Apache Mesos Essentials

Build and execute robust and scalable applications using Apache Mesos

Dharmesh Kakadia
In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'Running Mesos'
- A synopsis of the book’s content
- More information on **Apache Mesos Essentials**
Dharmesh Kakadia is a research fellow at Microsoft Research, who develops the next-generation cluster management systems. Before coming to MSR, he completed his MS in research from the International Institute of Information Technology, Hyderabad, where he worked on improving scheduling in cloud and big data systems. He likes to work at the intersection of systems and data and has published research in resource management at various venues. He is passionate about open source technologies and plays an active role in various open source communities. You can learn more about him at @DharmeshKakadia on Twitter.
Mesos makes it easier to develop and manage fault-tolerant and scalable distributed applications. Mesos provides primitives that allow you to program for the aggregated resource pool, without worrying about managing resources on individual machines. With Mesos, all your favorite frameworks, ranging from data processing to long-running services to data storage to Web serving, can share resources from the same cluster. The unification of infrastructure combined with the resilience built into Mesos also simplifies the operational aspects of large deployments. When running on Mesos, failures will not affect the continuous operations of applications.

With Mesos, everyone can develop distributed applications and scale it to millions of nodes.

What this book covers

Chapter 1, Running Mesos, explains the need for a data center operating system in the modern infrastructure and why Mesos is a great choice for it. It also covers how to set up singlenode and multimode Mesos installations in various environments.

Chapter 2, Running Hadoop on Mesos, discusses batch data processing using Hadoop on Mesos.

Chapter 3, Running Spark on Mesos, covers how to run Spark on Mesos. It also covers tuning considerations for Spark while running on Mesos.

Chapter 4, Complex Data Analysis on Mesos, demonstrates the various options for deploying lambda architecture on Mesos. It covers Storm, Spark Streaming, and Cassandra setups on Mesos in detail.
Chapter 5, *Running Services on Mesos*, introduces services and walks you through the different aspects of service architecture on Mesos. It covers the Marathon, Chronos, and Aurora frameworks in detail and helps you understand how services are deployed on Mesos.

Chapter 6, *Understanding Mesos Internals*, dives deep into Mesos fundamentals. It walks you through the implementation details of resource allocation, isolation, and fault tolerance in Mesos.

Chapter 7, *Developing Frameworks on Mesos*, covers specifics of framework development on Mesos. It helps you learn about the Mesos API by building a Mesos framework.

Chapter 8, *Administering Mesos*, talks about the operational aspects of Mesos. It covers topics related to monitoring, multitenancy, availability, and maintenance along with REST API and configuration details.
Running Mesos

This chapter will give you a brief overview of Apache Mesos and cluster computing frameworks. We will walk you through the steps for setting up Mesos on a single-node and multi-node setup. We will also see how to set up a Mesos cluster using Vagrant and on Amazon EC2. Throughout this book, we will refer to Apache Mesos and Mesos interchangeably. We will cover the following topics in this chapter:

- Modern data centers
- Cluster computing frameworks
- Introducing Mesos
- Why Mesos?
- A single-node Mesos cluster
- A multi-node Mesos cluster
- A Mesos cluster on Amazon EC2
- Running Mesos using Vagrant
- The Mesos community

Modern data centers

Modern applications are highly dependent on data. The manifold increase in the data generated and processed by organizations is continually changing the way we store and process it. When planning modern infrastructure for storing and processing the data, we can no longer hope to simply buy hardware with more capacity to solve the problem. Different frameworks for batch processing, stream processing, user-facing services, graph processing, and ad hoc analysis are every bit as important as the hardware they run on. These frameworks are the applications that power the data center world.
Running Mesos

The size and variety of big data means traditional scale-up strategies are no longer adequate for modern workloads. Thus, large organizations have moved to distributed processing, where a large number of computers act as a single giant computer. The cluster is shared by many applications with varying resource requirements, and the efficient sharing of resources at this scale among multiple frameworks is the key to achieving high utilization. There is a need to consider all these machines as a single warehouse scale computer. Mesos is designed to be the kernel of such computers.

Traditionally, frameworks run in silos and resources are statically partitioned among them, which leads to an inefficient use of resources. The need to consider a large number of commodity machines as a single computer, and the ability to share resources in an elastic manner by all the frameworks requires a cluster computing framework. Mesos is inspired by the idea of sharing resources in a cluster between multiple frameworks while providing resource isolation.

