HOL6663: Build your own cloud environment with Solaris 11 RAD + REST

Xiaosong Zhu

Principal Software Engineer, Oracle

Yu Wang

Principal Software Engineer, Oracle

Yan Zhang

Software Engineer, Oracle

Table of Contents

Introduction

Remote Administration Daemon (RAD), a new service in Solaris11, may provide secure, remote administrative access to an Oracle Solaris system. RAD provides a programmatic interface to OS subsystems such as Solaris Zones, ZFS file system, and so on.

In this lab, you will learn how to use RAD to build a simplest IaaS environment, the most common actions like query, request, boot and modify virtual machines are simulated in the lab. Solaris Zones is the default virtualization technology on Solaris, you will learn how to use JAVA client to list zones, show zones' properties, and create zones, also you will learn how to control and modify zones by RESTful interface.

Prerequisites

This hands-on lab assumes you have some basic knowledge about the following technologies.

1. Java language programming
2. Oracle Solaris or a similar UNIX or Linux OS

Hardware/Software Requirements

- Disk space requirement: 40 GB
- Oracle VM Virtualbox 4.3.14 or later (host OS: Windows 7/8, Oracle Enterprise Linux)
- At least 4 cpus, and 8G Memory, as lab may need two virtual box instances
    - hol6663-vbox1 : 2 vcpus, 4G memory, hostname: hol6663-1
    - hol6663-vbox2: 1 vcpus, 2G memory, hostname: hol6663-2
- Operating System: Solaris 11.3

Preparing the Environment

1. Download and Install Oracle Solaris 11.3 in virtualbox
   - You can download Oracle Solaris 11.3 virtualbox template at http://www.oracle.com/technetwork/server-storage/solaris11/downloads/vm-templates-2245495.html
   - Import the virtual box images for *hol6663-1* and *hol6663-2*.

	hol6663-1	hol6663-2
cpu	2	1
memory	4G	2G
network	Host-only	Host-only Open "Advanced" sub-menu, change the MAC address, and make it different form hol6663-1
General->advanced->shared clipboard	bidirectional	bidirectional

   - *Please initialize the OS and set the environ*ment with following configurations

Parameters	Machine 1	Machine 2
hostname	`hol6663-1`	`hol6663-2`
IP address	`192.168.100.1/24`	`192.168.100.2/24`
Username/password	`solaris:solaris11` `root: solaris11`	`solaris:solaris11` `root: solaris11`
publisher	Should be accessible. You'd better download the repository, and set it as the local repository	Should be accessible
Install packages	Jdk8: *`pkg install jdk-8`*	
/root/.profile	Add following lines to */root/.profile* *`export JAVA_HOME=/usr/java/`* *`export ANT_HOME=/root/rad/tools/apache-ant-1.9.6`* *`export PATH=/usr/bin:/usr/sbin:/$ANT_HOME/bin:.`*	
/etc/hosts	Add hol6663-2 to */etc/hosts* as following: *`192.168.100.2  hol6663-2`*	
Activate the resource pools	# *`pooladm -e`*	# *`pooladm -e`*
Others	Open the root authority for console login and remote login	Open the root authority for console login and remote login

2. Zones setup

   Create two local zones in hol6663-1, and one local zone in hol6663-2.

- You can configure and install local zone according to following documents.

  Introduction to Oracle® Solaris Zones

  Creating and Using Oracle® Solaris Zones

- Make sure the three zones are setup with following configuration

Status	Keys	hol6663-1		hol6663-2
Configuration	zonename	zone1	zone2	zone1
	zonepath	/system/zones/zone1	/system /zones/zone2	/system /zones/zone1
	Configured	Yes	Yes	Yes
Installation	zoneadm -z *zonename* install	Yes	No	No
Initialization	Initialized by zlogin -C *zonename*	Yes	No	No

3. Prepare libraries and tool.
   Download the following libraries, which are necessary in building.
   Json-simple.1.1.1.jar: https://json-simple.googlecode.com/files/json-simple-1.1.1.jar
   HttpClient4.5: http://hc.apache.org/downloads.cgi
   Jstl-1.2.jar: http://www.java2s.com/Code/JarDownload/jstl/jstl-1.2.jar.zip

   And download the following tools which may be used in compiling and deployment.
   Apache-ant 1.9.6 : http://ant.apache.org/bindownload.cgi
   Apache-tomcat 8.0: http://tomcat.apache.org/download-80.cgi

4. Add source code
   All the source codes are listed in appendix A and appendix B, following them to add source code for the lab.

Notes for Users

- The virtual environment is consisted of two Vitualbox instances, hol6663-vbox1 and hol6663-vbox2. The VM hol6663-vbox1 works as the control node. It will manage all the machines in the virtual environment include itself. And most of the lab exercises are done on hol6663-vbox1. The VM hol6663-vbox2 works as the remote node, and will be managed by hol6663-vbox1 in the lab.
- Both of the Vitualbox instances, hol6663-vbox1 and hol6663-vbox2, are startup. If not, please start them up.
- The lab prefers the GNOME desktop environment over Oracle Solaris 11 (with desktop packages installed).
- In order to open a terminal window in GNOME, right-click any point on the background of the desktop, and select *Open Terminal* in the pop-up menu (as shown in Figure 1).

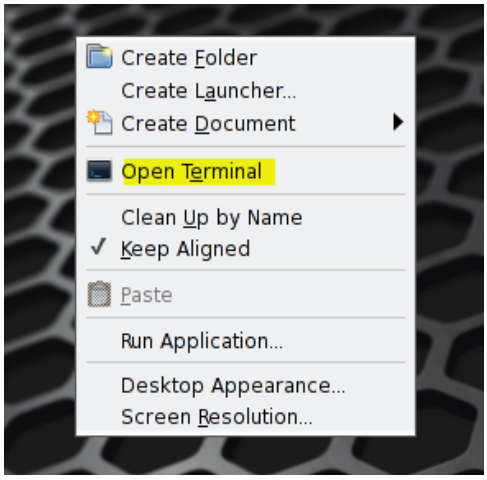

Figure 1. Open a terminal in Solaris 11

Exercise 1: Using RAD to Query Virtual Machines in the Cloud (15 Minutes)

In this exercise, you will learn how to start RAD services on physical machines, and setup RAD connections between the control machine and clients. Solaris Zone is used as the virtualization technology in this lab. All of the virtual machines are Solaris Zone based. You will use RAD to list zones in the cloud environment, and show properties of a selected zone by a JAVA client.

What is RAD?

The Remote Administration Daemon commonly referred to by its acronym and command name, RAD, is a standard system service that offers secure, remote administrative access to an Oracle Solaris system. The Remote Administration Daemon is the central point where system developers can expose their components for configuration or administration, and where the various programmatic consumers can go to perform those activities. RAD employs a client/server design. RAD itself acts as a server that services remote procedure calls, while RAD consumers are the clients

RAD is an integrated technology included in Oracle Solaris 11 that exposes a set of programmatic interfaces through a set of modules in a unified way to Oracle Solaris technologies. The client-side language bindings currently supported are C, Java, Python, and REST APIs over HTTP or HTTPS. Administrators and developers can use these client bindings to connect locally or remotely to systems, to view, and to manage system configuration much like the existing Oracle Solaris command-line interfaces.

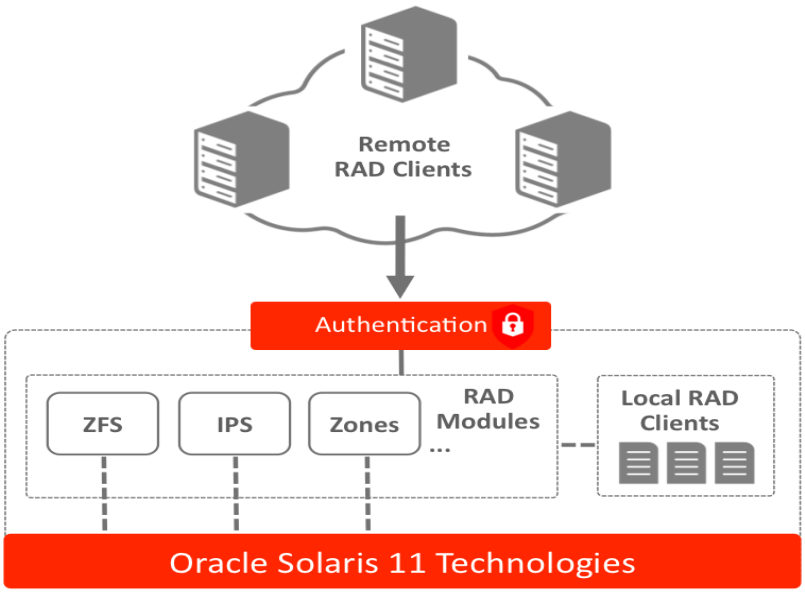

Figure 2. A high-level architecture of RAD

What is a Solaris Zone?

A Solaris Zone is an implementation of operating system-level virtualization technology for x86 and SPARC systems. A Solaris zone is the combination of system resource controls and the boundary separation. Zones act as

completely isolated virtual servers within a single operating system instance. By consolidating multiple sets of application services onto one system and by placing each into isolated virtual server containers, system administrators can reduce cost and provide most of the same protections of separate machines on a single machine.

Step 1: Check SMF service instances for RAD are enabled

1. Logging in to the machine hol6663-1 with username/password: *root/solaris11*, as the RAD task through connection will perform with privileges available to the user account that is running the program. Click the right mouse button on the desktop and choose *Open Terminal* to bring up a terminal window (as shown in Figure 1). To meet the permission requirement of zone administration, we used user *root* directly in this lab.

2. Check RAD services are enabled on the machine.

In Oracle Solaris 11.3, there are 3 new SMF service instances responsible for RAD connections, we may use UNIX socket for local machine, and TLS connection for remote machine in this lab. Please check the following two SMF services are shown as *online* on both of the machines:

- The svc:/system/rad:local service instance manages local connections through UNIX sockets.
- The svc:/system/rad:remote service instance manages secure remote TLS connections over TCP sockets.

```
root@hol6663-1:~# svcs rad
STATE          STIME    FMRI
online          7:42:01 svc:/system/rad:remote
...
online         15:40:31 svc:/system/rad:local
```

The service *svc:/system/rad:local* is enabled by default, if *svc:/system/rad:remote* is shown as disabled, please enable it, otherwise skip the following command.

```
root@hol6663-1:~# svcadm enable rad:remote
```

The public key for the remote rad SSL service is created in the file named "cert.perm" under the directory "/etc/rad/" when you enable the service, make a file link for “cert.perm” with hostname, if you will use the hostname in later exercises.

```
root@hol6663-1:~# ln -s /etc/rad/cert.perm /etc/rad/hol6663-1
```

Step 2: Connect to RAD

1. We have created *ConnectMachine.java* under */root/rad/java/oow2015/hol6663/exercises* to handle the different types of connections with RAD.

```
root@hol6663-1:~# cd /root/rad/java/oow2015/hol6663/exercises
root@hol6663-1:~/rad/java/oow2015/hol6663/exercises# gedit ConnectMachine.java
```

An editable window pops up which contains the source code, and we can add connection source code in this window.

2. You can find two RAD packages are imported for RAD connection.

- com.oracle.solaris.rad.connect – The classes for connecting to a RAD instance
- com.oracle.solaris.rad.client – The client implementation of the RAD protocol plus associated functionality

```
// Import RAD packages here

import com.oracle.solaris.rad.client.*;
import com.oracle.solaris.rad.connect.*;
```

All the RAD java libraries are located in */usr/lib/rad/java*, now the version number is 1.

```
root@hol6663-1:~# ls /usr/lib/rad/java
authentication_1.jar  files_1.jar           smf_old_1.jar
authentication.jar    files.jar             smf_old.jar
config_1.jar          kstat_1.jar           smf.jar
config.jar            kstat.jar             time_1.jar
container_1.jar       modules_1.jar         time.jar
container.jar         modules.jar           usermgr_1.jar
control_1.jar         network_1.jar         usermgr.jar
control.jar           network.jar           zfsmgr_1.jar
dlmgr_1.jar           pam_1.jar             zfsmgr.jar
dlmgr.jar             pam.jar               zonemgr_1.jar
errors_1.jar          panels_1.jar          zonemgr.jar
errors.jar            panels.jar            zonesbridge_1.jar
evscntl_1.jar         rad.jar               zonesbridge.jar
evscntl.jar           smf_1.jar
```

3. Connect to the local instance

Connection with a RAD instance is provided by the class *com.oracle.solaris.rad.connect.Connection* .

When connect to the local instance, implicit authentication by using syscall getpeerucred(3C) is performed against your user id directly. Add the following bold source code to the method *localConnect()*, it will return a *Connection* instance which provides a standard interface to interact with RAD.

```
    public Connection localConnect() throws IOException {
        //=============================================================//
        // Please input the Source code for local connection here     //
        //=============================================================//
        // Open a RAD connection on a UNIX Socket
        URIConnection uri = new URIConnection("unix:///");
        con = uri.connect(null);
        //=============================================================//
        // Input end                                                   //
        //=============================================================//
        return con;
    }
```

4. Connecting to a Remote Instance and Authenticating

In Solaris 11, the RAD service *svc:/system/rad:remote* provides TLS connection by default for remote machines.

The public key for the remote rad SSL service is stored in the file named "*cert.perm*" under the directory "*/etc/rad/*" of that physical machine. For the simplicity of coding, when we access remote RAD through SSL, we just copy this cert file to the control machine, name it as "*/etc/rad/<machine_name>*", and add this certificate to the trust store of the control machine.

Copy the following bold source code to the method *tlsConnect()* to provide remote TLS link to RAD instance.

```
    public Connection tlsConnect(String hostIP) throws IOException {
        //=============================================================//
        // Please input the Source code for remote connection here    //
        //=============================================================//
        // Open a TLS connection over TCP
        // Make connection with certification information
        Set<String> certs = new HashSet<String>();
        String cert_file = "/etc/rad/" + hostIP;
        certs.add(cert_file);
        con = Connection.connectTLS(hostIP, certs);

         // Perform a PAM login
        RadPamHandler hdl = new RadPamHandler(con);
```

```
        hdl.login("C", "root", "solaris11");
        //=============================================================//
        // Input end                                                   //
        //=============================================================//
        return con;
    }
```

Authentication is necessary for connecting with the remote RAD instance, and we performed a PAM login by class *com.oracle.solaris.rad.client.RadPamHandler*. In the demo program, we just connected to the remote machine in a non-interactive fashion, *"C"* stands for the locale, username is "*root*", and the password is "*solaris11*".

Press the button *Save* 

in the toolbar to save the file after typing. You can keep the gedit window open, as we may use it in later steps.

Step 3: List zones

To get a first glance of the virtual environment, we may need to list all the virtual machines in the cloud.

1. A public class *ZoneQuery* is already created in */root/rad/java/oow2015/hol6663/exercises*, which may include all the query method in this lab. Press the button Open 

in the toolbar of gedit, a window popped up for you to choose file, shown in figure 3, select *ZoneQuery.java* and press button *Open*.

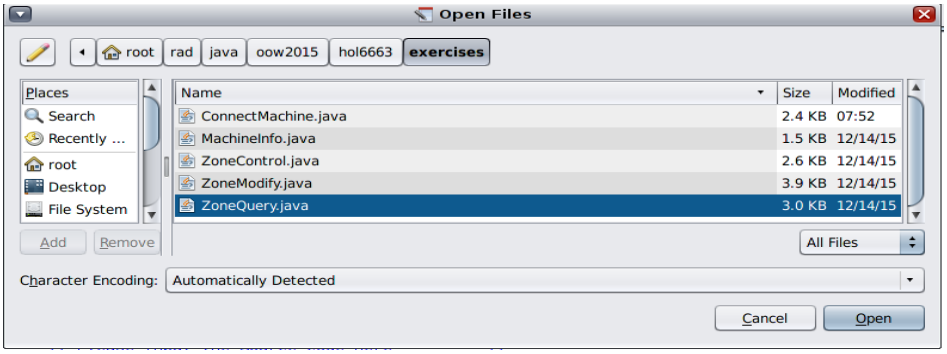

Figure 3. Select file in gedit

The file *ZoneQuery.java* is opened in gedit window, seen in figure 4.

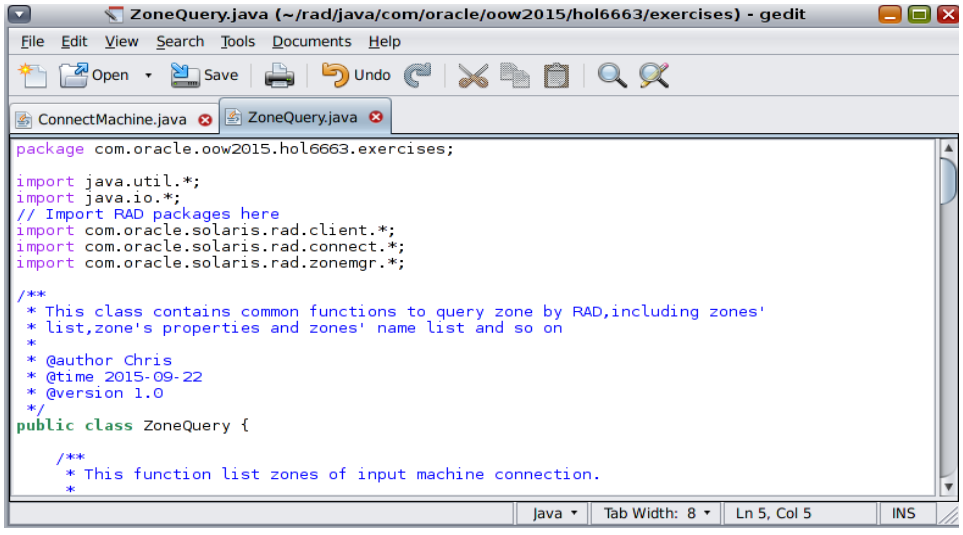

Figure 4. Show and edit files in gedit

2. We import a new RAD package *com.oracle.solaris.rad.zonemgr* for zone management.

```
// Import RAD packages here
import com.oracle.solaris.rad.client.*;
import com.oracle.solaris.rad.connect.*;
import com.oracle.solaris.rad.zonemgr.*;
```

You can learn more about the package by man.

```
root@hol6663-1:~# man zonemgr
```

3. Add the following bold source code in method *listZones()*, it will get zones list from the connected RAD server, and show the zone name, zone status, install path, brand type, and IP type of each zones

We can get a list of object names of type *ADRName* by *Connection. listobjects()*, then loop through these names to get the underlying objects associated to those names. This corresponds to how RAD internally maps between JAVA and the interfaces that use the Abstract Data Representation [ADR] definition language.

For each zone object, we can get all kinds of properties by method *getResourceProperties()*, properties are grouped into several resources, here we just concern about the properties in resource "global". In next step we will talk more about resources and properties.

```
    public static void listZones(Connection con) throws IOException {
        // Print out list header of the zones
        System.out.println();
        System.out.format("  %-16s %-11s %-16s %-10s %-10s\n", "NAME", "STATUS",
                "ZONEPATH", "BRAND", "IP");
        //=========================================//
        // Please input the Source code here       //
        //=========================================//
        // Get Zones list, and loop through the zones list
        List<ADRName> zoneList = con.listObjects(new Zone());
        for (ADRName name : zoneList) {
            Zone zone = con.getObject(name);
            String zonepath = "";
            String ip = "exclusive";

            // Get the common properties of the zone
            Resource global_filter = new Resource("global", null, null);
            List<Property> props = zone.getResourceProperties(global_filter, null);
            for (Property prop : props) {
                if (prop.getName().equals("zonepath")) {
                    zonepath = prop.getValue();
                }
                if (prop.getName().equals("ip-type")) {
                    ip = prop.getValue();
                }
            }

            // Print Zone Name, Status, Zonepath, Zone Brand, and IP type in your format
            System.out.format("%-16s %-11s %-16s %-10s %-10s\n",
                    zone.getname(), zone.getstate(), zonepath, zone.getbrand(), ip);
        }
        //=========================================//
        // Input end                               //
        //=========================================//
    }
```

Press the button *Save* in the toolbar to save *ZoneQuery.java* after typing.

4. We have prepared a console program called *StartRad* in */root/rad/java*, the interface and the common functions and ready in *StartRad*, after we fulfilled the function of RAD connection, and zone query, let's compile and run it to check the output.

Click the right mouse button on the desktop and choose *Open Terminal* to bring up a terminal window, run *ant* in this terminal to build and run the program.

```
root@hol6663-1:~# cd /root/rad/java
root@hol6663-1:~/rad/java# ant -e
```

This is the console output, you can find that all the exercises are listed here.

```
…
================ HOL6663: RAD Exercises =================
    Exercise 1. Using RAD to Query Cloud
         a) List zones
         b) Show zone's property
    Exercise 2. Using RAD to Manage cloud
         c) Add a remote machine to the cloud
         d) request a zone from the cloud
         e) Install zone
============================================================
Please choose an exercise by input a letter (a/b/c/d/e),
then press the Enter key to confirm :    a
```

Input option *a*, and press *Enter* key to list zones. In the very beginning, we have only one machine in the cloud, the control machine itself, which is shown as local machine. You can try option *a* in later exercise, after you add more machines to the cloud.

```
List zones on local machine:
  NAME              STATUS      ZONEPATH          BRAND            IP
  zone1             installed   /system/zones/zone1 solaris    exclusive
  zone2             configured  /system/zones/zone2 solaris    exclusive
  Continue?(y/n,default is y)
```

Press enter to continue running.

Step 4: Show zones properties

In Step 3, we have gotten all the zones in the cloud, now we can choose a zone to see how it is configured.

1. Go to the gedit window, and select the tab contains *ZoneQuery.java*
2. Input the bold source code to fulfill the method of *showZoneProp()* in class *ZoneQuery*.

