Cloudeploy 2.0
Design Document

2016-08-16

Catalogue

Cloudeploy 2.0 System Design

1. Introduction

Cloudeploy 2.0 is an open-source platform for DevOps (Development and Operations). Its functions include deployment, configuration, system scaling, fault-tolerance and so on. With the services of Cloudeploy 2.0, one can easily manage cloud based applications and middleware in a distributed cluster through web browser instead of CLI (command-line interface).

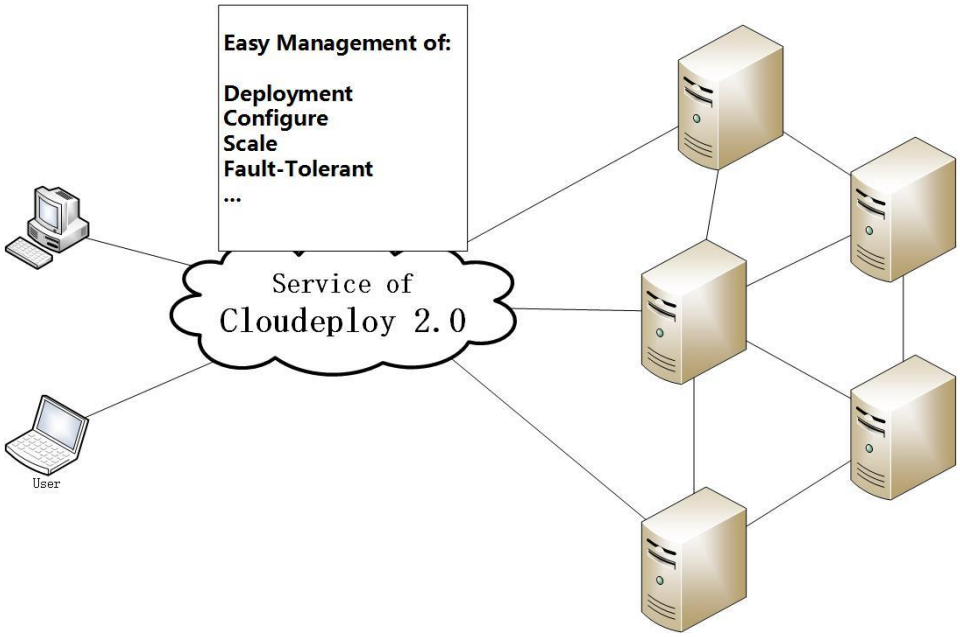

Figure 1.1 System Schematic

1.1 Purpose

This document serves as the blueprint for the software development and implementation of the Cloudeploy 2.0.

Cloudeploy 2.0 supports continuous deployment and configuration for cloud based applications. The functional requirements of Cloudeploy 2.0 are shown in document of “Requirement Analysis of Cloudeploy 2.0”.

2. Architecture

The architecture of the whole system is shown below.

