package com.arp.service.rpcloud.container;

import com.arp.feign.jaxrs.ARPFeignClientConfig;

import com.wordnik.swagger.config.ConfigFactory;

import com.wordnik.swagger.config.ScannerFactory;

import com.wordnik.swagger.config.SwaggerConfig;

import com.wordnik.swagger.jaxrs.config.DefaultJaxrsScanner;

import com.wordnik.swagger.jersey.JerseyApiReader;

import com.wordnik.swagger.model.ApiInfo;

import com.wordnik.swagger.reader.ClassReaders;

import org.mybatis.spring.annotation.MapperScan;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.client.discovery.EnableDiscoveryClient;

import org.springframework.cloud.netflix.feign.EnableFeignClients;

@SpringBootApplication(scanBasePackages = { "com.arp" })

//设置basePackages以扫描依赖jar包中的@FeignClient注解，否则依赖jar中的远程接口将无法使用

@EnableFeignClients(defaultConfiguration = { ARPFeignClientConfig.class }, basePackages = { "com.arp" })

@EnableDiscoveryClient

@MapperScan("com.arp")

public class ContainerServerApplication implements CommandLineRunner{

public static void main(String[] args) {

SpringApplication.run(ContainerServerApplication.class, args);

}

//@Override

public void run(String... args) throws Exception {

SwaggerConfig config = ConfigFactory.config();

config.setBasePath("/v1");

config.setApiVersion("1.0.0");

ApiInfo info = new ApiInfo("container服务接口","container服务接口，服务ID为fs-jycloud-container","","","","");

config.setApiInfo(info);

ScannerFactory.setScanner(new DefaultJaxrsScanner());

ClassReaders.setReader(new JerseyApiReader());

}

}

package com.arp.service.rpcloud.container.resource;

import java.math.BigDecimal;

import java.util.Date;

import java.util.List;

import java.util.Map;

import javax.ws.rs.GET;

import javax.ws.rs.POST;

import javax.ws.rs.Path;

import javax.ws.rs.Produces;

import javax.ws.rs.QueryParam;

import javax.ws.rs.core.MediaType;

import javax.ws.rs.core.Response;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Component;

import org.springframework.web.bind.annotation.RequestBody;

import com.alibaba.fastjson.JSONObject;

import com.github.pagehelper.PageInfo;

import com.arp.common.rpcloud.commonapi.model.CloudContainerAssets;

import com.arp.common.rpcloud.commonapi.util.DateUtil;

import com.arp.common.rpcloud.remote.bean.ResponseBean;

import com.arp.common.rpcloud.remote.bean.container.ContainerDetailInfoBean;

import com.arp.common.rpcloud.remote.bean.container.CreateContainerParam;

import com.arp.common.rpcloud.remote.bean.container.MigrationHistory;

import com.arp.common.rpcloud.remote.bean.container.OutTimeContainer;

import com.arp.common.rpcloud.remote.bean.container.PhysicalComputerBean;

import com.arp.service.rpcloud.container.service.ContainerBusinessService;

import com.arp.service.rpcloud.container.service.CreateContainerService;

import com.arp.service.rpcloud.container.util.ContainerErrorCode;

import com.arp.util.ResponseUtil;

import com.wordnik.swagger.annotations.Api;

import com.wordnik.swagger.annotations.ApiOperation;

import com.wordnik.swagger.annotations.ApiParam;

/\*\*

\* =======================================================

\*

\* @Version ：0.0.1

\* @Description ：容器业务层接口resource

\* ========================================================

\*/

@Path("/businessContainer")

@Api(value = "/businessContainer", description = "container业务服务", produces = MediaType.APPLICATION\_JSON)

@Produces(MediaType.APPLICATION\_JSON)

@Component

public class ContainerBusinessResource {

private static final Logger logger = LoggerFactory.getLogger(ContainerBusinessResource.class);

@Autowired

private ContainerBusinessService containerBusinessService;

@Autowired

private CreateContainerService createContainerService;

@POST

@Path("/selectHostList")

@ApiOperation(value = "带条件查询的列表",httpMethod = "POST", notes = "带条件查询的列表")

public ResponseBean<PageInfo<Map<String, Object>>> selectHostList(

@ApiParam(name = "userId",value = "用户id") @QueryParam("userId") String userId,

@ApiParam(name = "mobile",value = "用户手机号码") @QueryParam("mobile") String mobile,

@ApiParam(name = "status",value = "操作类型： 1 开机 ，2 关机， 3 删除 ，4 挂起 ， 5 恢复挂起") @QueryParam("status") Integer type,

@ApiParam(name = "phyIp",value = "物理机ip") @QueryParam("phyIp") String phyIp,

@ApiParam(name = "containerId",value = "容器id") @QueryParam("containerId") String containerId,

@ApiParam(name = "webname",value = "主机名称") @QueryParam("webname") String name,

@ApiParam(name = "hostIp",value = "主机ip") @QueryParam("hostIp") String hostIp,

@ApiParam(name = "label",value = "标签") @QueryParam("label") Integer label,

@ApiParam(name = "keyword",value = "关键字") @QueryParam("keyword") String keyword,

@ApiParam(name = "page",value = "页码",required = true) @QueryParam("page") Integer page,

@ApiParam(name = "pageNum",value = "每页数量",required = true) @QueryParam("pageNum") Integer pageNum){

logger.info("容器列表页 userId:{}",userId);

ResponseBean<PageInfo<Map<String, Object>>> pageInfoResponseBean = null;

try {

pageInfoResponseBean = containerBusinessService.selectHostList(userId, mobile, type, phyIp, containerId, name, hostIp,label, keyword, page, pageNum);

} catch (Exception e) {

logger.error("底层请求异常",e);

pageInfoResponseBean = new ResponseBean<>();

pageInfoResponseBean.setCodeAndMsg(ContainerErrorCode.code\_100102.getCode(),ContainerErrorCode.code\_100102.getMsg());

}

return pageInfoResponseBean;

}

@POST

@Path("/updateContainerConfig")

@ApiOperation(value = "动态更新容器配置",httpMethod = "POST", notes = "动态更新容器配置，更新cpu和内存")

public ResponseBean<Void> updateContainerConfig(@ApiParam(name = "containerId",value = "containerId",required = true ) @QueryParam("containerId") String containerId,

@ApiParam(name = "cpu",value = "cpu数量",required = true) @QueryParam("cpu") Integer cpu,

@ApiParam(name = "memory",value = "内存大小M", required = true) @QueryParam("memory") Integer memory){

logger.info("更新容器配置 "+containerId+" "+cpu + " "+ memory);

ResponseBean<Void> responseBean = null;

try {

responseBean = createContainerService.updateContainerConfigCost(containerId,cpu,memory);

} catch (Exception e) {

logger.error("底层请求异常",e);

responseBean = new ResponseBean<>();

responseBean.setCodeAndMsg(ContainerErrorCode.CODE\_100204.getCode(),ContainerErrorCode.CODE\_100204.getMsg());

}

return responseBean;

}

@GET

@Path("/getContainerDetailInfo")

@ApiOperation(value = "查询容器详细信息",httpMethod = "GET", notes = "查询容器详细信息,参数id是容器主键")

public ResponseBean<ContainerDetailInfoBean> getContainerDetailInfo(@ApiParam(name = "containerId",value = "containerId",required = true ) @QueryParam("containerId") Integer containerId){

logger.info("更新容器配置 "+containerId);

ResponseBean<ContainerDetailInfoBean> responseBean = null;

try {

responseBean = createContainerService.containerDetailInfo(containerId);

} catch (Exception e) {

logger.error("底层请求异常",e);

responseBean = new ResponseBean<>();

responseBean.setCodeAndMsg(ContainerErrorCode.CODE\_100204.getCode(),ContainerErrorCode.CODE\_100204.getMsg());

}

return responseBean;

}

@POST

@Path("/updateContainerConfigCallback")

@ApiOperation(value = "动态扩展容器配置回调",httpMethod = "POST", notes = "动态扩展容器配置回调")

public void updateContainerConfigCallback(

@ApiParam(name = "result",value = "回调参数",required = true ) @RequestBody JSONObject result

){

this.createContainerService.updateContainerConfigCallback(result);

}

@POST

@Path("/isLittleThen")

@ApiOperation(value = "查询配置是否小于当前配置",httpMethod = "POST", notes = "查询配置是否小于当前配置，动态修改配置不能往小改，只能往大改")

public ResponseBean<Boolean> isLittleThen(

@ApiParam(name = "containerId",value = "containerId",required = true ) @QueryParam("containerId") String containerId,

@ApiParam(name = "cpu",value = "cpu数量",required = true) @QueryParam("cpu") Integer cpu,

@ApiParam(name = "memory",value = "内存大小M", required = true) @QueryParam("memory") Integer memory){

logger.info("更新容器配置 "+containerId+" "+cpu + " "+ memory);

ResponseBean<Boolean> responseBean = new ResponseBean<>();

boolean littleThen = createContainerService.isLittleThen(cpu, memory, containerId);

responseBean.setData(littleThen);

responseBean.setCodeAndMsg(ContainerErrorCode.SUCCESS.getCode(),ContainerErrorCode.SUCCESS.getMsg());

return responseBean;

}

@POST

@Path("/operationContainer")

@ApiOperation(value = "容器对应操作",httpMethod = "POST", notes = "操作类型： 1 开机 ，2 关机， 3 删除 ，4 挂起 ， 5 恢复挂起， 10 彻底删除")

public ResponseBean<Void> operationContainer(

@ApiParam(name = "containerId",value = "containerId",required = true ) @QueryParam("containerId") String containerId,

@ApiParam(name = "type",value = "type操作类型",required = true) @QueryParam("type") Integer type,

@ApiParam(name = "isAdmin",value = "前端还是管理端",required = true) @QueryParam("isAdmin") Integer isAdmin){

logger.info("容器对应操作 "+containerId+" "+type );

ResponseBean<Void> responseBean = createContainerService.operationContainer(containerId, type,isAdmin,0);

return responseBean;

}

@POST

@Path("/operationContainerCallback")

@ApiOperation(value = "容器操作回调",httpMethod = "POST", notes = "容器操作回调")

public void operationContainerCallback(

@ApiParam(name = "result",value = "回调参数",required = true ) @RequestBody JSONObject result

){

this.createContainerService.operationContainerCallback(result);

}

@POST

@Path("/operationContainerList")

@ApiOperation(value = "批量操作容器",httpMethod = "POST", notes = "批量操作容器1 开机 ，2 关机， 3 删除 ，4 挂起 ， 5 恢复挂起 ， 6 恢复删除 ， 7创建出错，10 彻底删除")

public ResponseBean<Void> operationContainerList(

@ApiParam(name = "containerId",value = "containerId列表",required = true ) @QueryParam("containerId") List<String> containerIdList,

@ApiParam(name = "type",value = "type操作类型",required = true) @QueryParam("type") Integer type,

@ApiParam(name = "isAdmin",value = "是前端还是管理端",required = true) @QueryParam("isAdmin") Integer isAdmin){

//logger.info("容器对应操作 "+containerIdList+" "+type );

