

# Processing Vehicle data with Storm and Kafka on Microsoft Azure Data Science Core

---

In this example, we'll show you how to deploy a [Storm](#) topology that reads data from the [Kafka](#) messaging system. You can use the Kafka client application to send vehicle real-time information from anywhere to the Kafka cluster. The Storm topology will translate those coordinates into JSON objects, use [GeoJSON](#) to identify the coordinates on the Bing map, and then keep a running record of vehicle's speed, temperature, RPM and gear usage rate. For persistence, the real-time data is stored in [Microsoft Azure Table Storage service](#). The topology also writes data to [Redis](#), which is how this web application gets the data. This web app is written in [Node.js](#), uses Socket.IO and the express web application framework to read the data from Redis and display it via [d3js](#).

## Use Azure Management Portal to Create a Windows Azure Data Analysis VM

---

1. Log in to the [Windows Azure Management Portal](#).
2. Click on the **Virtual Machines** tab and click on **Images** near the top of the screen.

virtual machines

VIRTUAL MACHINE INSTANCES **IMAGES** DISKS

3. Click on **Browse VM Depot** in the bar on the bottom



4. Select **Ubuntu** on the left and then select the **Azure Data Analysis** image.

## Choose an image

ALL  
CENTOS  
OPENSUSE  
SUSE ENTERPRISE SERVER  
**UBUNTU**

A-Z

Apache Solr 4.3.1-1 (Ubuntu 12.10)

Apache Solr 4.4.0-0 (Ubuntu 12.10)

Apache Solr 4.5.0-0 (Ubuntu 12.10)

Apache TomEE Plus - 1.6 Snapshot

apache\_openmeetings\_2\_1\_1\_Ubu...

Artifactory 3.0.2-0 (Ubuntu 12.10)


Artifactory 3.0.3-0 (Ubuntu 12.10)

**Azure Data Analysis**

biolinux

BitNami concrete5 Stack 5.6.1.2-0 (...)

Cl...

 **Azure Data Analysis**  
★★★★★ (0)

Includes R, openmpi and loads of packages. Include Spark 0.8, Shark 0.7 and Storm and Kafka. Also includes iPython notebook + a whole heap of libraries. [More](#)

|              |  |
|--------------|--|
| PUBLISHER    | elastacloud  |
| PUBLISH DATE | 10/6/2013  |
| PACKAGES     | azure sdk, git, hive, ipython, ipython-notebook, kafka, nodejs, npm, numpy, openjdk, openjre, pandas, python, scala, scikit-learn, scipy, shark, spark, storm, tornado |
| SIZE         | 29.3 GB  |

→

5. Choose the **Image Region** that your storage account is in (i.e. the region you created your affinity group in) from the drop down box, then select your storage account from the drop down box.

# Choose a storage account

Select a storage account to copy the image to

IMAGE REGION

East Asia

STORAGE ACCOUNT IN IMAGE REGION

portalvhds3j870py2t90m

## ATTENTION

Microsoft is not responsible for images provided & licensed by the community members and does not screen for security, compatibility or performance. Users should perform their own due diligence before using the community images. Support for images is provided by the community and not by Windows Azure customer support. By using unsupported images, you might forfeit Windows Azure availability SLA.



## Azure Data Analysis

★★★★★ (0)

Includes R, openmpi and loads of packages. Include Spark 0.8, Shark 0.7 and Storm and Kafka. Also includes iPython notebook + a whole heap of libraries. [More](#)

PUBLISHER [elastacloud](#)

PUBLISH DATE 10/6/2013

PACKAGES azure sdk, git, hive, ipython, ipython-notebook, kafka, nodejs, npm, numpy, openjdk, openjre, pandas, python, scala, scikit-learn, scipy, shark, spark, storm, tornado

SIZE 29.3 GB



6. Click the check mark button to continue and wait for the disk image to download from the VM Depot to your storage account. You can click on **Details** in the status bar to see the transfer progress.

Copying image 'Azure-Data-Analysis'...

Storage container 'communityimages' created.

Started copying image 'Azure-Data-Analysis'.

Copying image 'Azure-Data-Analysis': 16 % completed...

NEW

CREATE

BROWSE VM DEPOT

DELETE

7. Once the image has copied you'll need to register it. Select the image and click **Register** in the bar on the bottom.