Cluster computing frameworks

In modern clusters, the computing requirements of different frameworks are radically different, and organizations need to run multiple frameworks and share data and resources between them. Resource managers face challenging and competing goals:

- Efficiency: Efficiently sharing resources is the prime goal of cluster management software.
- Isolation: When multiple tasks are sharing resources, one of the most important considerations is to ensure resource isolation. Isolation combined with proper scheduling is the foundation of guaranteeing service level agreements (SLAs).
- Scalability: The continuous growth of modern infrastructure requires cluster managers to scale linearly. One important scalability metric is the delay experienced in decision-making by the framework.
- Robustness: Cluster management is a central component, and robust behavior is required for continuous business operations. There are many aspects contributing to robustness, from well-tested code to fault-tolerant design.
- Extensible: Cluster management software is a huge development in any organization and has been used for decades. During an operation, the changes in the organization policy and/or the hardware invariably require change in how the cluster resources are managed. Thus, maintainability becomes an important consideration for large organizations. It should be configurable considering constraints (for example location, hardware) and support for multiple frameworks.
Introducing Mesos

Mesos is a cluster manager aiming for improved resource utilization by dynamically sharing resources among multiple frameworks. It was started at the University of California, Berkeley in 2009 and is in production use in many companies, including Twitter and Airbnb. It became an Apache top-level project in July 2013 after nearly two years in incubation.

Mesos shares the available capacity of machines (or nodes) among jobs of different natures, as shown in the following figure. Mesos can be thought of as a kernel for the data center that provides a unified view of resources on all nodes and seamless access to these resources in a manner similar to what an operating system kernel does for a single computer. Mesos provides a core for building data center applications and its main component is a scalable two-phased scheduler. The Mesos API allows you to express a wide range of applications without bringing the domain-specific information into the Mesos core. By remaining focused on core, Mesos avoids problems that are seen with monolithic schedulers.

The following components are important for understanding the overall Mesos architecture. We will briefly describe them here and will discuss the overall architecture in more detail in Chapter 6, Understanding Mesos Internals.
Running Mesos

The master

The master is responsible for mediating between the slave resources and frameworks. At any point, Mesos has only one active master, which is elected using ZooKeeper via distributed consensus. If Mesos is configured to run in a fault-tolerant mode, one master is elected through the distributed leader election protocol, and the rest of them stay in standby mode. By design, Mesos' master is not meant to do any heavy lifting tasks itself, which simplifies the master design. It offers slave resources to frameworks in the form of resource offers and launches tasks on slaves for accepted offers. It also is responsible for all the communication between the tasks and frameworks.

Slaves

Slaves are the actual workhorses of the Mesos cluster. They manage resources on individual nodes and are configured with a resource policy to reflect the business priorities. Slaves manage various resources, such as CPU, memory, ports, and so on, and execute tasks submitted by frameworks.

Frameworks

Frameworks are applications that run on Mesos and solve a specific use case. Each framework consists of a scheduler and executor. A scheduler is responsible for deciding whether to accept or reject the resource offers. Executors are resource consumers and run on slaves and are responsible for running tasks.

Why Mesos?

Mesos offers huge benefits to both developers and operators. The ability of Mesos to consolidate various frameworks on a common infrastructure not only saves on infrastructure costs, but also provides operational benefits to the Ops teams and simplifies developers' view of the infrastructure, ultimately leading to business success. Here are some of the reasons for organizations to embrace Mesos:

- Mesos supports a wide variety of workloads, ranging from batch processing (Hadoop), interactive analysis (Spark), real-time processing (Storm, Samza), graph processing (Hama), high-performance computing (MPI), data storage (HDFS, Tachyon, and Cassandra), web applications (play), continuous integration (Jenkins, GitLab), and a number of other frameworks. Moreover, meta-scheduling frameworks, such as Marathon and Aurora can run most of the existing applications on Mesos without any modification. Mesos is an ideal choice for running containers at scale. This flexibility makes Mesos very easy to adopt.
• Mesos improves utilization through elastic resource sharing between various frameworks. Without a common data center operating system, different frameworks have to run on siloed hardware. Such static partitioning of resources leads to resource fragmentation, limiting the utilization and throughput. Dynamic resource sharing through Mesos drives higher utilization and throughput.

• Mesos is an open source project with a vibrant community. The Mesos pluggable architecture makes it easy to customize it for the organization's needs. Combined with the fact that Mesos runs on a wide range of operating systems and hardware choices, it provides the widest range of options and guards against vendor lock-in. Thus, developing against the Mesos API provides many choices of infrastructure for running them. It also means that the Mesos applications will be portable across bare metal, virtualized infrastructure, and cloud providers.

• Probably, the most important benefit of Mesos is empowering developers to build modern applications with increased productivity. As developers move from developing applications for a single computer to a program against data centers, they need an API that allows them to focus on their logic and not on the nitty-gritty details of the distributed infrastructure. With Mesos, the developers do not have to worry about the distributed aspects and can focus on the domain-specific logic of the application. Mesos provides a rich API to develop scalable and fault-tolerant distributed applications, as we will see in Chapter 7, Developing Frameworks on Mesos.

• Operating a large infrastructure is challenging. Mesos simplifies infrastructure management by providing a unified view of resources. It brings a lot of agility and deploying new services takes a shorter time with Mesos since there is no separate cluster to be allocated. Mesos is extremely Ops-friendly and treats infrastructure resources like cattle and not pets. What this means is that Mesos is resilient in the face of failures and can automatically ensure high availability, without requiring manual intervention. Mesos supports multitenant deployment with strong isolation, which is essential for operating at scale. Mesos provides full-featured REST, web, and command-line interfaces and integrates well with the existing tools, as we will see in Chapter 8, Administering Mesos.

• Mesos is battle-tested at Twitter, Airbnb, HubSpot, eBay, Netflix, Conviva, Groupon, and a number of other organizations. Mesos catering to the needs of a wide variety of use cases across different companies is proof of Mesos's versatility as a data center kernel.
Mesos also offers significant benefits over traditional virtualization-based infrastructure:

- Most of the applications do not require strong isolation provided by virtual machines and can run on container-based isolation in Mesos. Since containers have much lower overheads than to VMs, this not only leads to higher consolidation but also has other benefits, such as fast start-up time and so on.
- Mesos reduces infrastructure complexity drastically compared to VMs.
- Achieving fault tolerance and high availability using VMs is very costly and hard. With Mesos, hardware failures are transparent to applications, and the Mesos API helps developers in embracing failures.

Now that we have seen the benefits of running Mesos, let's create a single-node Mesos cluster and start exploring Mesos.

**Single-node Mesos clusters**

Mesos runs on Linux and Mac OS X. A single machine Mesos setup is the simplest way of trying out Mesos, so we'll go through it first. Currently, Mesos does not provide binary packages for different operating systems, and we need to compile it from the source. There are binary packages available by community.

**Mac OS**

Homebrew is a Linux-style package manager for Mac. Homebrew provides a formula for Mesos and compiles it locally. We need to perform the following steps to install Mesos on Mac:

1. Install Homebrew from [http://brew.sh/](http://brew.sh/).
2. Homebrew requires Java to be installed. Mac has Java installation by default, so we just have to make sure that `JAVA_HOME` is set correctly.
3. Install Mesos using Homebrew with the following command:

   ```
   mac@master:~ $ brew install mesos
   ```

   Although Homebrew provides a way to try out Mesos on Mac, the production setup should run on Linux.
Fedora

Starting from Fedora 21, the Fedora repository contains the Mesos packages. There are `mesos-master` and `mesos-slave` packages to be installed on the master and slave respectively. Also, there is a `mesos` package, which contains both the master and slave packages. To install the `mesos` package on Fedora version \( \geq 21 \), use the following command:

```
fedora@master:~ $ sudo yum install -y mesos
```

Now we can continue with the Start Mesos section to run Mesos. For Fedora Version \( \leq 21 \), we have to install the dependencies and Mesos from the source, similar to CentOS as explained in the following section.

Installing prerequisites

Mesos requires the following prerequisites to be installed:

- g++ (\( \geq 4.1 \))
- Python 2.6 developer packages
- Java Development Kit (\( \geq 1.6 \)) and Maven
- The cURL library
- The SVN development library
- Apache Portable Runtime Library (APRL)
- Simple Authentication and Security Layer (SASL) library

Additionally, we will need autoconf (Version 1.12) and libtool if we want to build Mesos from the git repository. The installation of this software differs for various operating systems. We will show you the steps to install Mesos on Ubuntu 14.10 and CentOS 6.5. The steps for other operating systems are also fairly similar.

CentOS

Use the following commands to install all the required dependencies on CentOS:

1. Currently, the CentOS default repository does not provide a SVN library \( \geq 1.8 \). So, we need to add a repository, which provides it. Create a new `wandisco-svn.repo` file in `/etc/yum.repos.d/` and add the following lines:

```
centos@master:~ $ sudo vim /etc/yum.repos.d/wandisco-svn.repo
```

```
[WandiscoSVN]
name=Wandisco SVN Repo
```

```
Running Mesos

baseurl=http://opensource.wandisco.com/centos/6/svn-1.8/
RPMS/$basearch/
   enabled=1
   gpgcheck=0

Now, we can install libsvn using the following command:

    centos@master:~ $ sudo yum groupinstall -y "Development Tools"

2. We need to install Maven by downloading it, extracting it, and putting it in PATH. The following commands extract it to /opt after we download it and link mvn to /usr/bin:

    centos@master:~ $ wget http://mirror.nexcess.net/apache/maven/maven-3.0.5/binaries/apache-maven-3.0.5-bin.tar.gz
    centos@master:~ $ sudo tar -zxf apache-maven-3.0.5-bin.tar.gz -C /opt/
    centos@master:~ $ sudo ln -s /opt/apache-maven-3.0.5/bin/mvn /usr/bin/mvn

3. Install the other dependencies using the following command:

    centos@master:~ $ sudo yum install -y python-devel java-1.7.0-openjdk-devel zlib-devel libcurl-devel openssl-devel cyrus-sasl-devel cyrus-sasl-md5 apr-devel subversion-devel

Ubuntu

Use the following command to install all the required dependencies on Ubuntu:

    ubuntu@master:~ $ sudo apt-get -y install build-essential openjdk-6-jdk python-dev python-boto libcurl4-nss-dev libssl-dev libapix-dev libsvn-dev maven

Build Mesos

Once we have installed all the required software, we can follow these steps to build Mesos:

1. Download the latest stable release from http://mesos.apache.org/downloads/. At the time of writing, the latest release is 0.21.0. Save the mesos-0.21.0.tar.gz file in some location. Open the terminal and go to the directory, where we have saved the file or you can directly run the following command on the terminal to download Mesos:

    ubuntu@master:~$ wget http://www.apache.org/dist/mesos/0.21.0/mesos-0.21.0.tar.gz
2. Extract Mesos with the following command and enter the extracted directory. Note that the second command will remove the downloaded .tar file, and rename the version name from the extracted folder:

```
ubuntu@master:~ $ tar -xzf mesos-*.tar.gz
ubuntu@master:~ $ rm mesos-*.tar.gz ; mv mesos-* mesos
ubuntu@master:~ $ cd mesos
```

3. Create a build directory. This will contain the compiled Mesos binaries. This step is optional, but it is recommended. The build can be distributed to slaves instead of recompiling on every slave:

```
ubuntu@master:~/mesos $ mkdir build
ubuntu@master:~/mesos $ cd build
```

4. Configure the installation by running the configure script:

```
ubuntu@master:~/mesos/build $ ../configure
```

The configure script supports tuning the build environment, which can be listed by running `configure --help`. If there are any dependencies missing, then the configure script will report, and we can go back and install the missing packages. Once the configuration is successful, we can continue with the next step.

5. Compile it using make. This might take a while. The second step is make check:

```
ubuntu@master:~/mesos/build $ make
ubuntu@master:~/mesos/build $ make check
```

The make check step builds the example framework, and we can now run Mesos from the build folder directly without installing it.

6. Install Mesos using the following command:

```
ubuntu@master:~/mesos/build $ make install
```

The list of commands that Mesos provides is as follows:

<table>
<thead>
<tr>
<th>Command</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>mesos-local.sh</td>
<td>This command launches an in-memory cluster within a single process.</td>
</tr>
<tr>
<td>mesos-tests.sh</td>
<td>This command runs the Mesos test case suite.</td>
</tr>
<tr>
<td>mesos.sh</td>
<td>This is a wrapper script used to launch the Mesos commands. Running without any arguments shows all the available commands.</td>
</tr>
</tbody>
</table>
Running Mesos

<table>
<thead>
<tr>
<th>Command</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>gdb-mesos-*</td>
<td>This command launches the corresponding processes in debugging mode using gdb.</td>
</tr>
<tr>
<td>lldb-mesos-*</td>
<td>This command launches the corresponding processes in debugging mode using lldb.</td>
</tr>
<tr>
<td>valgrind-mesos-*</td>
<td>This command launches the corresponding Valgrind instrumentation framework.</td>
</tr>
<tr>
<td>mesos-daemon.sh</td>
<td>This command starts/stops a Mesos daemon process.</td>
</tr>
<tr>
<td>mesos-start-cluster.sh</td>
<td>This command starts and stops the Mesos cluster on nodes in the [install-prefix]/var/mesos/deploy/masters and [install-prefix]/var/mesos/deploy/slaves files.</td>
</tr>
<tr>
<td>mesos-stop-cluster.sh</td>
<td>This command starts and stops mesos masters on nodes listed in the masters file.</td>
</tr>
<tr>
<td>mesos-start-masters.sh</td>
<td>This command starts and stops the mesos slaves on nodes listed in the slaves file.</td>
</tr>
<tr>
<td>mesos-stop-masters.sh</td>
<td>This command starts and stops the mesos slaves on nodes listed in the slaves file.</td>
</tr>
</tbody>
</table>

We can now start the local Mesos cluster using the mesos-local command, which will start both the master and slave in a single process and provide a quick way to check the Mesos installation.

Start Mesos

Now we are ready to start the Mesos process. First, we need to create a directory for the Mesos replicated logs with read-write permissions:

```bash
ubuntu@master:~ $ sudo mkdir -p /var/lib/mesos
```

```bash
ubuntu@master:~ $ sudo chown `whoami` /var/lib/mesos
```

Now, we can start the master with the following command, specifying the directory we created:

```bash
ubuntu@master:~ $ mesos-master --work_dir=/var/lib/mesos
```

```
```
The output here lists the build version, various configurations that the master has used, and the master ID of the cluster. The slave process should be able to connect to the master. The slave process can specify the IP address or the hostname of the master by the \texttt{--master} option. In the rest of the book, we will assume that the machine on which the master is running has the hostname \texttt{master} and should be replaced with an appropriate hostname or IP address.

\begin{verbatim}
ubuntu@master:~ $ mesos-slave --master=master:5050
I1228 07:33:32.415992 4654 main.cpp:144] Version: 0.21.0
I1228 07:33:32.416199 4654 containerizer.cpp:100] Using isolation: posix/cpu,posix/mem
I1228 07:33:32.443282 4654 main.cpp:165] Starting Mesos slave
I1228 07:33:32.447244 4654 slave.cpp:169] Slave started on master:5051
I1228 07:33:32.448254 4654 slave.cpp:289] Slave resources: cpus(*):2; mem(*):1961; disk(*):35164; ports(*):[31000-32000]
I1228 07:33:32.462025 4654 slave.cpp:602] New master detected at master@master:5050
\end{verbatim}

The output confirms the connection to the master and lists the slave resources. Now, the cluster is running with one slave ready to run the frameworks.
Running test frameworks

Mesos includes various example test frameworks written in C++, Java, and Python. They can be used to verify that the cluster is configured properly. The following test framework is written in C++, and it runs five sample applications. We will run it using the following command:

```
ubuntu@master:~/mesos/build/src $ ./test-framework --master=master:5050
```

```
I1228 08:53:13.303910  6044 sched.cpp:137] Version: 0.21.0
I1228 08:53:13.312556  6065 sched.cpp:234] New master detected at master:
master :5050
 Attempting to register without authentication
20141228-085231-251789322-5050-5407-0001
Registered!
Received offer 20141228-085231-251789322-5050-5407-03 with mem(*):1961;
disk(*):35164; ports(*):[31000-32000]; cpus(*):2
Launching task 0 using offer 20141228-085231-251789322-5050-5407-03
Task 0 is in state TASK_RUNNING
Task 0 is in state TASK_FINISHED
Received offer 20141228-085231-251789322-5050-5407-04 with mem(*):1961;
disk(*):35164; ports(*):[31000-32000]; cpus(*):2
Launching task 2 using offer 20141228-085231-251789322-5050-5407-04
Task 2 is in state TASK_RUNNING
Task 2 is in state TASK_FINISHED
Received offer 20141228-085231-251789322-5050-5407-05 with mem(*):1961;
disk(*):35164; ports(*):[31000-32000]; cpus(*):2
Launching task 4 using offer 20141228-085231-251789322-5050-5407-05
Task 4 is in state TASK_RUNNING
Task 4 is in state TASK_FINISHED
```
Here the output shows the framework connected to the master and receives the resource offers from the master. It also shows the various states of the tasks it has launched. The Java example framework is included in the `src/example/java` folder:

```
ubuntu@master:~/mesos/build/src/examples/java $ ./test-framework
master:5050
```

```
I1228 08:54:39.290570  7224 sched.cpp:137] Version: 0.21.0
```

```
I1228 08:54:39.302083  7250 sched.cpp:234] New master detected at master@master:5050
```

```
I1228 08:54:39.302613  7250 sched.cpp:242] No credentials provided. Attempting to register without authentication
```

```
Registered! ID = 20141228-085231-251789322-5050-5407-0002
Received offer 20141228-085231-251789322-5050-5407-06 with cpus: 2.0 and mem: 1961.0
```

```
Launching task 0 using offer 20141228-085231-251789322-5050-5407-06
```

```
Status update: task 1 is in state TASK_RUNNING
```

```
Status update: task 0 is in state TASK_RUNNING
```

```
Status update: task 1 is in state TASK_FINISHED
Finished tasks: 1
```

```
Status update: task 0 is in state TASK_FINISHED
Finished tasks: 2
```

```
Received offer 20141228-085231-251789322-5050-5407-07 with cpus: 2.0 and mem: 1961.0
```

```
Launching task 2 using offer 20141228-085231-251789322-5050-5407-07
```

```
Launching task 3 using offer 20141228-085231-251789322-5050-5407-07
```

```
Status update: task 2 is in state TASK_RUNNING
```

```
Status update: task 2 is in state TASK_FINISHED
Finished tasks: 3
```

```
Status update: task 3 is in state TASK_RUNNING
```

```
Status update: task 3 is in state TASK_FINISHED
Finished tasks: 4
```

Running Mesos

Received offer 20141228-085231-251789322-5050-5407-08 with cpus: 2.0 and mem: 1961.0
Launching task 4 using offer 20141228-085231-251789322-5050-5407-08
Status update: task 4 is in state TASK_RUNNING
Status update: task 4 is in state TASK_FINISHED
Finished tasks: 5
I1228 08:54:41.788455  7248 sched.cpp:1286] Asked to stop the driver
I1228 08:54:41.788652  7248 sched.cpp:752] Stopping framework '20141228-085231-251789322-5050-5407-0002'
I1228 08:54:41.789008  7224 sched.cpp:1286] Asked to stop the driver

Similarly, the Python example framework is included in the src/example/python folder and shows frameworkId and the various tasks states:

ubuntu@master:~mesos/build/src/examples/python $./test-framework master:5050
I1228 08:55:52.389428  8516 sched.cpp:137] Version: 0.21.0
I1228 08:55:52.422859  8562 sched.cpp:234] New master detected at master@master:5050
I1228 08:55:52.424178  8562 sched.cpp:242] No credentials provided. Attempting to register without authentication
Registered with framework ID 20141228-085231-251789322-5050-5407-0003
Received offer 20141228-085231-251789322-5050-5407-09 with cpus: 2.0 and mem: 1961.0
Launching task 0 using offer 20141228-085231-251789322-5050-5407-09
Launching task 1 using offer 20141228-085231-251789322-5050-5407-09
Task 0 is in state TASK_RUNNING
Task 1 is in state TASK_RUNNING
Task 0 is in state TASK_FINISHED
Received message: 'data with a \\x00 byte'
Task 1 is in state TASK_FINISHED
Received message: 'data with a \\x00 byte'
Received offer 20141228-085231-251789322-5050-5407-010 with cpus: 2.0 and mem: 1961.0
Launching task 2 using offer 20141228-085231-251789322-5050-5407-010
Launching task 3 using offer 20141228-085231-251789322-5050-5407-010
Task 2 is in state TASK_RUNNING
Task 2 is in state TASK_FINISHED
Task 3 is in state TASK_RUNNING
Task 3 is in state TASK_FINISHED
Received message: 'data with a \x00 byte'
Received message: 'data with a \x00 byte'
Received offer 20141228-085231-251789322-5050-5407-011 with cpus: 2.0 and mem: 1961.0
Launching task 4 using offer 20141228-085231-251789322-5050-5407-011
Task 4 is in state TASK_RUNNING
Task 4 is in state TASK_FINISHED
All tasks done, waiting for final framework message
Received message: 'data with a \x00 byte'
All tasks done, and all messages received, exiting
I1228 08:55:54.136085  8561 sched.cpp:1286] Asked to stop the driver
I1228 08:55:54.136261  8516 sched.cpp:1286] Asked to stop the driver

Mesos Web UI
Mesos provides a web UI for reporting information about the Mesos cluster. It can be accessed from <master-host>:<port>; in our case, this will be http://master:5050. This includes the slaves, aggregated resources, frameworks, and so on. Here is the screenshot of the web interface:

Mesos web interface
Multi-node Mesos clusters

We can repeat the previous procedure to manually start `mesos-slave` on each of the slave nodes to set up the cluster, but this is labor-intensive and error-prone for large clusters. Mesos includes a set of scripts in the deploy folder that can be used to deploy Mesos on a cluster. These scripts rely on SSH to perform the deployment. We need to set up a password less SSH. We will set up a cluster with two slave nodes (slave1, slave2) and a master node (master).

Let’s configure our cluster to make sure that they have connectivity between them after installing all the prerequisites on all the nodes. The following commands will generate a `ssh` key and will copy them to both the slaves:

```
ubuntu@master:- $ ssh-keygen -f ~/.ssh/id_rsa -P ""
ubuntu@master:- $ ssh-copy-id -i ~/.ssh/id_rsa.pub ubuntu@slave1
ubuntu@master:- $ ssh-copy-id -i ~/.ssh/id_rsa.pub ubuntu@slave2
```

We need to copy the compiled Mesos to both the nodes at the same location, as in the master:

```
ubuntu@master:- $ scp –R build slave1:[install-prefix]
ubuntu@master:- $ scp –R build slave2:[install-prefix]
```

Create a masters file in the `[install-prefix]/var/mesos/deploy/masters` directory with an editor of your own choice to list the masters one per line, which in our case will be only one:

```
ubuntu@master:- $ cat [install-prefix]/var/mesos/deploy/masters
master
```

Similarly, the `slaves` file will list all the nodes that we want to be Mesos slaves:

```
ubuntu@master:- $ cat [install-prefix]/var/mesos/deploy/slaves
slave1
slave2
```
Now, we can start the cluster with the `mesos-start-cluster` script and use `mesos-stop-cluster` to stop it:

```
ubuntu@master:~ $ mesos-start-cluster.sh
```

This, in turn, calls `mesos-start-masters` and `mesos-start-slaves` that will start the appropriate processes on the master and slave nodes. The script looks for any environment configurations in `[install-prefix]/var/mesos/deploy/mesos-deploy-env.sh`. Also, for better configuration management, the master and slave configuration options can be specified in separate files in `[install-prefix]/var/mesos/deploy/mesos-master-env.sh` and `[install-prefix]/var/mesos/deploy/mesos-slave-env.sh`.

### Mesos cluster on Amazon EC2

The Amazon Elastic Compute Cloud (EC2) provides access to compute the capacity in a pay-as-you-go model through virtual machines and is an excellent way of trying out Mesos. Mesos provides scripts to create Mesos clusters of various configurations on EC2. The `mesos-ec2` script located in the `ec2` directory allows launching, running jobs, and tearing down the Mesos clusters. Note that we can use this script even without building Mesos, but you will need Python (>=2.6). We can manage multiple clusters using different names.

We will need an AWS keypair to use the `ec2` script, and our access and secret key. We have to make our keys available via an environment variable. Create and download a keypair via the AWS Management Console (https://console.aws.amazon.com/console/home) and give them 600 permissions:

```
ubuntu@local:~ $ chmod 600 my-aws-key.pem
ubuntu@local:~ $ export AWS_ACCESS_KEY_ID=<your-access-key>
ubuntu@local:~ $ export AWS_SECRET_ACCESS_KEY=<your-secret-key>
```

Now we can use the EC2 scripts provided with Mesos to launch a new cluster using the following command:

```
ubuntu@local:~/mesos/ec2 $ ./mesos-ec2 -k <your-key-pair> -i <your-identity-file> -s 3 launch ec2-test
```
Running Mesos

This will launch a cluster named ec2-test with three slaves. Once the scripts are done, it will also print the Mesos web UI link, in the form of <master-hostname>:8080. We can confirm that the cluster is up by going to the web interface. The script provides a number of options, a few of which are listed in the following table. We can list all the available options of the script by running mesos-ec2 --help:

<table>
<thead>
<tr>
<th>Command</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>--slave or -s</td>
<td>This is the number of slaves in the cluster</td>
</tr>
<tr>
<td>--key-pair or -k</td>
<td>This is the SSH keypair for authentication</td>
</tr>
<tr>
<td>--identity-file or -i</td>
<td>This is the SSH identity file used for logging into the instances</td>
</tr>
<tr>
<td>--instance-type or -t</td>
<td>This is a slave instance type, must be 64-bit</td>
</tr>
<tr>
<td>--ebs-vol-size</td>
<td>This is the size of an EBS volume used to store the persistent HDFS data.</td>
</tr>
<tr>
<td>--master-instance-type or -m</td>
<td>This is a master instance type, must be 64-bit</td>
</tr>
<tr>
<td>--zone or -z</td>
<td>This is the Amazon availability zone for launching instances</td>
</tr>
<tr>
<td>--resume</td>
<td>This flag resumes the installation from the previous run</td>
</tr>
</tbody>
</table>

We can use the login action to log in to the launched cluster by providing a cluster name, as follows:

```
ubuntu@local:/mesos/ec2 $ ./mesos-ec2 -k <your-key-pair> -i <your-identity-file> login ec2-test
```

The script also sets up a HDFS instance that can be used via commands in the /root/ephemeral-hdfs/ directory.

Finally, we can terminate a cluster using the following command. Be sure to copy any important data before terminating the cluster:

```
ubuntu@local:/mesos/ec2 $ ./mesos-ec2 destroy ec2-test
```

The script also supports advance functionalities, such as pausing and restarting clusters with EBS-backed instances. The Mesos documentation is a great source of information for any clarification. It is worth mentioning that Mesosphere (http://mesosphere.com) also provides you with an easy way of creating an elastic Mesos cluster on Amazon EC2, Google Cloud, and other platforms and provides commercial support for Mesos.
Running Mesos using Vagrant

Vagrant provides an excellent way of creating portable virtual environments and thus provides an easy way to try Mesos running in a virtual machine. We will see how to create a single-node and multi-node Mesos cluster on virtual machines using Vagrant:

1. Download and install Vagrant from https://www.vagrantup.com/downloads.html. Vagrant works on all the major operating systems.

2. This Vagrant setup uses additional Vagrant plugins. Install them using the following command:
   
   ```
   ubuntu@local:~ $ vagrant plugin install vagrant-omnibus
evagrant-berkshelf vagrant-hosts vagrant-cachier vagrant-aws
   ```

3. Download Vagrant configuration from https://github.com/everpeace/vagrant-mesos/ or clone them using `git` and `cd` to the directory:
   
   ```
   ubuntu@local:~ $ git clone https://github.com/everpeace/vagrant-mesos.git ; cd vagrant-mesos
   ```

4. For a single-node cluster setup, `cd` to the standalone directory and run the `vagrant up` command. This will create one virtual machine that will run the Mesos master, slave, and ZooKeeper instances. The Mesos UI will be available at http://192.168.33.10:5050:
   
   ```
   ubuntu@local:~ $ cd standalone ; vagrant up
   ```

5. For a multi-node setup, `cd` to the multinode directory. We can configure how many virtual machines can be created for the Mesos masters, slaves, and ZooKeeper instances in the `cluster.yml` file. By default, it will create five virtual machines that run as one ZooKeeper, two Mesos masters, and two Mesos slave instances. The Mesos web UI in multi-node setup will be available at http://172.31.1.11:5050:
   
   ```
   ubuntu@local:~ $ cd multinode ; vagrant up
   ```

6. The Mesos cluster should be up and running. We can log in to these machines via `ssh` using `vagrant ssh`. A single-node setup assigns them the master and slave as hostnames, while a multi-node setup names the hosts as master1, slavel, and so on:
   
   ```
   ubuntu@local:~ $ vagrant ssh master # to login to master
   ubuntu@local:~ $ vagrant ssh slave # to login to slave
   ```
7. We can bring down the virtual machines using the `halt` command. This allows the virtual machines to be booted again with everything set up using the `up` command. Finally, the `destroy` command will destroy all the virtual machines created by Vagrant. Note that we have to execute the `vagrant destroy` commands from the `standalone` or `multinode` directory accordingly:

