Scaling Internet of Things
Data Movement with
Amazon* Kinesis*

Collect, cache, and distribute high-throughput, low-latency machine
data coming from Intel® Gateway Solutions for the Internet of Things.

Turning Raw Data into Actionable Information in a Connected World
Collecting, storing, and analyzing high-throughput information can help companies stay
up to date on their business, customers, and assets. In the past, these capabilities required
complex software and a lot of infrastructure that was expensive to buy, provision, and
manage. Today, Amazon* Kinesis* makes it easy to set up high-capacity pipes that can
collect and distribute data in real time, at any scale – enabling fast movement of machine
data from edge to cloud for consumption by applications to make quick and decisive data-
driven actions.

So, instead of locking data away in large files that are not readily accessible, companies may
utilize Intel® Gateway Solutions for the Internet of Things (Intel® Gateway Solutions for the
IoT) to send each event to Amazon Kinesis, making them available for real-time processing.
As such,

• Data can be continuously analyzed without waiting until the end of the business day
• Key business metrics can be closely monitored via dynamic dashboards
• Data can be securely shared with third parties for additional applications

This paper explains how to stream data from an Intel® processor-based IoT gateway to
Amazon Kinesis, and how applications can receive streaming data from Amazon Kinesis, as
depicted in Figure 1.

Figure 1. High-Throughput Data Streaming from Gateway to Cloud via Amazon* Kinesis*
Intel® Gateways
A key building block of end-to-end IoT solutions, IoT gateways connect downstream to devices with sensors and controllers, and connect upstream to compute clouds on the Internet. Intel Gateway Solutions for the Internet of Things are powerful and versatile platforms for IoT gateway implementation. They support different types of machine and network connectivity, and include a secure and pre-validated solution stack for data aggregation and forwarding. In addition, they deliver the computing performance needed for many other tasks such as executing data analytics and business logic, as well as coordinating and managing a large number of devices.

Amazon® Kinesis*
When streaming real-time data from many data sources to the cloud, it is important to consider the ability of the cloud to sustain throughput and minimize latency. Providing exceptional capabilities on both these fronts, Amazon Kinesis is a fully-managed service for real-time reception and distribution of streaming data at a massive scale. It can continuously receive, cache, and forward terabytes of data per hour coming from hundreds of thousands of sources, while ensuring guaranteed throughput, low latency, and high reliability. It also provides an API for third-party applications to retrieve streaming data. In addition, Amazon Kinesis can readily interwork with other Amazon Web Services (AWS) components, including Amazon Simple Storage Service, Amazon Redshift, and Amazon Elastic Map Reduce.

In IoT deployments, edge gateways can stream data to Amazon Kinesis over the Internet, and Amazon Kinesis can distribute the data to end-user applications, other AWS components, or other cloud applications like big data analytics and data visualization.

Data Model
Amazon Kinesis is a real-time data distribution service based on a simple I/O abstraction called a “stream,” which is an ordered sequence of immutable data records. The data capacity of a stream is specified in units of “shards,” with one shard supporting up to 1000 writes and five reads per second, and up to a maximum of 1 MB data written and 2 MB data read per second. Once a stream object is instantiated, data records can be pushed into the stream on the input side, and data records can be pulled out of the stream on the output side by one or more applications, as illustrated in Figure 2.
Kinesis Programming with Python

Software, including Kinesis, can interact with AWS via HTTP-based APIs. Language bindings are available to simplify programming to AWS using Java, Python, Ruby, Javascript, etc.

The Python binding for AWS is known as "boto", which allows a Python program to access Amazon Kinesis via the construct of a “connect” object. The object encapsulates parameters for connecting to Amazon Kinesis and has methods for stream query, stream creation, data write to streams, and data read from streams, etc. Common functions are summarized in the following:

- Create the Connect Object
  ```python
kinesis = boto.connect_kinesis(
    "access_key_id",
    "secret_access_key"
    proxy = "ip_address",
    proxy_port = port
  )
  ```
- Check for a Stream
  ```python
  stream = None
  try:
    stream = kinesis.describe_stream(stream_name)
  ...
  except ResourceNotFoundException as rnfe:
    ...
  ```
- Create a Stream
  ```python
  kinesis.create_stream(stream_name, shard_count)
  ```
- Write to a Stream
  ```python
  response = kinesis.put_record(
    stream_name = stream_name,
    data = record,
    partition_key = partition_key
  )
  ```
- Read from a Stream
  ```python
  response = kinesis.get_shard_iterator(
    stream_name,
    shard_id,
    iterator_type
  )

  response = kinesis.get_records(next_iterator, limit=25)
  if len(response['Records']) > 0:
    ....
    next_iterator = response['NextShardIterator']
  ```
Amazon* Web Services (AWS) Signup

An Amazon account is required in order to use any AWS services, including Amazon Kinesis, so the first step is to create an Amazon account.