Zone configuration data consists of two kinds of entities: resources and properties.

- Each resource has a type, and each resource can have a set of one or more properties.
- The properties have names and values. The set of properties is dependent on the resource type.

Zones have several resources which include *global*, *anet/net*, *capped-memory*, *dedicated-cpu* and so on. If we want to show all the configuration of the zone, we may use method *getResources(filter, scope)* with no filter and no scope input to get the whole resources list, then go through the list and print all the properties of the selected resource by *getResouceProperties(res, properties)*.

```
     public static void showZoneProp(Connection con, String zonename)
            throws IOException {
        //get zone object
```

```
        Zone zone = getZone(con, zonename);
        if (zone == null) {
            System.out.println("  Show Zone Property ERROR: Can't find the zone " + zonename);
            return;
        }
        //============================================//
        // Please input the Source code here          //
        //============================================//
        // Go through all the resources
        List<Resource> resources = zone.getResources(null, null);
        for (Resource res : resources) {
            // Print the resource's type first
            System.out.println(res.getType());

            // Get all the properties of specified resource
            List<Property> proplist = zone.getResourceProperties(res, null);
            // Go through the properties list and print the proterty with value
            for (Property prop : proplist) {
                if (prop.getValue() != null && !prop.getValue().isEmpty()) {
                    System.out.format("      %-16s : %-20s\n", prop.getName(), prop.getValue());
                }
            }
        }
        //============================================//
        // Input end                                  //
        //============================================//
    }
```

A static method *getZone()* will return a *Zone* object by inputing the connected machine and the zone name, it is widely used in the whole lab.

```
     public static Zone getZone(Connection con, String zonename)
            throws IOException {
        if (con == null) {
            return null;
        }
        // iterate all the zones
        for (ADRName name : con.listObjects(new Zone())) {
            Zone zone = con.getObject(name);
            if (zone.getname().equals(zonename)) {
                return zone;
            }
        }
        return null;
    }
```

Press the button *Save* in the gedit toolbar to save *ZoneQuery.java* after typing.

3. Go to the terminal we used to run program, stop the StartRad by *ctrl + C*, then compile and run it again.

```
root@hol6663-1:~/rad/java# ant -e
…
================ HOL6663: RAD Exercises =================
    Exercise 1. Using RAD to Query Cloud
         a) List zones
         b) Show zone's property
    Exercise 2. Using RAD to Manage cloud
         c) Add a remote machine to the cloud
         d) request a zone from the cloud
         e) Install zone
=========================================================
Please choose an exercise by input a letter (a/b/c/d/e),
then press the Enter key to confirm :    b
```

Input option *b*, and press *Enter* key. It will list all the available machines in the cloud, there's only one machine now, and we will add more machines to the cloud in the later exercise. Choose the machine by entering a number, for example *1*.

```
Show zone properties on a specified machine

There are 1 machines in the cloud, please choose one:
  1. hol6663-1 (local)
Please input a number and press Enter to confirm (default option is 1) : 1
List zones on local machine:
NAME             STATUS      ZONEPATH         BRAND      IP
zone1            installed   /system/zones/zone1 solaris    exclusive
zone2            configured  /system/zones/zone2 solaris    exclusive
 Please input zone name, and press Enter key
zone1
```

Now input the zone name you are interested in, like *zone1*, after you press the *Enter* key, the detail configuration of the zone will be listed.

```
global
      zonename         : zone1
      zonepath         : /system/zones/zone1
      brand            : solaris
      autoboot         : false
      autoshutdown     : shutdown
      ip-type          : exclusive
anet
      linkname         : net0
      lower-link       : auto
      configure-allowed-address : true
      mac-address      : auto
      auto-mac-address : 2:8:20:d4:ab:b9
      tmp-id           : 0
  Continue?(y/n,default is y)
```

4. Open a new terminal window, and check the zone's configuration by command line.

```
root@hol6663-1:~# zonecfg -z zone1 info
zonename: zone1
zonepath: /system/zones/zone1
brand: solaris
autoboot: false
autoshutdown: shutdown
bootargs:
file-mac-profile:
pool:
limitpriv:
scheduling-class:
ip-type: exclusive
hostid:
tenant:
fs-allowed:
anet:
     linkname: net0
     lower-link: auto
     allowed-address not specified
     configure-allowed-address: true
     defrouter not specified
     allowed-dhcp-cids not specified
     link-protection: mac-nospoof
     mac-address: auto
     auto-mac-address: 2:8:20:88:e1:1c
     mac-prefix not specified
     mac-slot not specified
```

```
    vlan-id not specified
    priority not specified
    rxrings not specified
    txrings not specified
    mtu not specified
    maxbw not specified
    bwshare not specified
    rxfanout not specified
    vsi-typeid not specified
    vsi-vers not specified
    vsi-mgrid not specified
    etsbw-lcl not specified
    cos not specified
    pkey not specified
    linkmode not specified
    evs not specified
    vport not specified
```

The output of RAD program and command is similar, but in our program we just show the properties with value and omit the properties which are not specified.

Summary

In this exercise, we got a first glance at the RAD-based cloud. We learned how to use JAVA client to connect to the RAD instance in client machines, and query zones by RAD APIs including listing zones and getting zones properties. In the next exercise, we will learn how to use RAD APIs to manage the cloud.

Exercise 2: Using RAD to Manage the Virtual Machines in the Cloud (20 Minutes)

In this exercise, we will simulate two common scenes in the cloud system: adding a physical machine to the cloud and requesting a virtual machine from the cloud. RAD is used in the backend to get machine information, manage the zones, which include zone creation, modifying the basic configuration and installation.

Infrastructure as a service (IaaS)

IaaS refers to online services that abstract user from the detail of infrastructure like physical computing resources, location, data partitioning, scaling, security, backup etc. A hypervisor, such as Xen, KVM, VMware ESX/ESXi, or Hyper-V runs the virtual machines as guests. Pools of hypervisors within the cloud operational system can support large numbers of virtual machines and the ability to scale services up and down according to customers' varying requirements.

Zone-Wide Resource Control

Zone-wide resource control limits the total resource usage of all process entities within a zone

Oracle Solaris Zones allow you to create an isolated environment in which users can log in and do what they want without affecting anything outside that zone. Each Oracle Solaris Zone contains a complete resource-controlled environment, which allows you to allocate or limit the resource for all the process within a zone, like CPUs, memory, storages and network.

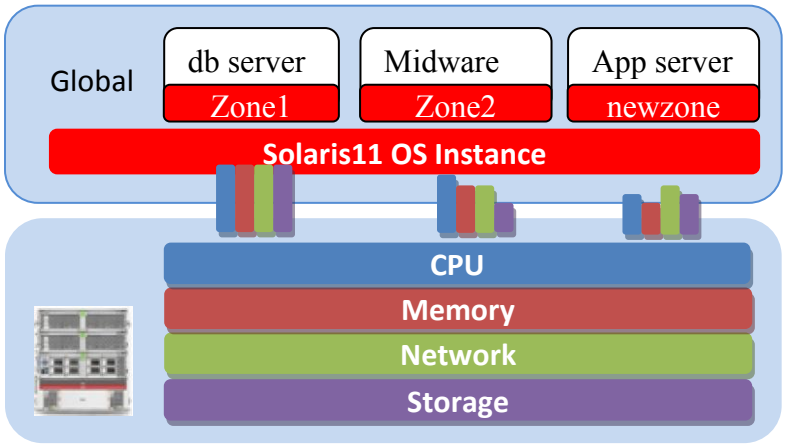

Figure 5 Solaris zones with resources

Step 1: Add a remote machine to the cloud

In Exec1, we found that the cloud just contains one machine hol6663-1 by default, which is also the control machine of the cloud. So we may expand the cloud by adding a new machine hol6663-2 to the cloud.

1. Switch to the VitualBox instance hol6663-vbox2, login to hol6663-2 with *root/solaris11*.

Click the right mouse button on the desktop and choose *Open Terminal* to bring up a terminal window, let's check the RAD services for hol6663-2.

```
root@hol6663-2:~# svcs rad
STATE          STIME    FMRI
online         13:27:02 svc:/system/rad:local
…
disabled         13:27:47 svc:/system/rad:remote
```

The service *svc:/system/rad:local* is enabled by default, if *svc:/system/rad:remote* is shown as disabled, please enable it, otherwise skip the following command.

```
root@hol6663-2:~# svcadm enable rad:remote
```

The public key for the remote rad SSL service is created in the file named "cert.perm" under the directory "/etc/rad/" when you enable the service, copy it to the control machine, and rename it with the hostname.

```
root@hol6663-2:~# scp /etc/rad/cert.perm root@hol6663-1:/etc/rad/hol6663-2
```

2. When we add a new machine to the cloud, we will check the resources of the machine, like CPU number and memory size. We can get the information by command *kstat*.

```
root@hol6663-2:~# kstat cpu_info |grep instance
module: cpu_info                        instance: 0

root@hol6663-2:~# kstat -n system_pages |grep physmem
    physmem                         1040158
```

3. Now let's realize it by using RAD JAVA APIs. A new RAD package *kstat* is used to get the resource information. Switch to the VitualBox instance hol6663-vbox1, go to the gedit window and press button *open* to open the file *MachineInfo.java* which is used to get CPU and memory information of selected machine.

4. Add the bold source code to the method *getMachineCPU()*, we get CPU number by counting the number of kstat *cpu_info* module. RAD interface *Connection* provides a wildcard pattern searching to help you find specific objects by name matching.

```
    public static int getMachineCPU(Connection con) throws IOException {
            int cpuNum = 0;
            //=============================================//
            // Please input the Source code here          //
            //=============================================//
            // Using ADRGlobPattern wildcard pattern search
            // to find all the cpu_info instance
            String keys[] = {"class", "module", "instance"};
            String values[] = {"misc", "cpu_info", "*"};
            ADRGlobPattern pat = new ADRGlobPattern(keys, values);

            // All the ADRGlobPattern matched object will
            // be returned by listObjects
            cpuNum = con.listObjects(new Kstat(), pat).size();
            //=============================================//
            // Input end                                   //
            //=============================================//
            return cpuNum;
        }
```

In last exercise, we go through the zone objects list and get a zone by matching zone's name. Actually we can refine the search by extending the name definition. ADRGlobPattern takes an array of keys and an array of values and extends the name used for the search.

5. Memory statistic information can be gotten by kstat *unix/system_pages* module. Add the bold source code to the method *getMachineMem()*, and get the machine physical memory pages by item "*physmem*".

```
    public static int getMachineMem(Connection con) throws IOException {
```

```
            int mem = 0;
            //============================================//
            // Please input the Source code here          //
            //============================================//
            // Memory statistic information are in kstats system_pages
            String keys[] = {"name"};
            String values[] = {"system_pages"};
            ADRGlobPattern pat = new ADRGlobPattern(keys, values);

            int pageSize = 4096;
            for (ADRName name : con.listObjects(new Kstat(), pat)) {
                Kstat kstat = con.getObject(name);
                // kstat show the statistic information of the time stamp you query
                Ksdata data = kstat.getsnapshot().getData();
                // the item physmem show physical memory all available on the machine in pages
                for (Kstat_named named : data.getNAMED()) {
                    if (named.getName().equals("physmem")) {
                        mem = (int) (named.getValue().getUINT32().getValue() * pageSize / 1024 /
1024);
                    }
                }
            }
            //============================================//
            // Input end                                  //
            //============================================//
            return mem;
        }
    }
```

Press the button Save in the toolbar to save *MachineInfo.java* after typing.

6. Delete the file .server.json in /root/rad/java/oow2015/hol6663 which contains all the machines' information in the cloud. Because the file is automatically created in the first time we run StartRad in Exercise 1, the CPU number and memory size are initialized as 0 as we don't fulfill the functionality at that time. We don't design the program to refresh the existing machines information, so just delete it, and let the program to create it automatically in next running.

```
root@hol6663-1:~/rad/java# rm  /root/rad/java/oow2015/hol6663/server.json
```

7. Go to the window we used to run program, stop the StartRad by ctrl + C, then compile and run it again.

```
root@hol6663-1:~/rad/java# ant -e
…
================ HOL6663: RAD Exercises =================
    Exercise 1. Using RAD to Query Cloud
         a) List zones
         b) Show zone's property
    Exercise 2. Using RAD to Manage cloud
         c) Add a remote machine to the cloud
         d) request a zone from the cloud
         e) Install zone
-----------------------------------------------------------
Please choose an exercise by input a letter (a/b/c/d/e),
then press the Enter key to confirm :    c
```

Input *c* and press *Enter* to add a machine to the cloud, we can find there's only one machine in the cloud now, input the machine's name(*hol6663-2*) we want to add.

```
  There are 1 machines in the cloud :
    1. hol6663-1
  Please input the server name you want to add to virtual environment:
hol6663-2
```

After the machine is added to the cloud, we get the following output, otherwise it will output the error information, and ask you to input the machine name again.

```
Add hol6663-2 sucessfully !
There are 2 machines in the cloud :
  1. hol6663-1
  2. hol6663-2
Continue?(y/n,default is y)
```

The machine information will be logged after it is added to the cloud. In next step when we request a virtual machine from the cloud, the cloud will go through the machine list to check resources, and return a right machine to create the zone. In this lab we concentrated on how to use RAD API to access the OS subsystem, not the cloud provision, so we will not talk about the algorithm of choosing the machine here.

Step 2: request Zones

In this exercise, a user raised a request for a virtual machine, which may be configured with 1 virtual CPUs and 1G memory. After the cloud chose a qualified machine, let's create a zone with dedicated CPUs and capped memory on that machine.

1. Go to the gedit window, press button *open* to open the file *ZoneControl.java* which is specified for the zone management by RAD.
2. Add the following bold source code to the method *create()*, and press button *save* in the toolbar to save file.

Here we get *ZoneManager* instance first, which works as the zone factory and used to create or delete zones in RAD, then create a standard zone with 3 inputs: zone name, zone path and a default template

```
    public boolean create(Connection con, String zonename)
            throws IOException {
        //============================================//
        // Please input the Source code here          //
        //============================================//
        // ZoneManger works as zones factory to create or delete zone configuration
        ZoneManager zmgr = con.getObject(new ZoneManager());
        try {
            // input zonename, zonepath and template to create a zone
            zmgr.create(zonename, "/system/zones/" + zonename, "SYSsolaris");
            return true;
        } catch (RadObjectException oe) {
            //RAD Error handling
            Result res = (Result) oe.getPayload();
            System.out.println(res.getCode());
            if (res.getStderr() != null) {
                System.out.println(res.getStderr());
            }
        }
        //============================================//
        // Input end                                  //
        //============================================//
        return false;
    }
```

Step 3: Add zone-wide resource control

In Step 2, we have created a standard zone, but what user requested is a zone with 1 dedicated CPUs and 1G memory, so we need to add some resources to the zone.

1. In the gedit window, press button *open* to open another file *ZoneModify.java* which is specified for the change zone properties.

2. When zone is created with default template, it shares CPU and memory with the whole machine. So we need to add resources to the zone by Zone interface *addResource()*, otherwise if the zone is configured with some resources, interface *setResourceProperties()* should be used instead to change the resource properties.

All modifications to the zones configuration must be done inside a transaction. At most one transaction for any given Zone instance in one RAD session can be active at any given time. Transactions are not opened automatically, so Zone interface *editConfig()* method must be called before modification while the transaction should be committed using *commitConfig*() after modification.

Add the bold source code to the method *addZoneProp()* with input arguments :

con -- machine connection

zonename -- the zone we want to add resources

type -- define the added resources

props -- contains the resource properties in list

```
        public static void addZoneProp(Connection con, String zonename, String type, List<Property>
props)
            throws IOException {
        // Get Zone object by zonename
        Zone zone = ZoneQuery.getZone(con, zonename);
        if (zone != null) {
            //============================================//
            // Please input the Source code here          //
            //============================================//
            // Create resource with properties
            Resource resource = new Resource(type, props, null);
            try {
                // Add resource to the zone configuration
                // We need to call editConfig before changing zone configuration
                // And call commitConfig after all the changes are finished
                zone.editConfig(null);
                zone.addResource(resource, null);
                zone.commitConfig();
            } catch (RadObjectException oe) {
                //RAD Error handling
                Result res = (Result) oe.getPayload();
                System.out.println(res.getCode());
                if (res.getStderr() != null) {
                    System.out.println(res.getStderr());
                }
            }
            //============================================//
            // Input end                                  //
            //============================================//
        }
    }
```

3. We will call *addZoneProp()* to add memory cap to the zone , the property physical memory is used to set the max-rss for this zone.

Add the bold source code to the method *addZoneMemoryProp()*, the zone resource type is *capped-memory*, and the property of the resource is set with *physical*.

```
    public static void addZoneMemoryProp(Connection con, String zonename, String memory)
            throws IOException {
        //============================================//
        // Please input the Source code here          //
        //============================================//
```

```
        // Zone's memory limitation is defined by resource "capped-memory"
        // Set the property "physical" of "capped-memory" for memory size
        List<Property> memprop = new ArrayList<Property>();
        memprop.add(new Property("physical", memory, PropertyValueType.PROP_SIMPLE, null, null));
        // Call "addZoneProp" in the class to add resource "capped-memory" to zone
        addZoneProp(con, zonename, "capped-memory", memprop);
        //============================================//
        // Input end                                  //
        //============================================//
    }
```

4. Call *addZoneProp()* again to add dedicated CPUs to the zone, this resource will create a pool and processor set for exclusive use by the zone when it boots. These processors are not available for other zones or the global zone while the zone is running.

Add the bold source code to the method *addZoneCPUProp()*. The zone resource type is *dedicated-cpu*, and the property of the resource is set by *ncpus*, it provides a convenient way to assign CPUs to zone, which means the CPUs dedicated to the zone can vary on each boot or live zone reconfiguration.

```
    public static void addZoneCPUProp(Connection con, String zonename, String cpuNumber)
            throws IOException {
        //============================================//
        // Please input the Source code here          //
        //============================================//
        // Zone's assigned CPUs is defined by resource "dedicated-cpu"
        // Set the property "ncpus" of "dedicated-cpu" for the CPU numbers assigned to the zone
        List<Property> cpuprop = new ArrayList<Property>();
        cpuprop.add(new Property("ncpus", cpuNumber, PropertyValueType.PROP_SIMPLE, null, null));
        // Call "addZoneProp" in the class to add resource "dedicated-cpu" to zone
        addZoneProp(con, zonename, "dedicated-cpu", cpuprop);
        //============================================//
        // Input end                                  //
        //============================================//
    }
```

Press the button *Save* in the toolbar to save the file after typing.

5. Go to the terminal we used to run program, stop the StartRad by ctrl + C, then compile and run it again.

```
root@hol6663-1:~/rad/java# ant -e
…

================ HOL6663: RAD Exercises =================
    Exercise 1. Using RAD to Query Cloud
         a) List zones
         b) Show zone's property
    Exercise 2. Using RAD to Manage cloud
         c) Add a remote machine to the cloud
         d) request a zone from the cloud
         e) Install zone
=========================================================
Please choose an exercise by input a letter (a/b/c/d/e),
then press the Enter key to confirm :    d
```

Input *d* then press *enter* to request a virtual machine from the cloud. It is asked to input CPUs and memory you request for the zone, here we input *1* for CPU number, and *1024* for memory size(MB).

```
  Please input cpu number(eg: 1),and press Enter key!
  Default is -1, meaning not specify zone's cpu number!
1
  Please input memory size(The unit is MB,eg:512),and press Enter key!
  Default is -1, meaning not specify zone's memory size!
1024
```

The application will choose an appropriate machine to create the zone. In this lab *hol6663-1* have 2 CPUs and 4G memory while *hol6663-2* has only 1 CPUs and 2G memory, so the application will choose *hol6663-1* to create the zone.

```
  The suitable server is hol6663-1
  Please input the zone name, and press Enter key:
newzone
```

After creation, the application will list all the zones on the machine.

```
  NAME              STATUS      ZONEPATH          BRAND      IP
  zone1             installed     /system/zones/zone1 solaris    exclusive
  zone2             configured  /system/zones/zone2 solaris    exclusive

  newzone           configured  /system/zones/newzone solaris    exclusive
```

You can check the zone property by choosing option *b* of the *StartRad*, or you can also check by command line

```
root@hol6663-1:~/rad/java# zonecfg -z newzone info
zonename: newzone
zonepath: /system/zones/newzone
brand: solaris
autoboot: false
autoshutdown: shutdown
…
ip-type: exclusive
hostid:
tenant:
fs-allowed:
anet:
     linkname: net0
     lower-link: auto
     …
dedicated-cpu:
     ncpus: 1
     cpus not specified
     cores not specified
     sockets not specified
capped-memory:
     physical: 1G
```

Step 4: Install Zones

Zone installation is the last step of requesting zone, but it will take a period of time, so we separate it to an independent option in the lab.

In Step 3, we have created a zone as we wish on hol6663-1, now let's install it

1. Go to the gedit window, switch to the tab of file *ZoneControl.java*.

2. Similar to Step 2, we add the bold source code to the method *install*() , here we install the zone without any arguments, you can see detail options for install by *man zoneadm*.

```
    public boolean install(Connection con, String zonename)
            throws IOException {
        //=========================================//
        // Please input the Source code here       //
        //=========================================//
        //get the zone by name
        Zone zone = ZoneQuery.getZone(con, zonename);
        if (zone != null) {
            try {
                // zone install
                zone.install(null);
                return true;
            } catch (RadObjectException oe) {
                //RAD Error handling
                Result res = (Result) oe.getPayload();
                System.out.println(res.getCode());
                if (res.getStderr() != null) {
                    System.out.println(res.getStderr());
                }
            }
        } else
            System.out.println("Can't find zone " + zonename);
        //=========================================//
        // Input end                               //
        //=========================================//
        return false;
    }
```

Press the button *Save* in the toolbar to save the file after typing, now you can close all the gedit window.

