AppScale Documentation

Release 1

Honglei Liu

July 11, 2015

Contents

CHAPTER 1

Introduction

This document is meant to briefly introduce AppScale, give a look at what AppScale is and why we want to use it.

1.1 What is Google App Engine?

Google App Engine lets you run web applications on Google's infrastructure. App Engine applications are easy to build, easy to maintain, and easy to scale as your traffic and data storage needs grow. With App Engine, there are no servers to maintain: You just upload your application, and it's ready to serve your users.

You can serve your app from your own domain name (such as http://www.example.com/) using Google Apps. Or, you can serve your app using a free name on the appspot.com domain. You can share your application with the world, or limit access to members of your organization.

Google App Engine supports apps written in several programming languages. With App Engine's Java runtime environment, you can build your app using standard Java technologies, including the JVM, Java servlets, and the Java programming language—or any other language using a JVM-based interpreter or compiler, such as JavaScript or Ruby. App Engine also features two dedicated Python runtime environments, each of which includes a fast Python interpreter and the Python standard library. Finally, App Engine provides a Go runtime environment that runs natively compiled Go code. These runtime environments are built to ensure that your application runs quickly, securely, and without interference from other apps on the system.

With App Engine, you only pay for what you use. There are no set-up costs and no recurring fees. The resources your application uses, such as storage and bandwidth, are measured by the gigabyte, and billed at competitive rates. You control the maximum amounts of resources your app can consume, so it always stays within your budget.

App Engine costs nothing to get started. All applications can use up to 1 GB of storage and enough CPU and bandwidth to support an efficient app serving around 5 million page views a month, absolutely free. When you enable billing for your application, your free limits are raised, and you only pay for resources you use above the free levels.

1.2 What is AppScale?

AppScale implements Google's Cloud Platform, App Engine, in open source. In addition to executing your scalable web apps and mobile endpoints on Google's resources, AppScale lets you execute them everywhere else – on your laptop, behind your firewall, on your datacenter, or on your favorite public or private cloud infrastructure. AppScale is free and easy to use and thus brings Google's "best practices" to devs, giving them the ability to focus on their innovation and MVP without lock-in.

The AppScale code base, written primarily in the Python programming language, provides the glue code that links these technologies together dynamically with the apps that use them. AppScale automatically configures, deploys, and scales this software across a distributed system.

When AppScale is deployed, the AppScale tools automatically assign one or more roles to each deployed VM instance. Using this deployment model, AppScale is able to provide fault tolerance and elasticity automatically and in a straightforward manner, since any VM can take on any role on-demand.

1.3 Why AppScale?

The set of features that AppScale brings developers and that sets AppScale apart include:

- Only production-ready platform with full Google App Engine compatibility
- Unlocked deployment options – execute on your favorite public/private cloud system
- Easy installation and use
- The sweet spot between NoOps and DevOps: convenience vs flexibility and control
- Customer choice of services for API plug-ins
- Automatic configuration, deployment, and scaling and fault tolerance of apps and their service ecosystems
- Portable and hybrid app deployment across clouds and services
- Uniformity across development, test, and production deployments
- Open, free, customizable execution environment for web apps, services, and mobile backends
- AppScale products for programmer productivity, code/data backup, disaster recovery and more

CHAPTER 2

How to install?

This document shows the basic steps of installing AppScale on a laptop.

2.1 Install VirtualBox

Download and install VirtualBox on your host machine using these instructions.

2.2 Install Vagrant

Download and install Vagrant on your host machine using these directions.

2.3 Install AppScale tools on local machine

First, Install homebrew.

Make sure that homebrew is installed properly with `brew doctor`. Fix all issues if there are any.

Then, get latest appscale tools. If a newer version is available at the appscale download page, please update this page accordingly.

```
brew install wget
wget https://github.com/AppScale/appscale-tools/archive/1.6.9.tar.gz -O appscale-tools-1.6.9.tar.gz
tar xvf appscale-tools-1.6.9.tar.gz
./appscale-tools-1.6.9/osx/appscale_install.sh
```

Update PATH

```
export PATH="$PATH:/usr/local/appscale-tools/bin"
```

2.4 Register the Image with Vagrant

To register an AppScale virtual machine (VM) with vagrant, execute the following:

```
mkdir -p ~/appscale
cd ~/appscale
vagrant box add appscale1.6.9 http://download.appscale.com/download/AppScale%201.6.9%20VirtualBox%201
vagrant init
```

2.5 Configure and start your virtual machine

The `vagrant init` command above creates a `Vagrantfile` in the current directory. Next, we need to modify this file to give our VM an IP address that our host machine can access it on. To do this, open `Vagrantfile` and modify the following lines

```
config.vm.network :hostonly, "192.168.33.10"
config.vm.box = "appscale1.6.9"
config.vm.customize ["modifyvm", :id, "--memory", 2048]
```

Next, start your AppScale virtual machine, ssh into it, and change the root password:

```
vagrant up
vagrant ssh
sudo -s passwd
exit
```

2.6 Start and shutdown AppScale

Start AppScale by running:

```
appscale up
```

Shut down your apps and AppScale deployment by running:

```
appscale down
```

CHAPTER 3

How to use AppScale?

This document shows how to use AppScale tools to deploy, remove and stop an instance on AppScale.