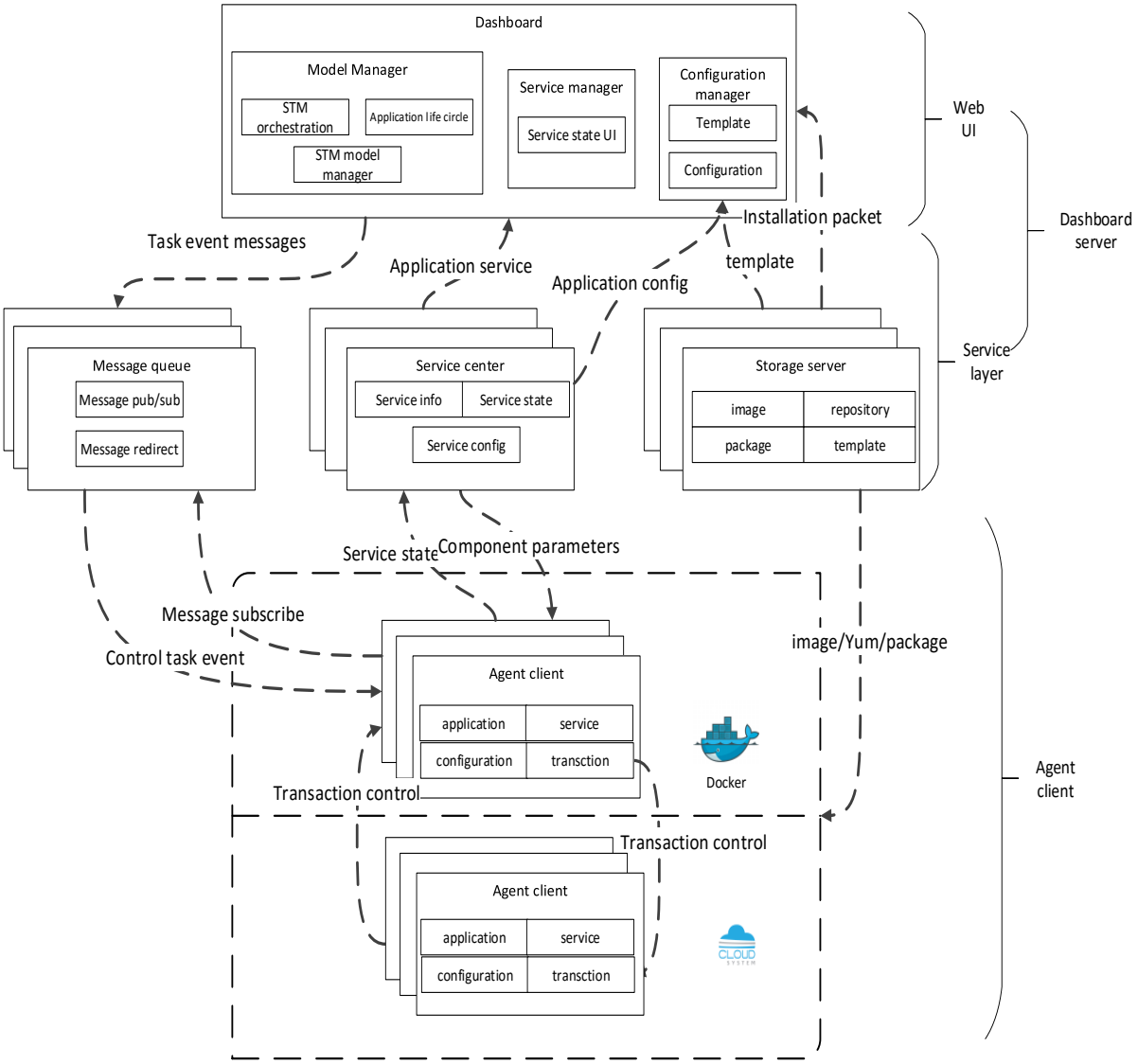

Figure 2.1 System Architecture

The architecture of Cloudeploy 2.0 consists of two parts: the web server and agent client on computers of cluster. Their relations are shown in figure 2.1. And the details are as followed.

Dashboard

Dashboard is an interface between user and system of Cloudeploy 2.0. It manages applications of users. Dashboard consists of three main management: model, configuration and service. Model manager is in charge of modeling task (called service

topology model, STM, we will discuss in detail later) and managing of application life cycle, including creating, updating, removing. Configuration manager handles parameters and configuration templates. Service manager shows the states of applications and its components in real time.

Message queue

Message queue controls communications between dashboard server and agent clients. It can send task event to specific agent computer. Parameters relationship and dependency is also realized in this part.

Service center

Runtime applications and its components become service for users. Service center manages the deployed components and their exposed parameters (called “attributes”). Health check is implemented in this module.

Storage server

This module is in charge of storing data of Cloudeploy 2.0, including configuration files, task packets and installation files. The data in message queue may contains URL that can access files in storage server.

Agent client

This module is running on target computer cluster. It executes task scripts that received from message queue. And then, it generates reports and sends them to Dashboard.

3. Related Skills

Table 3.1 shows the skills that used in Cloudeploy 2.0.

Table 3.1 skills in Cloudeploy 2.0

Roles	skills
Dashboard	JSP+JsPlumb(JavaScript) + SpringMVC
Message queue	Consul + Java
Service center	Consul
Storage server	SpringMVC
Agent client	Java + Thrift + Docker + consul-template

4. Detail Design

4.1. Application Task Description Model

We designed a service based topology model called STM (Service based Topology Model) to describe aspects of tasks (see in “requirement analysis” document), as figure 4.1 shows.

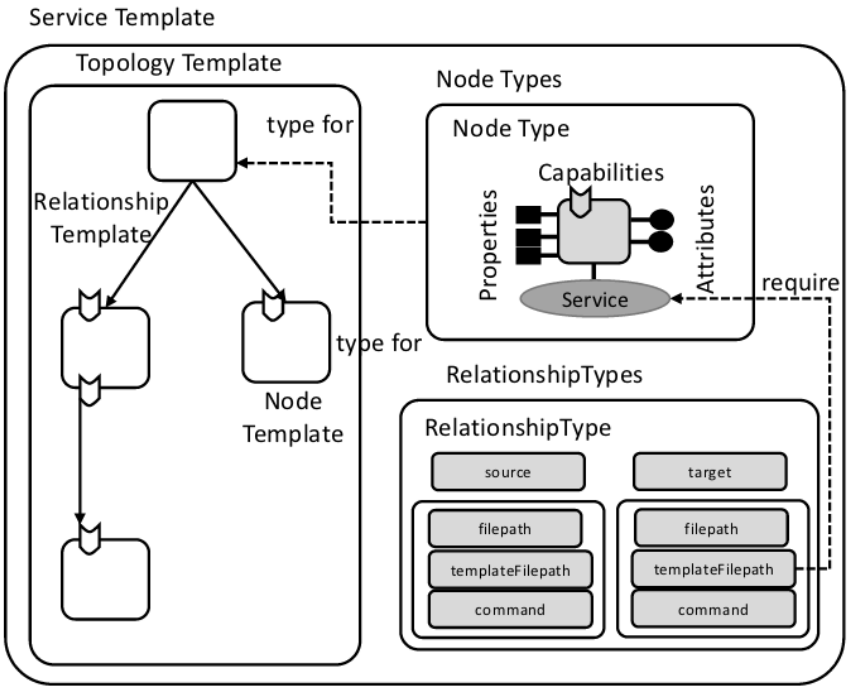

Figure 4.1 Model of STM

Topology template is abstract of application task. It is in shape of directed graph consisting of nodes and edges.

Node-type is an abstract of component type, whose runtime is in form of service. A node-type has properties, attributes and capabilities. Properties are parameters that independent of other components while attributes are parameters that required by other components. And capabilities are functions that a service of component can provide.

Relationship-type is an abstract of relations between components. It has a “source” implying the begin components and a ”target” implying the end

components.

4.2. Database Design

We choose MySQL as database of Cloudeploy 2.0. There are some entities stored in database, including types and instances in the graph of task, applications, components and parameters. So we design database as Figure 4.2 shows. The introductions of tables are as follows.