CREATE

BROWSE VM DEPOT

DELETE

REGISTER

8. Enter a name for the image, click the checkmark button, and wait for registration to complete.



# Register an image

NAME

Azure-Data-Analysis X

VHD URL

http://portalvhds3j870py2t90m.blob.core.windows.net



9. Stay on the Virtual Machines tab and click on **Virtual Machine Instances** near the top of the screen.

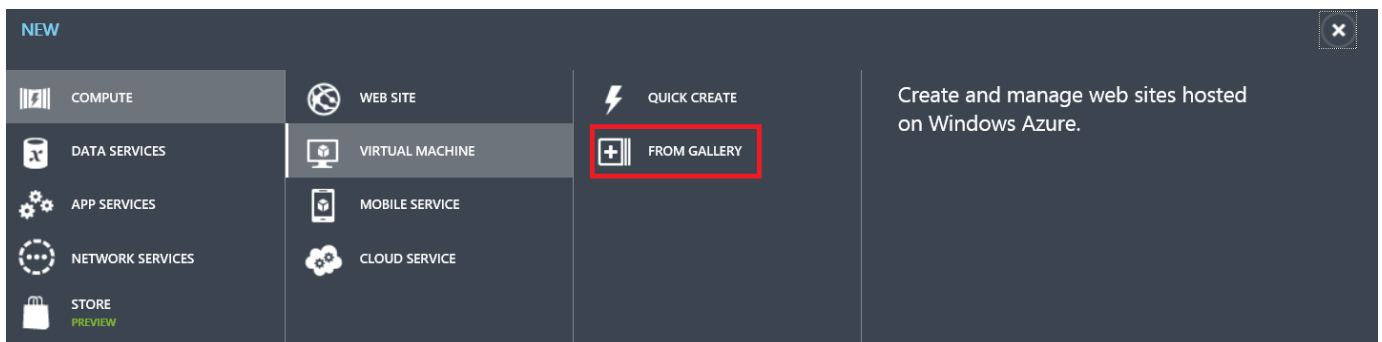
## virtual machines

VIRTUAL MACHINE INSTANCES

IMAGES

DISKS

10. Click on **New** in the bottom bar and select **From Gallery**.



11. Select **My Images** on the left and then select the Azure-Data-Analysis you just registered. Go to the next page.

# Virtual machine image selection

ALL

PLATFORM IMAGES

MY IMAGES

MY DISKS



Azure-Data-Analysis



Azure-Data-Analysis

Includes R, openmpi and loads of packages. Include Spark 0.8, Shark 0.7 and Storm and Kafka. Also includes iPython notebook + a whole heap of libraries. [More](#)

PUBLISHER [elastacloud](#) (VM Depot)

OS FAMILY Linux

SUBSCRIPTION Azpas300A0U8655

LOCATION East Asia

## PRICING INFORMATION

Pricing varies based on the subscription you select to provision your virtual machine.



12. Enter the virtual machine name, select the **Large** machine size from the drop down list, enter a new user name, check the **Provide a Password** box and enter the new user password. Go to the next page.

# Virtual machine configuration

**VIRTUAL MACHINE NAME** ?**SIZE****NEW USER NAME****AUTHENTICATION** ?☐ UPLOAD COMPATIBLE SSH KEY FOR AUTHENTICATION☒ PROVIDE A PASSWORD**NEW PASSWORD****CONFIRM****Azure-Data-Analysis**

Includes R, openmpi and loads of packages. Include Spark 0.8, Shark 0.7 and Storm and Kafka. Also includes iPython notebook + a whole heap of libraries. [More](#)

**PUBLISHER** [elastacloud](#) (VM Depot)

**OS FAMILY** Linux

**SUBSCRIPTION** Azpas300A0U8655

**LOCATION** East Asia

**PRICING INFORMATION**

Pricing varies based on the subscription you select to provision your virtual machine.



13. Enter a name for the new cloud service configuration and select your affinity group from the drop down box. Go to the next page.