ResponseBean<Void> responseBean = createContainerService.operationContainerList(containerIdList, type,isAdmin,0);

return responseBean;

}

@POST

@Path("/createContainer")

@ApiOperation(value = "创建主机",httpMethod = "POST", notes = "创建主机")

public Response createContainer(

@ApiParam(name = "param",value = "创建参数",required = true ) @RequestBody CreateContainerParam param,

@ApiParam(name = "userId",value = "用户id", required = true) @QueryParam("userId") String userId

){

ResponseBean<Void> result = this.createContainerService.createContainers(param, userId);

return ResponseUtil.getOkResponse(result);

}

@POST

@Path("/createContainerActive")

@ApiOperation(value = "创建主机回调",httpMethod = "POST", notes = "创建主机回调")

public void createContainerActive(

@ApiParam(name = "result",value = "回调参数",required = true ) @RequestBody JSONObject result

){

this.createContainerService.createContainerActive(result);

}

@POST

@Path("/cloneContainerCallback")

@ApiOperation(value = "克隆主机回调",httpMethod = "POST", notes = "克隆主机回调")

public void cloneContainerCallback(

@ApiParam(name = "result",value = "回调参数",required = true ) @RequestBody JSONObject result

){

this.createContainerService.cloneContainerCallback(result);

}

@POST

@Path("/commitImageCloneCallback")

@ApiOperation(value = "克隆主机提交镜像回调",httpMethod = "POST", notes = "克隆主机提交镜像回调")

public void commitImageCloneCallback(

@ApiParam(name = "result",value = "回调参数",required = true ) @RequestBody JSONObject result

){

this.createContainerService.commitImageCloneCallback(result);

}

@POST

@Path("/createGatewayContainer")

@ApiOperation(value = "创建网关容器",httpMethod = "POST", notes = "创建网关容器")

public ResponseBean<Void> createGatewayContainer(

@ApiParam(name = "name",value = "网关名", required = true) @QueryParam("name") String name,

@ApiParam(name = "publicIpInfoId",value = "公网ip info的主键", required = true) @QueryParam("publicIpInfoId") String publicIpInfoId,

@ApiParam(name = "userId",value = "用户id", required = true) @QueryParam("userId") String userId,

@ApiParam(name = "month",value = "时长") @QueryParam("month") Integer month

){

return this.containerBusinessService.createGatewayContainer(name, publicIpInfoId, userId, month);

}

@POST

@Path("/createGatewayContainerActive")

@ApiOperation(value = "创建网关容器回调",httpMethod = "POST", notes = "创建网关容器回调")

public void createGatewayContainerActive(

@ApiParam(name = "result",value = "回调参数",required = true ) @RequestBody JSONObject result

){

this.createContainerService.createContainerActive(result);

}

@POST

@Path("/getAssetCostGateway")

@ApiOperation(value = "计算网关资产费用", httpMethod = "POST", notes = "创建网关容器")

public ResponseBean<BigDecimal> getAssetCostGateway(

@ApiParam(required = true, name = "gateway", value = "gateway") @QueryParam("gateway") Integer gateway,

@ApiParam(required = true, name = "months", value = "months") @QueryParam("months") Integer months

) {

return this.containerBusinessService.getAssetCostGateway(gateway, months);

}

@POST

@Path("/cloneContainer")

@ApiOperation(value = "克隆容器",httpMethod = "POST", notes = "克隆容器")

public ResponseBean<String> cloneContainer(

@ApiParam(name = "containerId",value = "容器id", required = true) @QueryParam("containerId") String containerId,

@ApiParam(name = "month",value = "month 月份", required = true) @QueryParam("month") Integer month

){

return this.createContainerService.cloneContainer(containerId,month);

//return ResponseUtil.getOkResponse(result);

}

@POST

@Path("/getAllContainers")

@ApiOperation(value = "查询用户下的所有非网关容器",httpMethod = "POST", notes = "查询用户下的所有非网关容器")

public ResponseBean<List<CloudContainerAssets>> getAllContainers(

@ApiParam(name = "userId",value = "用户id", required = true) @QueryParam("userId") String userId,

@ApiParam(name = "containerId",value = "容器id") @QueryParam("containerId") String containerId,

@ApiParam(name = "webname",value = "容器名称") @QueryParam("webname") String webname

){

return containerBusinessService.getAllContainers(userId, containerId, webname);

}

@POST

@Path("/deleteGatewayContainer")

@ApiOperation(value = "删除网关容器网关容器",httpMethod = "POST", notes = "删除网关容器网关容器")

public ResponseBean<Void> deleteGatewayContainer(

@ApiParam(name = "containerId",value = "网关容器id", required = true) @QueryParam("containerId") String containerId,

@ApiParam(name = "isAdmin",value = "是否管理员", required = true) @QueryParam("isAdmin") boolean isAdmin

){

return this.containerBusinessService.deleteGatewayContainer(containerId, isAdmin);

//return ResponseUtil.getOkResponse(result);

}

@POST

@Path("/deleteGatewayContainerActive")

@ApiOperation(value = "删除网关回调",httpMethod = "POST", notes = "删除网关回调")

public void deleteGatewayContainerActive(

@ApiParam(name = "result",value = "回调参数",required = true ) @RequestBody JSONObject result

){

this.containerBusinessService.deleteGatewayContainerActive(result);

}

@POST

@Path("/migrateContainerCallback")

@ApiOperation(value = "迁移回调",httpMethod = "POST", notes = "提交镜像的回调")

public void migrateContainerCallback(

@ApiParam(name = "result",value = "回调参数",required = true ) @RequestBody JSONObject result

){

this.createContainerService.migrateContainerCallback(result);

}

@POST

@Path("/migrateContainerCallback1")

@ApiOperation(value = "迁移回调1",httpMethod = "POST", notes = "创建的回调")

public void migrateContainerCallback1(

@ApiParam(name = "result",value = "回调参数",required = true ) @RequestBody JSONObject result

){

this.createContainerService.migrateContainerCallback1(result);

}

//---------------迁移接口----------------

@POST

@Path("/migrateContainer")

@ApiOperation(value = "迁移容器",httpMethod = "POST", notes = "迁移容器")

public ResponseBean<List<ResponseBean<Void>>> migrateContainer(

@ApiParam(name = "containerIdList",value = "容器id数组", required = true) @QueryParam("containerIdList")List<String> containerIdList,

@ApiParam(name = "serviceId",value = "物理机的id", required = true) @QueryParam("serviceId")String serviceId) {

return this.containerBusinessService.migrateContainer(containerIdList, serviceId);

//return ResponseUtil.getOkResponse(responseBean);

}

@GET

@Path("/migrationHistory")

@ApiOperation(value = "迁移容器历史",httpMethod = "GET", notes = "迁移容器历史列表")

public ResponseBean<PageInfo<MigrationHistory>> migrationHistory(

@ApiParam(name = "containerId",value = "容器id，长字符串") @QueryParam("containerId")String containerId,

@ApiParam(name = "migrateStatus",value = "迁移状态") @QueryParam("migrateStatus")Integer migrateStatus,

@ApiParam(name = "startDate",value = "查询开始时间") @QueryParam("startDate")String startDate,

@ApiParam(name = "endDate",value = "查询结束时间") @QueryParam("endDate")String endDate,

@ApiParam(name = "page",value = "页码",required = true) @QueryParam("page")Integer page,

@ApiParam(name = "pageSize",value = "每页数量",required = true) @QueryParam("pageSize")Integer pageSize

) {

return this.createContainerService.migrationHistory(containerId,migrateStatus,startDate,endDate,page,pageSize);

}

@GET

@Path("/queryPhyComputer")

@ApiOperation(value = "需要迁移的列表",httpMethod = "GET", notes = "需要迁移的列表")

public ResponseBean<List<PhysicalComputerBean>> queryPhyComputer(

@ApiParam(name = "containerIp",value = "容器ip") @QueryParam("containerIp")String containerIp){

List<PhysicalComputerBean> phyComputer = this.containerBusinessService.queryPhyComputer(containerIp);

ResponseBean<List<PhysicalComputerBean>> responseBean = new ResponseBean<>();

responseBean.setData(phyComputer);

responseBean.setCodeAndMsg(ContainerErrorCode.SUCCESS.getCode(),ContainerErrorCode.SUCCESS.getMsg());

return responseBean;

}

//设置启动脚本

@POST

@Path("/setStartScript")

@ApiOperation(value = "设置启动脚本",httpMethod = "POST", notes = "设置启动脚本")

public ResponseBean<Void> setStartScript(

@ApiParam(name = "containerId",value = "容器id", required = true) @QueryParam("containerId")String containerId,

@ApiParam(name = "script",value = "脚本", required = true) @QueryParam("script")String script) {

return this.createContainerService.setStartScript(containerId, script);

//return ResponseUtil.getOkResponse(responseBean);

}

@POST

@Path("/setStartScriptCallback")

@ApiOperation(value = "设置启动脚本回调",httpMethod = "POST", notes = "设置启动脚本回调")

public void setStartScriptCallback(

@ApiParam(name = "result",value = "回调参数",required = true ) @RequestBody JSONObject result

){

this.createContainerService.setStartScriptCallback(result);

}

@POST

@Path("/getContainerById")

@ApiOperation(value = "查询容器信息",httpMethod = "POST", notes = "通过id查询信息")

public ResponseBean<CloudContainerAssets> getContainerById(

@ApiParam(name = "containerId",value = "容器ip") @QueryParam("containerId")String containerId){

CloudContainerAssets containerById = this.containerBusinessService.getContainerById(containerId);

ResponseBean<CloudContainerAssets> responseBean = new ResponseBean<>();

responseBean.setCodeAndMsg(ContainerErrorCode.SUCCESS.getCode(),ContainerErrorCode.SUCCESS.getMsg());

responseBean.setData(containerById);

return responseBean;

}

@POST

@Path("/editWebname")

@ApiOperation(value = "更新webname", httpMethod = "POST", notes = "更新webname")

public ResponseBean<Void> editWebname (

@ApiParam(required = true, name = "id", value = "容器主键") @QueryParam("id") String id,

@ApiParam(required = true, name = "webName", value = "webName") @QueryParam("webName") String webName

){

return containerBusinessService.editWebname(id,webName);

}

@POST

@Path("/editWebnameByContainerId")

@ApiOperation(value = "更新webname", httpMethod = "POST", notes = "更新webname,通过容器id更新webname")

public ResponseBean<Void> editWebnameByContainerId (

@ApiParam(required = true, name = "containerId", value = "容器id") @QueryParam("containerId") String containerId,

@ApiParam(required = true, name = "webName", value = "webName") @QueryParam("webName") String webName

){

return containerBusinessService.editWebnameByContainerId(containerId,webName);

}

// @POST

// @Path("/clearRecycled")

// @ApiOperation(value = "清空回收站", httpMethod = "POST", notes = "清空回收站，彻底清空回收站里的容器和存储。需要询问用户是否删除VHD镜像")

// public ResponseBean<Void> clearRecycled(

// @ApiParam(required = true, name = "userId", value = "userId") @QueryParam("userId") String userId,

