Task 2: Configure settings.py

To get started you only need to make a few customizations to the RA in the `setting.py` file. Since your RA project directory is in the `~/shared` directory you can access it from your host computer. The `settings.py` file is in the `~/shared/radocker/radocker` directory.

1. Locate and open `settings.py` in text editor or IDE.
2. Make sure your `settings.py` matches the values below. If not change them to match these settings:

```
TYPE_GROUP = 'NixServer'
RESOURCE_TYPES = ['NixDockerServer']
ENDPOINTS = ['cli']
```

3. If you made changes, save the file.

Continued on next page…

Task 3: Edit family.json

As part of configuring bpprov you need to specify the device family. The object name in the **`family.json`** file must match the resourceType value in the **`settings.py`** file.

The **`family.json`** file is in the **`~/shared/radocker/radocker/model`** directory.

1. From the Vagrant bash prompt, locate and open **`family.json`**.
2. If necessary, change the file to match the following (use the specified values):

```
{
   "NixDockerServer": {
       "dir": ".",
       "displayName": "NixDockerServer"
   }
 }
```

3. If you made changes, save the file.

Task 4: Edit device.json

As part of configuring bpprov you need to specify the devices. The family must match the resourceType value in the **`settings.py`** file.

The **`device.json`** file is in the **`~/shared/radocker/radocker/model`** directory.

1. From the Vagrant bash prompt, locate and open **`device.json`**.
2. If necessary, change the file to match the following:

```
{
"docker": {
      "family": "NixDockerServer",
      "dir": ".",
      "version": [
          "0.0.0.0"
      ]
   }
 }
```

3. If you made changes, save the file

Continued on next page…

Task 5: Customize Endpoints

Since the endpoint type is CLI you must ensure that the prompt from the device can be understood by the RA. The prompt is represented by a regular expression in the *endpoint-params.json* file.

The **`endpoint-params.json`** file is in the **`~/shared/radocker/radocker/model`** directory.

1. From the Vagrant bash prompt, locate and open **`endpoint-params.json`**.
2. Change the "prompt" value to match the following (Note: the only change made here is the ^ symbol at the beginning of the prompt value):

```
"prompt": "^[A-Za-z0-9\\-@()\\~:]+[>#\\$]",
```

3. Save the file.

Task 6: Test Your RA

1. From the Vagrant bash prompt, change to the RA project directory: **`~/shared/radocker`**.
2. Run the RA:

```
$ env/bin/radocker
```

3. Open a second SSH session to the Vagrant environment.
4. At the bash prompt enter the following command to verify your resourceType:

```
$ curl -s http://localhost:8080/api/v1/typeGroups | python -m json.tool
```

The output should include the following output:

```
...
"id": "NixServer",
"resourceTypes": [
                "/resourceTypes/NixDockerServer"
            ]
...
```

5. Stop the RA by going to the SSH session window running the RA and pressing CTRL+C.

Continued on next page…

Task 7: Copying Command Files

For this part of the lab you will copy files from the **`ra_sample_files/Lab_6`** into your RA package folder. Note: The contents of these files will be covered in later sections of the course. The purpose of this Task is to complete the process of creating a basic RA.

1. Copy the provided files named **`show-device.json`** and **`show-docker-version.json`** to the **`~/shared/radocker/radocker/model/commands`** directory.
2. Create a directory named **`fsms`** under the **`~/shared/radocker/radocker/model`** directory:

```
$ mkdir ~/shared/radocker/radocker/model/fsms
```

3. From the **`ra_sample_files/Lab_6/fsms`** directory copy the files named **`show-device.fsm`** and **`show-docker-version.fsm`** to the **`~/shared/radocker/radocker/model/fsms`** directory.
4. Copy the provided file in the **`Lab_6`** folder named **`show-device.tmpl`** to the **`~/shared/radocker/radocker/model/templates`** directory
5. Copy **`endpoints.json`** from the **`ra_sample_files/Lab_6`** directory to the **`~/shared/radocker/radocker/model`** directory.
6. From the Vagrant bash prompt, change to the RA project directory: **`~/shared/radocker`**.
7. Execute the following command:

```
env/bin/bpprov-cli command-run radocker/model cli commands/show-device.json
   '{}' --start
```

The output should be similar to the following (actual values and order may vary):

```
{
    "swImage": "Linux",
    "swVersion": "4.4.0-51-generic",
    "resourceType": "NixDockerServer",
    "serialNumber": "vagrant",
    "swType": "x86_64",
    "type": "docker",
    "deviceVersion": "1.12.6-cs13, build 0ee24d4"
}
```

Commented [RB1]: Update this to match current versions

8. Stop your RA by going to the SSH session running the RA and pressing Ctrl+C.

Please notify your instructor when you have completed the lab.

End of Lab

Lab 7: Create Custom RA Project

In this lab, you will be creating a new RA. You need to determine what kinds of devices the RA will communicate with and the name of the RA project. This RA will communicate with Ubuntu to manage docker containers.

Task 1: Render a New RA Project

1. From the Vagrant bash prompt, move to the **shared** directory.
2. Run the **paster** command to create the RA project. Use the values provided below.

```
$ paster
```

- **The name of the RA [MyRA]:** *RA-Class*
- **Python package name for your RA [raclass]:** *raclass*
- **The author of the RA [Ciena Engineer]:** *your_name*
- **The vendor of the RA [Ciena]:** *your_company*
- **List the endpoints supported by your RA. [['cli', 'snmp']]:** *['cli']*
- **General type or class of things the RA connects to, e.g. Ciena_Optical. [Demo]:** *DockerContainer*
- **List the subgroup or family of devices/domains that your RA manages, e.g. Ciena6500 [['Demo-A']]:** *docker*
- **List the specific types of devices/domains supported by this RA, e.g. CN6500 [['DEMO-A-1']]:** *linux*
- **Select your RA's configuration optimization settings:** *1*
- **Select if you'd like the virtualenv directory initialized for you:** *1*
- **Will you be running the RA as an Orchestrate resource provider? [Y/n]:** *Y*
- **Organization which maintains the RA. (URL is recommended) [http://ciena.com]:** *http://ciena.com*
- **The vendor's license for the RA. (URL is recommended). [http://ciena.com/license]:** *http://training/license*
- **RP ID (must be a valid UUID or urn such as urn:ciena:bp:ra:ciena6500) [urn:ciena:bp:ra:raclass]:** *urn:ciena:bp:ra:raclass*
- **ID for the RP's first domain (must be a valid UUID or urn such as urn:ciena:bp:domain:ciena6500) [urn:ciena:bp:domain:raclass]:** *urn:ciena:bp:domain:raclass*
- **Maintainer/owner of this RP [Ciena Engineer]: Ciena Student**
- **Email for the maintainer [ciena.engineer@email.com]:** *ciena.student@email.com*
- **Human-readable title of this RP [radocker]:** *RA Class Docker*
- **Describe the RP []:** *A manager for docker images*
- **RP version [0.0.1]:** *0.0.1*
- **Select your resource provider type:** *1 - domain manager*

Continued on next page…

3. Verify that the directory for your RA project was created by using the ls command. You should see a directory with the name of your RA.