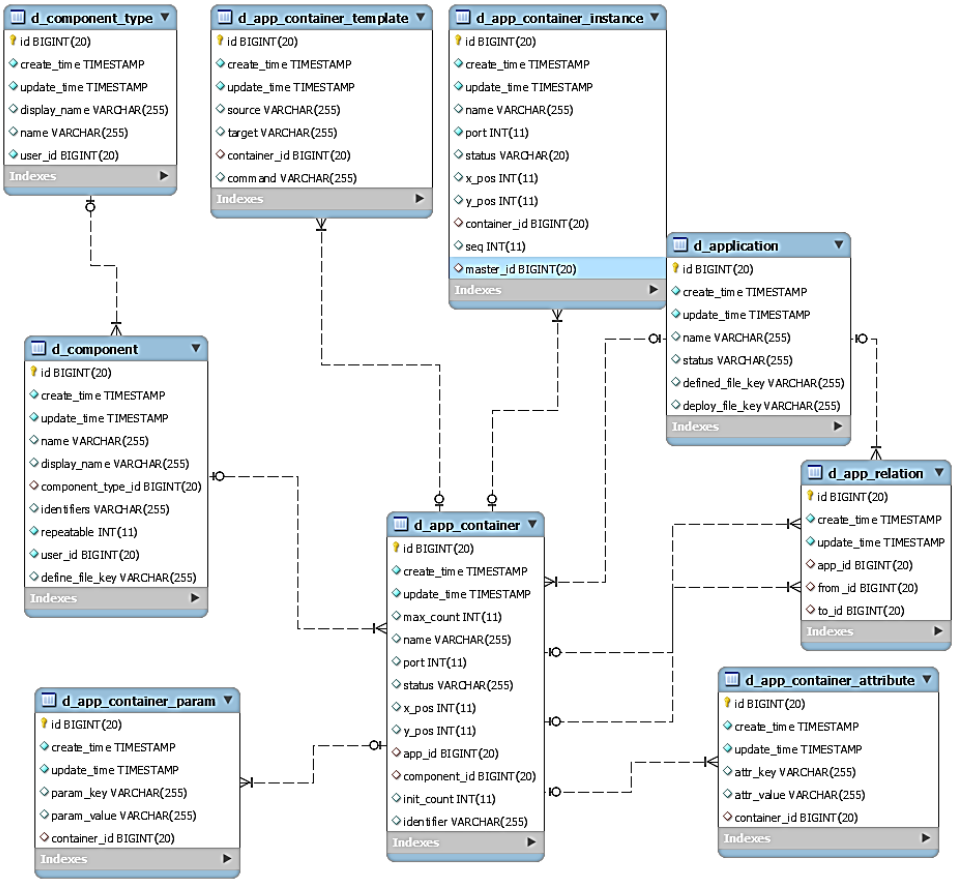

Figure 4.2 Design of Database

d_application: application list. Columns include create-time, name, state, STM file key.

d_app_container: components in application. Columns include name, max_numer, min_number, state, port, and node position in task graph.

d_app_container_instance: instance of components. Columns include name, port, states.

d_component: component types. Columns include name, type, file key definition

d_container_attributes, d_container_param, d_app_relation: the attributes, properties and relations of component instances.

4.3. Dashboard module

Figure 4.3 shows the main relations in this module. Firstly user designs a task on website panel. The elements in the task is modeled as a directed graph specifying the plan (named orchestration) of deployment or configuration. Then the orchestration is extracted and transformed into STM model (mentioned in 4.1) in form of yaml text files which stored in storage server. And after that, there will be a task event sent to message module. The data in task event includes the URL of STM model file.

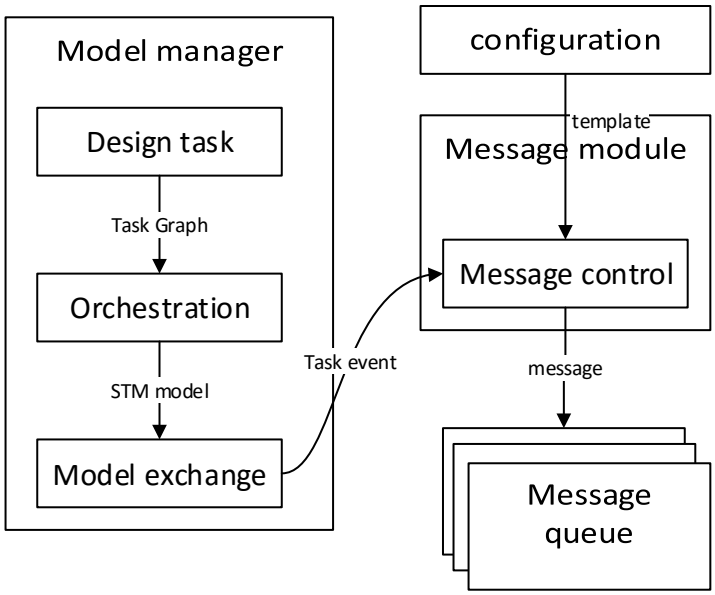

Figure 4.3 Design of Dashboard

4.3.1. Build Task Graph

Since we have defined the description model of task, the point is that how to help user design the task graph visually, easily and friendly. We choose JsPlumb, a JavaScript library, to paint topology graph online.

Nodes on painting panel represent node-type of STM model while edges represent relationship-type. We design a node-type repository which user can add a node by mouse click on. And add an edge by dragging mouse from node to node, as figure 4.4

shows.

For each node in task graph, double click to specify properties and attributes in detail as figure 4.5 shows.