// @ApiParam(required = true, name = "isDelVhd", value = "isDelVhd") @QueryParam("isDelVhd") boolean isDelVhd) {

// return containerBusinessService.clearRecycled(userId, isDelVhd);

// }

@POST

@Path("/checkWebname")

@ApiOperation(value = "检查webname是否重复", httpMethod = "POST", notes = "检查webname是否重复 true 为已经存在")

public Response checkWebname(

@ApiParam(required = true, name = "webname", value = "主机名") @QueryParam("webname") String webname,

@ApiParam(required = true, name = "userId", value = "用户id") @QueryParam("userId") String userId) {

ResponseBean<Boolean> result = this.containerBusinessService.checkWebname(webname, userId,null);

return ResponseUtil.getOkResponse(result);

}

// @POST

// @Path("/mountVolume")

// @ApiOperation(value = "动态挂载块存储", httpMethod = "POST", notes = "动态挂载块存储")

// public Response mountVolume(

// @ApiParam(required = true, name = "cloudContainerId", value = "容器主键id") @QueryParam("cloudContainerId") Integer cloudContainerId,

// @ApiParam(required = true, name = "vhdImageId", value = "vhd镜像主键id") @QueryParam("vhdImageId")Integer vhdImageId,

// @ApiParam(required = true, name = "isFormat", value = "是否格式化") @QueryParam("isFormat")boolean isFormat) {

// return ResponseUtil.getOkResponse(containerBusinessService.mountVolume(cloudContainerId,vhdImageId,isFormat));

// // return null;

// }

@POST

@Path("/mountVolumeBlock")

@ApiOperation(value = "动态挂载块存储", httpMethod = "POST", notes = "动态挂载块存储,此接口不创建块设备,默认有块设备")

public Response mountVolumeBlock(

@ApiParam(required = true, name = "cloudContainerId", value = "容器主键id") @QueryParam("cloudContainerId") String cloudContainerId,

@ApiParam(required = true, name = "vhdImageId", value = "vhd镜像主键id") @QueryParam("vhdImageId")String vhdImageId,

@ApiParam(required = true, name = "isFormat", value = "挂载的时候是否格式化") @QueryParam("isFormat")Boolean isFormat) {

return ResponseUtil.getOkResponse(createContainerService.mountVolumeBlock(cloudContainerId,vhdImageId,isFormat));

// return null;

}

@POST

@Path("/umountVolume")

@ApiOperation(value = "卸载块存储", httpMethod = "POST", notes = "卸载块存储")

public Response umountVolume(

@ApiParam(required = true, name = "cloudContainerId", value = "容器主键id") @QueryParam("cloudContainerId") String cloudContainerId,

@ApiParam(required = true, name = "vhdImageId", value = "vhd镜像主键id") @QueryParam("vhdImageId") String vhdImageId){

return ResponseUtil.getOkResponse(createContainerService.umountVolume(cloudContainerId,vhdImageId));

// return null;

}

@POST

@Path("/mountVolumeBlockCallback")

@ApiOperation(value = "动态挂载块设备回调",httpMethod = "POST", notes = "动态挂载块设备回调")

public void mountVolumeBlockCallback(

@ApiParam(name = "result",value = "回调参数",required = true ) @RequestBody JSONObject result

){

this.createContainerService.mountVolumeBlockCallback(result);

}

@POST

@Path("/umountVolumeCallback")

@ApiOperation(value = "动态卸载块设备回调",httpMethod = "POST", notes = "动态卸载块设备回调")

public void umountVolumeCallback(

@ApiParam(name = "result",value = "回调参数",required = true ) @RequestBody JSONObject result

){

this.createContainerService.umountVolumeCallback(result);

}

@POST

@Path("/getGatewayList")

@ApiOperation(value = "查询网关容器列表",httpMethod = "POST", notes = "查询网关容器列表")

public ResponseBean<List<Map<String, Object>>> getGatewayList(

@ApiParam(name = "userId", value = "用户id") @QueryParam("userId") String userId,

@ApiParam(name = "keywork", value = "查询关键字") @QueryParam("keywork") String keywork,

@ApiParam(name = "createTime", value = "创建日期") @QueryParam("createTime") String createTime,

@ApiParam(name = "mobile", value = "所属用户") @QueryParam("mobile") String mobile

){

ResponseBean<List<Map<String, Object>>> result = this.containerBusinessService.getGatewayList(userId, keywork, mobile);

return result;

}

// @POST

// @Path("/umountVolumeDelBlock")

// @ApiOperation(value = "卸载块存储", httpMethod = "POST", notes = "卸载块存储，bool参数是否删除块设备")

// public Response umountVolumeDelBlock(

// @ApiParam(required = true, name = "cloudContainerId", value = "容器主键id") @QueryParam("cloudContainerId") Integer cloudContainerId,

// @ApiParam(required = true, name = "vhdImageId", value = "vhd镜像主键id") @QueryParam("vhdImageId")Integer vhdImageId,

// @ApiParam(required = true, name = "isDelBlock", value = "是否删除块设备") @QueryParam("isDelBlock")boolean isDelBlock){

// return ResponseUtil.getOkResponse(containerBusinessService.umountVolume(cloudContainerId,vhdImageId,isDelBlock));

// }

@POST

@Path("/getGatewayPageList")

@ApiOperation(value = "查询网关容器列表",httpMethod = "POST", notes = "查询网关容器列表")

public ResponseBean<PageInfo<Map<String, Object>>> getGatewayList(

@ApiParam(name = "page", value = "页号") @QueryParam(value = "page") Integer page,

@ApiParam(name = "pageNum", value = "每页几条") @QueryParam(value = "pageNum") Integer pageNum,

@ApiParam(name = "userId", value = "用户id") @QueryParam("userId") String userId,

@ApiParam(name = "keywork", value = "查询关键字") @QueryParam("keywork") String keywork,

@ApiParam(name = "startTime", value = "开始日期") @QueryParam("startTime") String startTime,

@ApiParam(name = "endTime", value = "结束日期") @QueryParam("endTime") String endTime,

@ApiParam(name = "mobile", value = "所属用户") @QueryParam("mobile") String mobile

){

ResponseBean<PageInfo<Map<String, Object>>> result = this.containerBusinessService.getGatewayList(page, pageNum, userId, keywork, startTime, endTime, mobile);

return result;

}

@POST

@Path("/getContainerPrice")

@ApiOperation(value = "价钱",httpMethod = "POST", notes = "计算创建容器的价钱")

public ResponseBean<BigDecimal> getContainerPrice(

@ApiParam(name = "param",value = "创建参数",required = true ) @RequestBody CreateContainerParam param,

@ApiParam(name = "time",value = "time", required = true) @QueryParam("time") Long time

){

Date startDate = new Date();

Date endDate = DateUtil.addMonth(startDate, time.intValue());

Long timeMinute = (endDate.getTime()-startDate.getTime())/(1000\*60);//分钟数

ResponseBean<BigDecimal> responseBean = new ResponseBean<>();

BigDecimal containerPrice = this.createContainerService.getContainerPrice(param, timeMinute);

responseBean.setData(containerPrice);

return responseBean;

}

@POST

@Path("/updateContainerPwd")

@ApiOperation(value = "修改容器密码",httpMethod = "POST", notes = "修改容器密码")

public ResponseBean<Void> updateContainerPwd(

@ApiParam(name = "id",value = "容器id, int型",required = true ) @QueryParam("id") String id,

@ApiParam(name = "pwd",value = "pwd", required = true) @QueryParam("pwd") String pwd

){

return this.createContainerService.updateContainerPwd(id, pwd);

}

@POST

@Path("/updateContainerPwdCallback")

@ApiOperation(value = "修改容器密码回调",httpMethod = "POST", notes = "修改容器密码回调")

public void updateContainerPwdCallback(

@ApiParam(name = "result",value = "回调参数",required = true ) @RequestBody JSONObject result

){

this.createContainerService.updateContainerPwdCallback(result);

}

@POST

@Path("/getExpansionContainerPrice")

@ApiOperation(value = "扩容价格", httpMethod = "POST", notes = "扩容价格")

public ResponseBean<BigDecimal> getExpansionContainerPrice(

@ApiParam(name = "containerId", value = "containerId", required = true) @QueryParam("containerId") String containerId,

@ApiParam(name = "cpu", value = "cpu数量", required = true) @QueryParam("cpu") Integer cpu,

@ApiParam(name = "memory", value = "内存大小M", required = true) @QueryParam("memory") Integer memory

) {

return createContainerService.getExpansionContainerPrice(containerId, cpu,memory);

}

@POST

@Path("/getContainerRenew")

@ApiOperation(value = "主机续费", httpMethod = "POST", notes = "续费价格查询")

public ResponseBean<BigDecimal> getContainerRenew(

@ApiParam(name = "id", value = "id", required = true) @QueryParam("id") String id,

@ApiParam(name = "months", value = "months", required = true) @QueryParam("months") Integer months

) {

return createContainerService.getContainerRenew(id, months);

}

@POST

@Path("/getGatewayExpansionCostPurchase")

@ApiOperation(value = "网关续费续费价格", httpMethod = "POST", notes = "续费价格")

public ResponseBean<BigDecimal> getGatewayExpansionCostPurchase(

@ApiParam(name = "id", value = "id", required = true) @QueryParam("id") String id,

@ApiParam(name = "months", value = "months", required = true) @QueryParam("months") Integer months

) {

return containerBusinessService.getGatewayExpansionCostPurchase(id, months);

}

@POST

@Path("/umountShareVolume")

@ApiOperation(value = "解除容器和共享存储的对应关系", httpMethod = "POST", notes = "解除容器和共享存储的对应关系")

public ResponseBean<Void> umountShareVolume(

@ApiParam(name = "volumeId", value = "volumeId", required = true) @QueryParam("volumeId") String volumeId,

@ApiParam(name = "isTimeOut", value = "isTimeOut 是否是过期,过期自动删除不通知用户. 1是,0不是", required = true) @QueryParam("isTimeOut") Integer isTimeOut

) {

return createContainerService.umountShareVolume(volumeId, isTimeOut);

}

@POST

@Path("/umountShareVolumeCallback")

@ApiOperation(value = "解除容器和共享存储的对应关系回调", httpMethod = "POST", notes = "解除容器和共享存储的对应关系回调方法")

public void umountShareVolumeCallback(

@ApiParam(name = "result",value = "回调参数",required = true ) @RequestBody JSONObject result

) {

createContainerService.umountShareVolumeCallback(result);

}

@GET

@Path("/findOutTimeContainers")

@ApiOperation(value = "查询过期容器", httpMethod = "GET", notes = "查询过期容器")

public ResponseBean<List<OutTimeContainer>> findOutTimeContainers(

@ApiParam(name = "date", value = "date 相对时间", required = true) @QueryParam("date") Long date,

@ApiParam(name = "type", value = "type -1 要提醒的, 0 过期的, 1要回收的", required = true) @QueryParam("type") Integer type,

@ApiParam(name = "isGateway", value = "是否是网关", required = true) @QueryParam("isGateway") Integer isGateway