Task 2: Update settings.py

1. In a text editor, open **`settings.py`**. The file should be in the RA project directory: **`~/shared/RA-Class/raclass`**.
2. Make sure your **`settings.py`** matches the values below. If not change them to match these settings:

```
TYPE_GROUP = 'DockerContainer'

RESOURCE_TYPES = ['docker', ]

ENDPOINTS = ['cli', ]
```

3. If you made changes, save the file.

Task 3: Update family.json

1. In a text editor open **`family.json`** from the **`model`** directory. The file should be in the RA project directory: **`~/shared/RA-Class/raclass/model`**.
2. Change the file to match the following:

```
{

   "docker": {

       "dir": "docker",

       "displayName": "DOCKER"

   }

 }
```

3. Save the file.
4. From the Vagrant bash prompt, go to the **`commands`** directory within your RA project directory: **`~/shared/RA-Class/raclass/model/commands`**
5. Create a new directory within **`commands`** named **`docker`** to match the value that you provided in the **`family.json`** file:

```
$ mkdir docker
```

When you are done, you can compare your **`family.json`** file to the **`family.json`** in **`Answer Files/Lab_7`** directory.

Continued on next page…

Task 4: Update device.json

1. In a text editor open **`device.json`**. The file should be in the RA project directory: **`~/shared/RA-Class/raclass/model`**
2. change the file to match the following:

```
{
"linux": {
      "family": "docker",
      "dir": "docker/linux",
      "version": [
          "0.0.0.0"
      ]
   }
 }
```

3. Save the file.
4. From the Vagrant bash prompt, go to the **`commands`** directory within your RA project directory: **`~/shared/RA-Class/raclass/model/commands`**
5. Create a new directory within **`commands`** named **`docker/linux`** to match the value that you provided in the **`device.json`** file:

```
$ mkdir docker/linux
```

When you are done compare your new **`device.json`** file to the **`device.json`** file in the *Answer* **`Files/Lab_7`** directory.

Task 5: Update endpoint-params.json

Since the endpoint type is CLI you must ensure that the prompt from the device can be understood by the RA. The prompt is represented by a regular expression in the **`endpoint-params.json`** file.

The **`endpoint-params.json`** file is in the **`~/shared/RA-class/raclass/model`** directory.

1. From the Vagrant bash prompt, locate and open **`endpoint-params.json`**.
2. Change the “prompt” value to match the following (Note: the only change made here is the ^ symbol at the beginning of the prompt value):

```
"prompt": "^[A-Za-z0-9\\-@()\\~:]+[>#\\$]",
```

3. Save the file.

Continued on next page…

Task 6: Test Your RA

1. From the Vagrant bash prompt, move to the RA project directory.
2. Use the **make** command to prepare the RA:

```
$ make prepare-venv
```

3. Run the RA:

```
$ env/bin/raclass
```

4. In a second SSH session window, enter the following command to verify your RA is working:

```
curl -s http://localhost:8080/api/v1/raInfo | python -m json.tool
```

If you get errors or do not see the JSON output with information about your RA, go back through the lab and make sure you used the correct resourceType in all the files and that the files are in the correct directories.

Task 7: Create a git Repo

1. Once your RA is working correctly, stop the RA by going to the SSH session window of the Vagrant development environment that is running the RA and pressing CTRL+C.
2. Change to the RA project directory if you are not already there. The directory should be `~/shared/RA-Class`.
3. Create a git repo:

```
$ git init
$ git add -A
$ git commit -m "initial RA"
```

Please notify your instructor when you have completed the lab.

End of Lab

Lab 8: Create show-device.json

Task 1: Setup the Command File

1. In the `model/commands` directory open `show-device.json`.
2. Set the "endpoint" value to "cli".
3. Add "endpoint-parameters" and set the "command" value to **uname -snrmpo**.

```
"endpoint-parameters": {
"command": "uname -snrmpo"
 },
```

4. Leave the "out-path" as an empty array. You don't need any out-path translators for this command.
5. For testing we need to remove the "in-schema" key and value. While building the command, the command file will not pass validation so we will temporarily remove the "in-schema".
   Delete the "in-schema" parameter.
6. Set the "in-path" to be an empty array.
   Later you will add translators to the "in-path", but for now it should be empty.
7. Save the file.

Task 2: Test show-device.json

You will use the **bpprov-cli** command to test your command files. You must be in the Vagrant development environment to test your RA.

1. Copy the `endpoints.json` file from the `ra_sample_files/Lab_8` directory to the `model` directory of your RA project.
2. Move to your RA project directory and use the **bpprov** command to test the **show-device.json** command:

```
$ env/bin/bpprov-cli command-run raclass/model cli commands/show-device.json
   '{}' --start
```

   When you run the command, you may see a warning about "service_identity_module". You can ignore this warning. The output of the command should be:

```
"Linux vagrant 4.4.0-51-generic x86_64 x86_64 GNU/Linux"
```

You should copy the **bpprov-cli** command line and save it in a text file for easy access. You will be using the same command later in the course.

Please notify your instructor when you have completed the lab.

End of Lab

Lab 9: Create FSM

In this lab, you will create an FSM template using sample data. You will then test the template using the **bpprov-cli** command.

The data and template from this lab will not be used in your RA. This exercise allows you to practice using FSM and the **bpprov-cli** command.

instructions

1. Copy the `fsm_test.txt` file from the `ra_sample_files/Lab_9` directory into your RA project directory, for example: `~/shared/RA-Class/fsm_test.txt`
2. Open `fsm_test.txt`. Note the format of the data in the file. You should see three fields:
   - Index
   - ERP Name
   - VID Set
3. In your RA project directory create a new directory called `fsms` in the `model` directory.
4. Create a new text file named `test.fsm` and save it in the `fsms` directory. You will use this new file to create an FSM template.
5. In the `test.fsm` file create value definitions for each of the fields in the `fsm_test.txt` file. Use the following format:

```
Value <Value Name> (<insert regular expression>)
```

For example:

```
Value Index (\S+)
```

You must determine the correct regular expression for each value:

   - Index: Match any number one or more times
   - ERP Name: Match any character except line return one or more times
   - VID Set: Match a single VLAN or a range of VLANs

Continued on next page…

6. Below the value definitions create a state definition. State definitions are enclosed between Start and EOF:

```
Start
  <state definitions>


EOF
```

The state definition must match the format of the text input. Include variables for the value definitions using the format `${ValueName}`.

End the last state definition with `-> Record`.

For example:

```
   ^${Index} ${Name} -> Record
```

(Note: The example shown above will not work for this exercise. You will need to create a combination of variables and regular expressions for this exercise.)

7. Once you have the state definition complete save the file.
8. From the Vagrant bash prompt, use the **bpprov-cli** command to test the FSM template:

```
$ env/bin/bpprov-cli fsm-validate raclass/model/fsms/test.fsm
   --input=@fsm_test.txt
```

The output should be a list of lists with the values from the sample text file (see next page). If your output does not match or you get an error, check the regular expressions in your FSM template.