3. Go to the terminal we used to run program, stop the StartRad by ctrl + C, then compile and run it again.

```
root@hol6663-1:~/rad/java# ant -e
…
================ HOL6663: RAD Exercises =================
    Exercise 1. Using RAD to Query Cloud
         a) List zones
         b) Show zone's property
    Exercise 2. Using RAD to Manage cloud
         c) Add a remote machine to the cloud
         d) request a zone from the cloud
         e) Install zone
============================================================
Please choose an exercise by input a letter (a/b/c/d/e),
then press the Enter key to confirm :    e
```

Input option *e*, and press *Enter* key. Then the available machines in the virtual environment are listed here, choose the machine by entering a number, for example input *1* to choose hol6663-1.

```
Show zone properties on a specified machine

There are 1 machines in the cloud, please choose one:
  1. hol6663-1
Please input a number and press Enter to confirm (default option is 1) : 1
List zones on local machine:
NAME             STATUS      ZONEPATH         BRAND      IP
  zone1            installed     /system/zones/zone1 solaris    exclusive
  zone2            configured  /system/zones/zone2 solaris    exclusive
  newzone          configured  /system/zones/newzone solaris    exclusive
```

```
Please input the zone name that you want to install and press Enter key
newzone
```

All the zones on hol6663-1 are listed here, input the zone name you want to install. We will install zone *newzone* which is created in Step 3. After you press the *Enter* key, it will begin to install the zone.

It may take several minutes to do installation, leave the terminal alone, you can go ahead to the next exercise. After the zone is installed, it will output the installation status such as *zone is successfully installed*, and list all the zones of the machine. We will check the output when we run Exercise 4.

```
  Newzone is successfully installed!

NAME              STATUS      ZONEPATH          BRAND      IP
zone1             running     /system/zones/zone1 solaris    exclusive
zone2             configured  /system/zones/zone2 solaris    exclusive
newzone           installed   /system/zones/newzone solaris    exclusive
  Continue?(y/n,default is y)
```

Summary

In this exercise, we simulated the scenes of adding a machine to the cloud and requesting a virtual machine from the cloud. When a user asks for a virtual machine with specific requirements, like CPU number and memory size, the cloud will choose an appropriate machine to create a zone for the user. We have learned to use RAD JAVA APIs to implement the backend of the cloud management, which includes getting the machine information, creating and installing zones, and changing zone's properties according to the request.

Exercise 3: Using Rest to Control the Virtual Machines in the Cloud(15 Minutes)

In this exercise, a new REST interface to RAD is introduced, you will explore how to map URIs and HTTP operations to RAD components, and learn how to manage zones by sending HTTP request.

REST

REST stands for Representational State Transfer, which is an architecture style for World Wide Web. It relies on a stateless, client-server, communications and cacheable protocol -- and in virtually all cases, the HTTP protocol is used.

To the extent that systems conform to the constraints of REST they can be called RESTful. RESTful systems typically, but not always, communicate over the Hypertext Transfer Protocol with the same HTTP verbs (GET, POST, PUT, DELETE, etc.) which web browsers use to retrieve web pages and to send data to remote servers.

HTTP-based RESTful APIs are defined with these aspects:

- Each resource is addressable by a URI
- Resources are manipulated via their representations (e.g. XML, HTML, JSON, YAML etc.)
- A uniform interface is exposed for all resources (GET, PUT, POST, DELETE etc. in the case of HTTP)
- hypertext links to reference state
- hypertext links to reference-related resources

RAD REST client in Solaris 11.3

From Oracle Solaris 11.3 onwards, a new REST interface to RAD has been added. REST APIs are an increasingly popular way of interacting with system services across the network over both HTTP and HTTPS using an encoding payload such as JSON or XML.

A RESTful architecture uses resources, identified by a URI, as the primary interface through which to manipulate data, or it uses call methods to operate on that data. Resources can be accessed individually or as a collection of member resources. The RAD REST functionality supports HTTP GET, POST, PUT, and DELETE requests

URI Specification

RAD REST resources are accessed using URIs of the form:

http://<host>:<port>/<base>/{namespace}/[{version}]/{collection}[?query-params]
http://<host>:<port>/<base>/{namespace}/[{version}]/{collection}/{coll-id}[?query-params]
http://<host>:<port>/<base>/{namespace}/[{version}]/{collection}/{coll-id}/{sub-collection}[?query-params]
http://<host>:<port>/<base>/{namespace}/[{version}]/{collection}/{coll-id}/{sub-collection}/{sub-coll-id}[?query-params]

where,

<base> is the base url mapping to identify and route RAD requests.
{namespace} - represents the name associated with a RAD module (the module api name)
{version} - an optional version number that specifies the RAD module version
{collection} - identifies a collection resource (the interface type)

{coll-id} - an identifier or path to an individual resource within a collection
{sub-collection} - a collection nested within an individual resource
{sub-coll-id} - an identifier or path to an individual resource within a subcollection

Step 1: Start a RAD instance running in rad_http protocol

We will use *curl*(1) to send HTTP request. The simplest way to get a usable RAD instance is to run your own daemon and instruct it to run the rad_http protocol.

1. In both hol6663-1 and hol6663- 2, click the right mouse button on the desktop and choose *Open Terminal* to bring up a terminal window if necessary.
2. Run the following script to start a RAD instance running in rad_http protocol on port 6666 on both of the machines.

```
root@hol6663-1:~# cd rad/tools
root@hol6663-1:~/rad/tools# ./http6666.sh
```

```
root@hol6663-2:~# cd rad/tools
root@hol6663-2:~/rad/tools # ./http6666.sh
```

Note: Please leave the terminals along, just minimize the window and be careful not to close them in the following exercises.

You can take a look at the script:

```
root@hol6663-1:~/rad/tools # more http6666.sh
/usr/lib/rad/rad -d -m /usr/lib/rad/module/test/c -m /usr/lib/rad/module -m /usr/lib/rad/protocol -m
/usr/lib/rad/transport -t tcp:port=6666,proto=rad_http,localonly=false
```

-m moduledir : Add *moduledir* to the list of directories to scan and load modules from.

-t transpec : Instantiate a transport specified by transport specification *transpec*. We defined a TCP transport with following options:

proto : The protocol to use with this transport instance, here we use rad_http.
port : The port to listen on for connections, here we use 6666.
localonly : If true, rad will only listen for connections from the local machine, here we set it as false for remote connection.

Step 2: Connect to RAD server

1. Switch to hol6663-1, click the right mouse button on the desktop and choose *Open Terminal* to bring up a terminal window.

2. First, we'll need to set up the RAD connection as before. For this, we'll send a POST request to the */api/com.oracle.solaris.rad.authentication/1.0/Session* API and we'll also provide some credentials included in a JSON file. Create the file *body.json* as following, or you can choose your favorite way to create the file with the bold content.

```
root@hol6663-1:~/# cd rad
root@hol6663-1:~/rad# vi body.json
{
      "username": "root",
      "password": "solaris11",
      "scheme": "pam",
      "preserve": true,
      "timeout": -1
```

```
}
```

It provides both username and password that we used to authenticate by PAM over a UNIX socket located at*/system/volatile/rad/radsocket-http*. And mentioned that the cookie for connection should preserved for reconnection, and timeout argument is *-1*which means using a default timeout of 60 minutes.

3. We can make the *POST* request using *curl*, saving the authentication token to a file called *cookie.txt*:

-H: Extra header to include in the request when sending HTTP to a server.

--data-binary: Send the specified data (purely binary) in a POST request to the HTTP server, here we send the json file *body.json*.

-b: Pass the data to the HTTP server as a cookie.

-c: Specify to which file you want curl to write all cookies after a completed operation.

```
root@hol6663-1:~/rad# curl -H "Content-type: application/json" -X POST --data-binary @body.json \
http://hol6663-1:6666/api/com.oracle.solaris.rad.authentication/1.0/Session \
 -v -c cookie.txt -b cookie.txt

*   Trying 192.168.100.1...
* Failed to set TCP_KEEPALIVE on fd 4
* Connected to hol6663-1 (192.168.100.1) port 6666 (#0)
> POST /api/com.oracle.solaris.rad.authentication/1.0/Session HTTP/1.1
> User-Agent: curl/7.40.0
> Host: hol6663-1:6666
> Accept: */*
> Content-type: application/json
> Content-Length: 129
>
* upload completely sent off: 129 out of 129 bytes
< HTTP/1.1 201 Created
< Connection: Keep-Alive
< Content-Length: 164
< Expires: 0
< Pragma: no-cache
< Cache-Control: no-cache, no-store, must-revalidate
< Location: /api/com.oracle.solaris.rad.authentication/1.0/Session/_rad_reference/2048
* Added cookie _rad_instance="2048" for domain hol6663-1, path /api, expire 1445345212
< Set-Cookie: _rad_instance=2048; Path=/api; Max-Age=3600; HttpOnly
* Added cookie _rad_token="40f4c5ce-5b5c-48a9-9444-cc7df0af2de1" for domain hol6663-1, path /api,
expire 1445345212
< Set-Cookie: _rad_token=40f4c5ce-5b5c-48a9-9444-cc7df0af2de1; Path=/api; Max-Age=3600; HttpOnly
< Date: Tue, 20 Sep 2015 11:46:52 GMT
<
{
        "status": "success",
        "payload": {
                "href": "/api/com.oracle.solaris.rad.authentication/1.0/Session/_rad_reference/2048"
        }
* Connection #0 to host hol6663-1 left intact
```

4. Check the token we saved in *cook.txt*.

```
root@hol6663-1:~/rad# cat cookie.txt
```

Shown from the output, we have successfully obtained the token.

```
# Netscape HTTP Cookie File
# http://curl.haxx.se/docs/http-cookies.html
# This file was generated by libcurl! Edit at your own risk.

#HttpOnly_hol6663-1      FALSE  /api   FALSE  1445345212    _rad_instance2048
#HttpOnly_hol6663-1      FALSE  /api   FALSE  1445345212    _rad_token    40f4c5ce-5b5c-48a9-9444-
cc7df0af2de1
```

Step 3: Query and Manage Zones by REST

1. We can use the token we got in step 2 to communicate with RAD server. Let's get the zones list first, as what we do in Exercise 1.

```
root@hol6663-1:~/rad# curl -H "Content-type: application/json" \
-X GET http://hol6663-1:6666/api/com.oracle.solaris.rad.zonemgr/1.0/Zone -b cookie.txt
```

Here's the response:

```
{
        "status": "success",
        "payload": [
                {
                        "href": "api/com.oracle.solaris.rad.zonemgr/1.2/Zone/zone1"
                },
                {
                        "href": "api/com.oracle.solaris.rad.zonemgr/1.2/Zone/zone2"
                },
                {
                        "href": "api/com.oracle.solaris.rad.zonemgr/1.2/Zone/newzone"
                }
        ]
}
```

The resulting payload returned is very similar to the ADR name we got in Exercise 1. However, instead of returning a list of ADR names for the zone instances, we get back a URI. We can now use these URIs to reference individual zones.

2. Then we want to get zone properties of a specified zone, for example *newzone* which is created in Exercise 2, so the URI for *newzone* should be *http://192.168.100.1:6666/api/com.oracle.solaris.rad.zonemgr/1.2/Zone/newzone* .

Create null.json with no json object contained in the file, or you can choose your favorite way to create it with following bold contents.

```
root@hol6663-1:~/rad# vi null.json
```

```
{
}
```

3. To look at more properties of the zone, we will need to call a method on this interface. This is not considered a traditional RESTful operation, but many REST-based APIs use method invocation. To use method calls, we will need to include *_rad_method* within the URI to distinguish between the RAD instance and the method name.

Here null.json is used as the file for input arguments, which means we have no filter and scope request for the resource. So the command will return all the properties of the zone.

```
root@hol6663-1:~/rad# curl -H "Content-type: application/json" -X PUT --data @null.json \
http://hol6663-1:6666/api/com.oracle.solaris.rad.zonemgr/1.2/Zone/newzone/_rad_method/getResources
 -b cookie.txt

{
        "status": "success",
        "payload": [
                {
                        "type": "global",
                        "properties": [
                                {
                                        "name": "zonename",
                                        "value": "newzone",
                                        "type": "PROP_SIMPLE",
```

```
                                "listvalue": null,
                                "complexvalue": null
                        },
                        {
                                "name": "zonepath",
                                "value": "/system/zones/newzone",
                                "type": "PROP_SIMPLE",
                                "listvalue": null,
                                "complexvalue": null
                        },
                        {
                                "name": "brand",
                                "value": "solaris",
                                "type": "PROP_SIMPLE",
                                "listvalue": null,
                                "complexvalue": null
                        },
                        {
                                "name": "autoboot",
                                "value": "false",
                                "type": "PROP_SIMPLE",
                                "listvalue": null,
                                "complexvalue": null
                        },
                        …
                        {
                                "name": "ip-type",
                                "value": "exclusive",
                                "type": "PROP_SIMPLE",
                                "listvalue": null,
                                "complexvalue": null
                        },
                        …
                ],
                "parent": null
        },
        {
                "type": "anet",
                "properties": [
                        {
                                "name": "linkname",
                                "value": "net0",
                                "type": "PROP_SIMPLE",
                                "listvalue": null,
                                "complexvalue": null
                        },
                        …
                ],
                "parent": null
        },
        {
                "type": "capped-memory",
                "properties": [
                        {
                                "name": "physical",
                                "value": "1073741824",
                                "type": "PROP_SIMPLE",
                                "listvalue": null,
                                "complexvalue": null
                        }
                ],
                "parent": null
        },
        {
                "type": "dedicated-cpu",
```

```
                        "properties": [
                                {
                                        "name": "ncpus",
                                        "value": "1",
                                        "type": "PROP_SIMPLE",
                                        "listvalue": null,
                                        "complexvalue": null
                                },
                                …
                        ],
                        "parent": null
                }
        ]
}
```

4. Now let's boot the zone, as *newzone* may be still in installation, we boot *zone1* instead. It may take some time, as the installation of *newzone* is running at the same time.

```
root@hol6663-1:~/rad# curl -H "Content-type: application/json" -X PUT --data @null.json
 http://hol6663-1:6666/api/com.oracle.solaris.rad.zonemgr/1.2/Zone/zone1/_rad_method/boot  -b
 cookie.txt

{
        "status": "success",
        "payload": {
                "code": "NONE",
                "str": "",
                "stdout": null,
                "stderr": null
        }
}
```

Summary

In this exercise, we learned about the latest REST interface to RAD in Solaris11, and explored how HTTP operations to RAD components are mapped to URI. Then we learned how to use command curl to send simple HTTP requests to query and manage zones.

Exercise 4: Using Rest-based web application to Modify the Virtual Machines in the Cloud (10 Minutes)

We have learned how to use command *curl* to send HTTP request to manage zone in Exercise 3. Actually REST is widely used in web services. In this lab we will use Tomcat to setup a simple web application to use RAD RESTful APIs to modify zones' properties.

Step 1: Setup the Web Application

We have made a JAVA web application named *RestWebApp* on hol6663-1.

```
root@hol6663-1: ~/rad# ls
body.json      java            null.json      tools
cookie.txt     lib             RestfulWebApp
```

All the webpages are jsp based, and java servlet is used to add dynamic content to the web server

```
root@hol6663-1: ~/rad# cd RestWebApp
root@hol6663-1:~/rad/RestWebApp# ls
build.xml  src        WebRoot

root@hol6663-1:~/rad/RestfulWebApp# ls WebRoot/
cpuProperty.jsp  index.jsp        restful          WEB-INF

root@hol6663-1:~/rad/RestfulWebApp# ls -R src
src:
oow2015

src/oow2015:
hol6663

src/oow2015/hol6663:
ChangeCPUPropServlet.java  StartRadServlet.java
rest                       utility

src/oow2015/hol6663/rest:
ParseJSONInput.java   RestHttpClient.java   ZoneAdminByRest.java

src/oow2015/hol6663/utility:
JSONHelper.java  Server.java
```

The source codes of communicating with the RAD server by Restful API are located in *src/oow2015/hol6663/rest*. The file *RestHttpClient.java* maintains the http connection and request with RAD server, while the file *ZoneAdminByRest.java* contains all the zone operations used in the application.

1. We choose package *httpclient* to manage the HTTP transport. All the resources are formatted by JSON.

The HTTP transport between the control server and the RAD server are implemented in the *RestHttpClient.java* in */root/rad/RestWebApp/src/rest*. You can take a look at the source code.

```
root@hol6663-1:~/rad/RestfulWebApp# gedit src/oow2015/hol6663/rest/RestHttpClient.java
```

Here we only list the method *authentication()* which is used to do connection. We send httpPOST request to */api/com.oracle.solaris.rad.authentication/1.0/Session,* as what we did in Exercise 3 by command line.

```
    public boolean authentication(String loginFile) {
```

```
        // HTTP opertions are mapped to URIs
        // The URI for authentication is
        // http://httpMachine:httpPort/api/com.oracle.solaris.rad.authentication/1.0/Session
        String uri = getURI("/api/com.oracle.solaris.rad.authentication/1.0/Session", "");
        HttpPost httpPost = new HttpPost(uri);  // create a POST request
        httpPost.addHeader("Content-type", "application/json");

        String loginInfo = getEntity(loginFile); // get login info by file
        if (loginInfo != null) {
            CloseableHttpResponse res = null;
            try {
                httpPost.setEntity(new StringEntity(loginInfo));
                // Send the HTTP POST request with login infomation
                res = httpClient.execute(httpPost);
            } catch (Exception e) {
                System.out.println("  Failed in HTTP Authentication by file : " + loginFile);
                return false;
            } finally {
                close(res);
            }
            return true;
        }

        System.out.println("  Failed in Authentication by file : " + loginFile);
        return false;
    }
```

And the credential information is input by the *loginFile* which would be *body.json* under the */root/rad/RestWebApp/WebRoot/restful* directory.

Click the right mouse button on the desktop and choose *Open Terminal* to bring up a terminal window

```
root@hol6663-1:~# cd rad/RestfulWebApp
root@hol6663-1:~/rad/RestfulWebApp# more ./WebRoot/restful/body.json
{
      "username": "root",
      "password": "solaris11",
      "scheme": "pam",
      "preserve": true,
      "timeout": -1
}
```

2. The web application is used to modify the zone's dedicated CPUs, open the file *ZoneAdminByRest.java* in gedit window for detail information.

If the zone has been set with dedicated CPU, we made a method *setProp* to modify the exist property.

```
public void setProp(String zonename, String propFile)
            throws Exception {
        String apiAddr = getZoneAPIAddress(zonename);
        String prop = "{}";
        JSONObject obj = (JSONObject) JSONValue.parse(prop);
        try {
            restClient.sendPut(apiAddr, "/_rad_method/editConfig", obj, false);
            restClient.sendPut(apiAddr, "/_rad_method/setResourceProperties", propFile, true);
            restClient.sendPut(apiAddr, "/_rad_method/commitConfig", obj, false);
        } catch (Exception e) {
            System.out.println("  Set resource property failed for " + propFile);
        }
    }
```

To change the CPU number of a zone, the input parameters for *setResourceProperties* may be constructed in the following style.

```
{
```

```
    "filter": {
          "type": "dedicated-cpu"
    },
    "properties": [
       {
          "name" : "cpus",
          "value": "1"
        }
    ]
}
```

If the zone has not been set with dedicated CPU, we made a method *addResouce* to add property to zones.

```
public void addResource(String zonename, String resFile)
            throws Exception {
        String apiAddr = getZoneAPIAddress(zonename);
        String prop = "{}";
        JSONObject obj = (JSONObject) JSONValue.parse(prop);
        try {
            restClient.sendPut(apiAddr, "/_rad_method/editConfig", obj, false);
            restClient.sendPut(apiAddr, "/_rad_method/addResource", resFile, true);
            restClient.sendPut(apiAddr, "/_rad_method/commitConfig", obj, false);
        } catch (Exception e) {
            System.out.println("  Add resource failed for " + resFile);
        }
}
```

And the input parameters for *addResource* may be constructed in the following style.

```
{
 "resource": {
    "type": "dedicated-cpu",
    "properties": [
                          {
                "name" : "cpus",
                "value": "1"
          }
    ]
 }
}
```

3. Use ant to compile the application *RestfulWebApp*. The compiled war file will be copied to tomcat.

```
root@hol6663-1:~/rad/RestWebApp# ant
```

Apache-Tomcat 8 is already put under */root/rad* , check if the application is setup on webapps now.

```
root@hol6663-1:~/rad/RestWebApp# cd ../tools/apache-tomcat-8.0.27
root@hol6663-1:~/rad/tools/apache-tomcat-8.0.27# ls webapp
RestfulWebApp.war
```

4. Startup the tomcat

```
root@hol6663-1:~/rad/apache-tomcat-8.0.27# ./bin/startup.sh
```

Step 2: Run the Web Application to change property

1. Click the Firefox button on the top toolbar of hol6663-1, and input the URL http://hol6663-1:8080/RestfulWebApp

All the Zones within the cloud are listed in a table. We also show the machine name, zone status, and the dedicated CPU property of zones. Seen from figure 6, *zone1* is boot in Exercise 3, and *newzone* is successfully installed now. Most of the zones don't have dedicated CPUs except *newzone* which is created in Exercise 2.