) {

return createContainerService.findOutTimeContainers(new Date(date),type,isGateway);

}

@POST

@Path("/getwayRenew")

@ApiOperation(value = "网关续费", httpMethod = "POST", notes = "网关续费")

public ResponseBean<Void> getwayRenew(

@ApiParam(name = "id", value = "id", required = true) @QueryParam("id") String id,

@ApiParam(name = "month", value = "month", required = true) @QueryParam("month") Integer month

) {

return containerBusinessService.getwayRenew(id, month);

}

@GET

@Path("/delJunkDate")

@ApiOperation(value = "删除垃圾数据", httpMethod = "GET", notes = "如果容器在物理机底层删除,则删除垃圾数据,释放资源")

public ResponseBean<Void> delJunkDate(

@ApiParam(name = "containerId", value = "容器id,长id字符串", required = true) @QueryParam("containerId") String containerId

) {

return createContainerService.delJunkDate(containerId);

}

@POST

@Path("/updateUserDns")

@ApiOperation(value = "更新容器的dns分组", httpMethod = "POST", notes = "更新容器的dns分组")

public ResponseBean<Void> updateUserDns(

@ApiParam(name = "containerId", value = "containerId,容器的主键", required = true) @QueryParam("containerId") String containerId,

@ApiParam(name = "dnsId", value = "dnsId dns的主键", required = true) @QueryParam("dnsId") String dnsId

) {

return createContainerService.updateUserDns(containerId, dnsId);

}

@POST

@Path("/updateUserDnsCallback")

@ApiOperation(value = "更新容器dns分组回调", httpMethod = "POST", notes = "更新容器dns分组回调")

public void updateUserDnsCallback(

@ApiParam(name = "result",value = "回调参数",required = true ) @RequestBody JSONObject result

) {

createContainerService.updateUserDnsCallback(result);

}

@POST

@Path("/batchUpdateUserDns")

@ApiOperation(value = "批量更新容器的dns分组", httpMethod = "POST", notes = "批量更新容器的dns分组")

public ResponseBean<Void> batchUpdateUserDns(

@ApiParam(name = "containerIds", value = "containerIds,容器的主键", required = true) @QueryParam("containerIds") List<String> containerIds,

@ApiParam(name = "dnsId", value = "dnsId dns的主键", required = true) @QueryParam("dnsId") String dnsId

) {

return createContainerService.batchUpdateUserDns(containerIds, dnsId);

}

@POST

@Path("/batchSetStartScript")

@ApiOperation(value = "批量设置容器启动脚本", httpMethod = "POST", notes = "批量设置容器启动脚本")

public ResponseBean<Void> batchSetStartScript(

@ApiParam(name = "containerIds", value = "containerIds,长字符串", required = true) @QueryParam("containerIds") List<String> containerIds,

@ApiParam(name = "script", value = "脚本内容", required = true) @QueryParam("script") String script

) {

return createContainerService.batchSetStartScript(containerIds, script);

}

@POST

@Path("/startGateway")

@ApiOperation(value = "网关开机", httpMethod = "POST", notes = "网关开机")

public ResponseBean<Void> startGateway(

@ApiParam(name = "id", value = "containerId", required = true) @QueryParam("id") String id

) {

return createContainerService.startGateway(id);

}

@POST

@Path("/startGatewayActive")

@ApiOperation(value = "网关开机回调",httpMethod = "POST", notes = "网关开机回调")

public void startGatewayActive(

@ApiParam(name = "result",value = "回调参数",required = true ) @RequestBody JSONObject result

){

this.createContainerService.startGatewayActive(result);

}

/\*\*

\* dns字段已经删除

\*/

// @POST

// @Path("/dealwithOldDnsData")

// @ApiOperation(value = "处理dns分组模块上线之前的数据,更新dnsId为已有的数据", httpMethod = "POST", notes = "dns")

// public ResponseBean<Void> dealwithOldDnsData(

// ) {

// return createContainerService.dealwithOldDnsData();

// }

}

package com.arp.service.rpcloud.container.resource;

import com.arp.common.rpcloud.commonapi.model.CloudContainerAssets;

import com.arp.common.rpcloud.remote.bean.ResponseBean;

import com.arp.service.rpcloud.container.service.ContainerBusinessService;

import com.arp.service.rpcloud.container.service.ContainerService;

import com.arp.service.rpcloud.container.util.ContainerErrorCode;

import com.wordnik.swagger.annotations.Api;

import com.wordnik.swagger.annotations.ApiOperation;

import com.wordnik.swagger.annotations.ApiParam;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Component;

import javax.ws.rs.POST;

import javax.ws.rs.Path;

import javax.ws.rs.Produces;

import javax.ws.rs.QueryParam;

import javax.ws.rs.core.MediaType;

/\*\*

\*

\* =======================================================

\*

\* @Version ：0.0.1

\* @Description ：容器基础层接口

\* ========================================================

\*/

@Path("/container")

@Api(value = "/container", description = "container服务", produces = MediaType.APPLICATION\_JSON)

@Produces(MediaType.APPLICATION\_JSON)

@Component

public class ContainerResource {

private static final Logger logger = LoggerFactory.getLogger(ContainerResource.class);

@Autowired

private ContainerService containerService;

@Autowired

private ContainerBusinessService containerBusinessService;

// @POST

// @Path("/createContainer")

// @ApiOperation(value = "创建container",httpMethod = "POST", notes = "创建container")

// public ResponseBean<String> createContainer(@ApiParam(name = "image",value = "完整镜像地址",required = true ) @QueryParam("image") String image,

// @ApiParam(name = "cpu",value = "cpu数量",required = true) @QueryParam("cpu") int cpu,

// @ApiParam(name = "memory",value = "内存数量",required = true) @QueryParam("memory") int memory,

// @ApiParam(name = "major",value = "存储设备的版本号",required = true) @QueryParam("major") String major,

// @ApiParam(name = "hostname",value = "主机名称（可为空）",required = true) @QueryParam("hostname") String hostname,

// @ApiParam(name = "devName",value = "容器外存储路径（块存储）",required = true) @QueryParam("devName") String devName,

// @ApiParam(name = "userLocation",value = "用户容器内的存储挂载点（块存储）",required = true) @QueryParam("userLocation") String userLocation,

// @ApiParam(name = "volumesInfo",value = "容器的存储设备（文件存储），字典形式(key代表用户创建的文件存储名称，value代表用户需要挂在容器内的路径)",required = false) @QueryParam("volumesInfo") Map<String, Object> volumesInfo,

// @ApiParam(name = "deviceReadBps",value = "文件读速度",required = true) @QueryParam("deviceReadBps") String deviceReadBps,

// @ApiParam(name = "deviceWriteBps",value = "文件写速度",required = true) @QueryParam("deviceWriteBps") String deviceWriteBps,

// @ApiParam(name = "deviceReadIops",value = "io读速度",required = true) @QueryParam("deviceReadIops") String deviceReadIops,

// @ApiParam(name = "deviceWriteIops",value = "io写速度",required = true) @QueryParam("deviceWriteIops") String deviceWriteIops,

// @ApiParam(name = "dns",value = "dns地址，可写多个",required = true) @QueryParam("dns") List<Object> dns,

// @ApiParam(name = "capAdd",value = "容器需要的特殊权限，现在支持的：SYS\_ADMIN,NET\_ADMIN等。 特殊支持fuse，针对云盘的特殊需求传入FUSE",required = false) @QueryParam("capAdd") List<Object> capAdd,

// @ApiParam(name = "containerLabel",value = "特殊容器标示（如gateway 网关容器），可以不传或传-1",required = false) @QueryParam("containerLabel") String containerLabel,

// @ApiParam(name = "serviceId",value = "分配到哪台物理机的serviceId",required = true) @QueryParam("serviceId") String serviceId

// ) {

// logger.info("创建容器");

// ResponseBean<String> response = null;

// try {

// response = containerService.createContainer(image,cpu,memory,major,hostname,devName,userLocation,volumesInfo,deviceReadBps,deviceWriteBps,

// deviceReadIops,deviceWriteIops,dns,capAdd,containerLabel,serviceId);

// } catch (Exception e) {

// logger.error("底层调用异常");

// response = new ResponseBean<>();

// response.setCodeAndMsg(ContainerErrorCode.code\_100101.getCode(),ContainerErrorCode.code\_100101.getMsg());

// }

// return response;

// }

/\*\*

\* 02

\* @param containerId 容器id

\* @param type 操作类型： 0 创建中， 1 开机 ，2 关机， 3 删除 ，4 挂起 ， 5 恢复挂起 ， 6 恢复删除 ， 7创建出错，8 迁移中， 9 重启 ，10 彻底删除

\* @return

\*/

@POST

@Path("/operationContainer")

@ApiOperation(value = "创建container",httpMethod = "POST", notes = "创建container")

public ResponseBean<Void> operationContainer(@ApiParam(name = "containerId",value = "容器id",required = true ) @QueryParam("containerId") String containerId,

@ApiParam(name = "type",value = "操作类型： 0 创建中， 1 开机 ，2 关机， 3 删除 ，4 挂起 ， 5 恢复挂起 ， 6 恢复删除 ， 7创建出错，8 迁移中， 9 重启 ，10 彻底删除", required = true) @QueryParam("type") int type,

@ApiParam(name = "serviceId",value = "分配到哪台物理机的serviceId",required = true) @QueryParam("serviceId") String serviceId) {

logger.info("容器操作"+type+" "+containerId);

ResponseBean<Void> response = null;

try {

CloudContainerAssets containerById = containerBusinessService.getContainerById(containerId);

response = containerService.operationContainer(containerId, type,containerById.getStatus(), serviceId);

} catch (Exception e) {

logger.error("底层请求异常");

response = new ResponseBean<>();

response.setCodeAndMsg(ContainerErrorCode.code\_100102.getCode(),ContainerErrorCode.code\_100102.getMsg());

}

return response;