Continued on next page…

```
[
    [
        "1",
        "Training ERP                    ",
        "600-650"
    ],
    [
        "2",
        "Iris erp 500-550                ",
        "500-550"
    ],
    [
        "3",
        "Iris ERP 400-450                ",
        "400-450"
    ],
    [
        "4",
        "S1 Iris ERP 100-150             ",
        "100-150"
    ],
    [
        "6",
        "Gene1                           ",
        "300-350"
    ],
    [
        "7",
        "Arek1                           ",
        "200-250"
    ]
]
```

Please notify your instructor when you have completed the lab.

End of Lab

Lab 10: Add an FSM Template to show-device

In this lab you will create an FSM template to parse the output of the **uname –snrmpo** command and then add the template to the `show-device.json` file.

The output of the **uname –snrmpo** command is similar to the following (your exact values may vary):

```
Linux vagrant 4.4.0-51-generic x86_64 x86_64 GNU/Linux
```

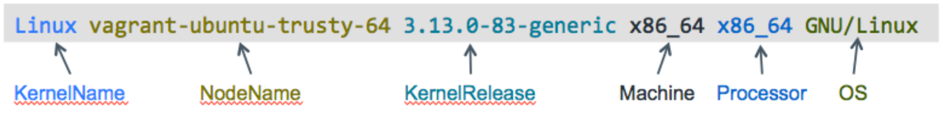
Task 1: Create FSM Template

1. From the Vagrant bash prompt, go to your RA project directory.
2. Write the output of the **uname** command to a text file:

```
$ uname -snrmpo > uname.txt
```

   This file will be used to test your FSM.

3. Create a new file named `show-device.fsm` and save it in the `fsms` directory in your RA package directory.
4. In the FSM template add value definitions for the following:
   - KernelName
   - NodeName
   - KernelRelease
   - Machine
   - Processor
   - OS
5. For each value specify a regular expression that matches the output field from the **uname –snrmpo** command.
6. Using the variable mapping below (note: your values may be different), create a state definition in using the format of the command



```
Linux vagrant-ubuntu-trusty-64 3.13.0-83-generic x86_64 x86_64 GNU/Linux
```

7. Save the file.
8. Test the file using the **bpprov-cli** command:

```
$ env/bin/bpprov-cli fsm-validate raclass/model/fsms/show-device.fsm
   --input=@uname.txt
```

Continued on next page…

The output should be:

```
[
    [
        "Linux",
        "vagrant-ubuntu-trusty-64",
        "3.13.0-101-generic",
        "x86_64",
        "x86_64",
        "GNU/Linux"
    ]
]
```

If your output does not match, or has errors you will need to troubleshoot the FSM template before moving the next part of the lab.

Task 2: Add FSM to show-device.json

1. Open the **show-device.json** file.
2. Within the "endpoint-parameters key", add the FSM template:

```
"endpoint-parameters": {
       "command": "uname -snrmpo",
       "fsm": "fsms/show-device.fsm"
   }
```

3. Save the file.
4. From the Vagrant bash prompt, test the command file using the **bpprov** command:

```
$ env/bin/bpprov-cli command-run raclass/model cli commands/show-device.json
   '{}' --start
```

The output should be the same as the output of the FSM template testing from Task #1.

Please notify your instructor when you have completed the lab.

End of Lab

Lab 11: Add in-path Runner

In this lab, you will add in-path translators to convert the output from the TextFSM.

1. Open the `show-device.json` file.
2. In the "in-path" key, add the "bpprov.translators.list.Flatten" translator:

```
"in-path": [{
         "type": "bpprov.translators.list.Flatten"
}]
```

3. Save the file.
4. From the Vagrant bash prompt, test the command using the **bpprov-cli** command:

```
$ env/bin/bpprov-cli command-run raclass/model cli commands/show-device.json
   '{}' --start
```

The output should be similar to the following:

```
[
    "Linux",
    "vagrant",
    "4.4.0-51-generic",
    "x86_64",
    "x86_64",
    "GNU/Linux"
]
```

Note the list is now a single list and is not nested.

5. In the `show-device.json` file add a second translator: "bpprov.translators.list.ToDict". **Don't forget to add a comma after the first translator**:

```
  {
   "type": "bpprov.translators.list.ToDict",
         "parameters": {
             "labels": [ "kernel", "node", "release", "machine", "processor",
   "os" ]
         }
     }
```

Continued on next page…

6. Save the file and test it using the **bpprov-cli** command:

```
env/bin/bpprov-cli command-run raclass/model cli commands/show-device.json
   '{}' --start
```

The output should be similar to the following:

```
{
    "node": "vagrant",
    "kernel": "Linux",
    "machine": "x86_64",
    "release": "4.4.0-51-generic",
    "os": "GNU/Linux",
    "processor": "x86_64"
}
```

At this point we have most of the information needed for the `show-device.json` command file: swImage, swVersion, resourceType, serialNumber, and swType. In the next labs, you will create another command to get the rest of the information so the following output can be created:

```
{
    "swImage": "GNU-Linux",
    "swVersion": "4.4.0-51-generic",
    "resourceType": "docker",
    "serialNumber": "vagrant",
    "swType": "Linux",
    "type": "linux",
    "deviceVersion": "1.12.6-cs13, build 0ee24d4"
}
```

Please notify your instructor when you have completed the lab.

End of Lab

Lab 12: Create show-docker-version.json

In this lab you will create the `show-docker-version.json` command file. This command file will be added to the `show-device.json` command to complete the required output of the `show-device.json` command file.

Task 1: Setup the Command File

1. From the `ra_sample_files/Lab_12` directory copy the `command-template.json` file to your RA project directory. Put the file in the `commands` directory.
2. Rename the file to `show-docker-version.json`.
3. Open `show-docker-version.json`.
4. Set the "endpoint" value to "cli".
5. Set the "type" value to "bpprov.runners.simple.Sequence".
6. In "endpoint-parameters" set the "command" value to "**docker -v**".
7. Set the "out-path" to be an empty array. You don't need any out-path translators for this command.
8. Delete the "in-schema" parameter.
9. Set the "in-path" to be an empty array.
   Later you will add translators to the in-path, but for now it can be empty.
10. Save the file.
11. From the Vagrant bash prompt, use the **bpprov-cli** command to test the command. You can use the same command you've been using, but change `show-device.json` to `show-docker-version.json`:

```
env/bin/bpprov-cli command-run raclass/model cli
   commands/show-docker-version.json '{}' --start
```

The output should be similar to:

```
"Docker version 1.12.6-cs13, build 0ee24d4"
```

(Note: The version and build number may be different, but the format should be the same.)

Task 2: Create an FSM

1. Create a new file named `show-docker-version.fsm` and save it in the `fsms` directory in your RA package directory.
2. Create the following value definitions:
   - Device: match any non-white space character
   - Version: match any non-white space character
   - Build: match any non-white space character
3. Create a state definition that records "Docker" in the **Device** variable, the version number in the **Version** variable and the build number in the **Build** variable. Hint: match for a literal comma character in the state definition to avoid having a comma in the Version variable.
4. Save the file.

Continued on next page…

5. In the `show-docker-version.json` file add the "fsm" template to the "endpoint-parameters" value:

```
"endpoint-parameters": {
       "command": "docker -v",
       "fsm": "fsms/show-docker-version.fsm"
   },
```

6. Save the file.
7. Use the **bpprov-cli** command to test the `show-docker-version.json` command file. The output of the command should be similar to:

```
[

    [

         "Docker",

         "1.12.6-cs13",

         "0ee24d4"

    ]

]
```

Task 3: Add in-path Translators

The output from the TextFSM needs to be flattened and dictified. In this part of the lab you will add two translators to the in-path to get a JSON object with the docker version and build number.