# Virtual machine configuration

**CLOUD SERVICE** ? ▼**CLOUD SERVICE DNS NAME** ✓ .cloudapp.net**REGION/AFFINITY GROUP/VIRTUAL NETWORK** ? ▼**AVAILABILITY SET** ? ▼**Azure-Data-Analysis**

Includes R, openmpi and loads of packages. Include Spark 0.8, Shark 0.7 and Storm and Kafka. Also includes iPython notebook + a whole heap of libraries. [More](#)

|                     |                        |
|---------------------|------------------------|
| <b>PUBLISHER</b>    | elastacloud (VM Depot) |
| <b>OS FAMILY</b>    | Linux                  |
| <b>SUBSCRIPTION</b> | Azpas300A0U8655        |
| <b>LOCATION</b>     | East Asia              |

**PRICING INFORMATION**

Pricing varies based on the subscription you select to provision your virtual machine.




14. We'll need to add several endpoints for the VM. To add an endpoint, enter a name for the endpoint like "HTTP" or "Kafka" in the field under **Name** in the Endpoints table. Add the following TCP endpoints:

1. HTTP: Port 80
2. User HTTP: Port 8080
3. Kafka: Port 9092
4. Zookeeper: 2181 Your endpoints should look like this:

# Virtual machine configuration

**ENDPOINTS** ?

| NAME   | PROTOCOL | PUBLIC PORT | PRIVATE PORT |
|--|----------|-------------|--------------|
| SSH  | TCP      | 22          | 22           |
| HTTP   | TCP      | 80          | 80           |
| User HTTP  | TCP      | AUTO        | 8080         |
| Kafka  | TCP      | AUTO        | 9092         |
| Zookeeper  | TCP      | AUTO        | 2181         |
| <div>ENTER OR SELECT A VALUE </div> |          |             |              |

**LEGAL TERMS**

By clicking the Submit button, I acknowledge that I am getting this software from elastacloud and that elastacloud's [legal terms](#) apply to it. Microsoft does not provide rights for third-party software.

**NOTE**

This is an unsupported image supplied by a third party. ?

**Azure-Data-Analysis**

Includes R, openmpi and loads of packages. Include Spark 0.8, Shark 0.7 and Storm and Kafka. Also includes iPython notebook + a whole heap of libraries. [More](#)

**PUBLISHER** elastacloud (VM Depot)

**OS FAMILY** Linux

**SUBSCRIPTION** Azpas300A0U8655

**LOCATION** East Asia

**PRICING INFORMATION**

Pricing varies based on the subscription you select to provision your virtual machine.



- Click the checkmark button and wait for the new VM to be created, provisioned, and started.
- If your local workstation runs Windows then you'll need to install an SSH client like [PuTTY](#) to connect to the Azure Data Science Core VM.
- Use your ssh client to connect to the VM.





bindings.

```
Linux
git clone https://github.com/zeromq/jzmq.git
cd jzmq
./autogen.sh
./configure
make
sudo make install
```

3. Update node, install npm and redis server.

```
Linux
sudo add-apt-repository ppa:chris-lea/node.js
sudo apt-get update
sudo apt-get install nodejs
sudo apt-get install redis-server
```

4. Install Maven. Many projects use the [Maven](#) project management tool for build management, dependency management, and packaging. Maven projects are described in a **pom.xml** file in the top-level of the project.

```
Linux
cd $HOME
wget http://www.interior-dsgn.com/apache/maven/maven-3/3.1.0/binaries/apache-maven-3.1.0-bin.tar.gz
tar xzf apache-maven-3.1.0-bin.tar.gz
echo 'export PATH=$HOME/apache-maven-3.1.0/bin:$PATH' >> ~/.bashrc
source ~/.bashrc
mvn --version
```

5. Install Leiningen. Some community provided extensions to Storm and Kafka are written in [Clojure](#), a dynamic programming language that targets the Java Virtual Machine. Clojure projects are managed via [Leiningen](#), which is a software management tool very similar to Maven. Installing Leiningen is very easy because it's distributed as a single, simple script file.

```
Linux
cd /bin
sudo wget https://raw.githubusercontent.com/technomancy/leiningen/stable/bin/lein
sudo chmod +x lein
```

6. Install Kafka. Our example uses the Kafka messaging system to accept messages from clients and queue them until they can be processed. Kafka can achieve very high message throughput and is designed to scale up as needed, so it's a great fit for the cloud.

```
Linux
cd $HOME
wget http://www.motorlogy.com/apache/incubator/kafka/kafka-0.7.2-incubating/kafka-0.7.2-incubating-src.tgz
tar xzf kafka-0.7.2-incubating-src.tgz
cd kafka-0.7.2-incubating-src
./sbt update
./sbt package
cd $HOME
echo 'export PATH=$HOME/kafka-0.7.2-incubating-src/bin:$PATH' >> ~/.bashrc
source ~/.bashrc
```

## Upload and Build the Example Code

1. Now we need to upload the source code under the training folder **storm-kafka-demo** to the remote linux machine. You can use any ftp tools or other ssh tools to upload the file from your local machine. Here we use the [pscp](#) on windows to upload the source code. (if you are using Linux or OS X please use your tools.)