3.1 Deploying Apps to AppScale

Once you've got AppScale started, you can deploy Google App Engine applications to it.

```
appscale deploy ~/path-to-your-app
```

3.2 Removing Apps from AppScale

Once you decide you no longer want to have AppScale host an application, you can run "appscale remove appname" to remove that app (presuming that your app was called "appname").

```
appscale remove guestbook
```

3.3 Seeing How AppScale is Doing

You can also use the AppScale Tools to query the status of your AppScale deployment, by running "appscale status".

```
appscale status
```

3.4 Logging into Your AppScale VMs

AppScale runs over Ubuntu Lucid virtual machines. Should you need to log into the machines for any reason, you can run "appscale ssh integer" to log into the machine with id "integer" as root. Since "appscale ssh 0" is the most common machine to log into, you can also use "appscale ssh" as shorthand for "appscale ssh 0".

```
appscale ssh
```

3.5 Stopping AppScale

Once you're done running AppScale and you want to tear it down, run "appscale down". If running in a cloud deployment, this terminates the machines that "appscale up" created, but if running in a cluster deployment, this stops all the API services we started when we ran "appscale up".

```
appscale down
```

CHAPTER 4

APIs

This document is meant to introduce the APIs AppScale provides, including Google App Engine APIs and some other APIs.

4.1 Google App Engine APIs

Foremost, AppScale provides implementations for the Google App Engine APIs. These APIs provide several scalable services which can be leveraged by a Google App Engine application. The Google App Engine APIs are Blobstore, Channel, Datastore, Images, Memcache, Namespaces, TaskQueue, Users, URL Fetch, and XMPP. Google App Engine provides an overview of the APIs and their functionality at http://code.google.com/appengine/docs/. AppScale emulates this functionality using open source software systems, tools, and services, as well as new components.

`Blobstore API` The Blobstore API enables users to store large entities of text or binary data.

`Channel API` The Channel API allows applications to push messages from the application server to a client's browser.

`Datastore API` The Datastore API allows for the persistence of data.

`Images API` The Images API facilitates programmatic manipulation of images.

`Memcache API` The Memcache API permits applications to store their frequently used data in a distributed memory grid.

`Namespace API` The Namespace API implements the ability to segregate data into different namespaces.

`Task Queue API` The Google App Engine platform lacks the ability to do threading or computations within a request greater than 30 seconds.

`Users API` The Users API provides authentication for web applications through the use of cookies.

`URL Fetch API` The URL Fetch API enables an application the ability to do POST and GET requests on remote resources.

`XMPP API` The XMPP API presents the ability to receive and send messages to users with a valid XMPP account.

4.2 MapReduce Streaming APIs

Through Hadoop Streaming, users can specify a Mapper and Reducer pro- gram that can run under the MapReduce programming paradigm. Fault-tolerance and data replication is automatically handled by the Hadoop Streaming framework, and AppScale exposes a simple API that interfaces to Hadoop Stream- ing. This API is:

- `putMRInput(data, inputLoc)`: Given a string ”data” and a Hadoop file system location “inputLoc”, creates a file on the Hadoop file system named “inputLoc”
- `runMRJob(mapper, reducer, inputLoc, outputLoc, config)`: Given the relative paths to a mapper and reducer file (relative to the main Python file being run), run a Hadoop MapReduce Streaming job. Input is fed to the program via the HDFS file named “inputLoc”, and output is fed to the HDFS file named “outputLoc”. If a hash is passed as ”config”, the key / value pairs are passed as configuration options to Hadoop Streaming.
- `getMROutput(outputLoc)`: Given a Hadoop file system location “outputLoc”, returns a string with the contents of the named file.
- `writeTempFile(suffix, data)`: Writes a file to /tmp on the local machine with the contents data. Is useful for passing a file with nodes to exclude from MapReduce jobs.
- `getAllIPs()`: Returns an array of all the IPs in the AppScale cloud. Is useful for excluding or including nodes based on some user-defined application logic.
- `getNumOfNodes()`: Returns an integer with the number of nodes in the AppScale cloud. Is useful for determining at MapReduce run time, how many Map tasks and Reduce tasks should be run for optimal performance (recommended value is 1 Map task per node).
- `getMRLogs(outputLoc)`: Returns a string with the MapReduce job log for the job whose output is located at outputLoc. Data is returned as a combination of XML and key / value pairs, in the standard Hadoop format.

4.3 EC2 APIs

AppScale provides an Amazon EC2 API. Users can use this API to spawn virtual machines and manage them entirely through an AppScale web service. The main functions this API provides mirror those of the EC2 command line tools:

- `ec2_run instances(options)`: Spawns virtual machines with the specified options (interpreted as command-line arguments).
- `ec2_describe instances()`: Returns the status of all machines owned by the current user.
- `ec2_terminate instances(options)`: Terminates virtual machines with the specified options (interpreted as command-

line arguments).

- `ec2_add keypair(options)`: Creates a new SSH key-pair that can be used to log in to virtual machines that are spawned with this key-pair’s name.
- `ec2_elete keypair(options)`: Deletes the named SSH key-pair from the underlying cloud infrastructure.
- `ec2_describe availability zones(options)`: In Eucalyptus, this returns information relating to the number of vir-

tual machines available to be spawned by users.

- `ec2_describe images()`: Returns information about the virtual machine images, ramdisks, and kernels that are available for use in the system.
- `ec2_reboot instances(options)`: Reboots virtual machines with the specified options (interpreted as command- line arguments).
- search