1. In the `show-docker-version.json` file add the following translators to the "in-path" in this order:
   - bpprov.translators.list.Flatten
   - bpprov.translators.list.ToDict
2. For "bpprov.translators.list.ToDict" add a parameter for "labels". The value of "labels" is an array with "dockerDevice", "dockerVersion" and "dockerBuild" as the values in the array.
3. Save the file.
4. From the Vagrant bash prompt, use the **bpprov-cli** command to test the `show-docker-version.json` command file. The output should be similar to:

```
{

    "dockerDevice": "Docker",

    "dockerVersion": "1.12.6-cs13",

    "dockerBuild": "0ee24d4"

}
```

End of Lab

Lab 13: Add a Command Aggregator

In this lab, you will add a translator to the `show-device.json` command file that gets data from the `show-docker-version.json` command file. The content of the translator is included in the `aggregator.json` file provided in the student sample files.

1. In the `ra_sample_files/Lab_13` directory open the `aggregator.json` file.
2. Copy the contents of the file.
3. In the `show-device.json`,file paste the copied content to the "in-path". Be sure it is the third translator in the "in-path".
4. Save the file.
5. From the Vagrant bash prompt, use the **bpprov-cli** command to test the `show-device.json` command file. The output should be similar to:

```
{
    "node": "vagrant",
    "kernel": "Linux",
    "dockerVersion": "1.12.6-cs13",
    "machine": "x86_64",
    "dockerDevice": "Docker",
    "dockerBuild": "0ee24d4",
    "release": "4.4.0-51-generic",
    "os": "GNU/Linux",
    "processor": "x86_64"
}
```

Please notify your instructor when you have completed the lab.

End of Lab

Lab 14: Complete show-device.json

Task 1: Add a Jinja Template

1. From the `ra_sample_files/Lab_14` directory copy `show-device.tmpl` to the `model/templates directory` in your RA project.
2. Open the `show-device.json` file.
3. Just before the in-path add the "in-schema" parameter:

```
"in-schema": "in-show-device.json",
```

As previously mentioned, this is required for the `show-device.json` command file.

4. Add a new translator after the last translator in the in-path. Use the following settings:

- **type**: bpprov.translators.template.Json
- **template** parameter: show-device.tmpl

5. Save the file.
6. From the Vagrant bash prompt, use the **bpprov-cli** command to test the `show-device.json` command file. The output should be:

```
{
    "swImage": "GNU-Linux",
    "swVersion": "4.4.0-51-generic",
    "resourceType": "docker",
    "serialNumber": "vagrant",
    "swType": "x86_64",
    "type": "linux",
    "deviceVersion": "1.12.6-cs13, build 0ee24d4"
}
```

This is the final output of `show-device.json`, which conforms to the schema.

Continued on next page…

Task 2: Test Your RA

1. From the Vagrant bash prompt, move to your RA project directory and start the RA:

```
$ env/bin/raclass
```

2. In another SSH session window use curl to create a session. Note that the "DockerContainer" value below (see "/typeGroups/DockerContainer") should be the "TYPE_GROUP" value from the `settings.py` file:

```
$ curl -s -H "Content-Type: application/json" -d '{"authentication": {"cli":
   {"username": "vagrant", "password": "vagrant"}}, "connection":
   {"hostname": "localhost", "cli": {"hostport": 22}}, "typeGroup":
   "/typeGroups/DockerContainer", "id": "1"}'
   http://localhost:8080/api/v1/sessions | python -m json.tool
```

3. Check to make sure the session connected:

```
$ curl -s -H "Content-Type: application/json"
   http://localhost:8080/api/v1/sessions | python -m json.tool
```

   In the response look for **"connectState": "CONNECTED"**. If the connection state is any other value check the RA window to see if there are any errors being reported.

   IMPORTANT NOTE: If the connectState is not set to "CONNECTED", then you must delete the connection before establishing the connection again. Step #5 of this Task demonstrates how to delete the connection. Delete the connection, fix the errors that caused the problem, stop your RA and then start from Step #1 of this task.

4. Get devices for the session by executing the "devices" API. This returns the results of the `show-device.json` command file.

```
curl -s -H "Content-Type: application/json"
   http://localhost:8080/api/v1/devices | python -m json.tool
```

Continued on next page…

You should see a response similar to:

```
{
    "items": [
        {
            "deviceVersion": "1.12.6-cs13, build 0ee24d4",
            "execute": {
                "href": "/devices/1/execute"
            },
            "id": "1",
            "resourceType": "docker",
            "serialNumber": "vagrant",
            "session": "/sessions/1",
            "swImage": "GNU-Linux",
            "swType": "x86_64",
            "swVersion": "4.4.0-51-generic",
            "type": "linux"
        }
    ],
    "itemsAvailable": 1,
    "kind": "#devices",
    "nextPageToken": null
}
```

Note the response includes the result of the **`show-device.json`** command file.

5. Remove the session (note: this step is really option as the session will be destroyed when you stop the RA in step #6):

```
$ curl -v -X DELETE http://localhost:8080/api/v1/sessions/1
```

6. Stop the RA by switching to the SSH session window in which it is running and pressing Control+c.

Please notify your instructor when you have completed the lab.

End of Lab

Lab 15: Resolving Errors

The goal of this lab is to guide you through the process of finding errors in your configuration files. Each of the following tasks will introduce an error in your existing RA.

Important note: This lab uses the RA you created in labs 5 and 6 and assumes that you have successfully completed these labs. Do not perform this lab on the RA that you created in labs 7-14.

Task 1: Error #1

1. In order to create these errors, first copy the contents of the `scripts` directory from your sample files into the `env/bin` directory of your RA (`~/shared/radocker/env/bin`).
2. In a SSH session window to the Vagrant development environment, switch to the RA project directory (`~/shared/radocker`)
3. Execute the following command:

```
$ bash env/bin/error1.sh
```

4. Execute the following **bpprov-cli** command:

```
$ env/bin/bpprov-cli command-run radocker/model cli commands/show-device.json
   '{}' --start
```

The **error1.sh** program created an error in your RA. Using the skills that you have learned in class, try to determine the cause of the error and fix the problem. The problem is considered fixed if you can successfully execute the **bpprov-cli** command in step #4.

If you can't determine the error within approximately 5 minutes, move to the next step.

5. Switch to the RA project directory (`~/shared/radocker`) and execute the following command to receive a hint on how to solve the problem:

```
$ bash env/bin/hint1.sh
```

Based on the hint provided, try to determine the cause of the error and fix the problem. The problem is considered fixed if you can successfully execute the **bpprov-cli** command in step #4.

If you can't determine the error within approximately 5 minutes, move to the next step.