On the remote linux machine, run the following command to make a new folder:

```
cd $HOME
mkdir storm-kafka-demo
```

On your local machine, open the command line console and run the following command:

```
cd "[Your Putty Folder]"
pscp.exe -r "[Your training Folder]\storm-kafka-demo\*" [username]@[DNSName]:/home/[username]/storm-kafka-demo
```

Replace the **[Your Putty Folder]** with the directory where the pscp.exe stored. Replace the **[Your training Folder]** with the directory where the training material stored. Replace the **[username]** with the username for you remote linux machine. Replace the **[DNSName]** with the dns name of you remote linux machine.

1. Use Leiningen to build the Storm uberjar. This produces a single, stand-alone file that can be submitted to a Storm cluster or launched as a locally-hosted server.

```
cd storm-kafka-demo
lein uberjar
```

2. Use Leiningen to build the Kafka client:

```
cd kafka-gps-client
lein uberjar
```

3. Use Node.js to build the web application:

```
cd ../node
npm install
```

The example is built and all necessary dependencies have been installed. Now we just need to run the example code.

## Launch Server Processes

There are several server processes that need to be started before we can launch the example. You can either follow the instructions below to launch all the servers from one console, or open several, separate console windows, connect to the VM in each window, and launch one server process per console. It's much easier to debug connectivity problems if you run each server in its own window. It is important to start the servers in the order shown below.

1. **Zookeeper:** Both Kafka and Storm use [Zookeeper](#) to track and manage server instances in their respective clusters. Zookeeper is also the endpoint for clients. Programs wishing to publish Kafka messages connect to the Zookeeper server to be forwarded to the least heavily loaded Kafka server.

```
zookeeper-server-start.sh ~/kafka-0.7.2-incubating-src/config/zookeeper.properties > ~/zookeeper-server.log 2>&1 &
```

This command starts the zookeeper server on **port 2181** as a background process and records all its output in the **zookeeper-server.log** file in your home directory. If you want to see what the server is doing, you can use the **tail** command:

```
tail -f ~/zookeeper-server.log
```

Press **Ctrl+C** to close tail.

2. **Kafka:** Once Zookeeper has started, start the Kafka server:

```
kafka-server-start.sh ~/kafka-0.7.2-incubating-src/config/server.properties > ~/kafka-server.log 2>&1 &
```

This command starts a single Kafka broker server on **port 9092** as a background process and records all its output in the **kafka-server.log** file in your home directory. If you want to see what the server is doing, you can use the **tail** command:

```
tail -f ~/kafka-server.log
```

Press **Ctrl+C** to close tail.

3. **Redis:** The Redis server provides a simple key-value store that works especially well with web applications written in Node.js. Start the redis server:

```
/usr/bin/redis-server > ~/redis-server.log 2>&1 &
```

4. To make this easier, we've included a convenience script, **startserverprocesses.sh** in *storm-kafka-demo* that will launch the server processes for you.

## Launch the Example

The example consists of three parts: a web application that presents vehicle real-time data, a Storm topology that processes the data, and a Kafka client application that produces the real-time data. We will start each part of the example in turn.