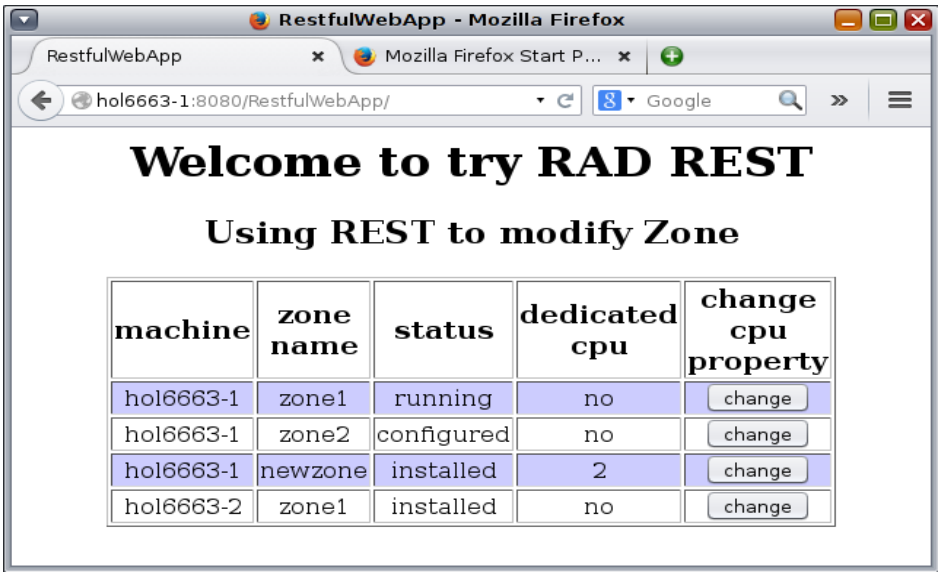

Figure 6. List zones in rest web application

Of course, you can run the browser on any machine in the same local net.

2. Click the button *change* for *zone1* of *hol6663-1*, go to a new page which contains all the properties of the selected zone. As shown in figure 7, there's no dedicated-cpu assigned to zone1.

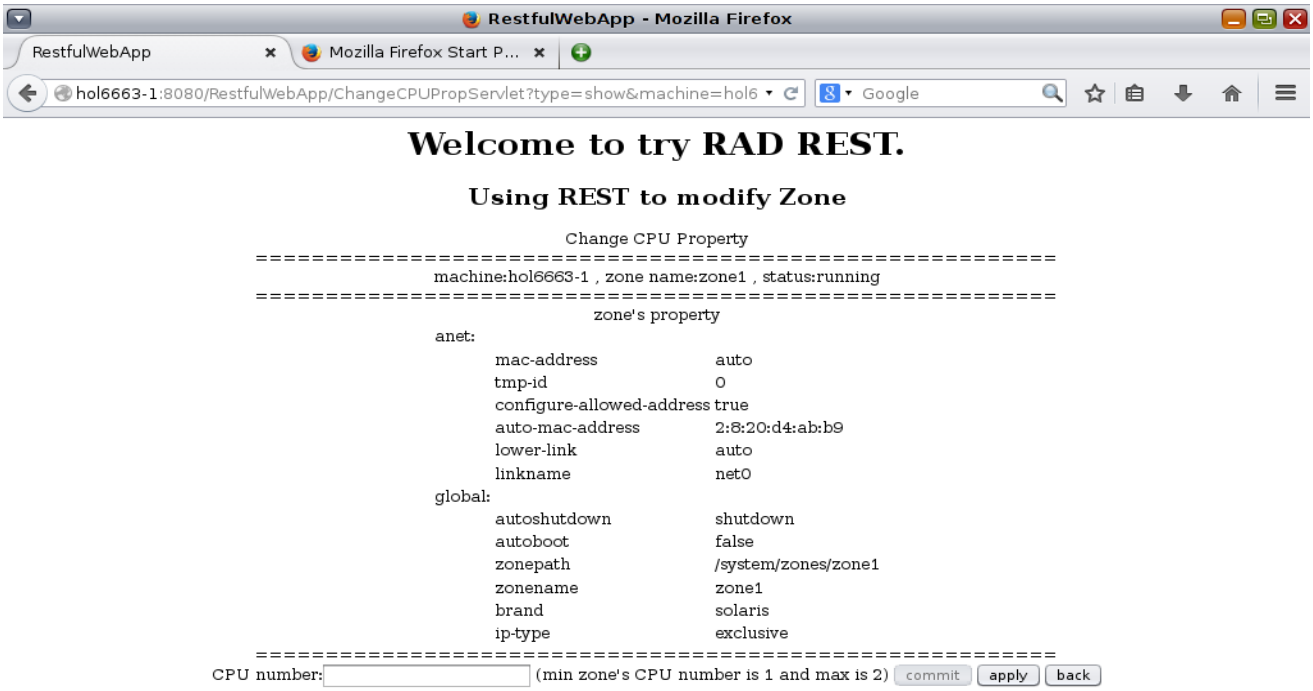

Figure 7. Show property in rest web application

3. In the bottom of the webpage, there's a line starting with "CPU number". Input *1* in the textbox and press the button *commit* to add dedicated CPU to zone1.

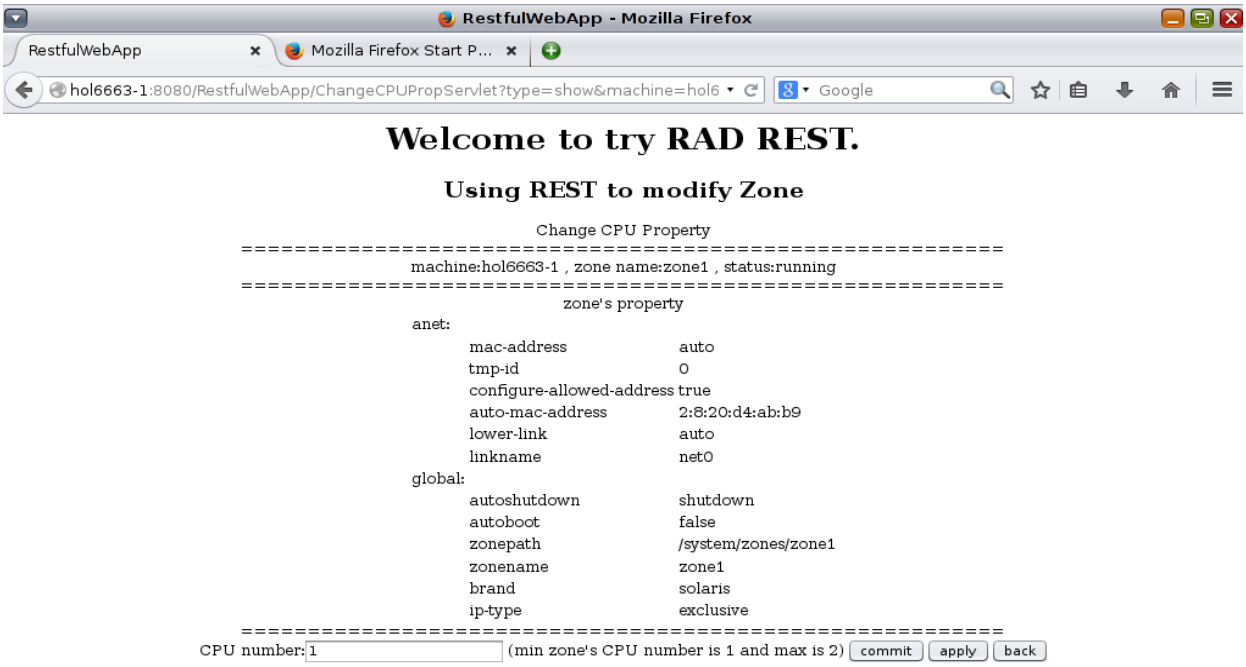

Figure 8. Input CPU number for selected zone1

After the property is added to zone1, the page is updated. In figure 9, we can find that 1 CPU is assigned to zone1 now.

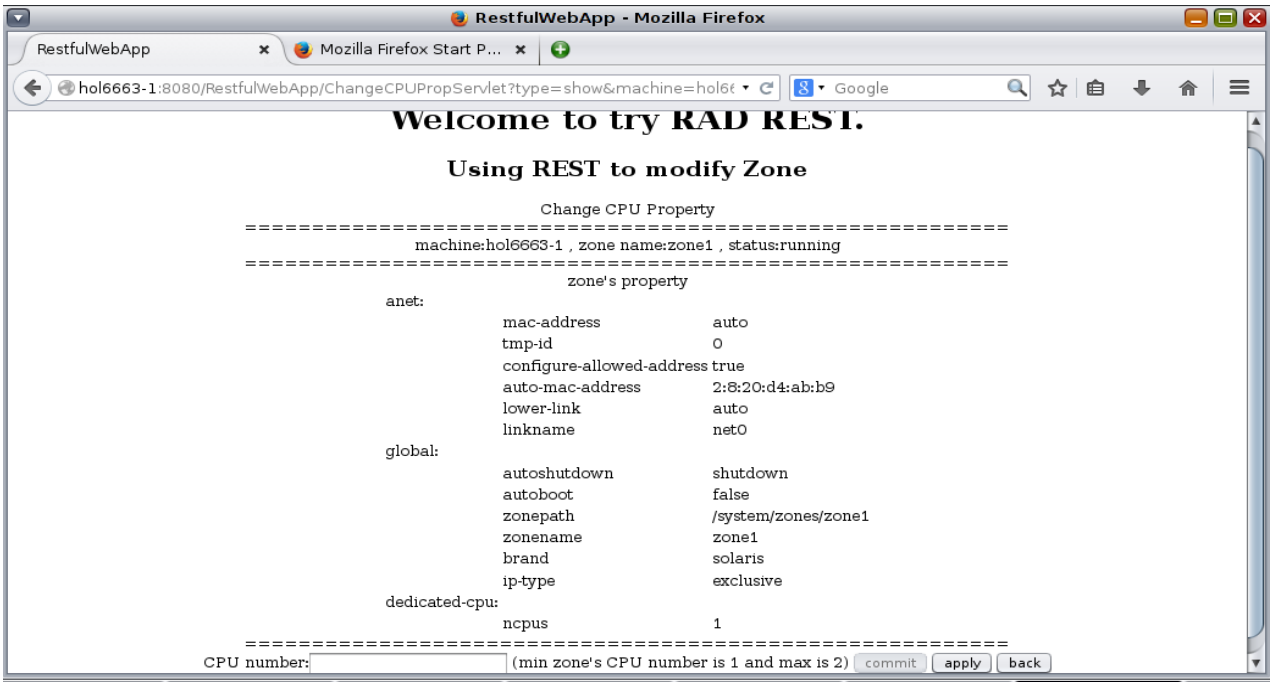

Figure 9. Updated zone properties

Step 3: Apply the zone modification immediately

1. We have modified the zone property of dedicated CPU in Step 2, but the modification will not affect the running zone.

Go to the gedit window. In the file *ZoneAdminByRest.java* , you can find a method *apply* which allows us to apply configuration immediately without rebooting the zone.

```
    public void apply(String zonename) throws Exception {
        String apiAddr = getZoneAPIAddress(zonename);
        String prop = "{}";
        JSONObject obj = (JSONObject) JSONValue.parse(prop);
        try {
            restClient.sendPut(apiAddr, "/_rad_method/apply", obj, false);
        } catch (Exception e) {
            System.out.println("  Apply zone configuration to zone failed  ");
        }
    }
```

2. Login to zone1 to check the CPU number of zone1.

Click the right mouse button on the hol6663-1 and choose *Open Terminal* to bring up a terminal window.

Login to zone1, and input *psrinfo* to get the CPU information of *zone1*. There are 2 CPUs available for *zone1*, because if we don't assign dedicated CPU the zone, it will share all the available CPUs on the machine.

```
root@hol6663-1:~# zlogin zone1
[Connected to zone 'zone1' pts/7]
Last login: Mon Oct 19 08:58:25 2015 on pts/6
Oracle Corporation      SunOS 5.11    11.3   June 2015
root@zone1:~# psrinfo
0    on-line   since 09/16/2015 11:25:09
1    on-line   since 09/16/2015 11:25:09
```

In another terminal window, run *poolstat* to check how many CPUs available on the machine. We can find all the CPUs contains in *pool_default*.

```
root@hol6663-1:~/rad# poolstat
                             pset
 id pool                 size used load
  0 pool_default            2 0.00 0.59
```

3. Back to the Firefox browser, there's another button *apply* near the button *commit*. Click the button *apply* to apply the modification to the running zone1. Then you will get a popup window that indicates the modification has been applied to the zone.

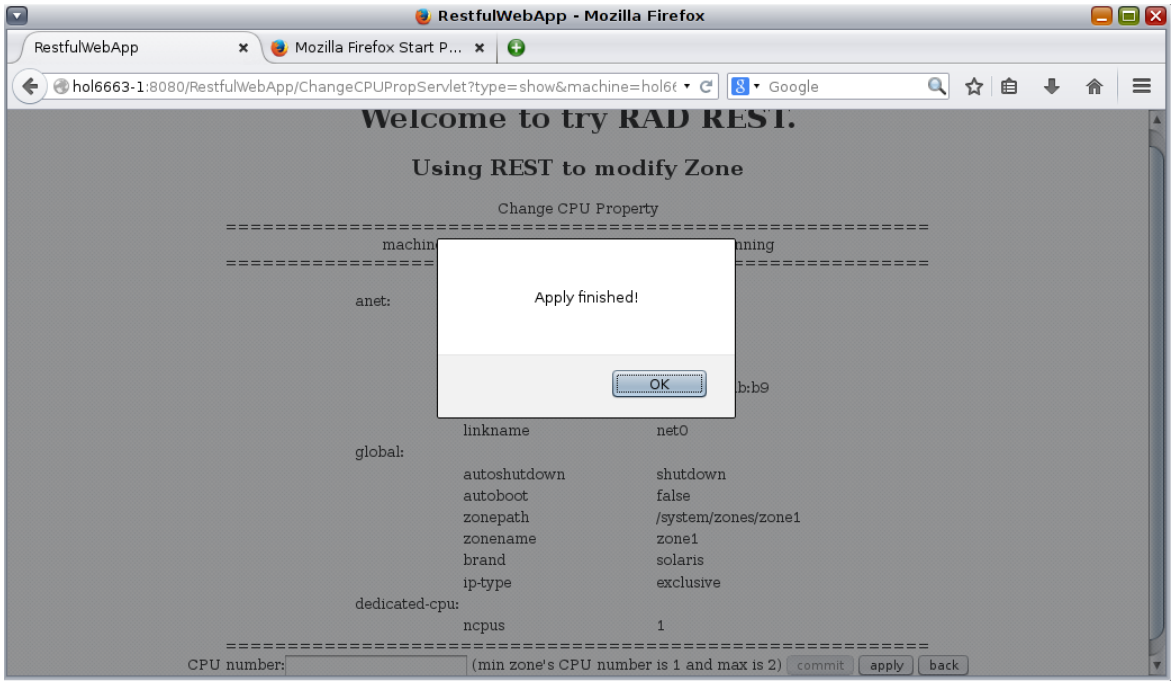

Figure 10. Apply finished

4. Check the CPU number of zone1 again

Go to the terminal window where you login in zone1, run *psrinfo* again. You will find zone1 has only 1 CPU now, but this cpu is dedicated to zone1 and will not be shared with other zones.

```
root@zone1:~# psrinfo
0    on-line   since 09/16/2015 11:25:09
```

In another terminal window, run *poolstat* to check how many cpus available on the machine. You can find there's only 3 cpus in pool_default and 1 cpu is assigned to zone1.

```
root@hol6663-1:~/rad# poolstat
                           pset
 id pool                 size used load
  4 SUNWtmp_zone1           1 0.00 0.00
  0 pool_default            1 0.00 0.08
```

Summary

In this exercise, based on Tomcat, we setup a simple web application, which can be used to change the property of zones. We can add property or modify the property of the zone. The modified configuration can be applied to the running zone immediately without rebooting it.

The REST APIs of RAD provide us a more popular way to interact with system services across the network over HTTP. By using these APIs, You can manage the system by HTTP operations, and all the resources and methods are identified by a URI.

Appendix A: Creating the Java code for RAD Client Exercise "StartRad"

Use the procedures in this appendix to create the directory structure, the .java files, and the ant build file used in the lab exercises 1 and exercises 2.

Note: Directory names and file names are case-sensitive.

Create the Directory Structure

1. All the exercises are located in the *rad* directory, which include the source code of RAD java client exercises and REST-based web application, the common libraries and tools used in test. In this lab, the *rad* directory is created under */root*. Of course, you can choose the directory by yourself.

```
root@hol6663-1:~ # mkdir -p /root/rad/tools
root@hol6663-1:~ # mkdir -p /root/rad/lib

root@hol6663-1:~ # ls /root/rad
lib            tools
```

2. The building tools are put in */root/rad/tools*, which include apache-ant, apache-tomcat, and the script to start a RAD instance running in rad_http protocol. You can download these tools by following links

   Apache-ant 1.9.6 : http://ant.apache.org/bindownload.cgi
   Apache-tomcat 8.0: http://tomcat.apache.org/download-80.cgi
   http666.sh is a script to start RAD instance and you can find the source code in Exercise 3.

```
root@hol6663-1:~ # cd /root/rad/tools
root@hol6663-1:~/rad/tools# ls
apache-ant-1.9.6      apache-tomcat-8.0.27  http6666.sh
root@hol6663-1:~/rad/tools# vi http666.sh
/usr/lib/rad/rad -d -m /usr/lib/rad/module/test/c -m /usr/lib/rad/module -m /usr/lib/rad/protocol -
m /usr/lib/rad/transport -t tcp:port=6666,proto=rad_http,localonly=false
root@hol6663-1:~/rad/tools# chmod 777 http666.sh
```

3. The common libraries are located in */root/rad/lib.*

   commons-logging-1.2.jar :http://mvnrepository.com/artifact/commons-logging/commons-logging/1.2
   httpcore-4.4.1.jar :http://mvnrepository.com/artifact/org.apache.httpcomponents/httpcore/4.4.1
   httpclient-4.5.jar :http://mvnrepository.com/artifact/org.apache.httpcomponents/httpclient/4.5
   json-simple-1.1.1.jar :http://mvnrepository.com/artifact/com.googlecode.json-simple/json-simple/1.1.1
   jstl-1.2.jar :http://www.java2s.com/Code/Jar/j/Downloadjstl12jar.htm

```
root@hol6663-1:~/rad/lib# ls
commons-logging-1.2.jar  httpcore-4.4.1.jar  jstl-1.2.jar httpclient-4.5.jar  json-simple-1.1.1.jar
```

4. Create the directories of source code. The source codes of RAD java client are created under */root/rad/java/,* while the source codes of REST-based web application are created under */root/rad/RestfulWebApp*

```
root@hol6663-1:~ # mkdir -p /root/rad/java
root@hol6663-1:~ # mkdir -p /root/rad/RestfulWebApp

root@hol6663-1:~ # ls /root/rad
java           lib            RestfulWebApp  tools
```

5. Create the directories for remote machine hol6663-2.

```
root@hol6663-2:~ # mkdir -p /root/rad/tools
```

```
root@hol6663-2:~ # cd /root/rad/tools
root@hol6663-2:~/rad/tools# vi http666.sh
/usr/lib/rad/rad -d -m /usr/lib/rad/module/test/c -m /usr/lib/rad/module -m /usr/lib/rad/protocol -
m /usr/lib/rad/transport -t tcp:port=6666,proto=rad_http,localonly=false
root@hol6663-2:~/rad/tools# chmod 777 http666.sh
```

Create the Build Script file and Java Files

1. The directory structure of StartRad is shown in figure 11.

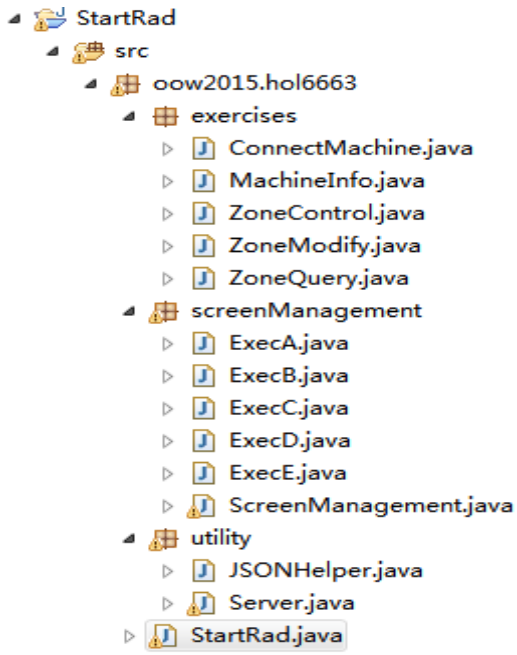

Figure 11: directory structure of RAD java client

2. The RAD java exercise "StartRad" is located under */root/rad/java/*. Create the ant build file *build.xml as following.*

```
<?xml version="1.0"?>
<project name="StartRad" default="run" basedir="./">
<property environment="env" />
<property name="radlib.dir" value="/usr/lib/rad/java" />
<property name="lib.dir" value="../" />
   <path id="project.classpath">
    <pathelement path="${basedir}/build/classes"/>
         <fileset dir="${radlib.dir}">
               <include name="**/*.jar" />
         </fileset>
         <fileset dir="${lib.dir}">
               <include name="**/*.jar" />
         </fileset>
   </path>
 <target name="run">
   <delete dir="${basedir}/build"  quiet="true"/>
   <mkdir dir="${basedir}/build/"/>
   <mkdir dir="${basedir}/build/classes"/>
   <javac destdir="${basedir}/build/classes" debug="false" verbose="no"  includeantruntime="true"
          encoding="UTF-8"  failonerror="true" fork="true">
          <src path="${basedir}" />
          <include name="**/*.java" />
```

```
            <include name="**/*.json" />
                 <exclude name="**/exercises.fullsrc/*.java"/>
            <classpath refid="project.classpath" />
   </javac>
 <java classname="oow2015.hol6663.StartRad">
             <classpath refid="project.classpath" />
</java>
</target>
</project>
```

Listing 1. Code for the */root/rad/java/build.xml* **file**

3. Create the java file StartRad.java for the main class in */root/rad/java/oow2015/hol6663*

Copy the code shown in Listing 2 and save it in a plain-text ASCII file named *StartRad.java*

```
package oow2015.hol6663;

import java.util.*;
import oow2015.hol6663.exercises.*;
import oow2015.hol6663.screenManagement.*;
import oow2015.hol6663.utility.Server;

public class StartRad {

    public static void main(String[] args) {
        try {
            ScreenManagement scManagement = new ScreenManagement();
            while (true) {
                String orderNumber = scManagement.printOptionsAndGetUserInput();
                switch (orderNumber) {
                    case "a":// list zones
                        List<String> ips = Server.getRemoteServerIPList();
                        ExecA.listZones(ips);
                        break;
                    case "b":// show zone's properties
                        List<String> machines = scManagement.getMachineOption(false);
                        while (true) {
                            System.out.println("  Show zone properties on a specified machine");
                            if (ExecA.listZones(machines)) {
                                break;// get zone list success!
                            }
                        }
                        Map<String, String> inputs =
scManagement.getShowZonePropertyInput(machines);
                        ExecB.showZoneProperty(inputs.get("ip"), inputs.get("zoneName"));
                        break;
                    case "c":
                        String machine = scManagement.AddServer();
                        boolean result = false;
                        if (machine != null) {
                            result = ExecC.addMachine(machine);
                        }
                        if (result) {
                            System.out.println("  Add " + machine + " sucessfully !");
                        }

                        int machineNum = Server.getServerList().size();
                        System.out.format("  There are %d machines in the cloud :\n", machineNum);
                        for (int i = 0; i < machineNum; i++) { //print ip list
                            System.out.format("    %d. %s\n", i + 1,
Server.getServerList().get(i).getServerName());
                        }
```

```
                        break;
                    case "d":// create zone
                        Map<String, Object> connAndZoneInfo = scManagement.getCreateZoneInfo();
                        if( connAndZoneInfo == null )
                            break;
                        ExecD.createZone(connAndZoneInfo);
                        break;
                    case "e":// install zone
                        List<String> servers = scManagement.getMachineOption(false);
                        while (true) {
                            System.out.println("  Install a zone on a specified machine");
                            if (ExecA.listZones(servers)) { break; }
                        }
                        Map<String, String> zone = scManagement.getInstallZoneInfo(servers);
                        ExecE exec22 = new ExecE();
                        exec22.installZone(zone.get("ip"), zone.get("zoneName"));
                        break;
                    default:
                        System.out.println("  Sorry ,invalide input,try again!");
                }
                System.out.println("  Continue?(y/n,default is y)");
                String flag = System.console().readLine().trim();
                if (flag.length() == 0 || flag.equals("y")) {
                    continue;
                } else {
                    break;
                }
            }
        } catch (Exception e) {
            System.out.println("System error!");
        }
    }
}
```

Listing 2. Code for the */root/rad/java/ oow2015/hol6663/StartRad.java* **file**

4. Create the java files for the "exercises" in */root/rad/java/oow2015/hol6663*

Copy the code shown in Listing 3 and save it in a plain-text ASCII file named *ConnectMachine.java*

```
package oow2015.hol6663.exercises;

import java.util.*;
import java.io.*;
// Import RAD packages here
import com.oracle.solaris.rad.client.*;
import com.oracle.solaris.rad.connect.*;

public class ConnectMachine {

    private Connection con = null;
    /**
     * Connect to local server
     *
     * @return Connection
     * @throws IOException
     */
    public Connection localConnect() throws IOException {
        //============================================================//
        // Please input the Source code for local connection here     //
        //============================================================//

        //============================================================//
        // Input end                                                  //
```

```
        //============================================================//
        return con;
    }

    /**
     * Connect to remote server
     *
     * @param hostIP remote server IP
     * @return Connection
     * @throws IOException
     */
    public Connection tlsConnect(String hostIP) throws IOException {
        //============================================================//
        // Please input the Source code for remote connection here    //
        //============================================================//

        //============================================================//
        // Input end                                                  //
        //============================================================//
        return con;
    }

    /**
     * Close the connection
     */
    public void closeConnect() {
        if (con != null) {
            try {
                con.close();
            } catch (IOException e) {
                // TODO Auto-generated catch block
               // e.printStackTrace();
            }
            con = null;
        }
    }
}
```

Listing 3. Code for the */root/rad/java/ oow2015/hol6663/exercises/ ConnectMachine.java* **file**

Copy the code shown in Listing 4 and save it in a plain-text ASCII file named *MachineInfo.java*

```
package oow2015.hol6663.exercises;

import java.io.*;
import com.oracle.solaris.rad.client.*;
import com.oracle.solaris.rad.connect.*;
import com.oracle.solaris.rad.kstat.*;

public class MachineInfo {
    /**
     * This function return the CPU number of the connected machine.
     *
     * @param con Connection RAD Connection to machine
     * @return int CPU number
     * @throws IOException
     */
    public static int getMachineCPU(Connection con) throws IOException {
        int cpuNum = 0;
        //============================================//
        // Please input the Source code here          //
        //============================================//

         //============================================//
        // Input end                                  //
        //============================================//
```

```
        return cpuNum;
    }

    /**
     * This function return the physical memory size of the connected machine.
     *
     * @param con Connection RAD Connection to machine
     * @return int memory size in MB
     * @throws IOException
     */
    public static int getMachineMem(Connection con) throws IOException {
        int mem = 0;
        //============================================//
        // Please input the Source code here          //
        //============================================//

        //============================================//
        // Input end                                  //
        //============================================//
        return mem;
    }
}
```

Listing 4. Code for the */root/rad/java/ oow2015/hol6663/exercises/ MachineInfo.java* **file**

Copy the code shown in Listing 5 and save it in a plain-text ASCII file named *ZoneControl.java*

```
package oow2015.hol6663.exercises;

import java.io.*;
import com.oracle.solaris.rad.client.*;
import com.oracle.solaris.rad.connect.*;
import com.oracle.solaris.rad.zonemgr.*;

public class ZoneControl {
    /**
     * This function creates a new zone by conn connection
     *
     * @param con Connection RAD Connection to machine
     * @param zonename String zone name
     * @return boolean if zone creating is successfully completed, return true, else return false
     * @throws IOException
     */
    public static boolean create(Connection con, String zonename) throws IOException {
        //============================================//
        // Please input the Source code here          //
        //============================================//

        //============================================//
        // Input end                                  //
        //============================================//
        return false;
    }

    /**
     * This function installs the zone by conn connection
     *
     * @param con Connection RAD Connection to machine
     * @param zonename String zone name
     * @return boolean if zone installation is successfully completed, return true, else return
false
     * @throws IOException
     */
    public static boolean install(Connection con, String zonename)  throws IOException {
        //============================================//
        // Please input the Source code here          //
```

```
        //============================================//

        //============================================//
        // Input end                                  //
        //============================================//
        return false;
    }

    /**
     * This function boots zone by conn connection
     *
     * @param con Connection RAD Connection to machine
     * @param zonename String zone name
     * @return boolean if zone booting is successfully completed, return true, else return false
     * @throws IOException
     */
    public static boolean boot(Connection con, String zonename)   throws IOException {
        Zone zone = ZoneQuery.getZone(con, zonename);
        if (zone != null) {
            try {
                zone.boot(null);
                return true;
            } catch (RadObjectException oe) {
                //RAD Error handling
                Result res = (Result) oe.getPayload();
                System.out.println(res.getCode());
                if (res.getStderr() != null) {
                    System.out.println(res.getStderr());
                }
            }
        }
        System.out.println("Can't find zone " + zonename);
        return false;
    }
}
```

Listing 5. Code for the */root/rad/java/ oow2015/hol6663/exercises/ ZoneControl.java* **file**

Copy the code shown in Listing 6 and save it in a plain-text ASCII file named *ZoneModify.java*

```
package oow2015.hol6663.exercises;

import java.util.*;
import java.io.*;
import com.oracle.solaris.rad.client.*;
import com.oracle.solaris.rad.connect.*;
import com.oracle.solaris.rad.zonemgr.*;

public class ZoneModify {
    /**
     * This function updates the property of specified zone
     *
     * @param con Connection RAD Connection to machine
     * @param zonename String zone name
     * @param type String resource type
     * @param props List<Property> updated properties
     * @throws IOException
     */
    public static void setZoneProp(Connection con, String zonename, String type, List<Property>
props)
            throws IOException {
        Zone zone = ZoneQuery.getZone(con, zonename); // Get Zone object by zonename
        if (zone != null) {
            Resource resource = new Resource(type, null, null); // Get resource by resource type
            try {
                // Change the property of the resource
```

```
                // We need to call editConfig before changing zone configuration
                // And call commitConfig after all the changes are finished
                zone.editConfig(null);
                zone.setResourceProperties(resource, props);
                zone.commitConfig();
            } catch (RadObjectException oe) {
                //RAD Error handling
                Result res = (Result) oe.getPayload();
                System.out.println(res.getCode());
                if (res.getStderr() != null) {
                    System.out.println(res.getStderr());
                }
            }
        }
    }

    /**
     * This function adds new resource to zone's configuration
     *
     * @param con Connection RAD Connection to machine
     * @param zonename String zone name
     * @param type String resource type
     * @param props List<Property> detail properties which will be added to the zone
     * @throws IOException
     */
    public static void addZoneProp(Connection con, String zonename, String type, List<Property>
props)
            throws IOException {
        Zone zone = ZoneQuery.getZone(con, zonename); // Get Zone object by zonename
        if (zone != null) {
            //============================================//
            // Please input the Source code here          //
            //============================================//

            //============================================//
            // Input end                                  //
            //============================================//
        }
    }

    /**
     * This function adds memory limitation to zone's configuration
     *
     * @param con Connection RAD Connection to machine
     * @param zonename String zone name
     * @param memory String memory size
     * @throws IOException
     */
    public static void addZoneMemoryProp(Connection con, String zonename, String memory)
            throws IOException {
        //============================================//
        // Please input the Source code here          //
        //============================================//

        //============================================//
        // Input end                                  //
        //============================================//
    }

    /**
     * This function adds dedicated cpus to zone's configuration
     *
     * @param con Connection RAD Connection to machine
     * @param zonename String zone name
     * @param cpuNumber String CPUs number
```

```
     * @throws IOException
     */
    public static void addZoneCPUProp(Connection con, String zonename, String cpuNumber)
            throws IOException {
        //============================================//
        // Please input the Source code here          //
        //============================================//

        //============================================//
        // Input end                                  //
        //============================================//
    }
}
```

Listing 6. Code for the */root/rad/java/ oow2015/hol6663/exercises/ ZoneModify.java* **file**

Copy the code shown in Listing 7 and save it in a plain-text ASCII file named *ZoneQuery.java*

```
package oow2015.hol6663.exercises;

import java.util.*;
import java.io.*;
import com.oracle.solaris.rad.client.*;
import com.oracle.solaris.rad.connect.*;
import com.oracle.solaris.rad.zonemgr.*;

public class ZoneQuery {
    /**
     * This function lists zones on specified machine.
     *
     * @param con Machine RAD Connection to machine
     * @throws IOException
     */
    public static void listZones(Connection con) throws IOException {
        // Print out list header of the zones
        System.out.println();
        System.out.format("  %-16s %-11s %-16s %-10s %-10s\n", "NAME", "STATUS",
                "ZONEPATH", "BRAND", "IP");
        //============================================//
        // Please input the Source code here          //
        //============================================//

        //============================================//
        // Input end                                  //
        //============================================//
    }

    /**
     * This function show specified zone's properties
     *
     * @param con Connection RAD Connection to machine
     * @param zonename String Zone name
     * @throws IOException
     */
    public static void showZoneProp(Connection con, String zonename)  throws IOException {
        Zone zone = getZone(con, zonename); //get zone object
        if (zone == null) {
            System.out.println("  Show Zone Property ERROR: Can't find the zone " + zonename);
            return;
        }
        //============================================//
        // Please input the Source code here          //
        //============================================//

        //============================================//
        // Input end                                  //
```

```
        //=========================================//
    }

    /**
     * This function gets zone object by zone name
     *
     * @param con Connection RAD Connection to machine
     * @param zonename String zone name
     * @return Zone Zone Object
     * @throws IOException
     */
    public static Zone getZone(Connection con, String zonename)   throws IOException {
        if (con == null) {
            return null;
        }
        // iterate all the zones
        for (ADRName name : con.listObjects(new Zone())) {
            Zone zone = con.getObject(name);
            if (zone.getname().equals(zonename)) {
                return zone;
            }
        }
        return null;
    }

    /**
     * This function return a list of all the zones' name on specified machine
     *
     * @param con Connection
     * @return List<String> zones' name list
     * @throws IOException
     */
    public static List<String> getZonesNameList(Connection con)  throws IOException {
        List<String> list = new ArrayList<String>();
        if (con != null) {
            for (ADRName name : con.listObjects(new Zone())) {
                Zone zone = con.getObject(name);
                list.add(zone.getname());
            }
        }
        return list;
    }
}
```

Listing 7. Code for the */root/rad/java/ oow2015/hol6663/exercises/ ZoneQuery.java* **file**

5. Create the java files for the "screenManagement" in */root/rad/java/oow2015/hol6663*

   Copy the code shown in Listing 8 and save it in a plain-text ASCII file named *ExecA.java*

```
package oow2015.hol6663.screenManagement;

import java.io.IOException;
import java.util.List;
import oow2015.hol6663.exercises.ConnectMachine;
import oow2015.hol6663.exercises.ZoneQuery;
import com.oracle.solaris.rad.connect.*;

public class ExecA {

    public static boolean listZones(List<String> remoteHosts) {
        ConnectMachine conMachine = new ConnectMachine();
        Connection con = null;
        try {
            // List zones on local machine first
            con = conMachine.localConnect();// connect to local server
```

```
            if (con != null) {
                System.out.println("  List zones on local machine:");
                ZoneQuery.listZones(con);// list local zone
                conMachine.closeConnect();
            }

            // List zones on remote machines
            if (remoteHosts != null) {
                // iterate remote hosts
                for (int i = 0; i < remoteHosts.size(); i++) {
                    try {
                        con = conMachine.tlsConnect(remoteHosts.get(i)); // connect remote host
                        if (con != null) {
                            System.out.format("  List zones on %s:\n", remoteHosts.get(i));
                            ZoneQuery.listZones(con); // list zones
                            conMachine.closeConnect(); //close connection
                        }
                    } catch (IOException e) { return false; }
                }
            }
            return true;
        } catch (IOException e) { return false; }
    }
}
```

Listing 8. Code for the */root/rad/java/ oow2015/hol6663/ screenManagement/ExecA.java* **file**

Copy the code shown in Listing 9 and save it in a plain-text ASCII file named *ExecB.java*

```
package oow2015.hol6663.screenManagement;

import java.util.*;
import java.io.IOException;
import oow2015.hol6663.exercises.ConnectMachine;
import oow2015.hol6663.exercises.ZoneQuery;
import com.oracle.solaris.rad.connect.*;

public class ExecB {

    public static void showZoneProperty(String ip, String zoneName) {
        ConnectMachine conMachine = new ConnectMachine();
        Connection con;
        try {
            if (ip == null) {   // Login as local server
                con = conMachine.localConnect();
            } else {            // Login as a remote server
                con = conMachine.tlsConnect(ip);
            }

            String zonename = zoneName;
            List<String> zonelist = ZoneQuery.getZonesNameList(con); //zonelist:zone names on
server
            while (true) {
                if (zonelist.contains(zonename))//zone named zoneName exists
                {//show zone properties
                    ZoneQuery.showZoneProp(con, zonename);
                    break;
                } else // Can't find the zone named with zoneName
                {   // Input zone name again
                    System.out.println("  Sorry, invalide zone name, please try again!");
                    zonename = System.console().readLine().trim();
                    continue;
                }
            }
            conMachine.closeConnect();//close connection
        } catch (IOException e) {}
```

```
    }
}
```

Listing 9. Code for the */root/rad/java/ oow2015/hol6663/ screenManagement / ExecB.java* **file**

Copy the code shown in Listing 10 and save it in a plain-text ASCII file named *ExecC.java*

```
package oow2015.hol6663.screenManagement;

import java.io.File;
import java.util.*;
import java.net.*;
import org.json.simple.*;
import oow2015.hol6663.exercises.ConnectMachine;
import oow2015.hol6663.exercises.MachineInfo;
import oow2015.hol6663.utility.JSONHelper;
import com.oracle.solaris.rad.connect.Connection;

public class ExecC {

    @SuppressWarnings("unchecked")
    public static boolean addMachine(String machine) {
        try {
            JSONObject servers = JSONHelper.convertFileToJsonObject(new File(
                        JSONHelper.jsonFilePath + "server.json"));
            Map<String, Object> jsonMap = JSONHelper.parseJSONObject2Map(servers.toString());
            JSONArray serverArray = new JSONArray();
            List<Object> serverList = (List<Object>) jsonMap.get("server");
            for (int i = 0; i < serverList.size(); i++) {
                Map<String, Object> server = (Map<String, Object>) serverList.get(i);
                if (((String) server.get("serverName")).equals(machine)) {
                    System.out.println(" Already in cloud: " + machine);
                    return false;
                }
                JSONObject serverObj = new JSONObject(server);
                serverArray.add(serverObj);
            }

            InetAddress address = InetAddress.getByName(machine);
            ConnectMachine conMachine = new ConnectMachine();
            Connection con;
            try {
                con = conMachine.tlsConnect(machine);
                if (con != null) {
                    int cpu_num = MachineInfo.getMachineCPU(con);
                    int mem = MachineInfo.getMachineMem(con);
                    JSONObject json = JSONHelper.getServerJson(address.getHostName(),
                        address.getHostAddress(), cpu_num, mem, "remote", cpu_num - 1, mem - 512);
                    serverArray.add(json);

                    JSONObject object = new JSONObject();
                    object.put("info", "machine list");
                    object.put("server", serverArray);
                    JSONHelper.saveAsFile(object.toString(),JSONHelper.jsonFilePath +
"server.json");

                    conMachine.closeConnect();
                    return true;
                }
            } catch (Exception e) {
                conMachine.closeConnect();
                return false;
            }
        } catch (UnknownHostException e) {
            System.err.println("Unable to lookup " + machine);
            return false;
```

```
        }
        return false;
    }
}
```

Listing 10. Code for the */root/rad/java/ oow2015/hol6663/ screenManagement/ExecC.java* **file**

Copy the code shown in Listing 11 and save it in a plain-text ASCII file named *ExecD.java*

```
package oow2015.hol6663.screenManagement;

import java.util.*;
import java.io.IOException;
import oow2015.hol6663.exercises.ConnectMachine;
import oow2015.hol6663.exercises.ZoneControl;
import oow2015.hol6663.exercises.ZoneModify;
import oow2015.hol6663.exercises.ZoneQuery;
import com.oracle.solaris.rad.connect.*;

public class ExecD {

    public static void createZone(Map<String, Object> connAndZoneInfo) {
        ConnectMachine conMachine = new ConnectMachine();
        //get server connection
        Connection con = (Connection) connAndZoneInfo.get("connection");
        //get zone infomation
        @SuppressWarnings("unchecked")
        Map<String, String> map = (Map<String, String>) connAndZoneInfo.get("zoneInfo");
        String zonename = map.get("zoneName");
        try {
            //create zone on server, if success,return true
            boolean res = ZoneControl.create(con, zonename);
            if (res) {// create success
                if (!map.get("memory").equals("-1m"))//-1:memory size is not specified
                {
                    // Set memory limitaion for created zone
                    ZoneModify.addZoneMemoryProp(con, zonename, map.get("memory"));
                }
                if (!map.get("cpuNumber").equals("-1"))//-1:cpu number is not specified
                {
                    // Set dedicated cpu for created zone
                    ZoneModify.addZoneCPUProp(con, zonename, map.get("cpuNumber"));
                }
                ZoneQuery.listZones(con);//show all the zone
            }
        } catch (IOException e) {
        } finally {
            conMachine.closeConnect();//close connection
        }
    }
}
```

Listing 11. Code for the */root/rad/java/ oow2015/hol6663/ screenManagement / ExecD.java* **file**

Copy the code shown in Listing 12 and save it in a plain-text ASCII file named *ExecE.java*

```
package oow2015.hol6663.screenManagement;

import java.util.*;
import java.io.IOException;
import oow2015.hol6663.exercises.ConnectMachine;
import oow2015.hol6663.exercises.ZoneControl;
import oow2015.hol6663.exercises.ZoneQuery;
import com.oracle.solaris.rad.connect.*;

public class ExecE extends Thread {
```

```
    private String ip;//server ip
    private String zoneName;//zone name

    @Override
    public void run() {
        ConnectMachine conMachine = new ConnectMachine();
        Connection con;
        try {
            if (ip == null) {
                con = conMachine.localConnect();//local server
            } else {
                con = conMachine.tlsConnect(ip);//remote server
            }

            List<String> zonelist = ZoneQuery.getZonesNameList(con);//zonelist:zone names on server
            String zonename = zoneName;
            while (true) {
                if (zonelist.contains(zonename))//zone named zoneName exists
                {
                    break;
                } else {//not exist
                    if (zonelist.size() == 0) {//no zone in the server
                        System.out.println("  No available zone to install!");
                        return;
                    } else {//invalidate input
                        System.out.println("  Sorry, invalidate input, please try again!");
                        zonename = System.console().readLine();
                    }
                }
            }

            //install zone,if success,return true else return false
            boolean res = ZoneControl.install(con, zonename);
            //finished
            if (res) {
                System.out.println("  " + zoneName + " is successfully installed!");
                ZoneQuery.listZones(con);//show zone list
            } else {
                System.out.println("  " + zoneName + " install failed!");
            }
        } catch (IOException e) {
        } finally {
            conMachine.closeConnect();//close connection
        }
    }

    public void installZone(String ip, String zoneName) {
        this.ip = ip;
        this.zoneName = zoneName;
        this.start();// start the thread to install zone
        System.out.println("  The zone is installing now! This may take a few minutes,you can go
ahead to the next exercise first!");
        while (this.isAlive())//determine wether the installing process is finished
        {//not finished
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {}
        }
    }

    public String getIp() {
        return ip;
    }

    public void setIp(String ip) {
```

```
        this.ip = ip;
    }

    public String getZoneName() {
        return zoneName;
    }

    public void setZoneName(String zoneName) {
        this.zoneName = zoneName;
    }
}
```

Listing 12. Code for the */root/rad/java/ oow2015/hol6663/ screenManagement / ExecE.java* **file**

Copy the code shown in Listing 13 and save it in a plain-text ASCII file named *ScreenManagement.java*

```
package oow2015.hol6663.screenManagement;

import java.io.*;
import java.util.*;
import java.net.*;
import org.json.simple.*;
import oow2015.hol6663.exercises.*;
import oow2015.hol6663.utility.*;
import com.oracle.solaris.rad.connect.Connection;

public class ScreenManagement {
    public ScreenManagement(){
        JSONHelper.jsonFilePath=System.getProperty("user.dir")+"/oow2015/hol6663/";
        File file=new File(JSONHelper.jsonFilePath + "server.json");
        if(!file.exists())
            initialLocalMachine();
    }

    @SuppressWarnings("unchecked")
    private static boolean initialLocalMachine() {
        try {
            InetAddress address = InetAddress.getLocalHost();
            ConnectMachine conMachine = new ConnectMachine();
            Connection con;
            MachineInfo machineInfo = new MachineInfo();
            con = conMachine.localConnect();// connect to local server
            if (con != null) {
                int cpu_num = machineInfo.getMachineCPU(con);
                int mem = machineInfo.getMachineMem(con);
                JSONObject json = JSONHelper.getServerJson(address.getHostName(),
address.getHostAddress(), cpu_num, mem, "local", cpu_num - 1, mem - 512);
                JSONArray serverArray = new JSONArray();
                serverArray.add(json);
                JSONObject object = new JSONObject();
                object.put("info", "machine list");
                object.put("server", serverArray);
                JSONHelper.saveAsFile(object.toString(),JSONHelper.jsonFilePath + "server.json");
                conMachine.closeConnect();
                return true;
            }
        } catch (IOException e) {
            // TODO Auto-generated catch block
           // e.printStackTrace();
            return false;
        }
        return false;
    }
    public String printOptionsAndGetUserInput() {
        String input = null;
        printRootOptions();// print
```

```
        input = System.console().readLine();
        return input.toLowerCase().trim();
    }

    public void printRootOptions() {
        System.out.println();
        System.out.println("================ HOL6663: RAD Exercises =================");
        System.out.println("    Exercise 1. Using RAD to Query Cloud");
        System.out.println("         a) List zones");
        System.out.println("         b) Show zone's property");
        System.out.println("    Exercise 2. Using RAD to Manage cloud ");
        System.out.println("         c) Add a remote machine to the cloud");
        System.out.println("         d) Request a zone from the cloud");
        System.out.println("         e) Install zone");
        System.out.println("=========================================================");
        System.out.println("  Please choose an exercise by input a letter (a/b/c/d/e),");
        System.out.println("  then press the Enter key to confirm :    ");

    }
    public Map<String, Object> getCreateZoneInfo() {
        Map<String, Object> result = new HashMap<>();
        Map<String, String> zoneInfo = new HashMap<>();
        while (true) {
            System.out.println();
            System.out.println("  Please input cpu number(eg: 1),and press Enter key!");
            System.out.println("  Default is -1,meaning not specify zone's cpu number!");
            String cpuNumber = System.console().readLine().trim();
            //determine whether the user input is reasonable
            if (cpuNumber == null ||( cpuNumber.trim().length()!=0 &&  (!isInt(cpuNumber.trim()) ||
Integer.parseInt(cpuNumber) > Server.getCloudMaxZoneCpuNumber() || Integer.parseInt(cpuNumber)<=0)))
{
                System.out.println("  Invalide input!Try again!The max cpu number is " +
Server.getCloudMaxZoneCpuNumber()+",the min is 1");
                continue;
            } else {
                 if(cpuNumber.trim().length() == 0) {
                       cpuNumber=-1+"";//default is -1
                 }
                while (true) {
                    System.out.println("  Please input memory size(The unit is MB,eg:512),and press
Enter key!");
                    System.out.println("  Default is -1,meaning not specify zone's memory size!");
                    String memory = System.console().readLine().trim();
                    if (memory == null ||( memory.trim().length() !=0 && (!isInt(memory.trim()) ||
Integer.parseInt(memory) > Server.getClouldMaxZoneMemory() || Integer.parseInt(memory) <512))) {
                        System.out.println("  Invalide input!Try again!The max memory size is " +
Server.getClouldMaxZoneMemory() + "MB,the min is 512MB");
                        continue;
                    } else {
                        if(memory.trim().length() == 0) {
                            memory=-1+"";//default is -1
                        }
                        //according to user's cpu number and memory size requirements and find a
suitable server in the cloud
                        Server server = getSuitableServer(Integer.parseInt(cpuNumber),
Integer.parseInt(memory));
                       if( server == null ) {
                           System.out.println(" Sorry, there's no suitable server in the cloud ");
                           return null;
                        }
                        System.out.println("  The suitable server is " + server.getServerName());
                        if (server.getIP().equals(Server.getLocalServer().getIP())) {
                            zoneInfo.put("ip", null);//local server
                        } else {
                            zoneInfo.put("ip", server.getIP());//remote server
```

```
                        }
                        zoneInfo.put("cpuNumber", cpuNumber.trim());
                        zoneInfo.put("memory", memory.trim() + "m");
                        break;
                    }
                }
                break;
            }
        }
        String name = null;
        while (true) {
            System.out.println("  Please input the zone name,and press Enter key:");
            name = System.console().readLine().trim();
            if (name != null && name.length() > 0) {
                zoneInfo.put("zoneName", name);
                break;
            } else {
                System.out.println("  Sorry ,invalide zone name!Try again!");
                continue;
            }
        }
        ConnectMachine conMachine = new ConnectMachine();
        Connection con;
        try {
            if (zoneInfo.get("ip") == null) {
                con = conMachine.localConnect();//connect local server
            } else {
                con = conMachine.tlsConnect(zoneInfo.get("ip"));//connect remote server
            }
            ZoneQuery zoneQuery = new ZoneQuery();
            String zonename = null;
            List<String> zonelist = zoneQuery.getZonesNameList(con); //zonelist:zone names on
server
            zonename = zoneInfo.get("zoneName");
            while (true) {
                if (zonelist.contains(zonename))//zone named zoneName exists
                {
                    System.out.println("  Sorry the zone name has exist,please input a new zone
name!");
                    zonename = System.console().readLine().trim();
                    continue;
                } else //no zone named zoneName
                {
                    zoneInfo.put("zoneName", zonename);
                    break;
                }
            }
            result.put("connection", con);
            result.put("zoneInfo", zoneInfo);
            return result;
        } catch (IOException e) {}
        return result;
    }

    public Map<String, String> getInstallZoneInfo(List<String> machines) {
        String zoneName = null;
        Map<String, String> installInfo = new HashMap<>();
        if (machines == null) {
            installInfo.put("ip", null);
        } else {
            installInfo.put("ip", machines.get(0));
        }

        while (true) {
```

```
            System.out.println("  Please input the zone name that you want to install and press
Enter key");
            zoneName = System.console().readLine().trim();
            if (zoneName != null && zoneName.trim().length() > 0) {
                installInfo.put("zoneName", zoneName);
                break;
            } else {
                System.out.println("  Sorry,invalide input!Try again!");
                continue;
            }
        }
        return installInfo;
    }

    public Map<String, String> getShowZonePropertyInput(List<String> machines) {
        Map<String, String> inputs = new HashMap<>();
        if (machines == null) {
            inputs.put("ip", null);
        } else {
            inputs.put("ip", machines.get(0));
        }
        while (true) {
            System.out.println("  Please input zone name and press Enter key");
            String zoneName = System.console().readLine().trim();
            if (zoneName != null && zoneName.length() != 0) {
                inputs.put("zoneName", zoneName);
                break;
            } else {
                System.out.println("  Please input zoneName,try again!");
                continue;
            }
        }
        return inputs;
    }

    /**
     * According user's requirements, this function finds a suitable server in
     * the cloud to create a zone
     *
     * @param cpu int zone's cpu number,-1:no specified value
     * @param memory int zone's memory size,-1:no specified value
     * @return Server
     */
    public Server getSuitableServer(int cpu, int memory) {
        List<Server> servers = Server.getServerList();
        List<Server> serverCpu = new ArrayList<>();//serverCpu contains servers which meets the cpu
requirement
        List<Server> serverCpu_Memory = new ArrayList<>();//serverCpu_Memory contains servers which
meets both the cpu and memory requirement
        for (int i = 0; i < servers.size(); i++)//iterate servers in cloud and find servers which
could meet cpu requirement
        {
            Server server = servers.get(i);
            if (cpu <= server.getMaxZoneCpuNumber()) {
                serverCpu.add(server);
            }
        }
        for (int j = 0; j < serverCpu.size(); j++)//iterate serverCpu and find servers which could
meet memory size requirement
        {
            Server server = serverCpu.get(j);
            if (memory <= server.getMaxZoneMemory()) {
                serverCpu_Memory.add(server);
            }
        }
```

```
        if (serverCpu_Memory.size() != 0) {
            //iterate serverCpu_Memory and return a server random
            Random random = new Random();
            int index = random.nextInt(serverCpu_Memory.size());//0~serverCpu_Memory.size()-1
random
            return serverCpu_Memory.get(index);
        }
        return null;
    }

    public boolean isInt(String s) {
        try {
            Integer.parseInt(s);
        } catch (NumberFormatException e) {
            return false;
        } catch (NullPointerException e) {
            return false;
        }
        // only got here if we didn't return false
        return true;
    }

    public ArrayList<String> getMachineOption(boolean showAll) {
        ArrayList<String> ags = new ArrayList<>();
        while (true) {
            System.out.println();
            int machineNum = Server.getServerList().size();
            System.out.format("  There are %d machines in the cloud, please choose one:\n",
machineNum);
            for (int i = 0; i < machineNum; i++)//print ip list
            {
                if (Server.getServerList().get(i).getIP().equals(Server.getLocalServer().getIP()))
{
                    System.out.format("    %d. %s\n", i + 1,
Server.getServerList().get(i).getServerName() + "(local)");
                } else {
                    System.out.format("    %d. %s\n", i + 1,
Server.getServerList().get(i).getServerName());
                }
            }
            if (showAll) {
                System.out.format("    %d. All\n", machineNum + 1);
            }

            System.out.println("  Please input a number and press Enter to confirm (default option
is 1) : ");

            String line;
            line = System.console().readLine().trim();//select an ip index
            if (line != null && line.length() != 0)//user input a index value
            {
                int select = -1;
                try {
                    select = Integer.parseInt(line);
                } catch (Exception e) {
                    select = -1;
                }

                if (select == 1) {
                    return null; //localserver
                } else if (select == machineNum + 1) //localserver + remote server
                {
                    if (showAll == false) {
                        System.out.println("  Invalide input,try again!");
                        continue;
```

```
                    } else {
                        for (int i = 0; i < Server.getServerList().size(); i++) {
                            ags.add(Server.getServerList().get(i).getIP()); //all servers
                        }
                        return ags;
                    }
                } else if (select > 1 && select <= machineNum) //remote server
                {
                    ags.add(Server.getServerList().get(select - 1).getIP());//selected remote
server
                    return ags;
                } else {
                    System.out.println("  Invalide input,try again!");
                    continue;
                }
            } else {//user does not input
                return null;//default
            }
        }
    }

    public String AddServer() {
        System.out.println();
        int machineNum = Server.getServerList().size();
        System.out.format("  There are %d machines in the cloud : \n", machineNum);
        for (int i = 0; i < machineNum; i++)//print ip list
        {
            System.out.format("    %d. %s\n", i + 1, Server.getServerList().get(i).getServerName());
        }
        while (true) {
            String machine;
            System.out.println("  Please input the server name you want to add to virtual
environment");
            machine = System.console().readLine().toLowerCase().trim();
            if (machine == null || machine.length() == 0)
                return null;
            try {
                InetAddress local = InetAddress.getLocalHost();
                if (local.getHostName().equals(machine)||local.getHostAddress().equals(machine)) {
                                System.err.println("Local machine is already in list ");
                    continue;
                }

                if (!InetAddress.getByName(machine).isReachable(3000)) {
                    System.err.println("The machine is not reachable ");
                    continue;
                }
            } catch (UnknownHostException e) {
                System.err.println("Unable to lookup " + machine);
                continue;
            } catch (IOException e) {
                System.err.println("Unable to reach " + machine);
                continue;
            }
            return machine;
        }
    }
}
```

Listing 13. Code for the */root/rad/java/ oow2015/hol6663/ screenManagement / ScreenManagement.java* **file**

6. Create the java files for the "utility" in */root/rad/java/oow2015/hol6663*

   Copy the code shown in Listing 14 and save it in a plain-text ASCII file named *JSONHelper.java*

```
package oow2015.hol6663.utility;
```

```
import java.io.*;
import java.util.*;
import org.json.simple.*;
import org.json.simple.parser.*;

public class JSONHelper {

    public static String jsonFilePath;

    /**
     * This funtion parses JSONArray string to a List
     *
     * @param jsonStr  String json String which could be parsed to a JSONArray,
     *        jsonStr likes "[{"xx":"xx"},{"xx":"yy"},"xx":"zz"]"
     * @return List<Object> Object is a Map object or a List object or a String object
     */
   public static List<Object> parseJSONArray2List(String jsonStr) {
          JSONParser parser = new JSONParser();
          List<Object> list = new ArrayList<Object>();
          try {
                JSONArray arry = (JSONArray) parser.parse(jsonStr);// parse json String to a
JSONArray
                // Iterate the array's elements, every element is a
                // JSONObject({"xx":"xx"}) ,JSONArray([...]) or String("xx":"zz")
                for (int i = 0; i < arry.size(); i++) {
                      if (arry.get(i) instanceof JSONObject)// JSONObject
                      {
                            list.add(parseJSONObject2Map(arry.get(i).toString()));
                      } else if (arry.get(i) instanceof JSONArray)// JsonArray
                      {
                            list.add(parseJSONArray2List(arry.get(i).toString()));
                      } else if (arry.get(i) instanceof String) {// String
                            list.add(arry.get(i).toString());
                      }
                }
          } catch (Exception e) {}
          return list;
   }

   /**
     * This funtion parses JSONObject string to a Map
     *
     * @param jsonStr  String json String which could be parsed to a JSONObject,
     *        jsonStr likes {"xx":"xx","yy":{"xx":"xx"},"zz":[...]}
     * @return Map<String, Object> key is a String ,like "xx","yy","zz", value
     *         is an object, may be a Map object or a List object or a String object
     */
   public static Map<String, Object> parseJSONObject2Map(String jsonStr) {
          JSONParser parser = new JSONParser();
          JSONObject json;
          Map<String, Object> result = new HashMap<>();
          try {
                json = (org.json.simple.JSONObject) parser.parse(jsonStr);// parse
                for (Object key : json.keySet()) { // iterate the JSONOject
                      Object value = json.get((String) key);
                      if (value instanceof JSONObject) {
                            result.put(key.toString(), parseJSONObject2Map(value.toString()));
                      } else if (value instanceof JSONArray){
                            result.put(key.toString(), parseJSONArray2List(value.toString()));
                      } else if (value instanceof String){
                            result.put(key.toString(), value.toString());
                      }
                }
          } catch (ParseException e) {}
```

```
        return result;
    }

    /**
     * This function creates a JSONObject,which will be as a parameter for
     * zone's setting or adding cpu configuration
     *
     * @param cpuNumber int cpu number
     * @param type String "set" or "add"
     * @return JSONObject for example {"resource": {"type":
     *         "dedicated-cpu","properties": [{"name" : "cpus","value": "1"}]}}
     */
    @SuppressWarnings("unchecked")
    public static JSONObject getCPUJsonObject(int cpuNumber, String type) {
        if (type.equals("set"))// set cpu configuration
        {
                JSONObject object = new JSONObject();
                JSONObject filterObject = new JSONObject();
                filterObject.put("type", "dedicated-cpu");
                JSONObject proObj = new JSONObject();
                proObj.put("name", "ncpus");
                proObj.put("value", cpuNumber + "");
                JSONArray arr = new JSONArray();
                arr.add(proObj);
                object.put("properties", arr);
                object.put("filter", filterObject);
                return object;
        } else// add cpu configuration
        {
                JSONObject object = new JSONObject();
                JSONObject obj1 = new JSONObject();
                obj1.put("name", "ncpus");
                obj1.put("value", cpuNumber + "");
                JSONArray arr = new JSONArray();
                arr.add(obj1);
                JSONObject obj2 = new JSONObject();
                obj2.put("properties", arr);
                obj2.put("type", "dedicated-cpu");
                object.put("resource", obj2);
                return object;
        }
    }

    /**
     * This function creates a JSONObject,which will be as a parameter for
     * zone's setting or adding memory configuration
     *
     * @param memory int memory size
     * @param type String "set" or "add"
     * @return JSONObject for example JSONObject for example {"resource":
     *         {"type": "dedicated-cpu","properties": [{"name" : "cpus","value": "1"}]}}
     */
    @SuppressWarnings("unchecked")
    public static JSONObject getMemoryJsonObject(int memory, String type) {
        if (type.equals("set"))// set memory configuration
        {
                JSONObject object = new JSONObject();
                JSONObject filterObject = new JSONObject();
                filterObject.put("type", "capped-memory");

                JSONObject proObj = new JSONObject();
                proObj.put("name", "physical");
                proObj.put("value", memory + "m");
                JSONArray arr = new JSONArray();
                arr.add(proObj);
```

```
                object.put("properties", arr);
                object.put("filter", filterObject);
                return object;
        } else// add memory configuration
        {
                JSONObject obj = new JSONObject();
                JSONObject obj1 = new JSONObject();
                obj1.put("name", "physical");
                obj1.put("value", memory + "m");
                JSONArray arr = new JSONArray();
                arr.add(obj1);

                JSONObject obj2 = new JSONObject();
                obj2.put("properties", arr);
                obj2.put("type", "capped-memory");
                obj.put("resource", obj2);
                return obj;
        }
    }

    /**
     * This function writes content to the a file with full path.
     * If filePath does not exist, create it. If filePath exists, overwrite the file
     *
     * @param content String this is a json String
     * @param filePath
     * @return the result file
     */
    public static File saveAsFile(String content, String filePath) {
        File file = new File(filePath);
        if(!file.exists())
        {
                try {
                        file.createNewFile();
                } catch (IOException e) {
                        System.out.println("Json file create failed!");
                }
        }
        FileWriter fwriter = null;
        try {
                fwriter = new FileWriter(file);
                fwriter.write(content);
        } catch (IOException ex) {
                ex.printStackTrace();
        } finally {
                try {
                        fwriter.flush();
                        fwriter.close();
                } catch (IOException ex) {
                        ex.printStackTrace();
                }
        }
        return file;
    }

    public static JSONObject convertFileToJsonObject(File file) {
        JSONParser parser = new JSONParser();
        if (file.exists()) {
                try {
                        StringBuffer sb = new StringBuffer();
                        String line;
                        BufferedReader rd = new BufferedReader(new InputStreamReader(
                                        new FileInputStream(file)));
                        while ((line = rd.readLine()) != null) {
                                sb.append(line);
```

```
                        }
                        rd.close();
                        return (JSONObject) parser.parse(sb.toString());// parse the string to
JSONObject
                } catch (FileNotFoundException e) {
                        System.out.println("  Could not read the file:" + file.getPath());
                        return null;
                } catch (IOException e) {
                        System.out.println("  Could not read the file:" + file.getPath());
                        return null;
                } catch (ParseException e) {
                        return null;
                }
          } else {
                System.out.println("  The file is not exist:" + file.getPath());
                return null;
          }
    }

     public static JSONObject getServerJson(String serverName, String iP, int cpuNumber, int memory,
String type, int maxZoneCpuNumber, int maxZoneMemory) {

        Map<String, String> machine = new HashMap<String, String>();
        machine.put("serverName", serverName);
        machine.put("IP", iP);
        machine.put("cpuNumber", String.valueOf(cpuNumber));
        machine.put("memory", String.valueOf(memory));
        machine.put("type", type);
        machine.put("maxZoneCpuNumber", String.valueOf(maxZoneCpuNumber));
        machine.put("maxZoneMemory", String.valueOf(maxZoneMemory));

        JSONObject serverObj = new JSONObject(machine);
        return serverObj;
     }
}
```

Listing 14. Code for the */root/rad/java/ oow2015/hol6663/ utility/ JSONHelper.java* **file**