6. Switch to the RA project directory (`~/shared/radocker`) and execute the following command to receive the answer on how to solve the problem:

```
$ bash env/bin/answer1.sh
```

7. Based upon the information provided by the answer, fix the problem and make sure the following **bpprov-cli** command executes successfully before continuing to the next step:

```
$ env/bin/bpprov-cli command-run radocker/model cli commands/show-device.json
   '{}' --start
```

Continued on next page…

Task 2: Error #2

1. In a SSH session window to the Vagrant development environment, switch to the RA project directory (**`~/shared/radocker`**)
2. Execute the following command:

```
$ bash env/bin/error2.sh
```

3. Execute the following **bpprov-cli** command:

```
$ env/bin/bpprov-cli command-run radocker/model cli commands/show-device.json
   '{}' --start
```

The **error2.sh** program created an error in your RA. Using the skills that you have learned in class, try to determine the cause of the error and fix the problem. The problem is considered fixed if you can successfully execute the **bpprov-cli** command in step #3.

If you can't determine the error within approximately 5 minutes, move to the next step.

4. Switch to the RA project directory (**`~/shared/radocker`**) and execute the following command to receive a hint on how to solve the problem:

```
$ bash env/bin/hint2.sh
```

Based on the hint provided, try to determine the cause of the error and fix the problem. The problem is considered fixed if you can successfully execute the **bpprov-cli** command in step #3.

If you can't determine the error within approximately 5 minutes, move to the next step.

5. Switch to the RA project directory (**`~/shared/radocker`**) and execute the following command to receive the answer on how to solve the problem:

```
$ bash env/bin/answer2.sh
```

6. Based upon the information provided by the answer, fix the problem and make sure the following **bpprov-cli** command executes successfully before continuing to the next step:

```
$ env/bin/bpprov-cli command-run radocker/model cli commands/show-device.json
   '{}' --start
```

Continued on next page…

Task 2: Error #3

1. In a SSH session window to the Vagrant development environment, switch to the RA project directory (**`~/shared/radocker`**)
2. Execute the following command:

```
$ bash env/bin/error3.sh
```

3. Execute the following **bpprov-cli** command:

```
$ env/bin/bpprov-cli command-run radocker/model cli commands/show-device.json
   '{}' --start
```

The **error3.sh** program created an error in your RA. Note: you will not see an actual error message, but there is a problem with the output. Look at it closely! Using the skills that you have learned in class, try to determine the cause of the error and fix the problem. The problem is considered fixed if you can successfully execute the **bpprov-cli** command in step #3.

If you can't determine the error within approximately 5 minutes, move to the next step.

4. Switch to the RA project directory (**`~/shared/radocker`**) and execute the following command to receive a hint on how to solve the problem:

```
$ bash env/bin/hint3.sh
```

Based on the hint provided, try to determine the cause of the error and fix the problem. The problem is considered fixed if you can successfully execute the **bpprov-cli** command in step #3.

If you can't determine the error within approximately 5 minutes, move to the next step.

5. Switch to the RA project directory (**`~/shared/radocker`**) and execute the following command to receive the answer on how to solve the problem:

```
$ bash env/bin/answer3.sh
```

6. Based upon the information provided by the answer, fix the problem and make sure the following **bpprov-cli** command executes successfully before continuing to the next step:

```
$ env/bin/bpprov-cli command-run radocker/model cli commands/show-device.json
   '{}' --start
```

Continued on next page…

Task 4: Error #4

1. In a SSH session window to the Vagrant development environment, switch to your RA project directory (`~/shared/radocker`)
2. Execute the following command:

```
$ bash env/bin/error4.sh
```

3. Start the RA:

```
$ env/bin/radocker
```

4. Execute the following **bpprov-cli** command:

```
$ env/bin/bpprov-cli command-run radocker/model cli commands/show-device.json
   '{}' --start
```

The **bpprov-cli** command should return the correct result.

5. Use curl to create a session.

```
$ curl -s -H "Content-Type: application/json" -d '{"authentication": {"cli":
   {"username": "vagrant", "password": "vagrant"}}, "connection":
   {"hostname": "localhost", "cli": {"hostport": 22}}, "typeGroup":
   "/typeGroups/NixServer", "id": "1"}'
   http://localhost:8080/api/v1/sessions | python -m json.tool
```

6. Check to make sure the session connected:

```
$ curl -s -H "Content-Type: application/json"
   http://localhost:8080/api/v1/sessions | python -m json.tool
```

In the response look for "connectState": "CONNECTED". If the connection state is any other value check the RA window to see if there are any errors being reported.

The **error4.sh** program created an error in your RA. Using the skills that you have learned in class, try to determine the cause of the error and fix the problem. The problem is considered fixed if you can get "connectState": "CONNECTED" when running the curl command in step #6.

If you can't determine the error within approximately 5 minutes, first stop the RA and then move to the next step.

7. Switch to the RA project directory (`~/shared/radocker`) and execute the following command to receive a hint on how to solve the problem:

```
$ bash env/bin/hint4.sh
```

Based on the hint provided, try to determine the cause of the error and fix the problem. IMPORTANT NOTE: you much start the connection by starting the RA and then using the curl command (see steps #3 and #5 in this Task) first!

The problem is considered fixed if you can get "connectState": "CONNECTED" when running the curl command in step #6.

If you can't determine the error within approximately 5 minutes, first stop the RA and then move to the next step.

8. Switch to your RA project directory (`~/shared/radocker`) and execute the following command to receive the answer on how to solve the problem:

```
$ bash env/bin/answer4.sh
```

Based upon the information provided by the answer, fix the problem and make sure can get "connectState": "CONNECTED" when running the following **curl** command:

```
$ curl -s -H "Content-Type: application/json"
   http://localhost:8080/api/v1/sessions | python -m json.tool
```

9. When finished, close the curl session with the following command:

```
$ curl -v -X DELETE http://localhost:8080/api/v1/sessions/1
```

10. Lastly, stop the running RA.

End of Lab

Lab 16: Create show-containers.json

Task 1: Download the busybox Docker Image

To test the `show-containers.json` command file that you will be creating, you need a docker image. Busybox is a light-weight docker image that will suit our needs for testing.

1. From the Vagrant bash prompt, use the **docker run** command to download and start a busybox container:

```
$ docker run -d busybox ping localhost
```

2. Verify the container is running:

```
$ docker ps –a
```

3. Test the **docker** command you will use in your bpprov command:

```
$ docker ps -a -q | xargs docker inspect
```

   You should see detailed information about all of the containers, including the busybox container.

The busybox image is now available to be used by the RA to create new containers. In later labs, you will create commands that use the busybox image to create containers.

Task 2: Setup Command File

1. Copy `command-template.json` from the `ra_sample_files/Lab_16` directory to the directory for your device type within the commands directory of the RA project that you worked on in Labs #7-14 (This should be `~/shared/RA-Class/raclass/model/commands/docker/linux`).
2. Rename the file to `show-containers.json`.
3. Open `show-containers.json`.
4. Set the "endpoint" value to "cli".
5. Set the "type" value to "bpprov.runners.simple.Sequence".
6. In "endpoint-parameters" set the "command" value to "**docker ps -a -q | xargs docker inspect**".
7. Set the "out-path" to be an empty array. You don't need any out-path translators for this command.
8. Delete the "in-schema" parameter.
9. For the "in-path" add one translator using the following information:
   - type: "bpprov.translators.call.Function"
   - parameters:
     - function: "json.loads"
10. Save the file and test it using the **bpprov-cli** command (remember that it is under the **~/shared/RA-Class/raclass/model/commands/docker/linux** directory). The command should return a very large JSON object.