With an Amazon account, sign up for AWS services as follows:

2. Follow the on-screen instructions.

Upon completing the sign-up process, a confirmation email is sent. At any time, it is possible to view current account activity and manage accounts by going to http://aws.amazon.com and clicking My Account/Console.

Obtain Access Keys for AWS

For authentication and security purposes, each API request to AWS must include a pair of “access keys”: access key ID and secret access key.

These keys can be created from the AWS Management Console. It is recommended to use Identity and Access Management (IAM) access keys instead of AWS root account access keys because IAM allows users to explicitly control access to AWS services and resources in their AWS account.

1. To create IAM access keys:
2. Open the IAM console.
3. From the navigation menu, click Users.
4. Select your IAM user name.
5. Click User Actions, and then click Manage Access Keys.
6. Click Create Access Key.
7. Click Download Credentials, and store the keys in a secure location.

The access keys look something like this:
- Access key ID example: AKIAIOSFODNN7EXAMPLE
- Secret access key example: wJalrXUttnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY

These keys should be specified in the creation of the boto "connect" object.

Now create a group in IAM. Choose the Amazon Kinesis Full Access Policy template when creating the group and add the user you created to the group.

Prepare Intel® Gateway for AWS

The following steps set up the software environment for running boto-based applications on an Intel processor-based IoT gateway featuring Wind River* Linux*.

1. From a PC (Windows* or Linux*), download the following python packages (as tarballs):
   - setuptools (https://pypi.python.org/pypi/setuptools)
   - boto (https://pypi.python.org/pypi/boto/)
2. Connect the Intel gateway to the subnet on the PC.
3. Boot up the gateway.
4. Discover or determine the IP address acquired by the gateway.
5. Access the gateway using SSH.
6. Create a "Project" directory under the home directory.
7. Use SCP to transfer the two tarballs downloaded above to the "Project" directory.
8. On the SSH window, unpack the two tarballs:
   \texttt{tar -xvf "tarball"}
9. Install setup tools:
   \texttt{cd setuptools..}
   \texttt{python setup.py install}
10. Install boto:
    \texttt{cd boto...}
    \texttt{python setup.py install}

\textbf{Example}

In this example, CPU utilization data is "moved" in real time from a Intel processor-based IoT gateway to a PC via Amazon Kinesis, as shown in Figure 3. This is done by running a "producer" application on the gateway to send data to Amazon Kinesis, and running a "consumer" application on a PC to pull data out of Amazon Kinesis.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{data-transfer-example.png}
\caption{Data Transfer Example}
\end{figure}

\textbf{Real-Time Data Producer}

The following Python code listing of the “producer” application runs on the gateway and publishes CPU utilization data to a Kinesis stream in five second intervals.
import boto
import json
import time
import datetime

from boto.kinesis.exceptions import ResourceNotFoundException

class CPUutil(object):
    def __init__(self):
        self.prev_idle = 0
        self.prev_total = 0
        self.new_idle = 0
        self.new_total = 0

    def get(self):
        self.read()
        delta_idle = self.new_idle - self.prev_idle
        delta_total = self.new_total - self.prev_total
        cpuut = 0.0
        if (self.prev_total != 0) and (delta_total != 0):
            cpuut = ((delta_total - delta_idle) * 100.0 / delta_total)
        return cpuut

    def read(self):
        self.prev_idle = self.new_idle
        self.prev_total = self.new_total
        self.new_idle = 0;
        self.new_total = 0;
        with open('/proc/stat') as f:
            line = f.readline()
            parts = line.split()
            if len(parts) >= 5:
                self.new_idle = int(parts[4])
                for part in parts[1:]:
                    self.new_total += int(part)

if __name__ == '__main__':
    cpuutil = CPUutil()

    kinesis = boto.connect_kinesis(
        "access_key",
        "secret_key",
        proxy = "",
        proxy_port = 0,
    )

    stream_name = "ts_test"
    shard_count = 1
    partition_key = "cpu-ts"

    stream = None
    try:
        stream = kinesis.describe_stream(stream_name)
print(json.dumps(stream,
    sort_keys=True,
    indent=2,
    separators=(\'', ', ')))

except ResourceNotFoundException as rnfe:
    if stream is None:
        print('Could
    not
    find
    existing
    stream:{0}'.format(stream_name))
        kinesis.create_stream(stream_name, shard_count)
        print('new stream created')

while (True):
    data = cpuutil.get ()
    current_time = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    print("timestamp =", current_time, "AND cpu util = ", data)

    record = json.dumps(
        {
            'timestamp': current_time,
            'value': data
        }
    )

    response = kinesis.put_record(stream_name=stream_name,
        data=record,
        partition_key=partition_key)

    print("- put seqNum:", response['SequenceNumber'])

time.sleep(5)