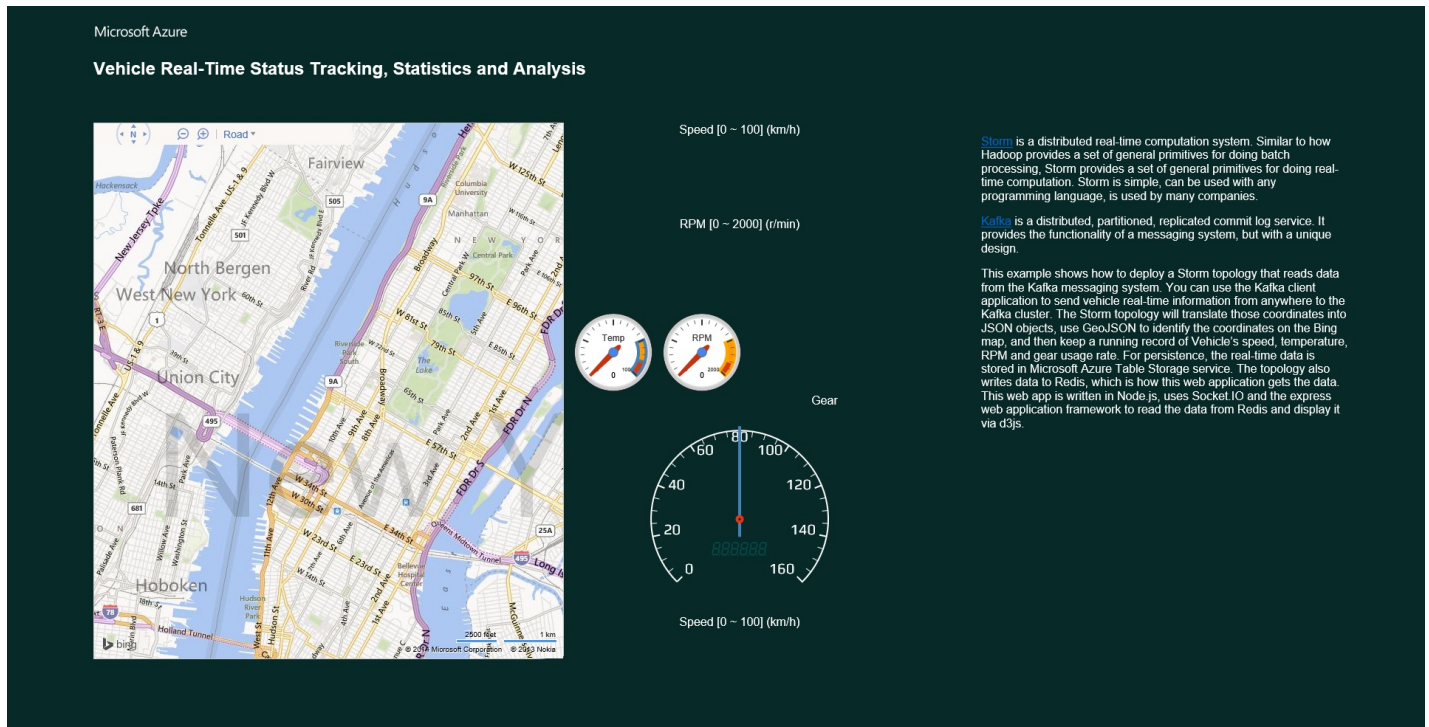
1. Start the web application. The application is served on **port 80** by default so we need to launch it as root.

```
cd $HOME/storm-kafka-demo/node
sudo node app.js
```

You should see output like this:

```
info - socket.io started
Listening on port 80
```

2. Open a web browser and navigate to the cloud service DNS name and you will see the demo application:



Nothing interesting is happening because we have not yet started the data stream. Leave this browser window open so you can see the effects of the following commands.

1. Open a second SSH connection to the VM (open on Windows or a new terminal on Linux or OS X) and launch the Storm topology:

```
cd $HOME/storm-kafka-demo
java -cp $(lein classpath) storm.example.KafkaGpsTopology
```

You should see a lot of output as the topology starts. Once it's up and running the output will look like:

```
4105 [Thread-25] INFO storm.kafka.PartitionManager - Starting Kafka 127.0.0.1:0 from offset 2185373
4106 [Thread-25] INFO backtype.storm.daemon.executor - Opened spout spout:(6)
4109 [Thread-25] INFO backtype.storm.daemon.executor - Activating spout spout:(6)
4224 [Thread-25] INFO storm.kafka.PartitionManager - Committing offset for 127.0.0.1:9092:0
4224 [Thread-25] INFO storm.kafka.PartitionManager - Comitted offset for 127.0.0.1:9092:0
```