}

}

package com.arp.service.rpcloud.container.service;

import java.math.BigDecimal;

import java.util.ArrayList;

import java.util.Collections;

import java.util.Date;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import java.util.concurrent.ExecutorService;

import javax.annotation.Resource;

import com.arp.common.rpcloud.commonapi.util.TransactionUtil;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.amqp.rabbit.core.RabbitTemplate;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import com.alibaba.fastjson.JSONObject;

import com.github.pagehelper.PageHelper;

import com.github.pagehelper.PageInfo;

import com.arp.common.rpcloud.commonapi.bean.NoticeBean;

import com.arp.common.rpcloud.commonapi.bean.RabbitmqParam;

import com.arp.common.rpcloud.commonapi.constants.Constants;

import com.arp.common.rpcloud.commonapi.mapper.CloudContainerAssetsMapper;

import com.arp.common.rpcloud.commonapi.mapper.CloudNetworkDeviceMapper;

import com.arp.common.rpcloud.commonapi.mapper.CloudVolumeContainerContactMapper;

import com.arp.common.rpcloud.commonapi.mapper.IntrIpParaMapper;

import com.arp.common.rpcloud.commonapi.mapper.IpMapper;

import com.arp.common.rpcloud.commonapi.mapper.PublicIpInfoMapper;

import com.arp.common.rpcloud.commonapi.mapper.RegistryImageMapper;

import com.arp.common.rpcloud.commonapi.mapper.ServerAssetsMapper;

import com.arp.common.rpcloud.commonapi.mapper.StorageVolumeMapper;

import com.arp.common.rpcloud.commonapi.mapper.TapBlockMapper;

import com.arp.common.rpcloud.commonapi.mapper.VhdImageMapper;

import com.arp.common.rpcloud.commonapi.model.CloudContainerAssets;

import com.arp.common.rpcloud.commonapi.model.CloudContainerAssetsExample;

import com.arp.common.rpcloud.commonapi.model.CloudNetworkDevice;

import com.arp.common.rpcloud.commonapi.model.CloudNetworkDeviceExample;

import com.arp.common.rpcloud.commonapi.model.IntrIpPara;

import com.arp.common.rpcloud.commonapi.model.PublicIpInfo;

import com.arp.common.rpcloud.commonapi.model.PublicIpInfoExample;

import com.arp.common.rpcloud.commonapi.model.RegistryImage;

import com.arp.common.rpcloud.commonapi.model.ServerAssets;

import com.arp.common.rpcloud.commonapi.model.ServerAssetsExample;

import com.arp.common.rpcloud.commonapi.model.TapBlock;

import com.arp.common.rpcloud.commonapi.model.VhdImage;

import com.arp.common.rpcloud.commonapi.util.DateUtil;

import com.arp.common.rpcloud.remote.bean.ResponseBean;

import com.arp.common.rpcloud.remote.bean.container.CreateContainerParam;

import com.arp.common.rpcloud.remote.bean.container.MigrateContainer;

import com.arp.common.rpcloud.remote.bean.container.PhysicalComputerBean;

import com.arp.common.rpcloud.remote.service.ip.IntranetIpRemoteService;

import com.arp.common.rpcloud.remote.service.network.NetworkBusinessRemoteService;

import com.arp.common.rpcloud.remote.service.scheduler.SchedulerRemoteService;

import com.arp.common.rpcloud.remote.service.transaction.AccountRemoteService;

import com.arp.common.rpcloud.remote.service.transaction.AssetCostRemoteService;

import com.arp.service.rpcloud.container.mapper.ContainerMapper;

import com.arp.service.rpcloud.container.mapper.CreateContainerMapper;

import com.arp.service.rpcloud.container.mapper.GatewayBusinessMapper;

import com.arp.service.rpcloud.container.mapper.RegistryImageBusinessMapper;

import com.arp.service.rpcloud.container.util.ContainerErrorCode;

import com.arp.service.rpcloud.container.util.ThreadPool;

/\*\*

\* =======================================================

\*

\* @Version ：0.0.1

\* @Description ：容器业务逻辑类

\* ========================================================

\*/

@Service

public class ContainerBusinessService {

private static final Logger logger = LoggerFactory.getLogger(ContainerBusinessService.class);

public static final Integer NORMAL\_CONTAINER = 0;//创建普通容器标志

public static final Integer GATEWAY\_CONTAINER = 1;//创建网关容器标志

public static final String IS\_CLUSTER = "0";//非集群

public static final String blockdata = "/blockdata";//块存储挂载点

@Autowired

private SchedulerRemoteService schedulerRemoteService;

@Autowired

private CloudContainerAssetsMapper cloudContainerAssetsMapper;

@Autowired

private NetworkBusinessRemoteService networkBusinessRemoteService;

@Autowired

private ContainerMapper containerMapper;

@Autowired

private StorageVolumeMapper storageVolumeMapper;

@Autowired

private TapBlockMapper tapBlockMapper;

@Autowired

private IpMapper ipMapper;

@Autowired

private PublicIpInfoMapper publicIpInfoMapper;

@Autowired

private RegistryImageMapper registryImageMapper;

@Autowired

private RegistryImageBusinessMapper registryImageBusinessMapper;

@Autowired

private ContainerService containerService;

@Autowired

private IntranetIpRemoteService intranetIpRemoteService;

@Autowired

private CloudNetworkDeviceMapper cloudNetworkDeviceMapper;

@Autowired

private ServerAssetsMapper serverAssetsMapper;

@Autowired

private CreateContainerMapper createContainerMapper;

@Autowired

private IntrIpParaMapper intrIpParaMapper;

@Autowired

private CloudVolumeContainerContactMapper cloudVolumeContainerContactMapper;

@Autowired

private VhdImageMapper vhdImageMapper;

@Resource(name = "gatewayParam")

private CreateContainerParam gatewayParam;

@Resource(name = "vhdPath")

private String vhdPath;

@Resource(name = "imgname")

private String imgname;

@Resource(name = "mkfs")

private String mkfs;

@Resource(name = "volumePath")

private String volumePath;

@Resource(name = "volumeName")

private String volumeName;

@Resource(name = "userLocation")

private String userLocation;

@Resource(name = "bridgeIn")

private String bridgeIn;

@Resource(name = "bridgeOut")

private String bridgeOut;

@Autowired

private CreateContainerService createContainerService;

@Autowired

private RabbitTemplate rabbitTemplate;

@Autowired

private GatewayBusinessMapper gatewayBusinessMapper;

@Autowired

private AssetCostRemoteService assetCostRemoteService;

@Autowired

private AccountRemoteService accountRemoteService;

private void setResult(ResponseBean<String> result, CloudContainerAssets container, String errorCode, String errorMsg) {

container.setStatus(ContainerService.CONTAINER\_ERROR);

this.cloudContainerAssetsMapper.updateByPrimaryKey(container);

result.setRetCode(errorCode);

result.setRetMsg(errorMsg);

}

/\*\*

\* 设置capAdd权限

\*

\* @param param

\* @param container

\*/

private void setCapAdd(CreateContainerParam param, CloudContainerAssets container) {

if (param.getCapAdd() == null) {

return;

}

String capAdd = setListToString(",", param.getCapAdd());

if (!capAdd.isEmpty())

container.setCapAdd(capAdd);

}

/\*\*

\* 把list转化为字符串，

\*

\* @param tag 分隔符，例如逗号，分号；等

\* @param list list

\* @return

\*/

private String setListToString(String tag, List<Object> list) {

StringBuilder str = new StringBuilder();

if (list.size() > 0) {

for (int i = 0; i < list.size(); i++) {

if (i == list.size() - 1) {

str.append(list.get(i));

} else {

str.append(list.get(i)).append(tag);

}

}

}

return str.toString();

}

/\*\*

\* @param status 主机状态

\* @param phyIp 物理机ip

\* @param containerId id

\* @param webname 服务器名称

\* @param hostIp 主机ip

\* @param label 标签

\* @param keyword 关键字

\* @param page 页号

\* @param pageNum 每页数量（长度）

\* @return ResponseBean

\*/

public ResponseBean<PageInfo<Map<String, Object>>> selectHostList(String userId, String mobile, Integer status, String phyIp, String containerId, String webname,

String hostIp, Integer label, String keyword, Integer page, Integer pageNum) {

Map<String, Object> selectParam = new HashMap<>();

selectParam.put("userId", userId);

selectParam.put("status", status);

selectParam.put("phyIp", phyIp);

selectParam.put("containerId", containerId);

selectParam.put("webname", webname);

selectParam.put("hostIp", hostIp);

selectParam.put("label", label);

selectParam.put("keyword", keyword);

selectParam.put("mobile", mobile);

//selectParam.put("page", page);

//selectParam.put("pageNum", pageNum);

PageHelper.startPage(page, pageNum);

List<Map<String, Object>> hostListInfoBeans = containerMapper.selectHostList(selectParam);

// 添加到期提醒

getEndTimeDay(hostListInfoBeans);

PageInfo<Map<String, Object>> date = new PageInfo<>(hostListInfoBeans);

ResponseBean<PageInfo<Map<String, Object>>> responseBean = new ResponseBean<>();

responseBean.setCodeAndMsg(ContainerErrorCode.SUCCESS.getCode(), ContainerErrorCode.SUCCESS.getMsg());

responseBean.setData(date);

return responseBean;

}

/\*\*

\* 根据containerId查询容器

\*

\* @param containerId

\* @return

\*/

private List<CloudContainerAssets> getCloudContainerAssetss(String containerId) {

CloudContainerAssetsExample cca = new CloudContainerAssetsExample();

cca.createCriteria().andContainerIdEqualTo(containerId);

return cloudContainerAssetsMapper.selectByExample(cca);

}

public CloudContainerAssets getContainerById(String containerId) {

return this.getCloudContainerAssetss(containerId).get(0);

}

/\*\*\*

\* 复制参数，生成克隆的参数。

\* @param cloudContainerAssets

\* @param newImgId

\* @return

\*/

private CreateContainerParam getCreateContainerParam(CloudContainerAssets cloudContainerAssets, String newImgId) {

RegistryImage registryImage = registryImageMapper.selectByPrimaryKey(newImgId);

CreateContainerParam param = new CreateContainerParam();

param.setRegistryImageId(newImgId);

param.setWebName(cloudContainerAssets.getWebname() + "-clone");

String hostName = cloudContainerAssets.getHostName();

if (hostName != null)

hostName = hostName.substring(0, hostName.lastIndexOf("-"));

param.setHostname(hostName);

param.setCpu(cloudContainerAssets.getCpu());

param.setMemory(cloudContainerAssets.getMemory());

param.setImageType(registryImage.getImageType());

param.setContainerLabel("-1");

String capAdd = cloudContainerAssets.getCapAdd();

List<Object> capAddList = getCapAddObjects(capAdd);

param.setCapAdd(capAddList);

return param;

}

private List<Object> getCapAddObjects(String capAdd) {

List<Object> capAddList = new ArrayList<>();

if (capAdd != null && !capAdd.isEmpty()) {

String[] split = capAdd.split(",");

Collections.addAll(capAddList, split);

}

return capAddList;

}

/\*\*

\* 多线程启动迁移

\*

\* @param containerIdList

\* @param serviceId

\* @return

\*/

public ResponseBean<List<ResponseBean<Void>>> migrateContainer(List<String> containerIdList, String serviceId) {

logger.info("serviceId=" + serviceId);

ResponseBean<List<ResponseBean<Void>>> responseBean = new ResponseBean<>();

if (serviceId == null || serviceId.isEmpty()) {

responseBean.setCodeAndMsg(ContainerErrorCode.code\_100602.getCode(), ContainerErrorCode.code\_100602.getMsg());

return responseBean;

}

List<ResponseBean<Void>> rList = new ArrayList<>();

ExecutorService instanse = ThreadPool.POOL.getInstanse();

for (String containerId : containerIdList) {

instanse.execute(() -> {

ResponseBean<Void> responseBean1 = createContainerService.migrateContainer(containerId, serviceId);

logger.info("migration code={},msg={}",responseBean1.getRetCode(),responseBean1.getRetMsg());

rList.add(responseBean1);

});

}

responseBean.setData(rList);

responseBean.setCodeAndMsg(ContainerErrorCode.SUCCESS.getCode(), ContainerErrorCode.SUCCESS.getMsg());

return responseBean;

}

/\*\*

\* 提交容器到新镜像，业务方法

\*

\* @param containerId containerId

\* @return

\*/

public ResponseBean<String> commitNewImage(String containerId) {

ResponseBean<String> responseBean = new ResponseBean<>();

//CloudContainerAssets cloudContainerAssets = cloudContainerAssetsMapper.selectByPrimaryKey(Integer.parseInt(containerId));

List<CloudContainerAssets> cloudContainerAssetss = this.getCloudContainerAssetss(containerId);

CloudContainerAssets cloudContainerAssets = cloudContainerAssetss.get(0);

String registryImageId = cloudContainerAssets.getRegistryImageId();

RegistryImage registryImage = registryImageMapper.selectByPrimaryKey(registryImageId);

//用户提交的镜像添加前缀user\_

int imageCount = registryImageBusinessMapper.countRegistryImage(registryImage.getFunction());

String newImageVersion = String.valueOf("user\_" + cloudContainerAssets.getUserId() + "\_" + imageCount);

ResponseBean<Void> responseBean1 = containerService.commitNewImage(containerId, registryImage.getFunction(), newImageVersion, null, cloudContainerAssets.getServerAssetsId());

if (ContainerErrorCode.SUCCESS.getCode().equals(responseBean1.getRetCode())) {

//版本加一

registryImage.setImageVersion(newImageVersion);

registryImage.setUserId(cloudContainerAssets.getUserId());

registryImage.setParentId(registryImage.getId());

registryImage.setId(null);//id设置为空，入库自增

registryImage.setUpdateTime(new Date());

//registryImage.setUserId(cloudContainerAssets.getUserId());

registryImageMapper.insert(registryImage);

responseBean.setData(registryImage.getId());

responseBean.setCodeAndMsg(ContainerErrorCode.SUCCESS.getCode(), ContainerErrorCode.SUCCESS.getMsg());

} else {

responseBean.setCodeAndMsg(ContainerErrorCode.code\_100501.getCode(), ContainerErrorCode.code\_100501.getMsg());

}

return responseBean;

}

/\*\*

\* 创建网关容器

\*

\* @param publicIpInfoId

\* @param userId

\* @return

\* @Description han.zhifeng

\*/

public ResponseBean<Void> createGatewayContainer(String name, String publicIpInfoId, String userId, Integer month) {

logger.info("创建网关参数：【{}】, 【{}】， 【{}】", name, publicIpInfoId, userId);

ResponseBean<Void> result = new ResponseBean<>();

Date startDate = new Date();

Date endDate = DateUtil.addMonth(startDate, month);

Long time = (endDate.getTime() - startDate.getTime()) / (1000 \* 60);//分钟数

ResponseBean<BigDecimal> amount = assetCostRemoteService.assetCostGateway(1, time);

logger.info("花费共计：【{}】", amount);

ResponseBean<Void> consumeResult = this.accountRemoteService.consume(userId, Constants.ACCOUNT\_TYPE\_BUYGATEWAY, amount.getData().toString(), String.format("类型： 网关， 名称： %s", name));

logger.info("扣费结果【{}】", consumeResult);

if (!consumeResult.getRetCode().equals(ContainerErrorCode.SUCCESS.getCode())) {

logger.info("扣费失败！");

return consumeResult;

}

CreateContainerParam param = gatewayParam.clone();

param.setPublicIpInfoId(publicIpInfoId);

param.setWebName(name);

ResponseBean<IntrIpPara> res = this.intranetIpRemoteService.distIpPara(userId);

logger.info("获取可用ip段结果：【{}】", res);

if (!ContainerErrorCode.SUCCESS.getCode().equals(res.getRetCode())) {

logger.info("获取可用ip段错误");

result.setRetCode(res.getRetCode());

result.setRetMsg(res.getRetMsg());

ResponseBean<Void> rechargeResult = this.accountRemoteService.recharge(userId, Constants.ACCOUNT\_TYPE\_REFUND, amount.toString(), String.format("类型： 网关， 名称： %s， 【创建失败退款】", name));

logger.info("获取可用ip段错误退款结果：【{}】", rechargeResult);

return result;

}

IntrIpPara intrIpPara = res.getData();

param.setIntrIpParaId(intrIpPara.getId());

ResponseBean<Void> r = createContainerService.createContainer(param, userId, 1, Constants.ACTIVE\_TYPE\_CREATE\_GATEWAY, startDate, endDate, time);

if (!ContainerErrorCode.SUCCESS.getCode().equals(res.getRetCode())) {

ResponseBean<Void> rechargeResult = this.accountRemoteService.recharge(userId, Constants.ACCOUNT\_TYPE\_REFUND, amount.toString(), String.format("类型： 网关， 名称： %s， 【创建失败退款】", name));

logger.info("创建网关容器失败，退款结果：【{}】", rechargeResult);

intrIpPara.setUserId(null);

intrIpPara.setStatus(Constants.UNUSED);

this.intrIpParaMapper.updateByPrimaryKey(intrIpPara);

PublicIpInfo ipInfo = new PublicIpInfo();

ipInfo.setId(publicIpInfoId);

ipInfo.setStatus(Constants.UNUSED);

this.publicIpInfoMapper.updateByPrimaryKeySelective(ipInfo);

}

return r;

}

/\*\*

\* 查询用户下的所有非网关容器

\*

\* @param userId

\* @return

\* @Description han.zhifeng

\*/

public ResponseBean<List<CloudContainerAssets>> getAllContainers(String userId, String containerId, String webname) {

ResponseBean<List<CloudContainerAssets>> result = new ResponseBean<>();

CloudContainerAssetsExample example = new CloudContainerAssetsExample();

CloudContainerAssetsExample.Criteria cr = example.createCriteria().andIsGatewayEqualTo(0).andUserIdEqualTo(userId).andStatusEqualTo(1);

if (containerId != null) {

cr.andContainerIdLike("%" + containerId + "%");

}

if (webname != null) {

cr.andWebnameLike("%" + webname + "%");

}

List<CloudContainerAssets> list = this.cloudContainerAssetsMapper.selectByExample(example);

result.setData(list);

result.setRetCode(ContainerErrorCode.SUCCESS.getCode());

result.setRetMsg(ContainerErrorCode.SUCCESS.getMsg());

return result;

}

/\*\*

\* 删除网关容器

\*

\* @param containerId

\* @return

\* @Description han.zhifeng

\*/

public ResponseBean<Void> deleteGatewayContainer(String containerId, boolean isAdmin) {

ResponseBean<Void> result = new ResponseBean<>();

Map<String, Object> param = new HashMap<>();

param.put("id", containerId);

param.put("status", Constants.USED);

int count = this.createContainerMapper.gatewayContainerIpCount(param);

if (count > 0) {

result.setRetCode(ContainerErrorCode.CODE\_100702.getCode());

result.setRetMsg(ContainerErrorCode.CODE\_100702.getMsg());

return result;

}

CloudContainerAssets container = this.cloudContainerAssetsMapper.selectByPrimaryKey(containerId);

if (container == null) {

result.setRetCode(ContainerErrorCode.CODE\_100703.getCode());

result.setRetMsg(ContainerErrorCode.CODE\_100703.getMsg());

return result;

}

CloudNetworkDeviceExample example = new CloudNetworkDeviceExample();

example.createCriteria().andStatusEqualTo(0).andCloudAssetsIdEqualTo(container.getId());

List<CloudNetworkDevice> list = this.cloudNetworkDeviceMapper.selectByExample(example);

List<Map<String, Object>> network = null;

if (!list.isEmpty()) {

network = new ArrayList<>();

for (CloudNetworkDevice device : list) {

Map<String, Object> networkMap = new HashMap<>();

if (device.getType().equals("1")) {

networkMap.put("bridge", this.bridgeOut);

} else {

networkMap.put("bridge", this.bridgeIn);

}

networkMap.put("interface\_name", device.getOuterDevice());

network.add(networkMap);

}

}

Map<String, Object> paramMap = new HashMap<>();

paramMap.put("container\_id", container.getContainerId());

paramMap.put("user\_id", container.getUserId());

paramMap.put("network", network);

Map<String, Object> baseMap = new HashMap<>();

baseMap.put("userId", container.getUserId());

baseMap.put("containerId", container.getId());

RabbitmqParam rp = new RabbitmqParam();

if (isAdmin) {

rp.setActiveType(Constants.ACTIVE\_TYPE\_DELETE\_GATEWAY\_ADMIN);

} else {

rp.setActiveType(Constants.ACTIVE\_TYPE\_DELETE\_GATEWAY);

}

rp.setBaseParam(baseMap);

rp.setMethodType(Constants.DELETE\_CONTAINER\_RESOURCE);

rp.setMethodParam(paramMap);

logger.info("==============");

logger.info("删除网关容器参数：【{}】", rp.toJson());

logger.info("==============");

rabbitTemplate.convertAndSend(Constants.TASK\_QUEUE + container.getServerAssetsId(), rp.toJson());

result.setRetCode(ContainerErrorCode.SUCCESS.getCode());

result.setRetMsg(ContainerErrorCode.SUCCESS.getMsg());

return result;

}

/\*\*

\* 删除网关容器回调

\*

\* @Description han.zhifeng

\*/

public void deleteGatewayContainerActive(JSONObject result) {

logger.info("删除网关容器回调结果：【{}】", result);

JSONObject r = result.getJSONObject("result");

JSONObject baseParam = result.getJSONObject("baseParam");

String userId = baseParam.getString("userId");

String containerId = baseParam.getString("containerId");

if (Constants.SUCCESS.equals(r.getString("status"))) {

CloudContainerAssets container = new CloudContainerAssets();

container.setId(containerId);

container.setStatus(10);

this.cloudContainerAssetsMapper.updateByPrimaryKeySelective(container);

CloudNetworkDeviceExample example = new CloudNetworkDeviceExample();

example.createCriteria().andStatusEqualTo(0).andCloudAssetsIdEqualTo(containerId);

List<CloudNetworkDevice> list = this.cloudNetworkDeviceMapper.selectByExample(example);

for (CloudNetworkDevice device : list) {

device.setStatus(Constants.DELETED);

this.cloudNetworkDeviceMapper.updateByPrimaryKey(device);

if ("2".equals(device.getType())) {//网关

IntrIpPara para = intrIpParaMapper.selectByPrimaryKey(device.getIpId());

if (para != null) {

para.setStatus(Constants.UNUSED);

para.setUserId(null);

para.setUpdateTime(new Date());

this.intrIpParaMapper.updateByPrimaryKey(para);

}

} else if ("1".equals(device.getType())) {//外网

PublicIpInfoExample infoExample = new PublicIpInfoExample();

infoExample.createCriteria().andPublicIpIdEqualTo(device.getIpId());

PublicIpInfo info = new PublicIpInfo();

info.setStatus(Constants.UNUSED);

this.publicIpInfoMapper.updateByExampleSelective(info, infoExample);