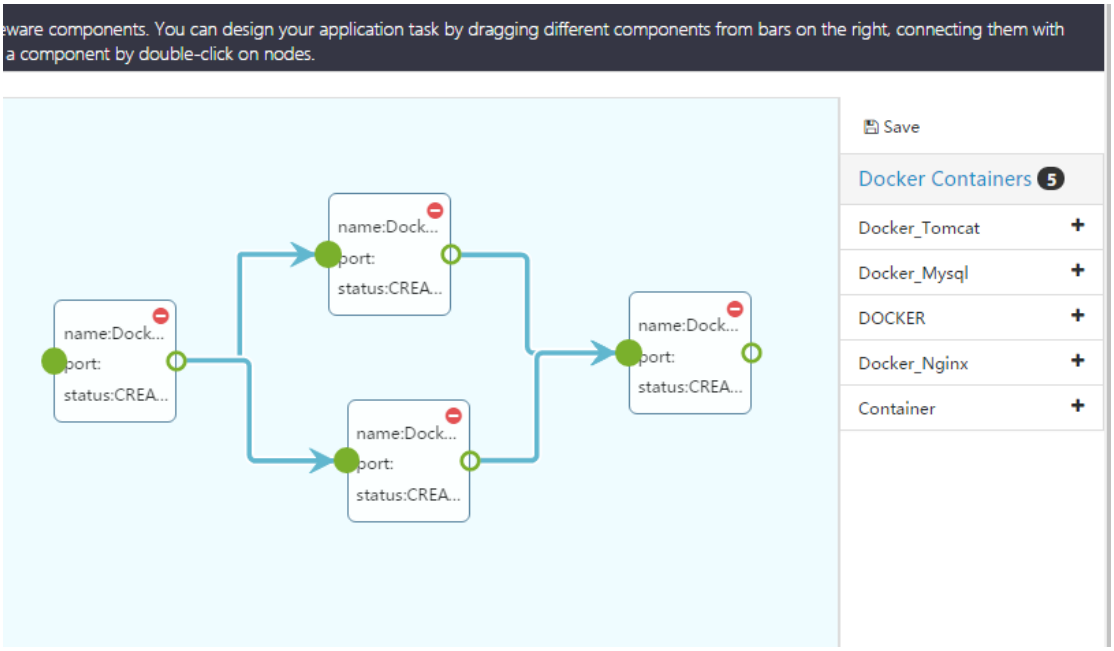

Figure 4.4 Task graph painting panel

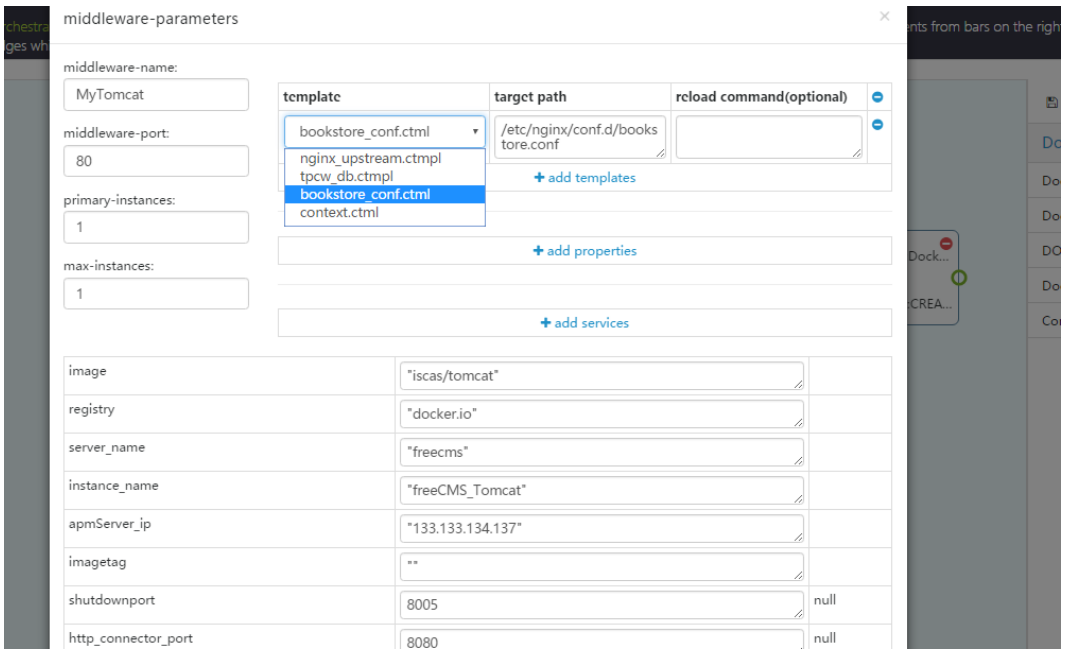

Figure 4.5 Node details

4.3.2. Model exchange

Task graph is in structure of JavaBean, so it will be transferred to STM model files. This process can be described by Figure 4.6. The orchestrator panel and type transfer generate STM model from task graph and store it in storage server.

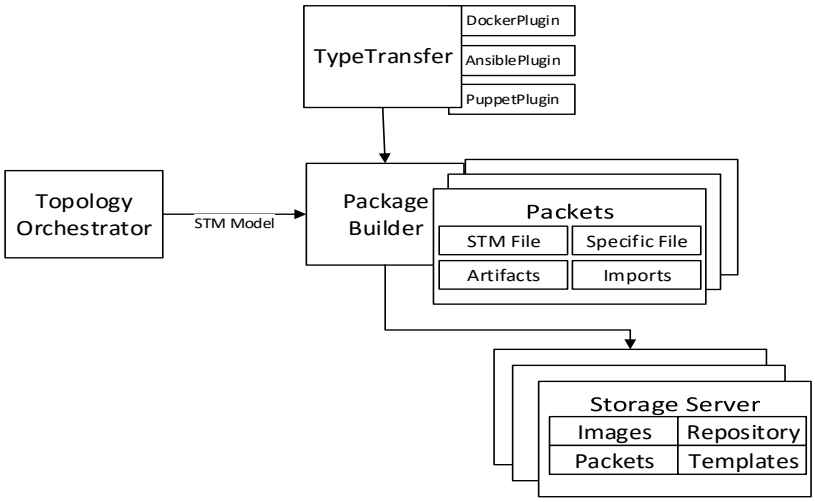

Figure 4.6 process of Model exchange

We traversal the task graph, for each element, identify its type and replace it by proper phrases. And figure 4.7 shows an example of the resulting file of the model transformation.

```
topologytemplate:
 nodetemplates:
 - name: bookstore_mysql
   type: cloudeploy.nodetype.mysql
   min:1
   max:1
 - name: bookstore_tomcat1
   type: cloudeploy.nodetype.tomcat
   min: 1
   max: 10
 - name: bookstore_nginx
   type: cloudeploy.nodetype.nginx
   min:1
   max:1
 relationshiptemplates:
 - name: tomcat_connectTo_mysql
   type: template_relationship
   source: bookstore_tomcat1
   target: bookstore_mysql
   templates:
     - name: context_config_template
       type: tomcat_db_relation
       template: tpcw_db.ctml
       configurationfile:
   /usr/local/tomcat/webapps
         /bookstore-tpcw/META-
INF/context.xml
       command:
     - name: nginx_connectTo_tomcat
       type: nginx_lb_relation
```

Figure 4.7 Example of Result File of Transformation

If one wants to add instances to deployed application, he/she can make it through task painting panel by to adding a copy from of the existing instance s. Then re-deploy the task just like it is a new task.

4.3.3. Scaling

If one want to add instances to deployed application, he/she can make it through task painting panel to copy from existing instances. Then re-deploy the task just like it is new task.

4.4. Message queue

Message includes deployment and configuration event. Message queue is to (1) receive event messages from dashboard server, (2) notify agent clients the arrival of new task event. The message queue is implemented with "consul", an excellent middleware of service discovery and distributed key-value style datacenter. Moreover it has the function of event trigger, which can fire events and notify clients to handle new message.

There are two kind of message on task event: *Node-Event* and *Config-Event*. *Node-Event* is about deployment and *Config-Event* is about configuration. The format of event message is in json.

Node-Event: *'{name: "eventName", target: "nodeId", payload: "data"}'*.

Config-Event: *'{name: "eventName", target: "nodeId", app: "appId", configKey : "key", configValue： "value"}*

And *Node-Event* consist of 4 kinds of specific action event: deploy-event, remove-event, start-event, and stop-event, corresponding to four application operations. In *Node-Event* "payload" is the URL of real task STM model file which can get from storage server through http request.

Figure 4.8 shows the main method communicating with consul in message queue. The implement of these methods are based on the library package of "*com.orbitz.consul*".

```
public interface EventService {
    public void pubDeployEvent(DeployEvent deployEvent);
    public void pubRemoveEvent(RemoveEvent removeEvent);
    public void pubStartEvent(StartEvent startEvent);
    public void pubStopEvent(StopEvent stopEvent);
    public void pubConfigEvent(ConfigEvent configEvent);
}
```

Figure 4.8 Interface in message queue

4.5. **Service center**

Deployed applications and its components become services, which can be accessed through the way of sockets. So here is a module to monitor the state of service center.

The main process in this module is showed in figure 4.9. When user define application task on dashboard and deploy the task, a check commands (there will be a default one) can be declared. And register the applications and components in service center if it succeeds in executing on agents. Then service center will execute the check command periodically, and return the results to dashboard. If there's an exception or an error when checking, there is a warning and an attempt to restart that application or components.

When it fails in restarting, there will be a "stopped" state to tag that service. Then fault-tolerance mechanism will start and dynamically update the components depending on the fail service. The way of "update" is to generate new configuration file which get rid of fail service attributes, and restart this components with the correct configuration.

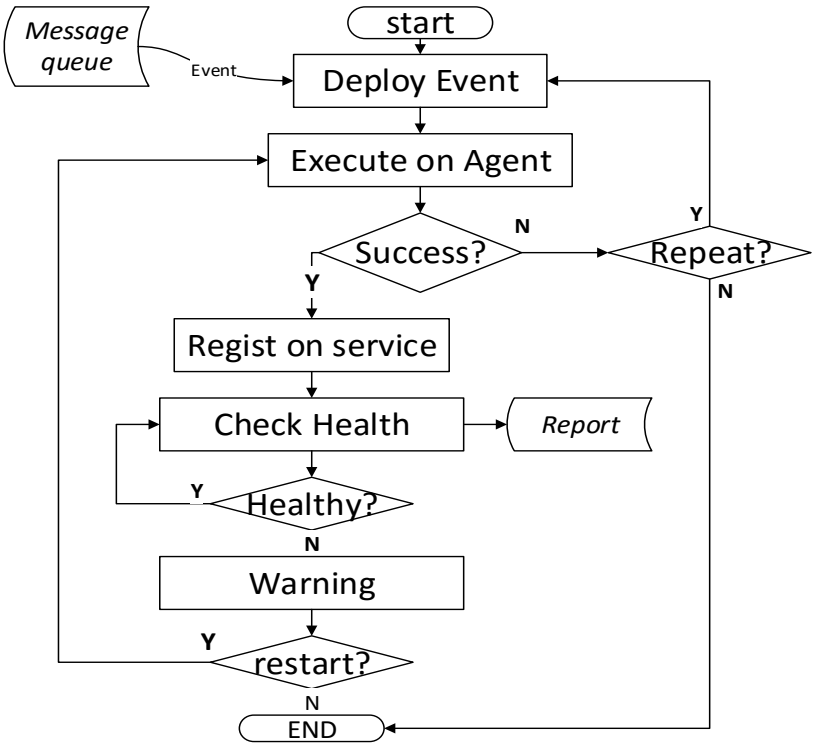

Figure 4.9 Example of Result File of Transformation

The implement of datacenter is based on consul, too. We register key-value pairs of (application-id, state) in consul and update the state with the results of health check. When visiting services center, we fetch the states by applications list, and handle the state data to show in web interface. Figure 4.10 shows the web UI of datacenter. We can see that the running services are in green color, such as consul and TPCWMySQL, while orange color denotes failing services. When click the service, it will show detail information.

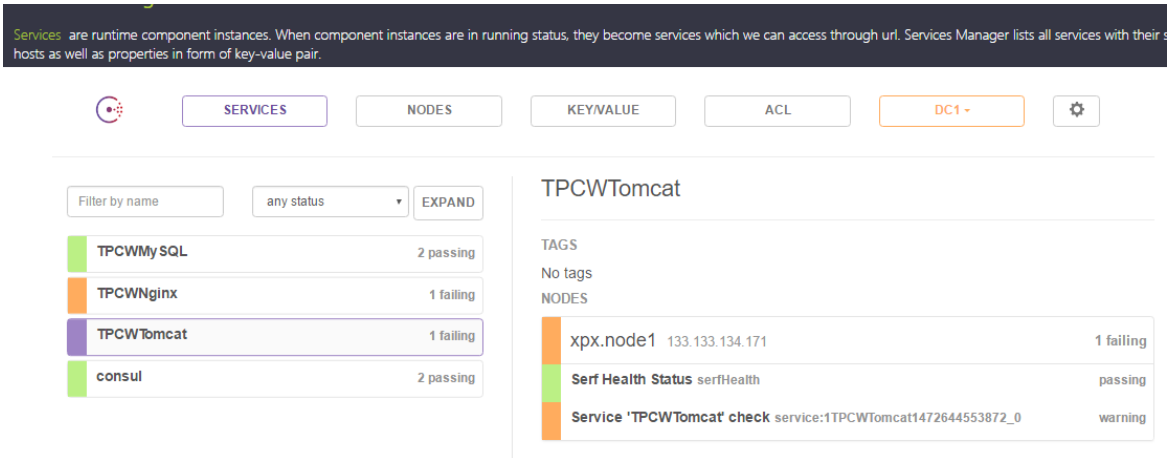

Figure 4.10 Web UI of service center

4.6. **Storage server**

Storage server is the global file data center. Agent or Dashboard can store or fetch different files from it. This module has its independent database structure as figure 4.11 shows.

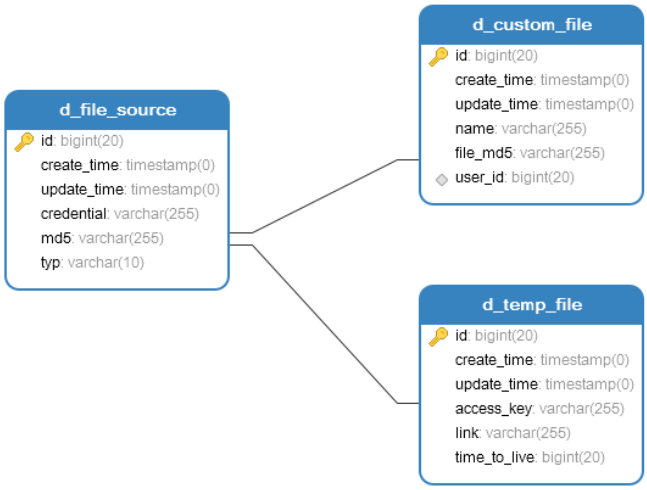

Figure 4.11 Database of Storage server

d_file_source: references to files in file system. Columns include create-time, file-path, file-type, and encode-md5.

d_custom_file: files list owned by user. This table is to identify files by file name and user.

d_temp_file: store temporary file that will be remove after some time.

The main interface design of this module is listed in table 2.

Table 2 Main interfaces of file process

```
public String saveFile(ByteSource byteSource) throws IOException;
public ByteSource findFile(String md5) throws FileNotFoundException;
public String generateDownloadURL(String md5);
public String readFileContent(String md5) throws IOException;
public String writeFileContent(String content) throws IOException;
public String generateTempKey();
public String saveTempFile(ByteSource byteSource, String accessKey)
        throws IOException;
```

```
public String getTempFileKeyByAccessKey(String accessKey);
public String savePuppetFile(ByteSource byteSource) throws IOException;
ZipBuilder buildZipBuilder();
String saveZip(ZipBuilder builder) throws IOException;
```

Method introduction:

saveFile: save data from byte Source to file system and return its md5-code.

findfile: get file by it's md5 code.

generateDownloadURL: get URL that can access to target file.

generateTempKey: generate specific key of temp file.

saveZip: compress file to save disk space.

The process of "saveFile" is showed in figure 4.12. Note that there's a comparison with md5-codes to prevent duplication of files.

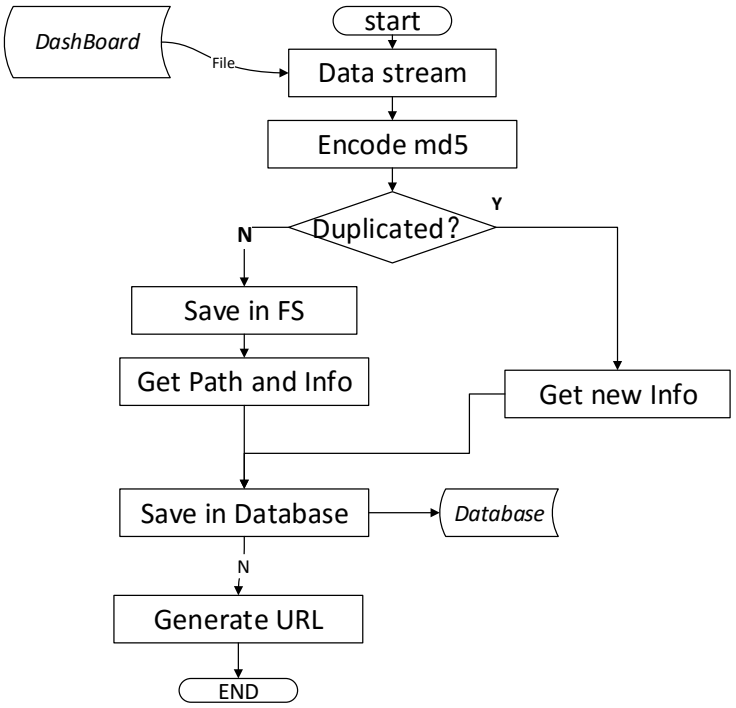

Figure 4.12 process diagram of "saveFile"

4.7. **Agent client**

4.7.1. Overview of Agent

Agent client is running on target machines or virtual machines, managing applications, components and their configurations. As shown in Figure 4.13, agent client consists of three main parts: Event manager, Application Manager, Dynamic Configurator.

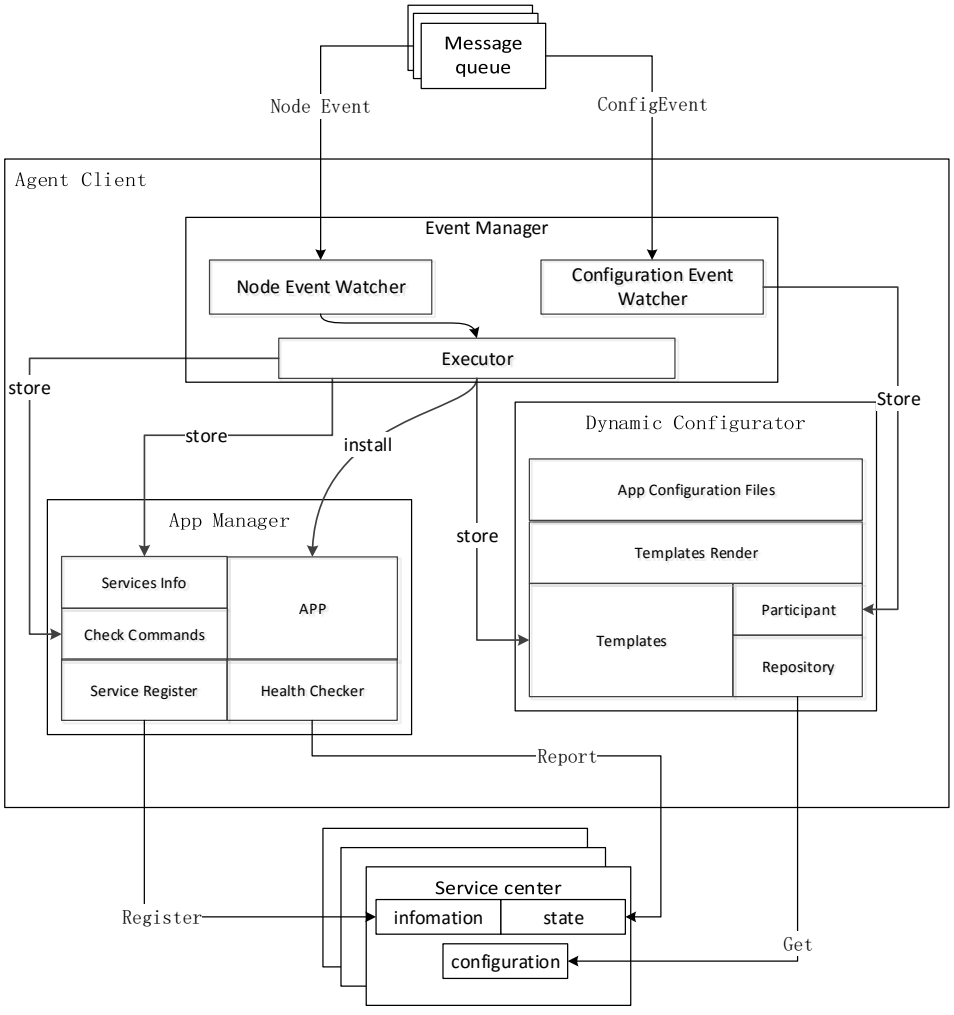

Figure 4.12 Architecture of Agent Client

Event Manager listens on message queue (implement based on consul), receive *Node-Event* and *Config-Event* (mentioned in 4.4). If it gets an event, it will call Application Manager (or Dynamic Configurator) to make handling of *Node-Event* (or

Configuration-Event).

Application manager controls applications and components on its host, including service information, check commands, service register, health checker, application states.

Dynamic Configurator manages properties and attributes of components as well as their configuration files. It can generate configuration files according to *Configure-Event*.