Continued on next page…

Task 3: Create a Jinja Template

In this part of the lab you will add a Jijna template to extract some of the data from the command response. The Jinja template is provided in the sample files directory.

1. Copy **`show-containers.tmpl`** from the **`ra_sample_files/Lab_16`** directory to **`model/templates`**.
2. In **`show-containers.json`** add a second translator to the in-path using the following information:
   - type: " bpprov.translators.template.Json"
   - parameters:
     - template: "show-containers.tmpl”
3. Save the file and test it using the **bpprov-cli** command.

Task 4: Removing Docker Containers

These instructions show you how to remove a container.

1. From the Vagrant bash prompt, get a list of running containers:

```
$ docker ps –a
```

2. In the list, find the container you want to remove.
3. Copy the container ID.
4. Remove the running container:

```
$ docker rm –f <container ID>
```

The **-f** flag forces the removal of a running container.

Please notify your instructor when you have completed the lab.

End of Lab

Lab 17: Create start-container.json

1. Copy `command-template.json` from the `ra_sample_files/Lab_17` directory to the directory for your device type within the commands directory of the RA project that you worked on in Labs #7-14 (This should be `~/shared/RA-Class/raclass/model/commands/docker/linux`). Rename the file `start-container.json`.
2. Set the "endpoint" value to "cli".
3. Set the "type" value to "bpprov.runners.simple.Sequence".
4. In "endpoint-parameters" set the "command value" to "**docker run {{ data.parameters }} {{ data.image }} {{ data.command }}**".
5. Set the "out-path" to be an empty array. You don't need any out-path translators for this command.
6. Delete the "in-schema" parameter.
7. Set the "in-path" to be an empty array. You don't need any in-path translators for this command.
8. Save the file.

   Note that "{{ data.parameters }}", "{{ data.image }}", and "{{ data.command }}" will be replaced by values that will be passed into the command file. You will test this command file in a future lab.

Please notify your instructor when you have completed the lab.

End of Lab

Lab 18: Create remove-container.json

1. Copy `command-template.json` from the `ra_sample_files/Lab_18` directory to the directory for your device type within the commands directory of the RA project that you worked on in Labs #7-14 (This should be `~/shared/RA-Class/raclass/model/commands/docker/linux`). Rename the file `remove-container.json`.
2. Open the `remove-container.json` file.
3. Set the "endpoint" value to "cli".
4. Set the "type" value to "bpprov.runners.simple.Sequence".
5. In "endpoint-parameters" set the "command" value to "**docker rm -f {{data.container }}**".
6. Set the "out-path" to be an empty array. You don't need any out-path translators for this command.
7. Delete the "in-schema" parameter.
8. Set the "in-path" to be an empty array. You don't need any in-path translators for this command.
9. Save the file.

   Note that "{{ data. container }}" will be replaced by a value that will be passed into the command file. You will test this command file in a future lab.

End of Lab

Lab 19: Test Your RA

Before moving on with creating a Resource Provider you should test your RA using the API. This ensures that a session can be created and commands can be executed with the session. Remember, to use curl command you must first start your RA!

1. Start a new session:

```
$ curl -s -H "Content-Type: application/json" -d '{"authentication": {"cli":
   {"username": "vagrant", "password": "vagrant"}}, "connection":
   {"hostname": "localhost", "cli": {"hostport": 22}}, "typeGroup":
   "/typeGroups/DockerContainer", "id": "1"}'
   http://localhost:8080/api/v1/sessions | python -m json.tool
```

2. Run the `start-container.json` command file via the REST API call:

```
$ curl -s -H "Content-Type: application/json" -d '{"command":
   "start-container.json", "parameters": {"image": "busybox", "command":
   "ping localhost", "parameters": "--name=temp"}}'
   http://localhost:8080/api/v1/devices/1/execute | python -m json.tool
```

3. Verify the container is running:

```
$ docker ps –a
```

   You should see a new container for the *busybox* image with the name *temp*.
4. Stop the container:

```
$ curl -s -H "Content-Type: application/json" -d '{"command":
   "remove-container.json", "parameters": {"container": "temp"}}'
   http://localhost:8080/api/v1/devices/1/execute | python -m json.tool
```

5. After you get the JSON response back from stopping the container, run the **docker ps –a** command to verify the container has been removed.
6. Remove the session:

```
$ curl -v -X DELETE http://localhost:8080/api/v1/sessions/1
```

At this point you can stop your RA. The next several labs do not require your RA to be running. You will restart it later when you onboard the RA to Orchestrate.

Please notify your instructor when you have completed the lab.

End of Lab

Challenge Lab: Create your own RA from scratch

This lab may be assigned by the instructor if time permits. If there is not enough time, this lab is also a good post-class exercise.

The goal of this lab is to create your own RA from scratch. Typically, when you create an RA, you know the device you want to communicate with and what commands you want to execute on the device. How you configure the RA to perform these commands is normally up to you.

Create a new RA project using the following general guidelines:

1. This will be a CLI-docker based RA
2. Hard code your show-device.json file (you can choose the values)
3. Additional commands:

   show-proc.json:
   Returns a JSON object that contains the following:
   - PID of first sleep command
   - User owner of first sleep command
   - Current date

   JSON Example:

```
{
    "Owner": "vagrant",
    "Date": "Sep 15 14:46:04 UTC 2016",
    "PID": "4594"
}
```

   Hint: OS commands:
   ps -fe | grep sleep | head -1
   date

   start-sleep.json:
   Executes the following command: `(nohup sleep {{input}} &)`

   kill-sleep.json:
   Stops all sleep commands
   Hint: OS command: `killall`

Continued on next page…

Helpful hints:

- Make use of the previous RAs that you have developed in class. Code reuse will make the process of creating this RA easier.
- Test each step using the **bpprov-cli** and/or **curl** commands. Don't create a bunch of files and then decide to test them all at once. It's easier to troubleshoot one file at a time.
- Realize that creating your first "from scratch" RA normally takes time and patience. Don't expect this to be a quick lab.

Please notify your instructor when you have completed the lab.

End of Lab

Lab 20: Adding RPSDK (Optional)

RPSDK is added to setup.py by the paster command. If you chose not to include an RP when you ran the **paster** command, but want to add an RP later you can follow this procedure.

If you added an RP during paster, you do not need to do this lab.

1. If your RA is running, stop it.
2. From the Vagrant bash prompt, move to your RA project directory and edit the `setup.py` file.
3. In the "install_requires" variable add "rpsdk[ra]":

```
install_requires=[
       'rasdk','rpsdk[ra]'
       ],
```

4. Save the file.
5. Since you added a package you need to re-run the **make** command. Start by moving to your RA project folder:

```
$ cd ~/shared/RA-Class
```

6. First remove the existing `env` directory, then run the `make` command

```
$ rm -rf env

$ make prepare-venv
```

7. You also need to run the **make requirements** command to set "rpsdk" as a requirement in the `requirements.txt` file:

```
$ make requirements
```

Please notify your instructor when you have completed the lab.