}

}

NoticeBean nb = new NoticeBean();

nb.setUserId(userId);

nb.setActiveType(result.getInteger("activeType"));

nb.setRetCode(ContainerErrorCode.SUCCESS.getCode());

nb.setRetMessage(ContainerErrorCode.SUCCESS.getMsg());

rabbitTemplate.convertAndSend(Constants.NOTICE\_WEB\_QUEUE, nb.toJson());

rabbitTemplate.convertAndSend(Constants.NOTICE\_ADMIN\_WEB\_QUEUE, nb.toJson());

} else {

logger.error("删除网关容器失败，错误码：【{}】", r.getJSONObject("error"));

NoticeBean nb = new NoticeBean();

nb.setUserId(userId);

nb.setActiveType(result.getInteger("activeType"));

nb.setRetCode(ContainerErrorCode.CODE\_100701.getCode());

nb.setRetMessage(ContainerErrorCode.CODE\_100701.getMsg());

rabbitTemplate.convertAndSend(Constants.NOTICE\_WEB\_QUEUE, nb.toJson());

rabbitTemplate.convertAndSend(Constants.NOTICE\_ADMIN\_WEB\_QUEUE, nb.toJson());

}

}

/\*\*

\* 主机迁移容器列表

\*

\* @param containerIp ip

\* @return

\*/

public List<PhysicalComputerBean> queryPhyComputer(String containerIp) {

List<PhysicalComputerBean> list = new ArrayList<>();

if (containerIp == null || containerIp.isEmpty()) {

List<ServerAssets> serverAssetss = serverAssetsMapper.selectByExample(new ServerAssetsExample());

for (ServerAssets serverAssets : serverAssetss) {

PhysicalComputerBean pb = new PhysicalComputerBean();

pb.setId(String.valueOf(serverAssets.getServerId()));

pb.setIp(serverAssets.getIp());

//pb.setName(serverAssets.getServerId());

pb.setName(serverAssets.getServerName());

Map<String, Object> param = new HashMap<>();

param.put("serviceId", serverAssets.getServerId());

List<Map<String, Object>> maps = containerMapper.selectMigrateContainer(param);

List<MigrateContainer> containerList = getContainerList(maps);

pb.setContainerList(containerList);

list.add(pb);

}

} else {

Map<String, Object> param = new HashMap<>();

param.put("ip", containerIp);

List<Map<String, Object>> maps = containerMapper.selectMigrateContainer(param);

List<MigrateContainer> containerList = getContainerList(maps);

ServerAssetsExample se = new ServerAssetsExample();

se.createCriteria().andServerIdEqualTo(String.valueOf(maps.get(0).get("serviceId")));

List<ServerAssets> serverAssetss = serverAssetsMapper.selectByExample(se);

ServerAssets serverAssets = serverAssetss.get(0);

PhysicalComputerBean pb = new PhysicalComputerBean();

pb.setIp(serverAssets.getIp());

pb.setId(String.valueOf(serverAssets.getServerId()));

//pb.setName(serverAssets.getServerId());

pb.setName(serverAssets.getServerName());

pb.setContainerList(containerList);

list.add(pb);

}

return list;

}

/\*\*\*

\* 热挂载块存储到容器

\* @param cloudContainerId 容器主键

\* @param containerId 容器id

\* @param serviceId serviceId

\* @param block 块设备

\* @return

\*/

public ResponseBean<Void> bandBlockAndContainer(String cloudContainerId, String containerId, String serviceId, TapBlock block) {

//挂载点现在是 ／blockdata+blockId

String mountPoint = blockdata + block.getVhdImageId();

//挂载

ResponseBean<Void> responseBean = containerService.mountVolume(containerId, mountPoint, block.getDevName(), serviceId);

if (ContainerErrorCode.SUCCESS.getCode().equals(responseBean.getRetCode())) {

block.setServerAssetsId(cloudContainerId);//设置容器id,key 主键

block.setStatus(Constants.USED);//已经使用

VhdImage vhdImage = vhdImageMapper.selectByPrimaryKey(block.getVhdImageId());

vhdImage.setStatus(Constants.USED);

int j = vhdImageMapper.updateByPrimaryKey(vhdImage);

int i = tapBlockMapper.updateByPrimaryKey(block);

if (j == 1 && i == 1) {

logger.debug("----绑定块设备成功----");

} else {

logger.error("绑定块设备，数据库update异常");

responseBean.setCodeAndMsg(ContainerErrorCode.CODE\_101003.getCode(), ContainerErrorCode.CODE\_101003.getMsg());

}

} else {

logger.error(ContainerErrorCode.CODE\_101001.getMsg());

}

return responseBean;

}

/\*\*\*

\* 更新webname

\* @param id

\* @param webName

\* @return

\*/

public ResponseBean<Void> editWebname(String id, String webName) {

CloudContainerAssets cloudContainerAssets = cloudContainerAssetsMapper.selectByPrimaryKey(id);

cloudContainerAssets.setWebname(webName);

ResponseBean<Void> responseBean = new ResponseBean<>();

if (!checkWebname(webName, cloudContainerAssets.getUserId(), id).getData()) {

int i = cloudContainerAssetsMapper.updateByPrimaryKeySelective(cloudContainerAssets);

if (i == 1) {

responseBean.setCodeAndMsg(ContainerErrorCode.SUCCESS.getCode(), ContainerErrorCode.SUCCESS.getMsg());

} else {

responseBean.setCodeAndMsg(ContainerErrorCode.CODE\_101003.getCode(), ContainerErrorCode.CODE\_101003.getMsg());

}

} else {

logger.error("主机名重复");

responseBean.setCodeAndMsg(ContainerErrorCode.CODE\_100308.getCode(), ContainerErrorCode.CODE\_100308.getMsg());

}

return responseBean;

}

/\*\*\*

\* 更新webname

\* @param containerId 容器id

\* @param webName

\* @return

\*/

public ResponseBean<Void> editWebnameByContainerId(String containerId, String webName) {

ResponseBean<Void> responseBean = new ResponseBean<>();

CloudContainerAssets cloudContainerAssets = this.getContainerById(containerId);

cloudContainerAssets.setWebname(webName);

int i = cloudContainerAssetsMapper.updateByPrimaryKeySelective(cloudContainerAssets);

if (i == 1) {

responseBean.setCodeAndMsg(ContainerErrorCode.SUCCESS.getCode(), ContainerErrorCode.SUCCESS.getMsg());

} else {

responseBean.setCodeAndMsg(ContainerErrorCode.CODE\_101003.getCode(), ContainerErrorCode.CODE\_101003.getMsg());

}

return responseBean;

}

public ResponseBean<Boolean> checkWebname(String webname, String userId, String id) {

if (id == null) id = "";

ResponseBean<Boolean> result = new ResponseBean<>();

List<Integer> states = new ArrayList<>();

states.add(1);//开机

states.add(2);//关机

states.add(3);//回收站

CloudContainerAssetsExample example = new CloudContainerAssetsExample();

example.createCriteria().andUserIdEqualTo(userId).andWebnameEqualTo(webname).andStatusIn(states).andIdNotEqualTo(id);

List<CloudContainerAssets> list = this.cloudContainerAssetsMapper.selectByExample(example);

if (list.isEmpty()) {

result.setData(false);

} else {

result.setData(true);

}

return result;

}

private List<MigrateContainer> getContainerList(List<Map<String, Object>> maps) {

List<MigrateContainer> containerList = new ArrayList<>();

for (Map<String, Object> map : maps) {

MigrateContainer mc = new MigrateContainer();

mc.setIp(String.valueOf(map.get("ip")));

mc.setName(String.valueOf(map.get("webname")));

mc.setContainerId(String.valueOf(map.get("containerId")));

containerList.add(mc);

}

return containerList;

}

public ResponseBean<List<Map<String, Object>>> getGatewayList(String userId, String keyword, String mobile) {

ResponseBean<List<Map<String, Object>>> result = new ResponseBean<>();

Map<String, Object> param = new HashMap<>();

param.put("userId", userId);

param.put("keywork", keyword);

param.put("mobile", mobile);

List<Map<String, Object>> list = this.gatewayBusinessMapper.getGatewayList(param);

// 添加到期提醒

getEndTimeDay(list);

result.setRetCode(ContainerErrorCode.SUCCESS.getCode());

result.setRetMsg(ContainerErrorCode.SUCCESS.getMsg());

result.setData(list);

return result;

}

public ResponseBean<PageInfo<Map<String, Object>>> getGatewayList(Integer page, Integer pageNum, String userId, String keyword, String startTime, String endTime, String mobile) {

ResponseBean<PageInfo<Map<String, Object>>> result = new ResponseBean<>();

Map<String, Object> param = new HashMap<>();

param.put("userId", userId);

param.put("keywork", keyword);

if (startTime != null) {

startTime = startTime + " 00:00:00";

param.put("startTime", startTime);

}

if (endTime != null) {

endTime = endTime + " 23:59:59";

param.put("endTime", endTime);

}

param.put("mobile", mobile);

PageHelper.startPage(page, pageNum);

List<Map<String, Object>> list = this.gatewayBusinessMapper.getGatewayList(param);

// 添加到期提醒

getEndTimeDay(list);

PageInfo<Map<String, Object>> info = new PageInfo<>(list);

result.setRetCode(ContainerErrorCode.SUCCESS.getCode());

result.setRetMsg(ContainerErrorCode.SUCCESS.getMsg());

result.setData(info);

return result;

}

private void getEndTimeDay(List<Map<String, Object>> list) {

// 添加到期提醒

for (Map<String, Object> map : list) {

if (Integer.valueOf(map.get("end\_time\_day").toString()) <= 0) {

map.put("end\_time\_day\_str", "已到期");

}

if (Integer.valueOf(map.get("end\_time\_day").toString()) > 0

&& Integer.valueOf(map.get("end\_time\_day").toString()) <= 30) {

map.put("end\_time\_day\_str", map.get("end\_time\_day") + "天后到期");

}

}

}

/\*\*

\* 计算网关购买费用

\*

\* @param gateway 大小

\* @param months 购买时常 (几月)

\* @return

\*/

public ResponseBean<BigDecimal> getAssetCostGateway(Integer gateway, Integer months) {

logger.info("开始计费时间{}, 结束计费时间 {}", TransactionUtil.getStartChargingTime(), TransactionUtil.getEndChargingTime(months));

long m = TransactionUtil.getMinute(TransactionUtil.getStartChargingTime(), TransactionUtil.getEndChargingTime(months));

logger.info("一共{}分钟", m);

ResponseBean<BigDecimal> data = assetCostRemoteService.assetCostGateway(gateway, TransactionUtil.getMinute(TransactionUtil.getStartChargingTime(), TransactionUtil.getEndChargingTime(months)));

logger.info("共{}收费元", data.getData().doubleValue());

return data;

}

/\*\*

\* 网关续费方法

\* @Description

\* @param id

\* @param month

\* @return

\*/

public ResponseBean<Void> getwayRenew(String id, Integer month){

ResponseBean<Void> result = new ResponseBean<>();

CloudContainerAssets containerAssets = this.cloudContainerAssetsMapper.selectByPrimaryKey(id);

Date startDate = containerAssets.getEndTime();

Date endDate = DateUtil.addMonth(startDate, month);