Copy the code shown in Listing 15 and save it in a plain-text ASCII file named *Server.java*

```
package oow2015.hol6663.utility;

import java.io.File;
import java.util.*;
import org.json.simple.JSONObject;

public class Server {
    private String serverName; // server name
    private String IP;         // machine's IP
    private int cpuNumber;     // CPU number of the machine
    private int memory;        // Memory size MB of the machine
    private String type;       // local or remote
    private int maxZoneCpuNumber;     // max CPU number that can be assigned to zone
    private int maxZoneMemory;        // max memory size that can be assigned to zone

    public Server(String serverName, String iP, int cpuNumber, int memory,
                String type, int maxZoneCpuNumber, int maxZoneMemory) {
          super();
          this.serverName = serverName;
          IP = iP;
          this.cpuNumber = cpuNumber;
          this.memory = memory;
          this.type = type;
          this.maxZoneCpuNumber = maxZoneCpuNumber;
          this.maxZoneMemory = maxZoneMemory;
    }
```

```
public int getMaxZoneCpuNumber() {
	return maxZoneCpuNumber;
}

public void setMaxZoneCpuNumber(int maxZoneCpuNumber) {
	this.maxZoneCpuNumber = maxZoneCpuNumber;
}

public int getMaxZoneMemory() {
	return maxZoneMemory;
}

public void setMaxZoneMemory(int maxZoneMemory) {
	this.maxZoneMemory = maxZoneMemory;
}

public String getServerName() {
	return serverName;
}

public void setServerName(String serverName) {
	this.serverName = serverName;
}

public String getIP() {
	return IP;
}

public void setIP(String iP) {
	IP = iP;
}

public int getCpuNumber() {
	return cpuNumber;
}

public void setCpuNumber(int cpuNumber) {
	this.cpuNumber = cpuNumber;
}

public int getMemory() {
	return memory;
}

public void setMemory(int memory) {
	this.memory = memory;
}

 @SuppressWarnings("unchecked")
 public static Server getLocalServer() {
     File jsonFile=new File(JSONHelper.jsonFilePath + "server.json");
     if (jsonFile.exists()){
         JSONObject servers = JSONHelper.convertFileToJsonObject(new File(
                    JSONHelper.jsonFilePath + "server.json"));
         Map<String, Object> jsonMap = JSONHelper.parseJSONObject2Map(servers.toString());
         List<Object> serverList = (List<Object>) jsonMap.get("server");
         for (int i = 0; i < serverList.size(); i++) {
             Map<String, Object> server = (Map<String, Object>) serverList.get(i);
             if (((String) server.get("type")).equals("local")) {
                 Server localserver = new Server(
                                 (String) server.get("serverName"),
                                 (String) server.get("IP"),
                                 Integer.parseInt(server.get("cpuNumber").toString()),
                                 Integer.parseInt(server.get("memory").toString()),
```

```
                                    "local",
                                    Integer.parseInt(server.get("maxZoneCpuNumber").toString()),
                                    Integer.parseInt(server.get("maxZoneMemory").toString()));
                    return localserver;
                }
            }
        }
        return null;
    }

    @SuppressWarnings("unchecked")
    public static List<Server> getRemoteServer() {
        File jsonFile=new File(JSONHelper.jsonFilePath + "server.json");
        List<Server> list = new ArrayList<>();
        if(jsonFile.exists()){
            JSONObject servers = JSONHelper.convertFileToJsonObject(new File(
                                                    JSONHelper.jsonFilePath + "server.json"));
            Map<String, Object> jsonMap = JSONHelper.parseJSONObject2Map(servers.toString());
            List<Object> serverList = (List<Object>) jsonMap.get("server");
            for (int i = 0; i < serverList.size(); i++) {
                Map<String, Object> server = (Map<String, Object>) serverList.get(i);
                if (((String) server.get("type")).equals("remote")) {
                    Server remoteServer = new Server(
                                    (String) server.get("serverName"),
                                    (String) server.get("IP"),
                                    Integer.parseInt(server.get("cpuNumber").toString()),
                                    Integer.parseInt(server.get("memory").toString()),
                                    "remote",
                                    Integer.parseInt(server.get("maxZoneCpuNumber").toString()),
                                    Integer.parseInt(server.get("maxZoneMemory").toString()));
                    list.add(remoteServer);
                }
            }
        }
        return list;
    }

    public static List<Server> getServerList() {
        List<Server> servers = new ArrayList<>();
        if(getLocalServer() != null)
            servers.add(getLocalServer());
        if(getRemoteServer().size()!=0)
            servers.addAll(getRemoteServer());
        return servers;
    }

    public static List<String> getRemoteServerIPList() {
        List<Server> servers = getRemoteServer();
        List<String> ips = new ArrayList<>();
        for (int i = 0; i < servers.size(); i++) {
            ips.add(servers.get(i).getIP());
        }
        return ips;
    }

    public static Server getServerbyName(String serverName) {
        List<Server> list = getServerList();
        for (int i = 0; i < list.size(); i++) {
            if (list.get(i).getServerName().equals(serverName)) {
                return list.get(i);
            }
        }
        return null;
    }
```

```
   public static int getCloudMaxZoneCpuNumber() {
        int cpuNumber=0;
        List<Server> servers = getServerList();
        for(Server server : servers) {
            if(server.getCpuNumber() > cpuNumber)
                cpuNumber = server.getCpuNumber();
        }
        return cpuNumber;
   }

   public static int getClouldMaxZoneMemory() {
        int memorySize=0;
        List<Server> servers = getServerList();
        for(Server server : servers) {
            if(server.getMemory() > memorySize)
                memorySize = server.getMemory();
        }
        return memorySize;
   }
}
```

Listing 15. Code for the */root/rad/java/ oow2015/hol6663/ utility/ Server.java* **file**

Appendix B: Creating the REST-based application

Use the procedures in this appendix to create the .java files, and build file of REST-based web application in Exercisec 4.

Create the Directory Structure

The source code of Rest-based web application is located in the */root/rad/RestfulWebApp*.

```
root@hol6663-1:~ # ls /root/rad/RestfulWebApp
build.xml            src             WebRoot
```

Shown from the directory structure of RestfulWebApp in figure 12, the build file is put in */root/rad/RestfulWebApp*, the main source codes are created in */root/rad/RestfulWebApp/src*, while the jsp and configuration files are created in */root/rad/RestfulWebApp/WebRoot*.

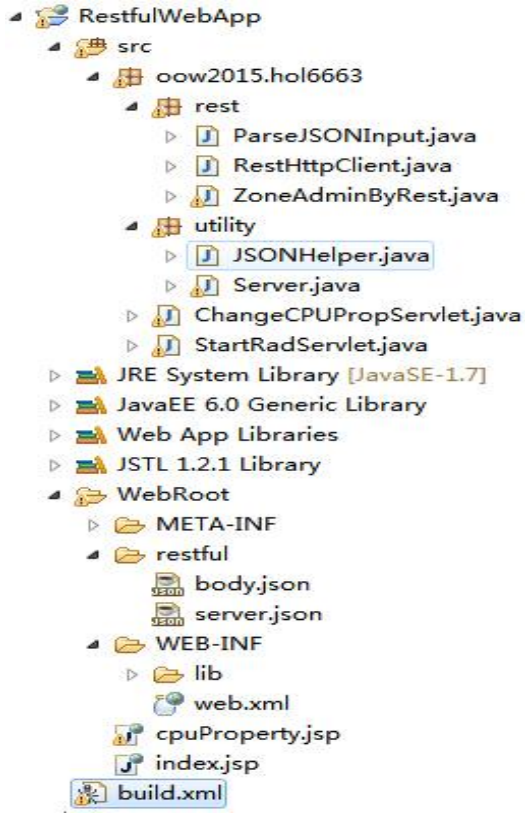

Figure 12 Directory structure of RestfulWebApp

Create the Build Script file

1. Create the ant build file *build.xml* for RestfulWebApp as following.

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="RestfulWebApp" default="deploy" basedir="./">
 <property environment="env" />
 <property name="webapp.name" value="RestfulWebApp" />
 <property name="catalina.home" value="../tools/apache-tomcat-8.0.27" />
 <property name="dist.dir" value="${basedir}/dist" />
```

```
<property name="ant.dir" value="../tools/apache-ant-1.9.6" />
<property name="webRoot.dir" value="${basedir}/WebRoot" />
<property name="src.dir" value="${basedir}/src" />
<property name="config.dir" value="${webRoot.dir}/restful" />
<property name="lib.dir" value="../lib" />
<property name="radlib.dir" value="/usr/lib/rad/java" />
<property name="build.dir" value="${basedir}/build" />
<!-- init classpath -->
<path id="project.classpath">
    <fileset dir="${lib.dir}">
           <include name="**/*.jar" />
    </fileset>
    <fileset dir="${radlib.dir}">
                        <include name="*.jar" />
    </fileset>
    <!-- add tomcat classpath -->
    <fileset dir="${catalina.home}/lib">
           <include name="*.jar" />
    </fileset>
    <!-- ant lib  -->
    <fileset dir="${ant.dir}">
           <include name="**/*.jar" />
    </fileset>
</path>

<!-- get the source compile classpath in a printable form -->
<pathconvert pathsep="${line.separator}|   |-- "
            property="echo.path.compile"
            refid="project.classpath">
</pathconvert>

<!-- show classpath jars -->
<target name="print_classpath">
    <echo message="|-- compile classpath"/>
    <echo message="|   |"/>
    <echo message="|   |-- ${echo.path.compile}"/>
</target>


<!-- remove provious dir -->
<target name="clear" description="remove old files">
    <delete dir="${build.dir}" />
    <delete dir="${dist.dir}" />
    <delete file="${catalina.home}/webapps/${webapp.name}.war" />
    <delete dir="${catalina.home}/webapps/${webapp.name}" />
</target>

<!-- create new dir -->
<target name="init" depends="clear" description="create new dir">
    <mkdir dir="${build.dir}/classes" />
    <mkdir dir="${dist.dir}" />
</target>

<!-- compile java -->
<target name="compile" depends="init" description="compile java">
    <echo message="begin compile..." />
    <javac srcdir="${src.dir}" destdir="${build.dir}/classes"
           includeantruntime="false" nowarn="on"
            deprecation="true" debug="true"
           encoding="UTF-8" classpathref="project.classpath"
           >
           <compilerarg line="-Xlint:unchecked" />
           <!-- <classpath refid="project.classpath" /> -->
    </javac>
    <copy todir="${build.dir}">
```

```
            <fileset dir="${config.dir}">
                  <include name="**/*.json" />
            </fileset>
    </copy>
    <echo message="end compile..." />
</target>

<!-- classes to  jar -->
<!--
    <target name="pack" depends="compile">
        <jar jarfile="${build.dir}/${webapp.name}.jar">
            <fileset dir="${build.dir}/classes">
                <include name="**/*.class"/>
            </fileset>
        </jar>
    </target>
-->

<!-- war package  -->
<target name="war" depends="compile" description="project to war">
    <echo message="begin war..." />
    <war destfile="${dist.dir}/${webapp.name}.war" basedir="${webRoot.dir}"
           webxml="${webRoot.dir}/WEB-INF/web.xml">
           <lib dir="${lib.dir}" />
           <lib dir="${radlib.dir}" />
           <classes dir="${build.dir}/classes" />
           <fileset dir="${webRoot.dir}">
                  <include name="***.*" />
           </fileset>
    </war>
    <echo message="end war..." />
</target>

<!-- copy war to tomcat deploy dir -->
<target name="deploy" depends="war" description="deploy project">
    <echo message="begin deploy..." />
    <copy file="${dist.dir}/${webapp.name}.war" todir="${catalina.home}/webapps" />
    <echo message="end deploy..." />
</target>
</project>
```

Listing 16. Code for the */root/rad/RestfulWebApp/build.xml* **file**

Create the Java files

1. Create the java files for the "rest" in */root/rad/RestfulWebApp/src/oow2015/hol6663*.

   Copy the code shown in Listing 17 and save it in a plain-text ASCII file named *RestHttpClient.java*

```
package oow2015.hol6663.rest;

import java.io.*;
import java.util.*;
import org.json.simple.*;
import org.json.simple.parser.*;
import org.apache.http.client.methods.*;
import org.apache.http.entity.StringEntity;
import org.apache.http.util.EntityUtils;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClients;
import oow2015.hol6663.utility.JSONHelper;

public class RestHttpClient {
```

```
private CloseableHttpClient httpClient;
JSONParser parser;
String httpMachine;
String httpPort;

public RestHttpClient(String machine, String port) {
    httpMachine = machine;
    httpPort = port;
    httpClient = HttpClients.createDefault();
    parser = new JSONParser();
}

/**
 * This function reads json file which contains the parameter for HTTP request,
 * parses it to JSONObject and return the string of JSONObject
 *
 * @param file String json file's with absolute path
 * @return String json
 */
private String getEntity(String file) {
    try {
        //file to  JSONObject
        JSONObject json = (JSONObject) parser.parse(new FileReader(file));
        System.out.println("  " + json.toString());
        return json.toString();// JSONObject to string and return
    } catch (Exception e) {
        System.out.println("Can't parse the HTTP entity by json file " + file );
        return null;
    }
}

/**
 * This function is to struct the RAD HTTP URI :
 *  uri=http://httpMachine:httpPort/apiAddr+subAddr
 *
 * @param apiAddress String
 * @param subAddress String
 * @return String uri
 */
public String getURI(String apiAddress, String subAddress) {
    return "http://" + httpMachine + ":" + httpPort + apiAddress + "/" + subAddress;
}

/**
 * This function will do RAD authentication by send HTTP POST to RAD server
 *
 *
 * @param loginFile String A JSON file contains login information
 * @return boolean authentication result
 * @throws IOException
 */
public boolean authentication(String loginFile) {
    //============================================================//
    // Please input the Source code for HTTP authentication here    //
    //============================================================//
    // HTTP opertions are mapped to URIs
    // The URI for authentication is
    // http://httpMachine:httpPort/api/com.oracle.solaris.rad.authentication/1.0/Session
    String uri = getURI("/api/com.oracle.solaris.rad.authentication/1.0/Session", "");
    HttpPost httpPost = new HttpPost(uri);  // create a POST request
    httpPost.addHeader("Content-type", "application/json");

    String loginInfo = getEntity(loginFile); // get login info by file
    if (loginInfo != null) {
```

```
            CloseableHttpResponse res = null;
            try {
                httpPost.setEntity(new StringEntity(loginInfo));
                // Send the HTTP POST request with login infomation
                res = httpClient.execute(httpPost);
            } catch (Exception e) {
                System.out.println("  Failed in HTTP Authentication by file : " + loginFile);
                return false;
            } finally {
                close(res);
            }
            return true;
        }
        //============================================================//
        // Input end                                                  //
        //============================================================//
        System.out.println("  Failed in Authentication by file : " + loginFile);
        return false;
    }

    /**
     * This function creates service uri and sends a "GET" request to it
     * uri=http://httpMachine:httpPort/apiAddr+subAddr
     *
     * @param apiAddr String
     * @param subAddr String
     * @param showOutput boolean true: print out server's return on screen; false: not print
     * @return JSONObject server's return
     * @throws IOException
     */
    public JSONObject sendGet(String apiAddr, String subAddr, boolean showOutput) {
        String uri = getURI(apiAddr, subAddr);
        HttpGet httpGet = new HttpGet(uri);
        httpGet.addHeader("Content-type", "application/json");
        JSONObject obj;
        CloseableHttpResponse res = null;
        try {
            res = httpClient.execute(httpGet);
            if (res.getStatusLine().getStatusCode() != 200) {
                System.out.println("  HTTP Get ERROR: " + res.getStatusLine().toString());
                return null;
            }
            String result = EntityUtils.toString(res.getEntity());
            obj = (JSONObject) JSONValue.parse(result);
        } catch (Exception e) {
            System.out.println("  HTTP Get ERROR: " + uri);
            return null;
        } finally {
            close(res);
        }
        return obj;
    }

    /**
     * This function creates service uri and sends a "POST" request to it
     * uri=http://httpMachine:httpPort/apiAddr+subAddr
     *
     * @param apiAddr String
     * @param subAddr String
     * @param inputFile String json file's absolute path
     * @param showOutput boolean true:print out server's return on screen; false:not print
     * @return JSONObject server's return
     * @throws Exception
     */
```

```
    public JSONObject sendPut(String apiAddr, String subAddr, String inputFile, boolean showOutput)
{
        JSONObject json = null;
        try {
            // json file to JSONObject
            json = (JSONObject) parser.parse(new FileReader(inputFile));
        } catch (Exception e) {
            System.out.println("  Failed in send PUT for input file : " + inputFile);
            return null;
        }
        return sendPut(apiAddr, subAddr, json, showOutput);
    }

    /**
     * This function creates service uri and sends a "POST" request to it
     * uri=http://httpMachine:httpPort/apiAddr+subAddr
     *
     * @param apiAddr String
     * @param subAddr String
     * @param inputFile String json file's absolute path
     * @param showOutput boolean true:print out server's return on screen; false:not print
     * @return JSONObject server's return
     * @throws Exception
     */
    public JSONObject sendPut(String apiAddr, String subAddr, JSONObject input, boolean showOutput)
{
        String uri = getURI(apiAddr, subAddr);
        System.out.println("  send put to " + uri);
        System.out.println("  json object is " + input.toString());
        HttpPut httpPut = new HttpPut(uri);
        httpPut.addHeader("Content-type", "application/json");
        try {
            httpPut.setEntity(new StringEntity(input.toString()));
        } catch (Exception e1) {
            e1.printStackTrace();
        }
        CloseableHttpResponse res = null;
        JSONObject obj = null;
        try {
            res = httpClient.execute(httpPut);
            if (res.getStatusLine().getStatusCode() != 200) {
                System.out.println("  HTTP Get ERROR: " + res.getStatusLine().toString());
                return null;
            }

            String result = EntityUtils.toString(res.getEntity());
            obj = (JSONObject) JSONValue.parse(result);
        } catch (Exception e) {
            System.out.println("  HTTP Put ERROR: " + uri);
            System.out.println("  json object is " + input.toString());
            return null;
        } finally {
            close(res);
        }
        return obj;
    }

    /**
     * This function closes HTTP connection
     *
     * @throws IOException
     */
    public void close() throws IOException {
        if( httpClient != null)
            httpClient.close();
```

```
        httpClient = null;
    }

    public void close(CloseableHttpResponse res) {
        try {
            if( res != null )
                res.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
        res = null;
    }

    /**
     * This function gets zones uris from JSONObject
     *
     * @param jsonObject JSONObject
     * @return List<String> every element is a zone's service uri, like
     * "http://serverIP:6666/api/com.oracle.solaris.rad.zonemgr/1.2/Zone/zone1"
     */
    @SuppressWarnings("unchecked")
    public List<String> getZonesURIList(JSONObject jsonObject) {
        List<String> uriList = new ArrayList<>();
        String json = jsonObject.toString();
        Map<String, Object> zones = JSONHelper.parseJSONObject2Map(json);
        List<Object> uris = (List<Object>) zones.get("payload");
        if (!zones.get("status").equals("success") || uris.size() == 0) {
            return null;
        }
        for (int i = 0; i < uris.size(); i++) {
            String uri = "http://" + httpMachine + ":6666/"
                    + ((Map<String, String>) uris.get(i)).get("href");
            uriList.add(uri);
        }
        return uriList;
    }

    /**
     * This function gets zone's name from JSONObject
     *
     * @param jsonObject JSONObject
     * @return List<String> every element is a zone name
     */
    public List<String> getZoneNameList(JSONObject jsonObject) {
        // each uri like:"http://serverIP:6666/api/com.oracle.solaris.rad.zonemgr/1.2/Zone/zone1"
        List<String> uris = getZonesURIList(jsonObject);
        List<String> zoneNames = new ArrayList<>();
        for (int i = 0; i < uris.size(); i++) {
            String uri = uris.get(i);
            int index = uri.split("/").length;
            String zoneName = uri.split("/")[index - 1];
            zoneNames.add(zoneName);
        }
        return zoneNames;
    }
}
```

Listing 17. Code for the */root/rad/RestfulWebApp/src/oow2015/hol6663/rest/RestHttpClient.java* **file**

Copy the code shown in Listing 18 and save it in a plain-text ASCII file named *ZoneAdminByRest.java*

```
package oow2015.hol6663.rest;

import java.util.*;
import java.io.*;
import org.json.simple.*;
```

```
import oow2015.hol6663.utility.JSONHelper;

public class ZoneAdminByRest {

    RestHttpClient restClient = null;
    private static String basePath;

    public static String getBasePath() {
        return basePath;
    }

    public static void setBasePath(String basePath) {
        ZoneAdminByRest.basePath = basePath;
    }

    public ZoneAdminByRest(String machine, String port) {
        restClient = new RestHttpClient(machine, port);
    }

     /**
     * Link to RAD server, and do authentication.
     *
     * @return boolean return the authentication status
     */
    public boolean authentication() {
        return restClient.authentication(basePath + "body.json");
    }

    /**
     * This function sends request to RAD server to get all the zones on the server
     *
     * @return List<String> zone names list
     */
    public List<String> getZoneNameList() {
        List<String> zoneNames = new ArrayList<>();
        //send request,false:not print out server's return json string
        JSONObject obj = restClient.sendGet("/api/com.oracle.solaris.rad.zonemgr/1.2/Zone", "",
false);
        zoneNames = restClient.getZoneNameList(obj);
        return zoneNames;
    }

    /**
     * This function gets zone's properties as a JSONObject
     *
     * @param zonename String zone name
     * @param filterType String filter zone's properties, filterType includes
     * "dedicated-cpu"/"capped-memory"/"global"/null If filterType is null,return all properties
     * @return JSONObject This JSONObject contains zone's properties
     */
    @SuppressWarnings("unchecked")
    public JSONObject getResource(String zonename, String filterType) {
        String apiAddr = getZoneAPIAddress(zonename);//get service address
        JSONObject obj = new JSONObject();
        try {
            if (filterType == null) {
                String prop = "{}";
                obj = (JSONObject) JSONValue.parse(prop);
            } else {
                JSONObject subobj = new JSONObject();
                subobj.put("type", filterType);
                obj.put("filter", subobj);
            }
            JSONObject res = restClient.sendPut(apiAddr, "/_rad_method/getResources", obj, false);
            if (res == null) {
```

```
            return null;
        }
        return res;
    } catch (Exception e) {
        System.out.println("  Get resource exception");
        e.printStackTrace();
        return null;
    }
}

/**
 * This function gets zone's status as a JSONObject
 *
 * @param zonename String zone name
 * @return JSONObject This JSONObject contains zone's detail info include status
 */
public JSONObject getZoneDetailInfo(String zonename) {
    String apiAddr;
    try {
        apiAddr = getZoneAPIAddress(zonename);
        JSONObject res = restClient.sendGet(apiAddr, "?_rad_detail=true", false);
        return res;
    } catch (Exception e) {
        e.printStackTrace();
    }
    return null;
}