End of Lab

Lab 21: Create Resource Definitions

Task 1: Create Resource Definitions

Note: Complete this for the RA that you created in labs #7-#19 (the raclass docker).

For this part of the lab you need to use the following:

- The name of the package used by your resource types will be: docker
- The full name of the docker container resource type: docker.resourceTypes.DockerContainer
- The full name of the docker image resource type: docker.resourceTypes.DockerImage

1. Copy **`resource_types_image.tosca`** and **`resource_types_container.tosca`** from the **`ra_sample_files/Lab_21`** directory to the **`model/resources/definitions`** directory.
2. Open the **`resource_types_container.tosca`** file.
3. Note the following:
    a. Verify the "package" name is set to "docker".
    b. Verify the resource name is set to "DockerContainer" (look at line #14).

   Based on these settings, the resource type identifier for this resource type will be docker.resourceTypes.DockerContainer
4. Do not close the **`resource_types_container.tosca`** file
5. Open the **`resource_types_image.tosca`** file.
6. Note the following:
    a. Verify the "package" name is set to "docker".
    b. Verify the resource name is set to "DockerImage" (look at line #14).

       Based on these settings, the resource type identifier for this resource type will be docker.resourceTypes.DockerImage
    c. Near the bottom of the file, in the "capabilities" property there is a property called "resourceType"s. Verify that this is set to "docker.resourceTypes.DockerContainer".
7. Go back to **`resource_types_container.tosca`**.
8. Near the bottom of the file, in the "requirements" property there is a property called "resourceType"s. Verify that this is set to "docker.resourceTypes.DockerImage".
9. Note: there should be no changes made to these files. Close each file.

Continued on next page…

Task 2: Update rp_config.yaml

1. Copy `rp_config.yaml` from the sample files directory to the `model/resources` directory of your RA project.
2. Open the file in a text editor or IDE.
3. Verify that the "id" value (on line #4) is set to "urn:organization:bp:ra:raclass" where "raclass" is the name of your RA project (found in `settings.py`) and organization is your company name (with no spaces or special characters).
4. NOTE: These identifiers will be used in several places. You must use the same value for the organization field every time you update an identifier with this format.
5. Update the value of contacts, title, and version to reflect you, your organization, and RA version.
6. Set the "id" value in the "domains" section (see line #51) to reflect your organization and RA name, for example "urn:ciena:bp:domain:raclass".
7. Verify that the "domains" properties includes the following:

```
domains:
   - id: "urn:ciena:bp:domain:raclass"
     title: "Docker" # max 32 chars
     description: "Domain for managing Docker containers"
     properties:
       - name: username
         title: "Username"
         optional: false
         description: "Username for a user with docker access"
         type: string
       - name: password
         title: "Password"
         optional: false
         obfuscate: true
         description: "Password for a user with docker access"
         type: string
```

8. At the same level as the "id", "title", "description", and "properties" keys of the "domains" key, add the following so the Resource Provider can tell BPO about the Resource Types for this Resource Adapter:

```
resource_types:
       - id: "docker.resourceTypes.DockerContainer"
         discoveryStrategy:
           - strategyType: "sync"
```

```
            pollingMode: "listOnly"
            pollingSettings:
              - pollingIntervalMs: 30000
      - id: "docker.resourceTypes.DockerImage"
        discoveryStrategy:
          - strategyType: "sync"
            pollingMode: "listOnly"
            pollingSettings:
              - pollingIntervalMs: 30000
```

Task 3: Create Relationships

1. In the **ra_sample_files/Lab_21** directory open **relationship.yaml**.
2. Copy the contents of the file.
3. Open **rp_config.yaml**.
4. Paste the contents of **relationship.yaml** at the bottom of the **rp_config.yaml** file. Relationships should be a top-level object at the same level as "domains", not a key within the "domains" values.
5. Verify that the "targetTypeId" is the name of the docker image resource identifier. This can be found in **rp_config.yaml** under "resource_types".
6. Verify that the "sourceTypeId" is the name of the docker container resource identifier. This can be found in **rp_config.yaml** under "resource_types".
7. Save the file.

   Important: compare your **rp_config.yaml file** to the file of the same name in the student answer files (**Answer Files/Lab_21** directory). If there are any differences, fix your file before moving on!

Continued on next page…

Task 4: Identify Products

1. Copy `products.json` to from the `ra_sample_files/Lab_21` directory to the `model/resources` directory. You will be replacing the existing `products.json` file.
2. Open `product.json`.
3. Verify that the "resource_type" values to the resource type identifier matches those defined in your definition files (the `model/resources/definitions/resource_types_container.tosca` and `model/resources/definitions/resource_types_image.tosca` files).
4. There is no need to make changes to this file. After reviewing the contents, close the file without saving.

Please notify your instructor when you have completed the lab.

End of Lab

Lab 22: Command Mapping

Task 1: Create commands.json

1. Copy the **`commands.json`** file from the **`ra_sample_files/Lab_22`** directory to the **`model/resources`** directory.
2. For each product defined in **`products.json`** create name:value pair:
   a. Use the "provider_product_id" from the **`products.json`** file as name.
   b. Specify a JSON file that will be for command listing. You will create the file later in this lab. For now, just specify the filename.

   For example:

```
"urn:cienatraining:bp:product:nix_container": "container.json"
```

   Note: There should be two settings, here; one for the docker container product and one for the docker image product. If you are in doubt as to what this file should look like, compare your **`products.json`** file to the answer file in the **`Answer Files/Lab_22`** directory
3. Save the file.

Task 2: Command Mapping Files

In this part of the lab you will create the command mapping file for each of the products specified in the **`commands.json`** file that was created in Task #1. The command mappings map CRUDL commands from Orchestrate to bpprov command files.

You must use the filenames specified in Task #1.

1. From the **`ra_rample_files/Lab_22`** directory, copy **`container.json`** and **`image.json`** to the **`model/resources`** directory.
2. Rename **`container.json`** to the filename specified in Task #1 for the command mapping for the docker container resource type (note: if the name of this file matches with the value that you created in the **`commands.json`** file, then there is no need to rename this file).
3. Rename **`image.json`** to the filename specified in Task #1 for the command mapping for the docker image resource type (note: if the name of this file matches with the value that you created in the **`commands.json`** file, then there is no need to rename this file).
4. Open the docker container command mapping file.
5. Verify that the "product" and "resource" values to match the values for the docker container product specified in the **`products.json`** file:

```
"product": "urn:ciena:bp:product:docker_container",

"resource": "docker.resourceTypes.DockerContainer"
```

6. Close the file.
7. Open the docker image command mapping file.

Continued on next page…

8. Verify that the "product" and "resource" values match the values for the docker image product specified in the **`products.json`** file:

```
"product": "urn:ciena:bp:product:docker_image",

"resource": "docker.resourceTypes.DockerImage",
```

9. Close the file.

Task 3: Copy Command Files

All the commands specified in the command mapping files need to be in the correct directory under **`model/commands`**. All of the **`.json`**, **`.tmpl`**, and **`.fsm`** files needed for the commands are provided in the **`ra_sample_files/Lab_22`** directory.

You must copy the files to the correct folder.