Long time = (endDate.getTime() - startDate.getTime()) / (1000 \* 60);//分钟数

ResponseBean<BigDecimal> amount = assetCostRemoteService.assetCostGateway(1, time);

logger.info("网关续费价格：【{}】", amount.getData());

ResponseBean<Void> consumeResult = this.accountRemoteService.consume(containerAssets.getUserId(), Constants.ACCOUNT\_TYPE\_BUYGATEWAY, amount.toString(), String.format("类型： 网关续费， 名称： %s, 时长：%s个月", containerAssets.getWebname(), month));

logger.info("网关续费扣费结果【{}】", consumeResult);

if (!consumeResult.getRetCode().equals(ContainerErrorCode.SUCCESS.getCode())) {

logger.info("扣费失败！");

return consumeResult;

}

CloudContainerAssets container = new CloudContainerAssets();

container.setId(containerAssets.getId());

container.setEndTime(endDate);

this.cloudContainerAssetsMapper.updateByPrimaryKeySelective(container);

result.setRetCode(ContainerErrorCode.SUCCESS.getCode());

result.setRetMsg(ContainerErrorCode.SUCCESS.getMsg());

return result;

}

/\*\*

\* 网关续费价格查询

\* @param id

\* @param months

\* @return

\*/

public ResponseBean<BigDecimal> getGatewayExpansionCostPurchase(String id, Integer months) {

ResponseBean<BigDecimal> responseBean = new ResponseBean<>();

responseBean.setRetCode(ContainerErrorCode.SUCCESS.getCode());

responseBean.setRetMsg(ContainerErrorCode.SUCCESS.getMsg());

CloudContainerAssets containerAssets = this.cloudContainerAssetsMapper.selectByPrimaryKey(id);

Date startDate = containerAssets.getEndTime();

Date endDate = DateUtil.addMonth(startDate, months);

Long time = (endDate.getTime() - startDate.getTime()) / (1000 \* 60);//分钟数

ResponseBean<BigDecimal> amount = assetCostRemoteService.assetCostGateway(1, time);

logger.info("续费 花费共计：【{}】", amount);

responseBean.setData(amount.getData());

return responseBean;

}

}

package com.arp.service.rpcloud.container.config;

import com.arp.common.rpcloud.commonapi.util.CloudProperty;

import com.arp.common.rpcloud.remote.bean.container.CreateContainerParam;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.bind.RelaxedPropertyResolver;

import org.springframework.context.EnvironmentAware;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.core.env.Environment;

import java.util.ArrayList;

import java.util.List;

@Configuration

public class ContainerConfig implements EnvironmentAware {

private static final String PUBLIC = "pub";

private static final String CONTAINER = "container";

private RelaxedPropertyResolver propResolver;

@Autowired

private CloudProperty property;

@Override

public void setEnvironment(Environment environment) {

propResolver = new RelaxedPropertyResolver(environment, "config.");

}

@Bean(name = "bridgeOut")

public String bridgeOut(){

return property.getProperty(PUBLIC, "bridgeOut");

}

@Bean(name = "bridgeIn")

public String bridgeIn(){

return property.getProperty(PUBLIC, "bridgeOut");

}

@Bean(name = "vhdPath")

public String vhdPath(){

return property.getProperty(CONTAINER, "vhdPath");

}

@Bean(name = "blockdata")

public String blockdata(){

return property.getProperty(CONTAINER, "blockdata");

}

@Bean(name = "imgname")

public String imgname(){

return property.getProperty(CONTAINER, "imgname");

}

@Bean(name = "mkfs")

public String mkfs(){

return property.getProperty(CONTAINER, "mkfs");

}

@Bean(name = "volumePath")

public String volumePath(){

return property.getProperty(CONTAINER, "volumePath");

}

@Bean(name = "volumeName")

public String volumeName(){

return property.getProperty(CONTAINER, "volumeName");

}

@Bean(name = "userLocation")

public String userLocation(){

return property.getProperty(CONTAINER, "userLocation");

}

@Bean(name = "consulServPath")

public String consulServPath(){

return property.getProperty(PUBLIC, "consulServPath");

}

@Bean(name = "zookeeperConnectionString")

public String zookeeperConnectionString(){

return property.getProperty(PUBLIC, "zookeeper");

}

@Bean(name = "openvpnConnIp")

public String openvpnConnIp(){

return property.getProperty(CONTAINER, "openvpnConnIp");

}

@Bean(name = "defaultDNS")

public String defaultDNS(){

return property.getProperty(CONTAINER, "defaultDNS");

}

@Bean(name = "webconsoleNetIP")

public String webconsoleNetIP(){

return property.getProperty(PUBLIC, "webconsoleNetIP");

}

@Bean(name = "webHostName")

public String webHostName(){

return property.getProperty(CONTAINER, "webHostName");

}

@Bean(name = "gatewayParam")

public CreateContainerParam gatewayParam(){

CreateContainerParam param = new CreateContainerParam();

param.setRegistryImageId(property.getProperty(CONTAINER, "gateway.registryImageId"));

param.setCpu(Integer.parseInt(property.getProperty(CONTAINER, "gateway.cpu")));

param.setMemory(Integer.parseInt(property.getProperty(CONTAINER, "gateway.memory")));

param.setImageType(property.getProperty(CONTAINER, "gateway.imageType"));

param.setContainerLabel(property.getProperty(CONTAINER, "gateway.containerLabel"));

param.setCloudStorage(0);

List<Object> dns = new ArrayList<Object>();

dns.add(property.getProperty(CONTAINER, "gateway.dns"));

param.setDns(dns);

List<Object> capAdd = new ArrayList<>();

capAdd.add(property.getProperty(CONTAINER, "gateway.capAdd"));

param.setCapAdd(capAdd);

param.setVolume(0);

param.setCloudStorage(0);

return param;

}

}

package com.arp.service.rpcloud.container.config;

import javax.annotation.Resource;

import org.apache.curator.framework.CuratorFramework;

import org.apache.curator.framework.CuratorFrameworkFactory;

import org.apache.curator.retry.ExponentialBackoffRetry;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

@Configuration

public class ZookeeperConfig {

@Resource(name="zookeeperConnectionString")

private String connectString;

@Bean

public CuratorFramework zkconfig(){

System.out.println("初始化CuratorFramework 。。。");

System.out.println("connectString:" + connectString);

CuratorFramework zk = CuratorFrameworkFactory.

builder()

.connectString(connectString)

.sessionTimeoutMs(5000)

.connectionTimeoutMs(5000)

.retryPolicy(new ExponentialBackoffRetry(1000, 3)).build();

zk.start();

return zk;

}

}

package com.arp.service.rpcloud.container.mapper;

import com.arp.common.rpcloud.remote.bean.container.ContainerDetailInfoBean;

import com.arp.common.rpcloud.remote.bean.container.MigrationHistory;

import com.arp.common.rpcloud.remote.bean.container.OutTimeContainer;

import org.apache.ibatis.annotations.Param;

import org.apache.ibatis.annotations.SelectProvider;

import org.springframework.stereotype.Repository;

import java.util.Date;

import java.util.List;

import java.util.Map;

/\*\*

\* =======================================================

\*

\* @Version ：0.0.1

\* @Description ：容器自定义mapper

\* ========================================================

\*/

@Repository

public interface ContainerMapper {

@SelectProvider(type = ContainerSqlProvider.class, method = "selectHost")

// @Results({

// @Result(column="id", property="id", jdbcType= JdbcType.INTEGER, id=true),

// @Result(column="container\_id", property="container\_id", jdbcType=JdbcType.VARCHAR),

// @Result(column="webname", property="webname", jdbcType=JdbcType.VARCHAR),

// @Result(column="status", property="status", jdbcType=JdbcType.INTEGER),

// @Result(column="subdomain\_id", property="subdomain\_id", jdbcType=JdbcType.INTEGER),

// @Result(column="id", property="id", jdbcType=JdbcType.INTEGER),

// @Result(column="hostname", property="hostname", jdbcType=JdbcType.VARCHAR),

// @Result(column="domain\_head", property="domain\_head", jdbcType=JdbcType.VARCHAR),

// @Result(column="ip", property="ip", jdbcType=JdbcType.VARCHAR),

// @Result(column="image\_name", property="image\_name", jdbcType=JdbcType.VARCHAR),

// @Result(column="registry\_image\_id", property="registry\_image\_id", jdbcType=JdbcType.INTEGER),

// @Result(column="image\_version", property="image\_version", jdbcType=JdbcType.INTEGER),

// @Result(column="cpu", property="cpu", jdbcType=JdbcType.INTEGER),

// @Result(column="memory", property="memory", jdbcType=JdbcType.INTEGER),

// @Result(column="update\_time", property="update\_time", jdbcType=JdbcType.DATE),

// @Result(column="create\_time", property="create\_time", jdbcType=JdbcType.DATE)

// })

List<Map<String,Object>> selectHostList(Map<String,Object> param);

@SelectProvider(type = ContainerSqlProvider.class, method = "selectHost")

int selectHostListCount(Map<String,Object> param);

@SelectProvider(type = ContainerSqlProvider.class, method = "selectMigrateContainer")

List<Map<String,Object>> selectMigrateContainer(Map<String,Object> param);

@SelectProvider(type = ContainerSqlProvider.class, method = "selectContainerInfo")

ContainerDetailInfoBean selectContainerInfo(Integer id);

@SelectProvider(type = ContainerSqlProvider.class, method = "selectMigrationHistory")

List<MigrationHistory> selectMigrationHistory(Map<String,Object> param);

//查询要提醒的,时间在超期前一段时间的

@SelectProvider(type = ContainerSqlProvider.class, method = "queryNoticeContainer")

List<OutTimeContainer> queryNoticeContainer(@Param("nowDate") Date nowDate, @Param("isGateway") Integer isGateway);

//查询过期的,需要做关机操作,所以查找过期且开机的

@SelectProvider(type = ContainerSqlProvider.class, method = "queryOutOfTimeContainer")

List<OutTimeContainer> queryOutOfTimeContainer(@Param("nowDate") Date nowDate, @Param("isGateway") Integer isGateway);

//查找需要回收资源的,需要回收资源,查找超过一定时间且是关机的,且是未回收的

@SelectProvider(type = ContainerSqlProvider.class, method = "queryDestoryContainer")

List<OutTimeContainer> queryDestoryContainer(@Param("nowDate") Date nowDate, @Param("isGateway") Integer isGateway);

}

package com.arp.service.rpcloud.container.mapper;

import java.util.List;

import java.util.Map;

import org.apache.ibatis.annotations.Select;

public interface CreateContainerMapper {

@Select({

"select \* from storage\_volume where id in ( ${ids} )"

})

List<Map<String, Object>> findStorageVolume(Map<String, Object> param);

@Select({

"select count(\*) from ip aa where aa.ip\_para\_id in (",

"select b.ip\_id from cloud\_container\_assets a right join cloud\_network\_device b on a.id = b.cloud\_assets\_id where a.is\_gateway = 1 and a.id=#{id} and b.type =2",

") and aa.status = #{status}"

})

int gatewayContainerIpCount(Map<String, Object> param);

}