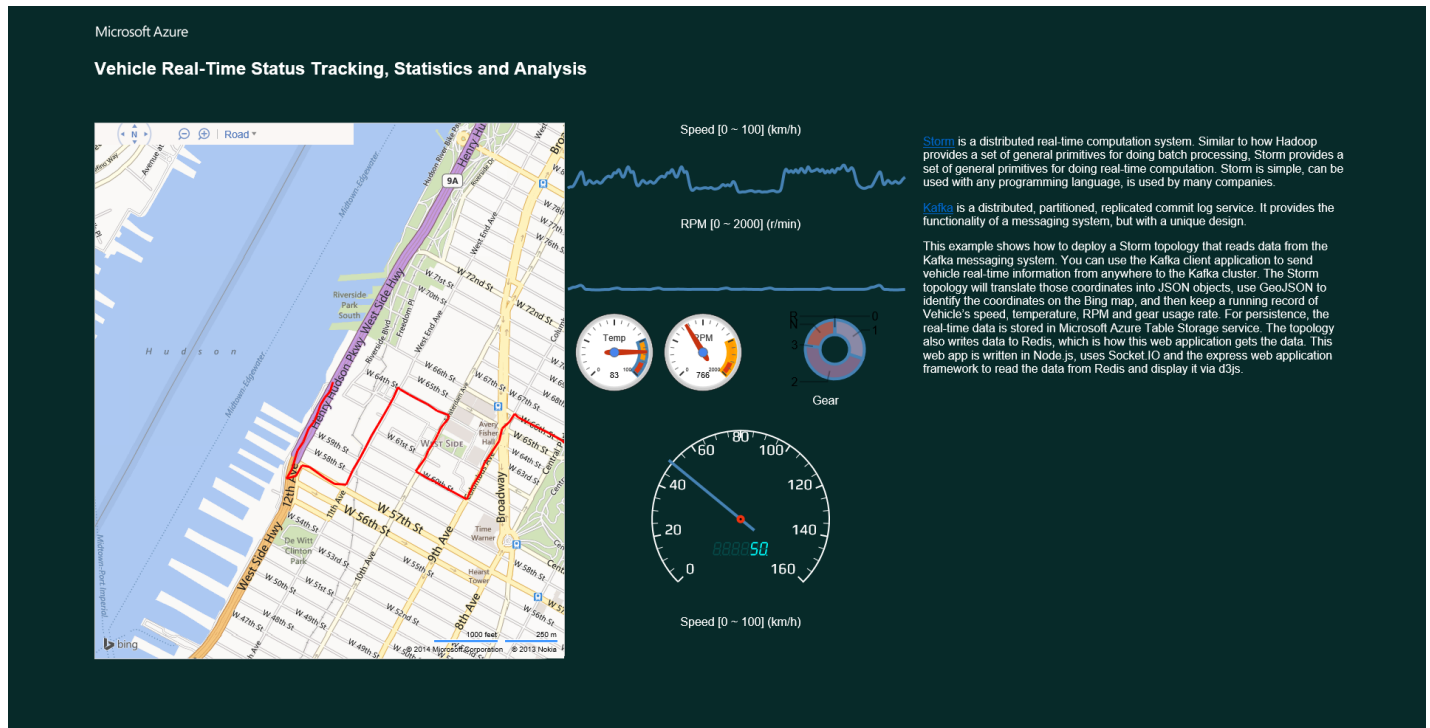
```
6241 [Thread-25] INFO storm.kafka.PartitionManager - Committing offset for 127.0.0.1:9092:0
6242 [Thread-25] INFO storm.kafka.PartitionManager - Comitted offset for 127.0.0.1:9092:0
```

You won't see any change your browser because although the topology has been started there is no data being sent to Kafka for the Storm topology to process.

1. Open a third SSH connection to the VM and start the Kafka client:

```
cd $HOME/storm-kafka-demo/kafka-gps-client
java -cp $(lein classpath) kafka.example.KafkaGpsDataProducer localhost
```

The client get the vehicle data and sends them to Kafka. Go back to your web browser and you'll see vehicle real-time data being plotted on the globe. The map will display the vehicle's path.



The client publishes data on the "gps" topic. **localhost** on the command line means we are connecting to Zookeeper on localhost to get connected to the Kafka server. You can specify the connection string as any of:

- `zookeeper_host`
- `zookeeper_host:port`
- `brokerid:kafkahost:kafka_port`

Copyright 2013 Microsoft Corporation. All rights reserved. Except where otherwise noted, these materials are licensed under the terms of the Apache License, Version 2.0. You may use it according to the license as is most appropriate for your project on a case-by-case basis. The terms of this license can be found in <http://www.apache.org/licenses/LICENSE-2.0>.