1. Open **`device.json`**. This file is in the **`model`** directory.
2. Note the location specified by the "dir" property. This is the location where you need to put the **`.json`** command files. Note that this should be the **`docker/linux`** directory which should be under the **`~/shared/RA-Class/raclass/model/commands`** directory.
3. Copy the **`.json`** command files that are identified in the **`container.json`** and **`image.json`** files from the **`ra_sample_files/Lab_2`** directory to the directory specified in by the "dir" property in the **`device.json`** file.
4. Copy the all of the **`.tmpl`** files to the **`model/templates`** directory.
5. Copy the **`.fsm`** file to the **`model/fsms`** directory.

Task 4: Modify Command File

The **`remove-container.json`** file in the **`~/shared/RA-Class/raclass/model/commands/docker/linux`** directory has a property that is no longer is used (data.container). Replace this property with data.id (the id property will be provided by the RA when the resource is created).

Please notify your instructor when you have completed the lab.

End of Lab

Lab 23: Create Session Templates

Task 1: Create Session Mapping File

1. In **`model/resources`** open the **`session-templates.json`** file.
2. Add the following object to the file:

```
"urn:ciena:bp:domain:raclass": {
         "template": "ra-session-create.tmpl"
     }
```

3. Note that the object name ("urn:ciena:bp:domain:raclass") is the domain identifier you specified in the **`rp_config.yaml`** file.
4. Save the file.

Task 2: Add an RA Session Template

1. Copy **`ra-session-create.tmpl`** from the **`ra_sample_files/Lab_23`** directory to the **`model/templates`** directory. This will overwrite the existing file.
2. Open the file.
3. Near the bottom of the file you'll see a "typeGroup" property. Update the last part of the path to the "TYPE_GROUP" value you specified in the **`settings.py`** file.
4. Save and close the file.

Please notify your instructor when you have completed the lab.

End of Lab

Lab 24: Onboard Your RA to Orchestrate

In this lab, you will onboard your RA into Orchestrate in your development environment. For this lab, you will need four SSH session windows:

- One to run bpocore-dev
- One to run the Orchestrate UI
- One to run your RA
- One for working in the Vagrant environment

Task 1: Start Orchestrate Docker Containers

Lab 2 provides instructions for starting the Orchestrate docker containers. Complete that lab and leave the containers running.

Task 2: Add UI Schema

A UI-Schema is needed by the Orchestrate web UI for each active product. For this RA there is only one active product: the docker container product. The UI-Schema is provided in the sample files folder.

1. Change to the **`~/shared/RA-class/raclass`** directory
2. Create a new directory using the following command (note: no spaces in the following command):

```
$ mkdir -p
   model/resources/ddui-schema/types/ddui/views/docker.resourceTypes.Docker
   Container
```

3. From the **`ra_sample_files/Lab_24/UI-files`** directory, copy the files **`create.json`**, **`detail.json`** and **`edit.json`** to the **`docker.resourceTypes.DockerContainer`** directory you created in step 1.

Task 3: Setup Environment Variables and Generate Keys

1. In the SSH session window that will run your RA, enter the following commands:

```
$ export BPOCORE_IP=$(docker inspect --format="{{ .NetworkSettings.IPAddress
   }}" bpocore-dev)
$ export DOCKER_BRIDGE_IP=$(docker inspect --format="{{
   .NetworkSettings.Gateway }}" bpocore-dev)
$ export MARKET_URL="http://$BPOCORE_IP:8181/bpocore/market/api/v1"
$ export ASSETS_URL=http://$BPOCORE_IP:8181/bpocore/asset-manager/api/v1
```

Continued on next page…

2. Generate ssh keys:

```
$ ssh-keygen -R $BPOCORE_IP

$ ssh-keyscan -H $BPOCORE_IP >> ~/.ssh/known_hosts
```

3. Move to your RA folder:

```
$ cd ~/shared/RA-Class
```

4. Start your RA with options needed for the dev environment:

```
$ env/bin/raclass --dev --declare-rp --no-clustering
```

Wait for the RA to finish starting before continuing. You should see a log message indicating the RA is ready, for example:

```
2016-08-22 21:46:56,170 INFO rpsdk.utils.ra market at
   http://172.17.0.2:8181/bpocore/market/api/v1 READY
```

If you get any error messages stop the RA and try to resolve the issue. Restart the RA once the issue has been resolved.

Continued on next page…

Task 4: Create a Domain

1. In a web browser go to the Orchestrate UI. The address is http://192.168.33.10:9980/orchestrate/.
2. Go to "Orchestration" -> "Domains".
3. Click the "Create" button.
4. From the list of available Resource Providers select your RP. This is the title that was specified in the `rp_config.yaml` file.
5. Enter a "Title".
6. Enter a "Description".
7. Leave the "Access URL" at the default value.
8. In the "Username" field enter **vagrant**.
9. In the "Password" field enter **vagrant**.
10. Click on the "Create" button.
    Your new domain will show up in the list of Domains. You may need to refresh the screen.
11. Click on the title of your Domain.

At this point if you do not see products and/or resources then your RA is not functioning correctly. Check the RA log window and the bpocore-dev log window for errors.

Once you address any issues, you will need to remove the domain from Orchestrate and add it again.

As you troubleshoot issues you may need to reset the Orchestrate environment by stopping the containers and restarting them. Always use the following order:

- Start bpocore-dev
- Start orchestrate-ui-dev
- Setup Environment Variables and Generate Keys (Task 3 of this lab) – Must be done if you restart bpocore-dev and orchestrate-ui-dev
- Wait for bpocore-dev to finish starting
- Start your RA using env/bin/raclass --dev --declare-rp --no-clustering

Please notify your instructor when you have completed the lab.

End of Lab

Lab 25: Start a Docker Container

In this lab, you will use Orchestrate to start a docker container. Your RA must be running and there should be no errors reported in the RA log window.

It's recommended that you monitor the bpocore-dev log as you complete this lab in case there are errors messages.

1. In the Orchestrate web UI select the "Network" -> "Product Catalog" menu items. This will take you to a list of all available products.
2. Click on the title of your docker container product.
3. Click the "Inventory" button. This will display a list of Resources instantiated from the docker container resource type.
4. Click the "Create" button.
5. In the "General" tab:
   a. Enter a "Label".
   b. Enter a "Description".
   c. Uncheck the "Auto Clean" box.
6. In the "Docker Container" tab:
   a. Enter a "Container Name".
   b. In the "Image" menu select the **busybox** image.
   c. In the "Command" field enter **ping localhost**.
7. Click the "Create" button. After a few moments, you should see your new resource in the Inventory list. You may need to refresh your screen.
8. Select the title of your new resource. Check the "Orch state" field. It should show "Active".
9. Open an SSH session to the Vagrant development environment and run the **docker ps -a** command. You should see the docker container you created listed.
10. In the Orchestrate UI terminate the resource:
    a. Select "Network" -> "Orchestrate Resources" menu items.
    b. In the text entry field along the top of the screen enter the title of the Resource to filter the Resources list.
    c. Select the box next to the "Resource" title.
    d. Select the "Delete" button and then select "Delete" on the confirm pop-up.
11. Go back to the SSH session window and re-run the **docker ps -a** command. The docker container you created should not be listed.

Please notify your instructor when you have completed the lab.

End of Lab