/**
 * This function adds new resources to zone's configuration
 *
 * @param zonename String zone name
 * @param resFile String a JSON file that defines the added resource
 * @throws Exception
 */
public void addResource(String zonename, String resFile) throws Exception {
    String apiAddr = getZoneAPIAddress(zonename);
    String prop = "{}";
    JSONObject obj = (JSONObject) JSONValue.parse(prop);
    try {
        restClient.sendPut(apiAddr, "/_rad_method/editConfig", obj, false);
        restClient.sendPut(apiAddr, "/_rad_method/addResource", resFile, true);
        restClient.sendPut(apiAddr, "/_rad_method/commitConfig", obj, false);
    } catch (Exception e) {
        System.out.println("  Add resource failed for " + resFile);
    }
}

/**
 * This function modify zone's properties
 *
 * @param zonename String zone name
 * @param propFile String a JSON file that defines the mdified properties
 * @throws Exception
 */
public void setProp(String zonename, String propFile) throws Exception {
    String apiAddr = getZoneAPIAddress(zonename);
    String prop = "{}";
    JSONObject obj = (JSONObject) JSONValue.parse(prop);
    try {
        restClient.sendPut(apiAddr, "/_rad_method/editConfig", obj, false);
        restClient.sendPut(apiAddr, "/_rad_method/setResourceProperties", propFile, true);
        restClient.sendPut(apiAddr, "/_rad_method/commitConfig", obj, false);
    } catch (Exception e) {
        System.out.println("  Set resource property failed for " + propFile);
```

```
        }
    }

     /**
     * This function makes zone's updates to effect immediately
     *
     * @param zonename String zone name
     * @throws Exception
     */
    public void apply(String zonename) throws Exception {
        String apiAddr = getZoneAPIAddress(zonename);
        String prop = "{}";
        JSONObject obj = (JSONObject) JSONValue.parse(prop);
        try {
            restClient.sendPut(apiAddr, "/_rad_method/apply", obj, false);
        } catch (Exception e) {
            System.out.println("  Apply zone configuration to zone failed  ");
        }
    }

    public String getStatusProperty(JSONObject res) {
        return ParseJSONInput.getStatusProperty(res);
    }

    @SuppressWarnings("unchecked")
    public String getCPUProperty(JSONObject res) {
        return ParseJSONInput.getCPUProperty(res);
    }

    public Map<String, Map<String, String>> getZoneProperty(JSONObject res) {
        return ParseJSONInput.getZoneProperty(res);
    }

    public void close() {
        try {
            if(restClient != null)
                restClient.close();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
        restClient = null;
    }

    private String getZoneAPIAddress(String zonename) {
        String uri = "/api/com.oracle.solaris.rad.zonemgr/1.2/Zone/" + zonename;
        return uri;
    }
}
```

Listing 18. Code for the */root/rad/RestfulWebApp/src/oow2015/hol6663/rest/ZoneAdminByRest.java* **file**

Copy the code shown in Listing 19 and save it in a plain-text ASCII file named *ParseJSONInput.java*

```
package oow2015.hol6663.rest;

import java.util.*;
import org.json.simple.JSONObject;
import oow2015.hol6663.utility.JSONHelper;

public class ParseJSONInput {
    @SuppressWarnings("unchecked")
    public static String getCPUProperty(JSONObject res) {
        try {
            if (res != null) {
                Map<String, Object> result = JSONHelper.parseJSONObject2Map(res.toString());
```

```
                List<Object> payload = (List<Object>) result.get("payload");
                if (result.get("status").equals("success") && payload == null) {
                    System.out.println("  Update success!");
                    return "null";
                }
                System.out.println("  zone infomation:");
                for (int j = 0; j < payload.size(); j++) {
                    Map<String, Object> body = (Map<String, Object>) payload.get(j);
                    System.out.println("  " + body.get("type"));
                    if (body.get("type").equals("dedicated-cpu")) {
                        List<Object> pro = (List<Object>) body.get("properties");
                        for (int i = 0; i < pro.size(); i++) {
                            Map<String, String> element = (Map<String, String>) pro.get(i);
                            if (element.get("value") != null ||
element.get("value").trim().length() != 0) {
                                return element.get("value");
                            }
                        }
                    }
                }
                return "no";
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return "no";
    }

    @SuppressWarnings("unchecked")
    public static Map<String, Map<String, String>> getZoneProperty(JSONObject res) {
        try {
            if (res != null) {
                Map<String, Object> result = JSONHelper.parseJSONObject2Map(res.toString());
                List<Object> payload = (List<Object>) result.get("payload");
                if (result.get("status").equals("success") && payload == null) {
                    System.out.println("  Update success!");
                    return null;
                }
                System.out.println("  zone infomation:");
                Map<String, Map<String, String>> tmp = new HashMap<String, Map<String, String>>();
                for (int j = 0; j < payload.size(); j++) {
                    Map<String, Object> body = (Map<String, Object>) payload.get(j);
                    System.out.println("  " + body.get("type"));
                    List<Object> pro = (List<Object>) body.get("properties");
                    Map<String, String> tmp2 = new HashMap<String, String>();
                    for (int i = 0; i < pro.size(); i++) {
                        Map<String, String> element = (Map<String, String>) pro.get(i);
                        if (element.get("value") != null&& element.get("value").trim().length() > 0)
{
                            tmp2.put(element.get("name"), element.get("value"));
                        }
                    }
                    tmp.put((String) body.get("type"), tmp2);
                }
                return tmp;
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }

    @SuppressWarnings("unchecked")
    public static String getStatusProperty(JSONObject res) {
        Map<String, Object> map = JSONHelper.parseJSONObject2Map(res.toString());
```

```
        try {
            String status = (String) ((Map<String, Object>) (((Map<String, Object>)
 map.get("payload")).get("Zone"))).get("state");
            return status;
        } catch (Exception e) {
            e.printStackTrace();
            return "";
        }
    }
}
```

Listing 19. Code for the */root/rad/RestfulWebApp/src/oow2015/hol6663/rest/ParseJSONInput.java* **file**

2. Copy "*utility*" directory of Appendix A to */root/rad/RestfulWebApp/src/oow2015/hol6663*.

   The java files in directory *utility* is same as files in Appendix A, just copy the whole directory to */root/rad/RestfulWebApp/src/oow2015/hol6663*

```
root@hol6663-1:~/rad/RestfulWebApp/src/oow2015/hol6663 #cp -r
 /root/rad/java/oow2015/hol6663/utility .
```

3. Create the servlet files in */root/rad/RestfulWebApp/src/oow2015/hol6663*.

   Copy the code shown in Listing 20 and save it in a plain-text ASCII file named *StartRadServlet.java*

```
package oow2015.hol6663;

import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import org.json.simple.JSONObject;
import oow2015.hol6663.rest.ZoneAdminByRest;
import oow2015.hol6663.utility.JSONHelper;
import oow2015.hol6663.utility.Server;

public class StartRadServlet extends HttpServlet {

    @Override
    public void init(ServletConfig config) throws ServletException {
        super.init(config);
        ServletContext s1 = this.getServletContext();
        String path = s1.getRealPath("/");
        JSONHelper.jsonFilePath=path + "restful/";
        ZoneAdminByRest.setBasePath(path + "restful/");
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        doPost(request, response);
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response) throws IOException
 {
        response.setContentType("text/html");
        List<Map<String, String>> list = new ArrayList<>();
        List<Server> servers = Server.getServerList();
        for (int i = 0; i < servers.size(); i++) {
            ZoneAdminByRest adminByRest = null;
            String machine = servers.get(i).getServerName();
            adminByRest = new ZoneAdminByRest(servers.get(i).getIP(), "6666");
            if (!adminByRest.authentication()) {
                adminByRest.close();
                continue;
            }
            List<String> zoneNames = adminByRest.getZoneNameList();
```

```
            if (zoneNames != null) {
                for (int j = 0; j < zoneNames.size(); j++) {
                    Map<String, String> zoneInfo = new HashMap<String, String>();
                    String zoneName = zoneNames.get(j);
                    JSONObject zone = adminByRest.getResource(zoneName, "dedicated-cpu");
                    String cpu = adminByRest.getCPUProperty(zone);
                    JSONObject statusJson = adminByRest.getZoneDetailInfo(zoneName);
                    String status = adminByRest.getStatusProperty(statusJson);
                    zoneInfo.put("machine", machine);
                    zoneInfo.put("zoneName", zoneName);
                    zoneInfo.put("dedicatedCpu", cpu);
                    zoneInfo.put("status", status);
                    list.add(zoneInfo);
                }
            }
            adminByRest.close();
        }
        request.getSession().setAttribute("zones", list);
        RequestDispatcher dispatcher = request.getRequestDispatcher("index.jsp");
        try {
           dispatcher .forward(request, response);
        } catch (ServletException e) {
           e.printStackTrace();
        }
    }
}
```

Listing 19. Code for the */root/rad/RestfulWebApp/src/oow2015/hol6663/StartRadServlet.java* **file**

Copy the code shown in Listing 20 and save it in a plain-text ASCII file named *ChangeCPUPropServlet.java*

```
package oow2015.hol6663;

import java.io.*;
import java.util.Map;
import javax.servlet.*;
import javax.servlet.http.*;
import org.json.simple.JSONObject;

import oow2015.hol6663.rest.ZoneAdminByRest;
import oow2015.hol6663.utility.JSONHelper;
import oow2015.hol6663.utility.Server;

public class ChangeCPUPropServlet extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        doPost(request, response);
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response) {
        response.setContentType("text/html");
        String machine = request.getParameter("machine");
        String zoneName = request.getParameter("zoneName");
        String type = request.getParameter("type");
        String status = request.getParameter("status");
        ZoneAdminByRest adminByRest = null;
        try {
            adminByRest = new ZoneAdminByRest(Server.getServerbyName(machine).getIP(), "6666");
            if (!adminByRest.authentication()) {
                return;
            } else if (type.equals("show")) {
                JSONObject zone = adminByRest.getResource(zoneName, null);
                Map<String, Map<String, String>> zoneProp = adminByRest.getZoneProperty(zone);
                request.getSession().setAttribute("zoneProp", zoneProp);
                request.getSession().setAttribute("machine", machine);
```

```
                request.getSession().setAttribute("zoneName", zoneName);
                request.getSession().setAttribute("status", status);
                request.getSession().setAttribute("cpuLimit",
 Server.getServerbyName(machine).getMaxZoneCpuNumber());
                RequestDispatcher dispatcher = request.getRequestDispatcher("cpuProperty.jsp");
                  dispatcher .forward(request, response);
            } else if (type.equals("commit"))//change
            {
                int cpuNumber = Integer.parseInt(request.getParameter("cpuNumber"));
                modifyZoneCpuProp(adminByRest, zoneName, cpuNumber);
                response.sendRedirect("ChangeCPUPropServlet?type=show&machine=" + machine +
 "&status=" + status + "&zoneName=" + zoneName);
            } else if (type.equals("apply")) {
                adminByRest.apply(zoneName);
                RequestDispatcher dispatcher = request.getRequestDispatcher("cpuProperty.jsp");
                  dispatcher .forward(request, response);
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            adminByRest.close();
        }
    }

    public void modifyZoneCpuProp(ZoneAdminByRest zoneAdmin, String zonename, int cpuNumber)
            throws Exception {
        JSONObject cpu = zoneAdmin.getResource(zonename, "dedicated-cpu");
        if (cpu != null && cpu.toString().contains("dedicated-cpu")) {        //update
            JSONObject json = JSONHelper.getCPUJsonObject(cpuNumber, "set");
            File jsonFile = JSONHelper.saveAsFile(json.toString(), ZoneAdminByRest.getBasePath() +
 "setprop_cpu.json");
            zoneAdmin.setProp(zonename, jsonFile.getAbsolutePath());
        } else {                                                          //add cpu resource
            JSONObject json = JSONHelper.getCPUJsonObject(cpuNumber, "add");
            File jsonFile = JSONHelper.saveAsFile(json.toString(), ZoneAdminByRest.getBasePath() +
 "addprop_cpu.json");
            zoneAdmin.addResource(zonename, jsonFile.getAbsolutePath());
        }
    }
}
```

Listing 20. Code for the */root/rad/RestfulWebApp/src/oow2015/hol6663/ChangeCPUPropServlet.java* **file**

Create the jsp files and configuration files

1. Create the jsp files in */root/rad/RestfulWebApp/WebRoot*.

   Copy the code shown in Listing 21 and save it in a plain-text ASCII file named *index.jsp*

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%
    String path = request.getContextPath();
    String basePath = request.getScheme() + "://" + request.getServerName() + ":" +
 request.getServerPort() + path + "/";
%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <base href="<%=basePath%>">
        <title>RestfulWebApp</title>
        <meta http-equiv="pragma" content="no-cache">
```

```
        <meta http-equiv="cache-control" content="no-cache">
        <meta http-equiv="expires" content="0">
        <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
        <meta http-equiv="description" content="This is my page">
        <!--
        <link rel="stylesheet" type="text/css" href="styles.css">
        -->
        <script type="text/javascript">
            function myFunction(machine, zoneName, status)
            {
                window.location = "/RestfulWebApp/ChangeCPUPropServlet?type=show&machine=" + machine
 + "&zoneName=" + zoneName + "&status=" + status;
            }
        </script>
    </head>
    <body>
        <h1 align="center"> Welcome to try RAD REST</h1>
        <h2 align="center"> Using REST to modify Zone</h2>
        <table width="50%" align="center" border="1" bordercolor="grey">
            <tr align="center" style="font-weight: bold;font-size:20px"><td>machine</td><td>zone
 name</td><td>status</td><td>dedicated cpu</td><td>change cpu property</td></tr>
            <c:forEach items="${zones}" var="zone" varStatus="status">
                <tr <c:if test="${status.count%2==1}"> bgcolor="#CCCCFE"</c:if> align="center">

 <td>${zone.machine}</td><td>${zone.zoneName}</td><td>${zone.status}</td><td>${zone.dedicatedCpu}</t
 d> <td><button type="button" title="change cpu property"  onclick="myFunction('${zone.machine}',
 '${zone.zoneName}', '${zone.status}')">change</button></td>
                </tr>
            </c:forEach>
        </table>
    </body>
</html>
```

Listing 21. Code for the */root/rad/RestfulWebApp/WebRoot/index.jsp* **file**

Copy the code shown in Listing 22 and save it in a plain-text ASCII file named *cpuProperty.jsp*

```
<%@ page language="java" import="java.util.*" pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%
    String path = request.getContextPath();
    String basePath = request.getScheme() + "://" + request.getServerName() + ":" +
 request.getServerPort() + path + "/";
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <base href="<%=basePath%>">
        <title>RestfulWebApp</title>
        <meta http-equiv="pragma" content="no-cache">
        <meta http-equiv="cache-control" content="no-cache">
        <meta http-equiv="expires" content="0">
        <meta http-equiv="keywords" content="keyword1,keyword2,keyword3">
        <meta http-equiv="description" content="This is my page">
        <!--
        <link rel="stylesheet" type="text/css" href="styles.css">
        -->
        <style type="text/css">
            p {
                font-size: 30px;
                font-weight: bold;
            }
        </style>
        <script type="text/javascript">
            function check(cpuLimit)
```

```
            {
                var cpuNumber = document.getElementById("cpuNumber").value;
                if (cpuNumber > cpuLimit || cpuNumber < 1)
                {
                    alert("Invalide input,try again!Max zone's cpu number is " + cpuLimit);
                    document.getElementById("commit").disabled = "disabled";
                }
                else
                {
                    document.getElementById("commit").disabled = "";
                }
            }
            function onSubmit(type)
            {
                var status = document.getElementById("status").value;
                if (type == "commit")
                {
                    document.getElementById("form").action =
"/RestfulWebApp/ChangeCPUPropServlet?type=commit";
                }
                else if (type == "apply")
                {
                    document.getElementById("form").action =
"/RestfulWebApp/ChangeCPUPropServlet?type=apply";
                    alert("Apply finished!");
                } else if (type == "back")
                {
                    document.getElementById("form").action = "/RestfulWebApp";
                }
                return true;
            }
            function load()
            {
                var status = document.getElementById("status").value;
                if (status == "running")
                {
                    document.getElementById("apply").disabled = "";
                }
                else
                {
                    document.getElementById("apply").disabled = "disabled";
                }
            }
        </script>
    </head>

    <body style="text-align: center;" onload="load()">
        <h1 align="center"> Welcome to try RAD REST.</h1>
        <h2 align="center"> Using REST to modify Zone</h2>
        <form id="form"  method="POST" onsubmit="onSubmit()">
            <input type="hidden" id="machine" name="machine" value="${machine}"/>
            <input type="hidden" id="zoneName" name="zoneName" value="${zoneName}"/>
            <input type="hidden" id="status" name="status" value="${status}"/>
            Change CPU Property <br/>
            ============================================================<br/>
            machine:${machine}  ,  zone name:${zoneName}   ,   status:${status}<br/>
            ============================================================<br/>
            zone's property<br/>
            <table align="center">
                <c:forEach items="${zoneProp}" var="prop">
                    <tr><td>${prop.key}:</td><td></td><td></td></tr>
                    <c:forEach items="${prop.value}" var="element" >
                        <tr><td></td><td>${element.key}</td><td>${element.value}</td></tr>
                    </c:forEach>
                </c:forEach>
```

```
            </table>
            ============================================================<br/>
            CPU number:<input type="text" name="cpuNumber" id="cpuNumber"
 onchange="check('${cpuLimit}')"/> (min zone's CPU number is 1 and max is ${cpuLimit})
            <button id="commit" name="commit" disabled="disabled"
 onclick="onSubmit('commit')">commit</button>
            <button disabled="disabled" id="apply" name="apply"
 onclick="onSubmit('apply')">apply</button>
            <button id="back" name="back" onclick="onSubmit('back')">back</button>
        </form>
    </body>
</html>
```

Listing 22. Code for the */root/rad/RestfulWebApp/WebRoot/cpuProperty.jsp* **file**

2. Create the configuration files in */root/rad/RestfulWebApp/WebRoot/restful*.

Copy the code shown in Listing 23 and save it in a plain-text ASCII file named *body.json*

```
{
      "username": "root",
      "password": "solaris11",
      "scheme": "pam",
      "preserve": true,
      "timeout": -1
}
```

Listing 23. Code for the */root/rad/RestfulWebApp/WebRoot/restful/body.json* **file**

Copy the code shown in Listing 23 and save it in a plain-text ASCII file named *server.json*

```
{
    "info": "machine list",
    "server": [{
            "serverName": "hol6663-1",
            "IP": "192.168.100.1",
            "cpuNumber": "4",
            "memory": "6144",
            "maxZoneCpuNumber": "2",
            "maxZoneMemory": "4096",
            "type": "local"
        },
        {
            "serverName": "hol6663-2",
            "IP": "192.168.100.2",
            "cpuNumber": "1",
            "memory": "4096",
            "maxZoneCpuNumber": "1",
            "maxZoneMemory": "2048",
            "type": "remote"
        }
    ]
}
```

Listing 24. Code for the */root/rad/RestfulWebApp/WebRoot/restful/server.json* **file**

3. Create the web deployment descriptor in */root/rad/RestfulWebApp/WebRoot/WEB-INF*.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-
 app_2_5.xsd" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
 http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
    <display-name>StartRad</display-name>
    <servlet>
        <description>This is the description of my J2EE component</description>
        <display-name>This is the display name of my J2EE component</display-name>
```

```
        <servlet-name>StartRadServlet</servlet-name>
        <servlet-class>oow2015.hol6663.StartRadServlet</servlet-class>
    </servlet>
    <servlet>
        <description>This is the description of my J2EE component</description>
        <display-name>This is the display name of my J2EE component</display-name>
        <servlet-name>ChangeCPUPropServlet</servlet-name>
        <servlet-class>oow2015.hol6663.ChangeCPUPropServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>StartRadServlet</servlet-name>
        <url-pattern>/StartRadServlet</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>ChangeCPUPropServlet</servlet-name>
        <url-pattern>/ChangeCPUPropServlet</url-pattern>
    </servlet-mapping>
    <welcome-file-list>
        <welcome-file>StartRadServlet</welcome-file>
    </welcome-file-list>
</web-app>
```

Listing 25. Code for the */root/rad/RestfulWebApp/WebRoot/restful/WEB-INF/web.xml* **file**

Summary

You have successfully completed the "Build your own cloud environment with Solaris 11 RAD + REST" hands-on lab! You have explored how to use RAD to query, request, boot and modify virtual machines (zones) in a smallest cloud. You have learned how to use a JAVA client to list zones, show zones' properties and control zones. Also you have learned how to use RESTful APIs to control and modify zones by HTTP. A REST-based web application has been setup in the last exercise. The application enable you to modify zones' properties by browser anyway.

See Also

- Getting Started with the Remote Administration Daemon on Oracle Solaris 11
- Secure Remote RESTful Administration with the Remote Administration Daemon
- Remote Administration Daemon Developer's Guide
- Oracle® Solaris Zones Configuration Resources

About the Authors

Xiaosong (Chris) Zhu is a Principal Software Engineer working for Oracle's ISV Engineering group. She is concentrated on Solaris and C/C++. Her duties include doing Solaris evangelizing and supporting local ISVs to run C/C++ applications best on Oracle Solaris and SPARC servers.

Yu Wang presently works for Oracle's ISV Engineering group as a Principal Software Engineer. His duties include supporting local ISVs and evangelizing about Oracle Solaris and Java technologies.

Yan Zhang is a Software Engineer working for Oracle's ISV Engineering group. She is concentrated on Solaris and Java.Her duties include supporting local ISVs and evangelizing about Oracle Solaris and Java technologies.