```
ubuntu@local:~ $ vagrant halt
ubuntu@local:~ $ vagrant destroy
```

This Vagrant setup also allows many different configurations and also supports you to launch the Mesos cluster on Amazon EC2. The `vagrant` files and the `README` file included in the repository will provide you with more details.

The Mesos community

Despite being a relatively young project, Mesos has a great community ([http://mesos.apache.org/community/](http://mesos.apache.org/community/)). There are a number of success stories of using Mesos by both small and large companies ([http://mesos.apache.org/documentation/latest/powered-by-mesos/](http://mesos.apache.org/documentation/latest/powered-by-mesos/)). Companies use Mesos for use cases, ranging from data analytics to web serving to data storing frameworks.

Case studies

Mesos is used by a number of companies in production to simplify infrastructure management. Here, we will see how some of the companies leverage Mesos.

Twitter

Twitter was the first adopter of Mesos and helped to mature the project during the Apache incubation. Twitter is a real-time conversation social platform. Twitter solved the famous fail whale problem, thanks to the reliability of the infrastructure. Twitter considers Mesos as its base for the entire infrastructure and runs a variety of jobs on the Mesos platform, including analytics, ad platform, typeahead service, and messaging infrastructure. All the new services built at Twitter use Mesos, and more importantly, it has changed the way developers think about resources in distributed environments. Developers now can think in terms of a shared pool of resources instead of thinking about individual machines. Twitter also built the Aurora scheduler framework to manage the long-running services on Mesos.
HubSpot

HubSpot makes inbound marketing products. HubSpot runs Mesos on Amazon EC2 to support more than 150 different types of services. Mesos improved resource utilization and ensured high availability without running multiple copies of services, leading to lower infrastructure costs. HubSpot noted that with Mesos, developers are able to launch new services much faster and scaling services have become much more reliable and easier to scale. HubSpot created the Singularity framework on Mesos and built Platform-as-a-Service (PaaS) to facilitate standardized deployment of services.

Airbnb

Airbnb is a community-driven rental company and was one of the early adopters of Mesos. Airbnb uses Mesos for running data analysis using Hadoop, Spark, Kafka as well as services, such as Cassandra and Rails. Airbnb also created the Chronos scheduler framework for Mesos. We will learn in detail about Aurora and Chronos in Chapter 5, Running Services on Mesos.

Twitter's stack was built on Ruby on Rails and JBoss-esque frameworks, which are mostly service-based in nature, while Airbnb, on the other hand, used Mesos more for data processing and is ETL in nature. Twitter runs Mesos on bare metal using Solaris Zones in a private infrastructure, while Airbnb runs it on top of virtual machines using VMware and Xen hypervisor on AWS. These validate that Mesos provides general and easy to use API as a kernel of modern distributed infrastructure that can run on a wide range of hardware choices and serves a variety of frameworks on top.

Mailing lists

Mesos maintains very accessible documentation at http://mesos.apache.org/documentation/latest/, detailing most parts of Mesos. When the documentation is not sufficient, the Mesos mailing lists provide an excellent medium to interact with other members and are an essential part of the Mesos community. The user mailing list (user@mesos.apache.org) and developer mailing list (dev@mesos.apache.org) actively discuss the development and usage of Mesos.
Summary
In this chapter, we gave an overview of the requirements of a modern cluster management framework and demonstrated how to set up Mesos clusters. We are ready to run various frameworks on Mesos, which is where we will turn to in the chapters to follow. We will start with Hadoop framework on Mesos in the next chapter.
Where to buy this book
You can buy Apache Mesos Essentials from the Packt Publishing website.
Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.
Click here for ordering and shipping details.