Dynamic Cloud Management System for Monitoring and Managing Services

Trevor Wright, Scott Fuller, Ashwin Reddi
tlw@cknowledge.com, wfuller@hotmail.com, msg2ash@gmail.com

Abstract

As cloud computing becomes more widely adopted, the size of cloud systems will necessarily become larger. In recent years, large companies, such as Google and Amazon, have become reliant on resources and services provided by cloud computing environments. While these large corporations have most likely implemented efficient, dynamic, and scalable systems for managing their cloud infrastructure, they are highly protected corporate secrets, and very few (if any) publicly-available cloud management systems exist that are easily configurable and offer a high degree of scalability. This paper proposes an innovative architecture for an extremely scalable cloud management system that provides dynamic management of cloud services with minimal configuration.

1. Introduction

Cloud computing systems are becoming more and more common in a large variety of organizations. The recent boom in popularity comes into focus only when you think about what the IT department of any organization requires: a way to increase capacity or modify capabilities on the fly without investing in new infrastructure, training new employees, or licensing new software [1]. Cloud computing is maturing at a rapid rate as a result of the low-cost scalability and availability it provides, but there is a lack of tools available which are designed to manage a highly dynamic cloud environment. Modern organizations require fast-paced distributed systems which can add , remove, start, or stop services automatically. Because of the sheer number of computers in an organization's cloud, it is no longer acceptable for the organization to waste man-hours manually configuring each node in the cloud to work with a cloud management system . Manual configuration, when the organization has hundreds or thousands of nodes in the cloud, does not scale well and is far too expensive.

The cloud management system (CMS) proposed in this paper addresses the problem of manual configuration of cloud management systems by outlining an architecture and implementation that provides dynamic node discovery and automatic configuration.

2. Existing Cloud Management Tools

2.1 Nagios

Nagios is a popular, open-source cloud management system that provides the ability to monitor IT infrastructure within an organization. The creators of Nagios claim that it is “designed with scalability and flexibility in mind” [2]. Although Nagios provides a reliable platform for monitoring the state (detecting if a service is on or off) of a remote service, it does not provide functionality to turn the services on or off, and it does not provide scalability or flexibility. The shortcomings of Nagios are apparent in cloud environments with more than a few nodes, because it requires manual configuration on both the monitoring server and each client to setup a service to be monitored.

2.2 Ganglia

Ganglia is another popular cloud management utility which was originally intended for use in clusters, grids, and high-performance computing, but has been adopted in cloud computing as well. Ganglia relies on a hierarchical structure of the cloud using multicast discovery and announcements, but it requires significant amount of configuration and setup to work as desired, and it does not provide functionality to turn services on or off [3].

3. Main Features of the Proposed CMS

3.1 Monitoring Service State

The system will monitor the state of services registered with the CMS. A service can exist in three possible states: RUNNING, STOPPED, or UNKNOWN. A service is in an UNKNOWN state if a network error occurs between the CMS monitoring server and the client on which the service is running.

3.2 Starting and Stopping Services

The CMS will allow an end-user to issue start or stop requests through a user interface on the head node to remote services. For example, a system administrator may wish to stop a database instance on one node and start it on another – the user will simply issue the request and the CMS will handle the starting and stopping of the service.

3.3 Dynamic Service Registration

The drawback to almost all other cloud monitoring solutions is that they require manual configuration of services on the monitoring server and on the client. The proposed CMS only requires configuration of a very simple file on each client. When the CMS client is started, it automatically registers its services with the monitoring head node.

4. Components of the CMS

4.1 Basic Technologies

Our CMS is built upon core components of the Java programming language, including Java Management Extensions (JMX) and Remote Method Invocation (RMI). Details about these technologies are not addressed in this paper, but more can be read about them in the official Java online documentation [4] [5].

4.2 Broker

The broker is the head node of the CMS. Without the broker, the CMS will not function – it is responsible for maintaining a view of the entire cloud, which includes the state of all services running, and the capability to start or stop any of these services. Although the broker can be seen as the "controller", it essentially just acts as a notification listener and a request issuer between the user and all nodes in the cloud, referred to as agents.

4.3 Agent

An agent is the CMS component which is installed on a node that has services to be monitored. The agent requires a properties file to be set which points to the IP address or URL of the machine on which the broker is running. The agent is responsible for notifying the broker of any services it is running, and updates the status of those services with the broker at a regular interval. Each agent discovers services it should check by looking for new service definitions in a specified folder on the local file system.

4.4 Service

A service is any process (or combination of processes) that are running on a node where an agent is installed. Services are loaded into an agent through service definitions, which are simple text files that have the following attributes:

Service Name - name of the service
Check Interval – how often the agent should check to see if this service is running
Check Command – the command to execute which will verify whether or not the service is running or stopped
Start Command – the command to execute when this service should be started
Stop Command – the command to execute when this service should be stopped