The design of class and interface is shown in figure 4.13.

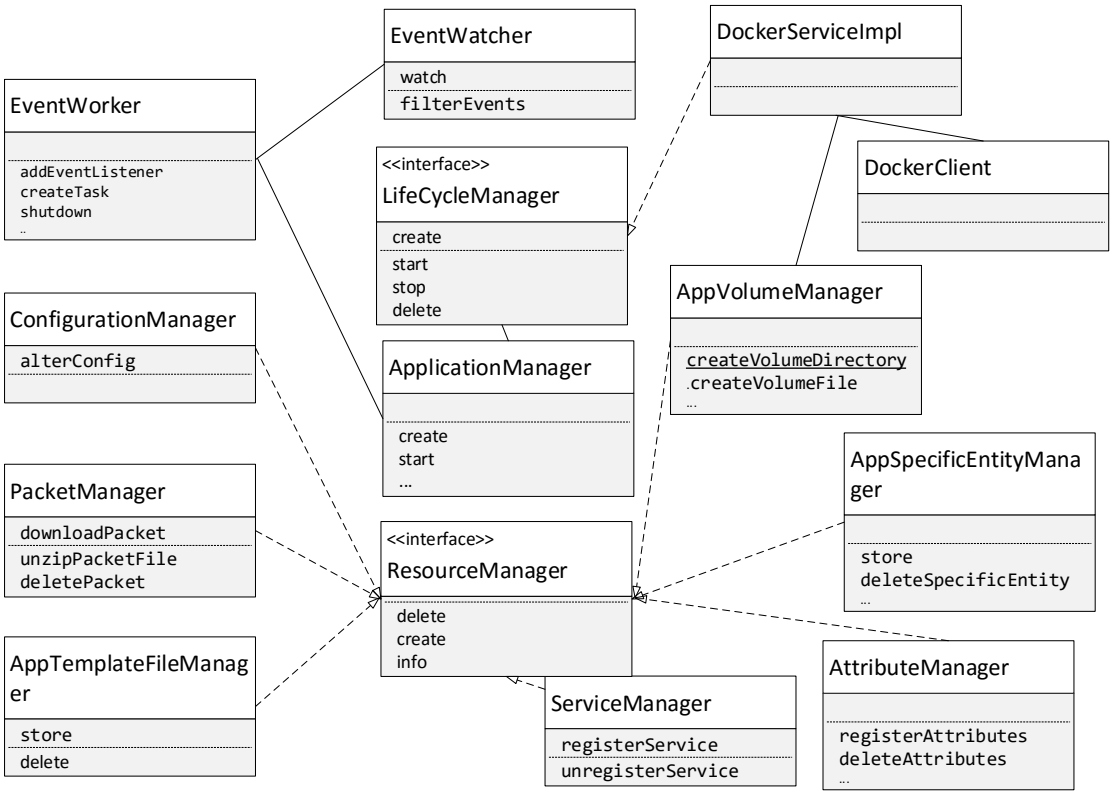

Figure 4.13 Class Diagram of Agent Client

Event watcher listens events from message queue.

Event worker extracts event data and call corresponding method in application manager.

LifeCycleManager controls applications through four action: “create”, “start”, “stop”, “remove”.

ResourceManager, packetManager and AppTemplateFileManager control the resources that applications will apply for example volumes in file system, packet of

deployment.

4.7.2. Dynamic configuration design

This module is to handle configuration changes, generate new configuration for component.

We design a template format that will be filed with new parameters. An example is showed in figure 4.14.

```
1  upstream bookstore-tpcw {
2     ip_hash  ;
3     keepalive {{key "service/bookstore-tpcw/keepalive"}};
4     {{range service "TPCWTomcat"}}
5       server {{.Address}}:{{.Port}} max_fails=0 fail_timeout=10s;
6     {{end}}
7  }
8   #output example
9 upstream bookstore-tpcw {
10    ip_hash  ;
11    keepalive 20;
12    server 133.133.134.110:8090 max_fails=0 fail_timeout=10s;
13    server 133.133.134.110:8091 max_fails=0 fail_timeout=10s;
14 }
```

Figure 4.14 An example of configuration template

Line 1 - 7 is original template. The “key” with “{{ }}” in template implies the following parameter is ready to be filled by fetching values in service center. The “service” limits the range of searching scope. After dynamic generation, line 9 - 14 is the output files, the value of keep-alive “20” and value of ”133.133.134.110”, ”8090” are all get from service center.

Once generation finishes, it will restart and send signals to service center, and the changes of this service will cause another dynamic configuration recursively.

4.7.3. Executor design

To execute the “create”, ”start”, ”stop”, ”remove” operations, we design the executor as plugins. The specific implementation is according to CMT (Configure

manage tools) that users apply. Now we choose docker as our executor. Docker is a famous and booming skill in virtual container. User can control instance of containers through RESTful URL. We use Docker-Java library “com. spotify. docker” package to realize through Java socket.

5. Summary

In this document, we introduce Cloudeploy 2.0 from system architecture to functions detail. In the process of designing, we adopt the idea of micro-service. Each module in the project is highly cohesive and low coupling. This makes the project easy to expand, for example we implement executor with docker, users also can use other managers like puppet or saltstack.

Another idea is that we use some open-source software and projects in Cloudeploy 2.0: JsPlumb, Maven, Tomcat, Docker, Consul and so on, from which we learnt a lot. This makes Cloudeploy 2.0 vibrant.

In future work, we intend to expand this project with system monitoring, so that users can automatically deploy, configure and monitoring applications and computer clusters in one platforms.