The code does the following:

- Connect to Amazon Kinesis.
- Check for existence of the stream container and create the stream if it does not exist.
- Determine CPU utilization and send update to Kinesis every five seconds.

Save the code in a file called "pushcpu.py" and then download (via SCP) or copy the file onto the gateway.
On the SSH window, navigate to the location where pushcpu.py is saved on the gateway and run the program:

```
python pushcpu.py
```

Sample output is shown in Figure 4.

```
```

Amazon Kinesis Dashboard

Amazon Kinesis monitors the activities of every stream in use and collects performance data such as throughput, latency, and read/write counts. Authorized users may access the AWS portal to view time-series graphs of the data.

To do so:

1. Log in to the AWS Management Console from [https://aws.amazon.com](https://aws.amazon.com).
2. Click “Kinesis” to enter the Kinesis dashboard, which lists the streams that have been created.
3. Click the stream name of interest to enter its “Stream Details” page, which displays several performance graphs, as shown in Figure 5.
**Real-Time Data Consumer**

The following Python code listing of the “consumer” application runs on the PC and periodically retrieves CPU utilization data from the Kinesis stream.

```python
import sys
import boto
import json
import time
import datetime
from boto.kinesis.exceptions import ResourceNotFoundException

if __name__ == '__main__':
    ...  # code block
```
kinesis = boto.connect_kinesis(
    "asddfsasdfsdfsdfsdf",
    "dasfdfsdfsdfsdfsdfsdfsdfs",
    proxy = "",
    proxy_port = 0,
)
prevdata = 0
stream_name = "ts_test"
shard_id = 0
iterator_type = 'LATEST'
stream = None
try:
    stream = kinesis.describe_stream(stream_name)
    print (json.dumps(stream, sort_keys=True, indent=2, separators=(',', ': ')))

    shards = stream['StreamDescription']['Shards']
    print('# Shard Count:', len(shards))
    shard_id = shards[0]['ShardId']
    print('# shardId:', shard_id)

    response = kinesis.get_shard_iterator(
        stream_name,
        shard_id,
        iterator_type
    )
    next_iterator = response['ShardIterator']
    print ('Getting next records using iterator:', next_iterator)

except ResourceNotFoundException as rnfe:
    print ('stream (0) not present or not ready', stream_name)

while (True):
    response = kinesis.get_records(next_iterator, limit=25)
    if len(response['Records']) > 0:
        print ('Got (0) Worker Records',len(response['Records']))
        records = response['Records']
        for record in records:
            data = record['Data']
            ts_data = json.loads(data)
            print("timestamp =", ts_data['timestamp'], "AND cpu util =", ts_data['value'])

    next_iterator = response['NextShardIterator']
    time.sleep(5)

The code does the following:
- Connect to Kinesis.
- Check for presence of the data stream. Obtain a start pointer to the stream.
- Read data from the stream every five seconds.

Save the code in a file called "pullcpu.py" on a PC with Python 2.7 and "boto" installed.

Navigate to the location where pullcpu.py is saved on the PC and run the program:
python pullcpu.py
Sample output is shown in Figure 6.

Visit [http://aws.amazon.com/kinesis](http://aws.amazon.com/kinesis) for more information about Amazon Kinesis.

To learn more about Intel solutions for the IoT, visit [www.intel.com/iot](http://www.intel.com/iot).

By using this document, in addition to any agreements you have with Intel, you accept the terms set forth below.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL’S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

A “Mission Critical Application” is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL’S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS’ FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked “reserved” or “undefined”. Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: [http://www.intel.com/design/literature.htm](http://www.intel.com/design/literature.htm).

Copyright © 2014 Intel Corporation. All rights reserved. Intel and the Intel logo are trademarks of Intel Corporation in the United States and/or other countries.

*Other names and brands may be claimed as the property of others.

Printed in USA 1214/MG/TM/PDF Please Recycle 331564-002US

November 2014