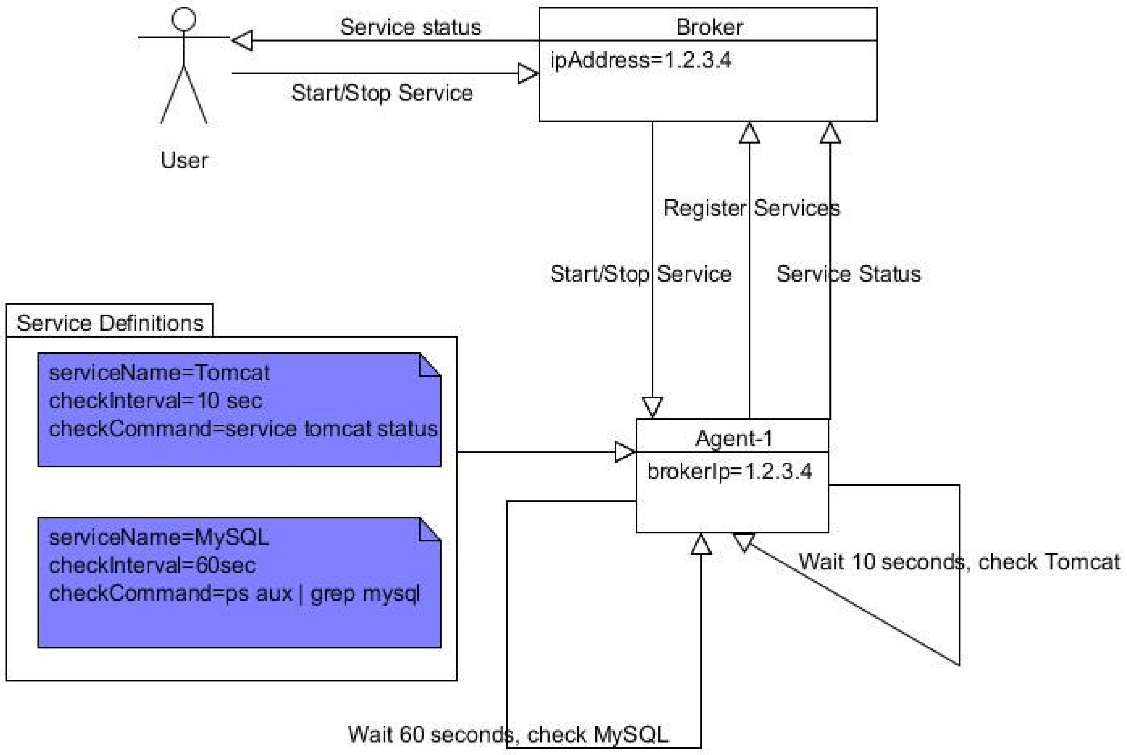
The local agent monitors the folder where the service definitions are placed, and any time a new definition is found, it registers the service with the broker. Once a service is successfully registered with the broker, the agent executes the check command at a regular interval, and reports the result to the broker. The broker can also issue start or stop commands, which the agent executes on demand.

4.4 Commands

A command can be any operating system level command that returns an exit status. The CMS makes the assumption that any command executed will return 0 on success, 1 on failure, or any other number on an unknown state. The commands can be a simple shell command (for example, on Linux, a user can issue the command "service tomcat status" to see if Tomcat is running), or it can run any executable that returns 0 on success and 1 on failure.

For example, we have created a simple Java application that wraps the Windows System Controller, and executes a query, start, or stop command for a specified service. The check command is set in the service definition as "java -jar cms-windows-controller.jar <service-name> query". The Jar file takes the input arguments, and calls the Windows command "sc query <service-name>", and returns 0 or 1, depending on whether or not the service is running.

A visual depiction of how the components interact is outlined in Figure 1, below:



5. Unresolved Problems

Although our CMS architecture enables an administrator to create a highly-dynamic and easily manageable cloud, there is one major unresolved issue that is essential to managing a cloud is dependencies between services.

In many environments, it is necessary for a service to be available on one machine before a service on another can start. For example, if a web site uses a load balancer to forward requests to a cluster of web servers, the web servers have a dependency on the load balancer – the web servers will not work if the load balancer is not running. Therefore, it would be useful to allow the user to define service dependencies. The service dependency is a basic rule-set or workflow. Using the previous example, the service dependencies between the load balancer and the web servers would read:

```
if user attempts to start the web server
then
    if load balancer is running
    then start web server
    else do not start web server
```

6. Summary

The proposed implementation of a CMS achieves flexibility, scalability, and ease of configuration. The various components are highly decoupled from each other and the underlying operating system by using a three-tiered architecture – the broker runs entirely operating system and implementation independent; the agent is operating system independent, as it only calls the commands specified in the service definitions. Although service definitions might not be portable to other systems, they are very simple to understand and can be ported to different environments quickly.

In addition, the service definitions provide the flexibility for a system administrator to define a service as a grouping of processes on one machine. For example, a service definition for "Web Server" could be created, where the checkCommand executes a custom script to verify that Apache, SSH, and FTP are all running on the same machine. The checkCommand would return 0 only if all of these services are running.

The proposed architecture is also extremely dynamic since each agent is responsible for checking its own services and reporting to the broker. The broker does not need to be configured to look at specific nodes – the broker just listens for status messages, and displays those messages to the user.

7. References

[1] Eric Knorr and Galen Gruman, "What cloud computing really means", *Info World,* http://www.infoworld.com/d/cloud-computing/what-cloud-computing-really-means-031
[2] Nagios, http://www.nagios.org/about/overview
[3] Ganglia, http://ganglia.sourceforge.net/
[4] Java JMX, http://java.sun.com/javase/technologies/core/mntr-mgmt/javamanagement/

[5] Java RMI, http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp