

Chapter 11

Tools for IoT

INTERNET OF THINGS

A Hands-On Approach



Arshdeep Bahga • Vijay Madisetti

Outline

- Infrastructure automation & configuration management tools:
 - Chef
 - Puppet
- NETCONF and YANG case studies
- IoT code generator tool

The Purpose of Configuration Management Tools



Key Differences

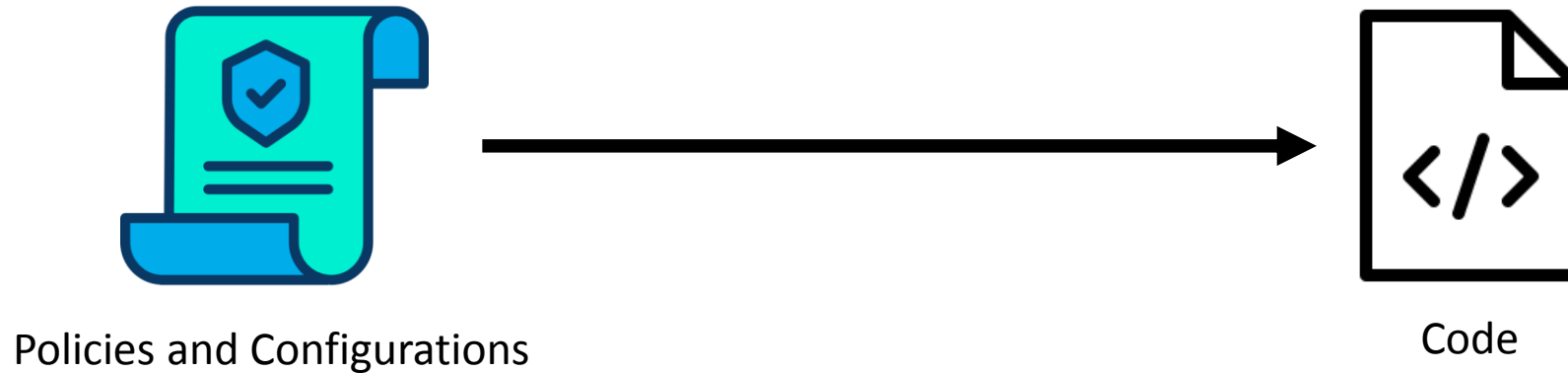
	Chef	Puppet
Scalability	High Scalability	High Scalability
Configuration Language	Imperative Language (Ruby)	Declarative Language (JSON/XML)
Ease of Setup	Not very easy	Not very easy
Availability	✓	✓
Management	Not very easy	Not very easy
Interoperability	High	High
Terms	Reciepes and Cookbooks	Manifests and Modules

Imperative Language: programming language that uses statements to change a program's state.

Declarative Language: programming language that expresses the logic of computation without describing the logic flow.

Introduction to Chef

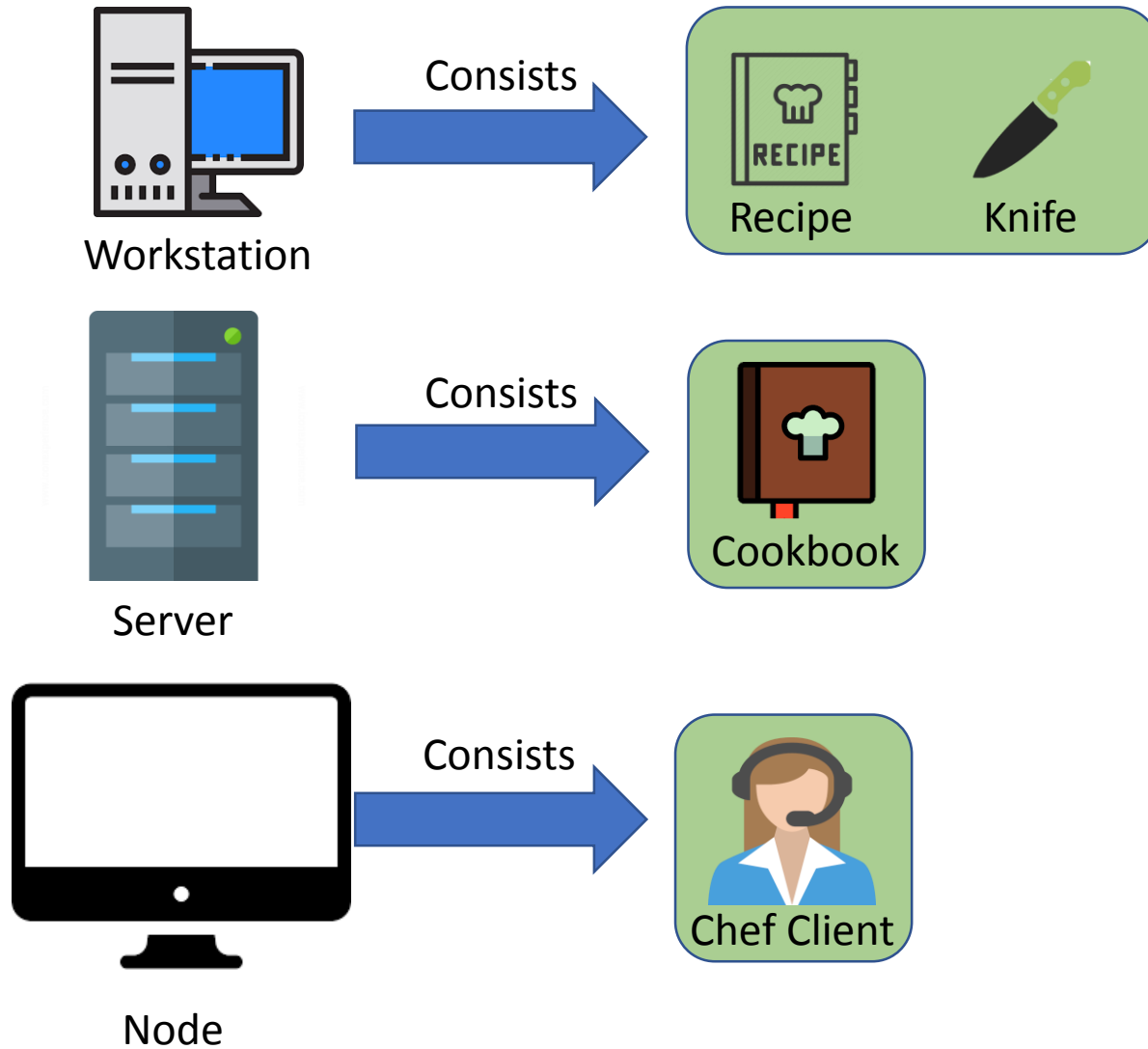
- Chef is an automation tool that converts infrastructure into code



This feature enables Chef to configure and manage multiple systems with ease



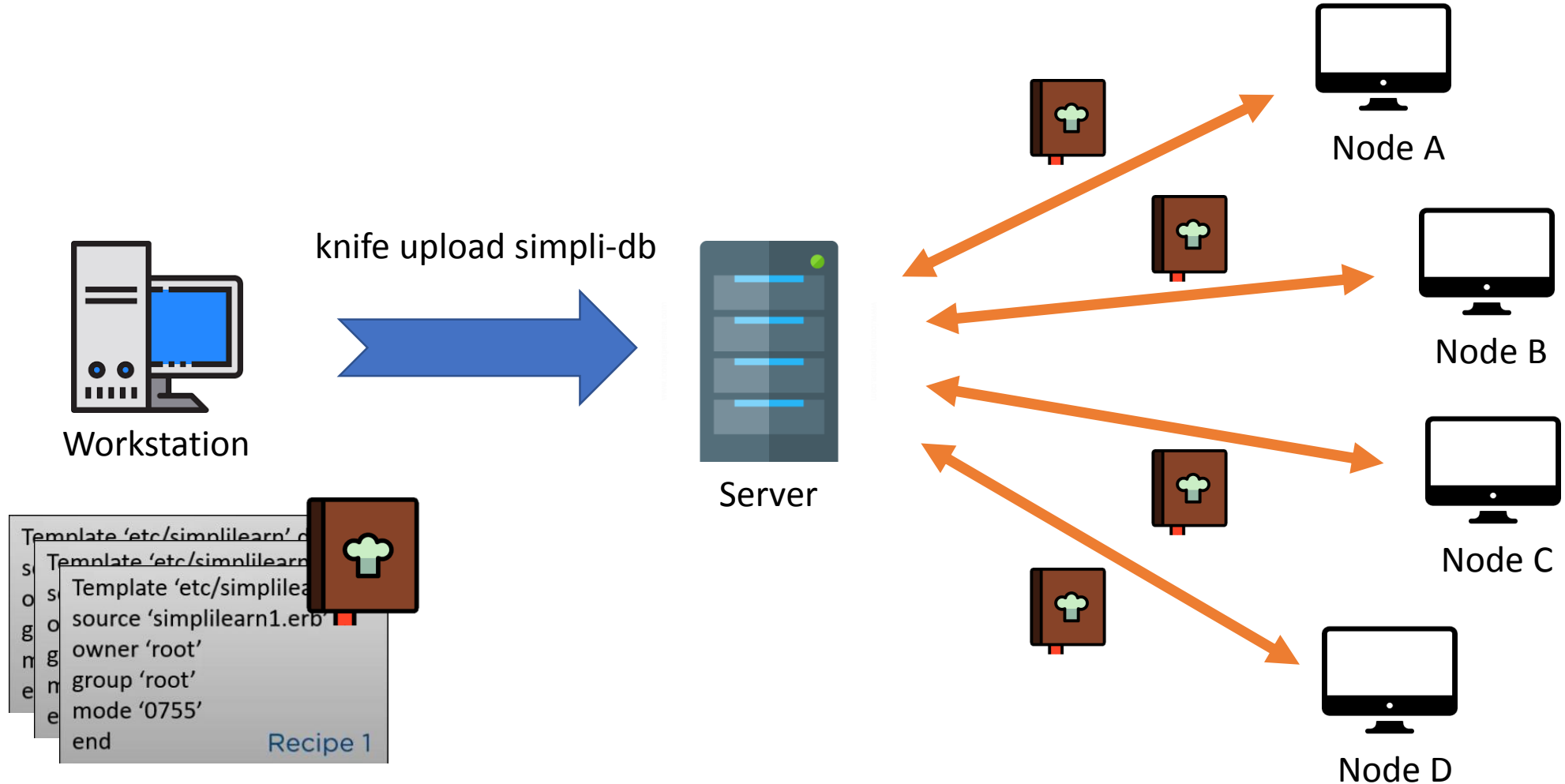
Components of Chef



Terms in Chef

- **Cookbook:** collection of recipes, attributes, templates and resources.
- **Recipe:** configuration element written in Ruby language that specifies various resources to be managed and how to manage the resources.
- **Resource:** fundamental unit of configuration.
- **Knife:** command utility that provides interface between workstation and server.
- **Run Lists:** ordered list of recipes and/or roles.

Architecture and Working of Chef



Setting up Chef – Hosted Enterprise Chef

3 steps needed to set up Chef Environment:

1. Set up Chef Server

(Signing up for a free trial Hosted Enterprise Chef and download a starter kit which includes PEM certificates)

2. Set up Workstation

Setting up Chef on workstation by simply run (for Linux workstation):

```
curl -L https://www.opscode.com/chef/install.sh | sudo bash
```

Above command will invoke the Chef's omnibus installer to install all you need to get started with Chef.

3. Setup Chef-client on nodes

Choose in which mode the node will run (cloud-based, physical or virtual). After choosing the mode, the node need to be bootstrapped. This process installs the Chef client and checks in with the Chef server. If Amazon EC2 is used, below is the knife command that need to be run on the Workstation:

```
knife bootstrap <IP-Address> -sudo -x ubuntu -I <keypair.pem> -N <nodeName>
```

For bootstrapping the node, EC2 keypair (PEM) is required.

Note: There are open source versions that can also be used to setup Chef Environment on your own node.

Chef Case Studies

There are 3 case studies shown in this Chapter:

1. Multi-tier Architecture Deployment

Infrastructure that consists of a number of tiers or layers.

2. Hadoop Cluster

Group of computers that are connected together to store and process large dataset.

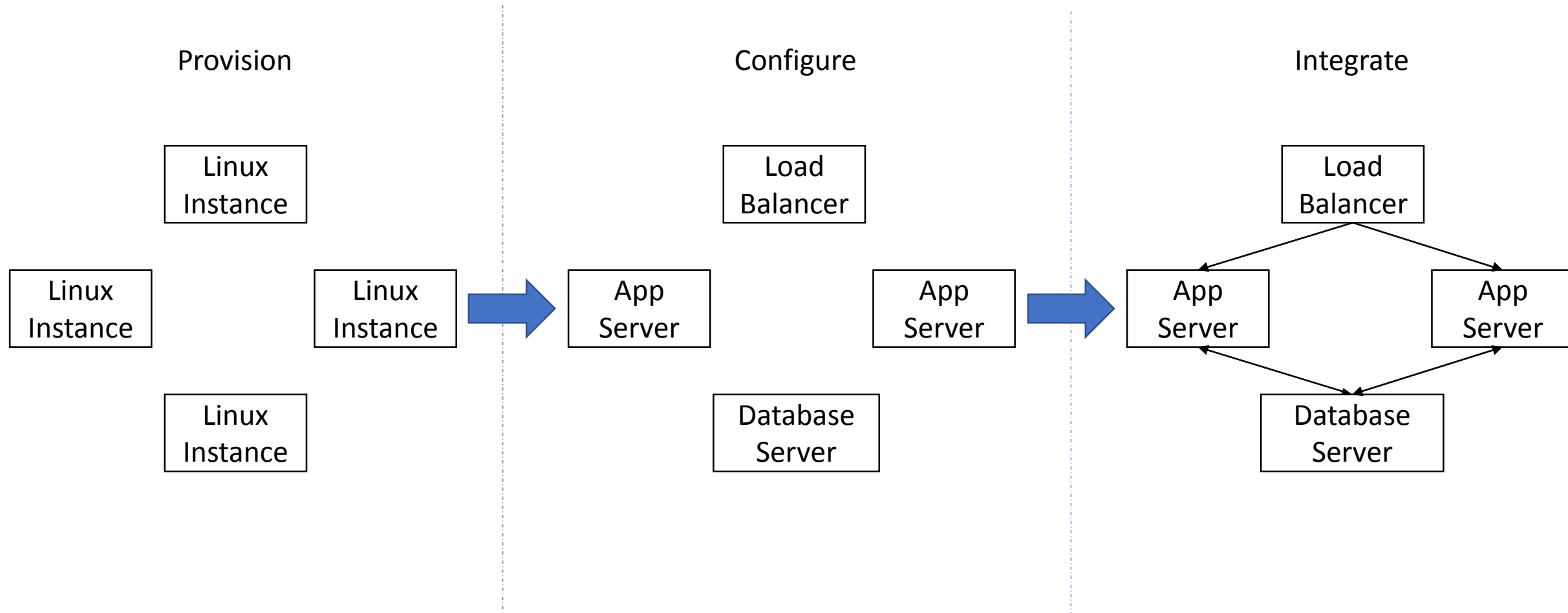
3. Storm Cluster

Real-time big data-processing system.

Multi-tier Architecture Deployment

- Multitier Application Architecture consists of
 - Presentation tier is a graphical user interface (GUI)
 - Application tier handles logic
 - Data tier stores information
- The most used multitier architecture is the three-tier application
 - The three tiers are logical, not physical, and may run on the same physical server

Three-tier Deployment with Chef

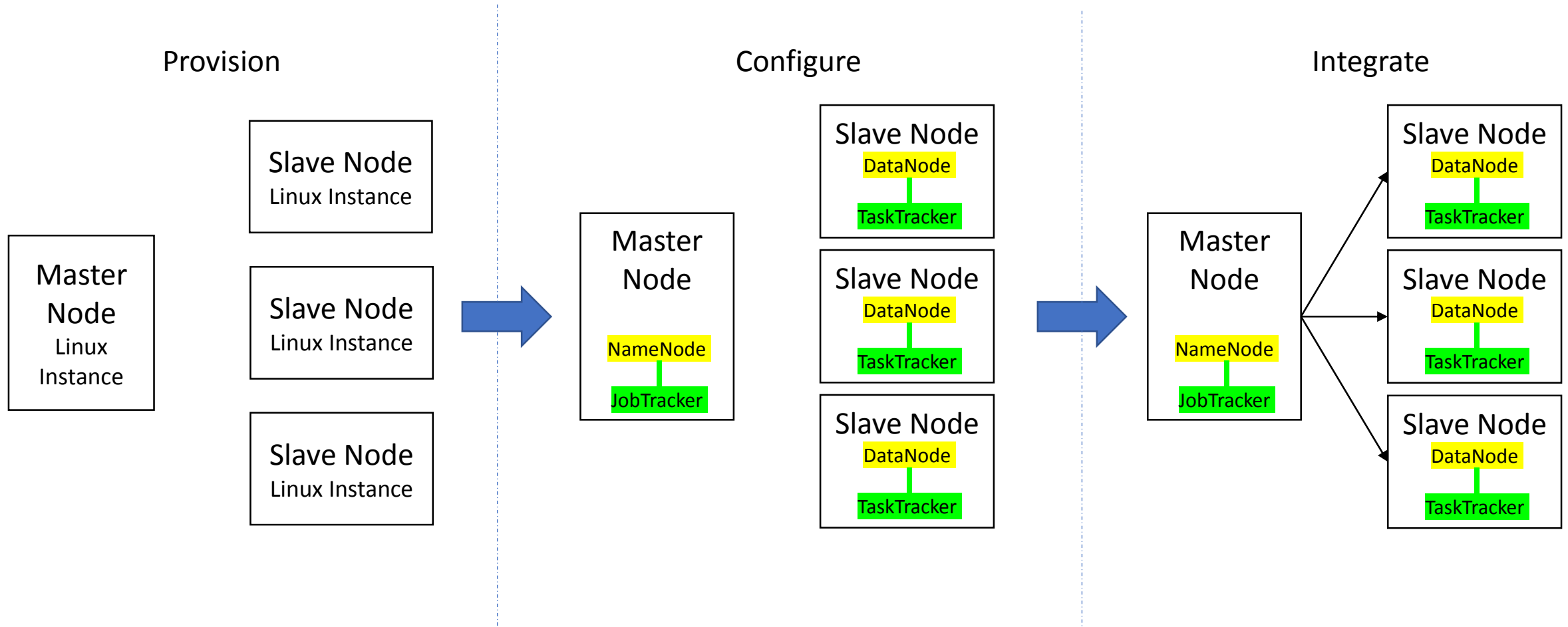


Hadoop Cluster

- Composed of a network of master and slave nodes
- The master nodes typically utilize higher quality hardware and include
 - NameNode
 - JobTracker.
- The slave nodes consist of virtual machines include
 - DataNode
 - TaskTracker

Do the actual work of storing and processing the jobs as directed by the master nodes.

Hadoop Cluster with Chef



Master Nodes

- The master nodes include a NameNode and JobTracker
 - NameNode is responsible to:
 - Store the metadata and another data related to DataNode.
 - Managing the file system namespace.
 - Control the access to the data blocks.
 - Check the availability of DataNodes.
 - JobTracker is responsible to:
 - Find the best TaskTracker nodes to execute task based on locality.
 - Monitor TaskTrackers
 - MapReduce execution

Slave Nodes

- The slave nodes include a DataNode and TaskTracker
 - DataNode is responsible to:
 - Main storage of data
 - Storing, creating, deleting jobs according to the instruction of NameNode
 - Send health report periodically to NameNode
 - TaskTracker is responsible to:
 - Execute Mapper and Reducer tasks
 - Signaling the progress to the JobTracker

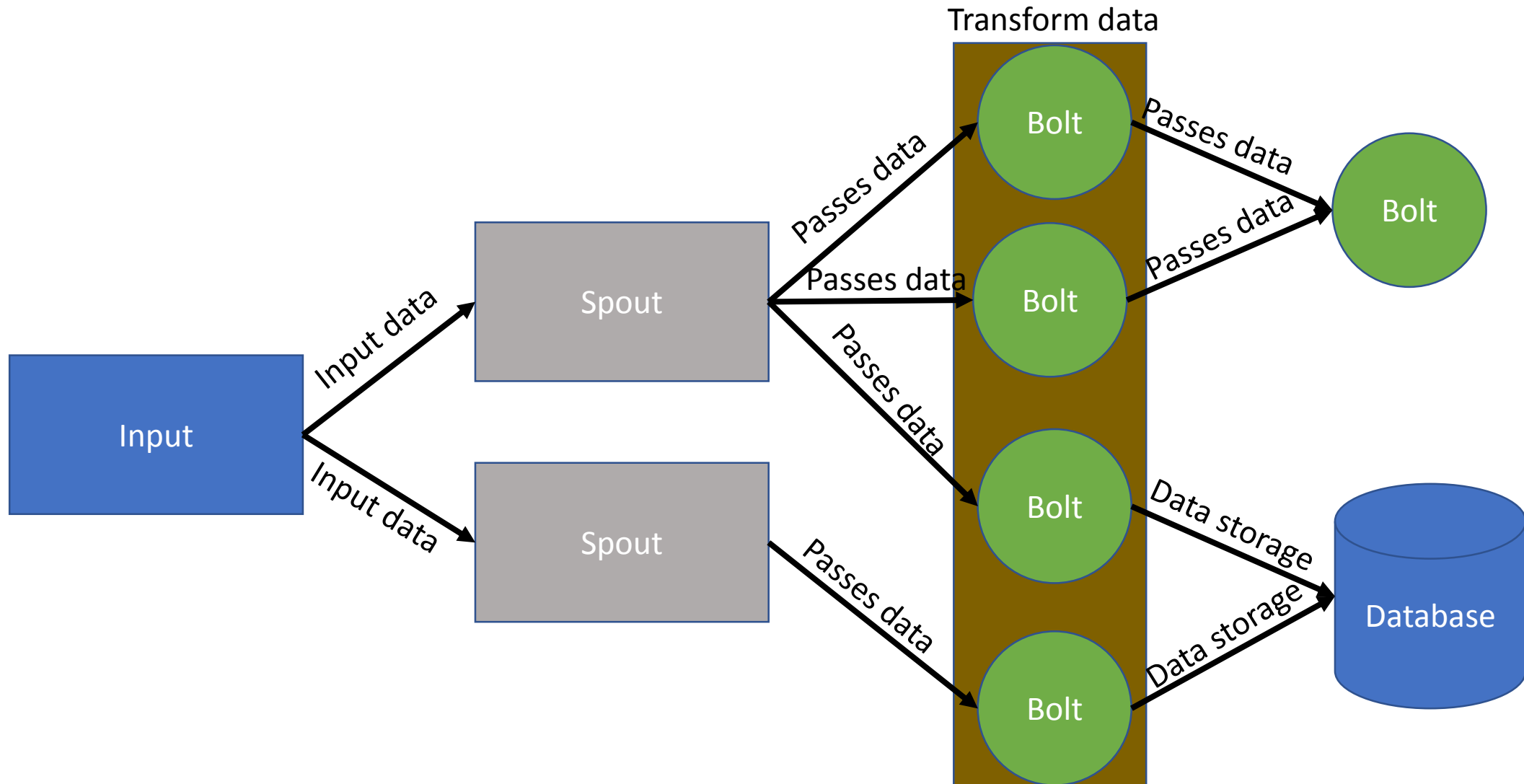
Advantages of Hadoop Cluster

- Scalable
- Flexible
- Fast
- Resilient to failure
- Cost effective

Storm Cluster

- Storm is a distributed, open-source and fault-tolerant system for processing **streams of data**.
- Each component has its own responsibility for a simple specific processing task.
 - The input stream of a Storm cluster is handled by a component called a **spout**.
 - The spout passes the data to a component called a **bolt** which then either sends the data in some storage, or passes it to some other bolt.
 - This arrangement of all the components (spout and bolts) and their connections is called a topology.

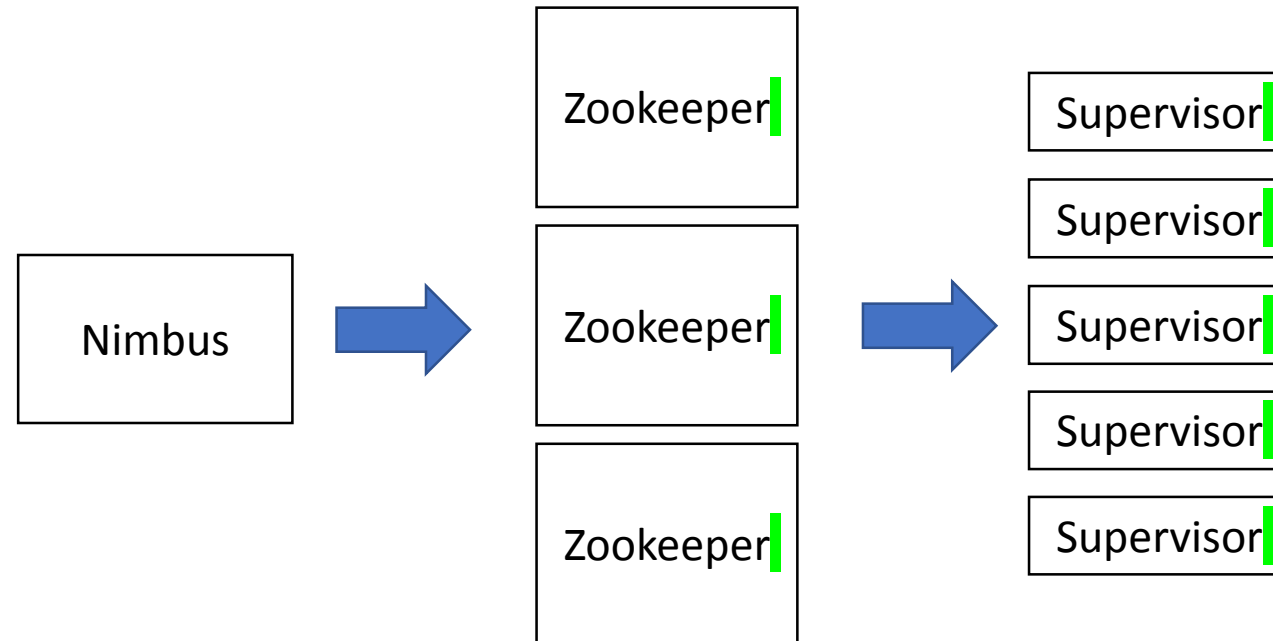
Architecture of Storm Cluster



Key Properties of Storm

- Extremely broad set of use cases
- Scalable
- Guarantees no data loss
- Extremely robust
- Fault-tolerant

Components of Storm Cluster (1/3)



Components of Storm Cluster (2/3)

- **Nimbus**

- Manage, coordinate and monitor topologies running on a cluster.
- Uploads computation for execution
- Launch workers across the cluster
- Track the status of all supervisor nodes and the tasks assigned to each.

- **ZooKeeper**

- ZooKeeper cluster is used by Storm to coordinate various processes.
- Storage for all of the states associated with the cluster and the various tasks submitted to the Storm.

Components of Storm Cluster (3/3)

- **Supervisor**

- Supervisor nodes are the worker nodes in a Storm cluster.
- Each supervisor node runs a supervisor daemon.
- Supervisor daemon responsible for creating, starting, and stopping worker process.

Storm Cluster vs Hadoop Cluster

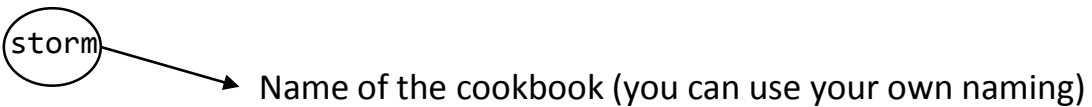
Storm Cluster	Hadoop Cluster
Real-time processing	Batch processing
With ZooKeeper based coordination	Without ZooKeeper based coordination
Nimbus and Supervisor	JobTracker and TaskTracker
Fast	Slow
Shutdown by the user or an unexpected unrecoverable failure	Sequential order system and completed eventually.
Nimbus dies nothing gets affected	MasterNode dies all the running jobs are lost.

Setting up Storm Cluster (1/8)

Creating cookbook and launch nodes

First step is to create a cookbook named storm with the following command:

```
knife cookbook create storm
```



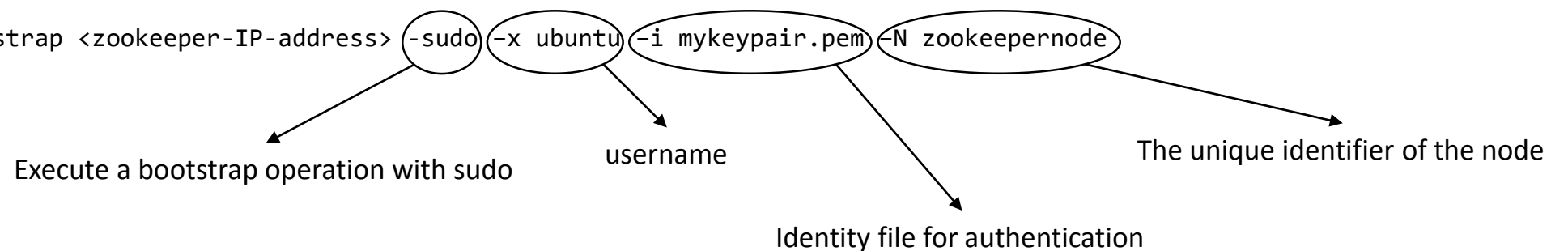
Name of the cookbook (you can use your own naming)

Launch three Amazon EC2 nodes (Nimbus, ZooKeeper and Supervisor) and bootstrap them with the following command:

```
knife bootstrap <nimbus-IP-address> -sudo -x ubuntu -i mykeypair.pem -N nimbusnode
```

```
knife bootstrap <supervisor-IP-address> -sudo -x ubuntu -i mykeypair.pem -N supervisornode
```

```
knife bootstrap <zookeeper-IP-address> -sudo -x ubuntu -i mykeypair.pem -N zookeepernode
```



Execute a bootstrap operation with sudo

username

Identity file for authentication

The unique identifier of the node

Setting up Storm Cluster (2/8)

Recipe for ZooKeeper

Recipe for setting up Zookeeper

```
setup_zk = Mixlib::ShellOut.new("echo \" deb [arch=amd64]  
http://archive.cloudera.com/cdh4/ubuntu/precise/amd64/cdh precise-cdh4 contrib\" >>  
/etc/apt/sources.list.d/cloudera.list", :cwd => '/home/ubuntu')  
setup_zk.run_command
```

```
setup_zk = Mixlib::ShellOut.new("echo \"deb-src  
http://archive.cloudera.com/cdh4/ubuntu/precise/amd64/cdh precise-cdh4 contrib\" >>  
/etc/apt/sources.list.d/cloudera.list", :cwd => '/home/ubuntu')  
setup_zk.run_command
```

```
setup_zk = Mixlib::ShellOut.new("apt-get -q -y update", :cwd => '/home/ubuntu')  
setup_zk.run_command
```

```
setup_zk = Mixlib::ShellOut.new("apt-get install -q -y zookeeper zookeeper-server", :cwd =>  
'/home/ubuntu')  
setup_zk.run_command
```

```
setup_zk = Mixlib::ShellOut.new("./zkServer.sh start", :cwd => '/usr/lib/zookeeper/bin/')  
setup_zk.run_command
```

This recipe is used to generate:

- SSH keys
- Setting up hosts
- Installing Java
- Setting up ZooKeeper
- Setting up Storm

Setting up Storm Cluster (3/8)

Recipe for Storm

Recipe for setting up Storm (1/3)

```
require 'chef/shell_out'
```

```
download_depend = Mixlib::ShellOut.new("apt-get install build-essential uuid-dev git pkg-config  
libtool autoconf automake", :cwd => '/home/ubuntu')  
download_depend.run_command
```

```
download_depend = Mixlib::ShellOut.new("apt-get install unzip", :cwd => '/home/ubuntu')  
download_depend.run_command
```

```
download_depend = Mixlib::ShellOut.new("wget http://download.zeromq.org/zeromq-2.1.7.tar.gz", :cwd  
=> '/home/ubuntu')  
download_depend.run_command
```

```
download_depend = Mixlib::ShellOut.new("tar -xzf zeromq-2.1.7.tar.gz", :cwd => '/home/ubuntu')  
download_depend.run_command
```

```
download_depend = Mixlib::ShellOut.new("./configure", :cwd => '/home/ubuntu/zeromq-2.1.7')  
download_depend.run_command
```

```
download_depend = Mixlib::ShellOut.new("make", :cwd => '/home/ubuntu/zeromq-2.1.7')  
download_depend.run_command
```

```
download_depend = Mixlib::ShellOut.new("make install", :cwd => '/home/ubuntu/zeromq-2.1.7')  
download_depend.run_command
```

Install dependencies

Download Zeromq

Un-tar the downloaded package

Configure Zeromq

Make and install Zeromq

Setting up Storm Cluster (4/8)

Recipe for Storm

Recipe for setting up Storm (2/3)

```
download_depend = Mixlib::ShellOut.new("export JAVA_HOME=/usr/lib/jvm/java-7-oracle", :cwd => '/home/ubuntu')
download_depend.run_command
```

Setting up JAVA_HOME

```
download_depend = Mixlib::ShellOut.new("git clone https://github.com/nathanmarz/jzmq.git", :cwd => '/home/ubuntu')
download_depend.run_command
```

Clone JZMQ

```
download_depend = Mixlib::ShellOut.new("touch classdist_noinst.stamp", :cwd => '/home/ubuntu/jzmq/src')
download_depend.run_command
```

Create a file

```
download_depend = Mixlib::ShellOut.new("CLASSPATH=../../:$CLASSPATH javac -d . org/zeromq/ZMQ.java
org/zeromq/ZMQException.java org/zeromq/ZMQQueue.java org/zeromq/ZMQForwarder.java
org/zeromq/ZMQStreamer.java", :cwd => '/home/ubuntu/jzmq/src')
download_depend.run_command
```

```
download_depend = Mixlib::ShellOut.new("./autogen.sh", :cwd => '/home/ubuntu/jzmq')
download_depend.run_command
```

Configure JZMQ

```
download_depend = Mixlib::ShellOut.new("./configure", :cwd => '/home/ubuntu/jzmq')
download_depend.run_command
```

```
download_depend = Mixlib::ShellOut.new("make", :cwd => '/home/ubuntu/jzmq')
download_depend.run_command
```

```
download_depend = Mixlib::ShellOut.new("make install", :cwd => '/home/ubuntu/jzmq')
download_depend.run_command
```

Make and install JZMQ

Setting up Storm Cluster (5/8)

Recipe for Storm

Recipe for setting up Storm (3/3)

```
download_storm = Mixlib::ShellOut.new("wget https://dl.dropbox.com/u/133901206/storm-0.8.2.zip", :cwd =>
'/home/ubuntu')
download_storm.run_command
```

Download Storm

```
download_storm = Mixlib::ShellOut.new("unzip storm-0.8.2.zip", :cwd => '/home/ubuntu')
download_storm.run_command
```

Unzip the downloaded package

```
download_storm = Mixlib::ShellOut.new("ln -s storm-0.8.2 storm", :cwd => '/home/ubuntu')
download_storm.run_command
```

Create a symbolic links

```
download_storm = Mixlib::ShellOut.new("chown -R ubuntu:ubuntu storm", :cwd => '/home/ubuntu')
download_storm.run_command
```

Gives the ownerships to ubuntu

```
# setup local directory
directory '/home/ubuntu/stormlocal' do
  owner "ubuntu"
  group "ubuntu"
  action :create
  recursive true
end
```

Setup Directory

Setting up Storm Cluster (6/8)

Chef Role

Chef role for setting up Storm cluster

```
name "storm_cluster_role"
description "Setup cluster nodes"
run_list [
  "recipe[storm::setup_hosts]",
  "recipe[storm::ssh_keys]",
  "recipe[storm::authorized_nodes]",
  "recipe[storm::setup_java]"
]
```

Chef role for setting up Zookeeper

```
name "storm_zookeeper_role"
description "Setup zookeeper node"
run_list [
  "recipe[storm::setup_zookeeper]"
]
```

ROLES

Chef role for setting up nodes for Storm cluster

```
name "storm_setup_role"
description "Setup storm nodes"
run_list [
  "recipe[storm::setup_storm]"
]
```

Setting up Storm Cluster (7/8)

Uploading cookbook and create roles

The Storm cookbook is then uploaded to the Chef server using the following commands:

```
Knife cookbook upload storm
```

Create roles on the server from the role files using the following commands:

```
Knife role from file storm_cluster_role.rb
```

```
Knife role from file storm_zookeeper_role.rb
```

```
Knife role from file storm_setup_role.rb
```

Add the roles to the run lists of the Nimbus, ZooKeeper and Supervisor nodes using the following commands:

```
Knife node run_list add nimbusnode 'role[storm_cluster_role]'
```

```
Knife node run_list add nimbusnode 'role[storm_setup role]'
```

```
Knife node run_list add supervisornode 'role[storm_cluster_role]'
```

```
Knife node run_list add supervisornode 'role[storm_setup role]'
```

```
Knife node run_list add zookeepernode 'role[storm_cluster_role]'
```

```
Knife node run_list add zookeepernode 'role[storm_zookeeper role]'
```

Setting up Storm Cluster (8/8)

Run chef-client from workstation

Chef-client is run on the nimbus, zookeeper and supervisor nodes (from the workstation) as follows:

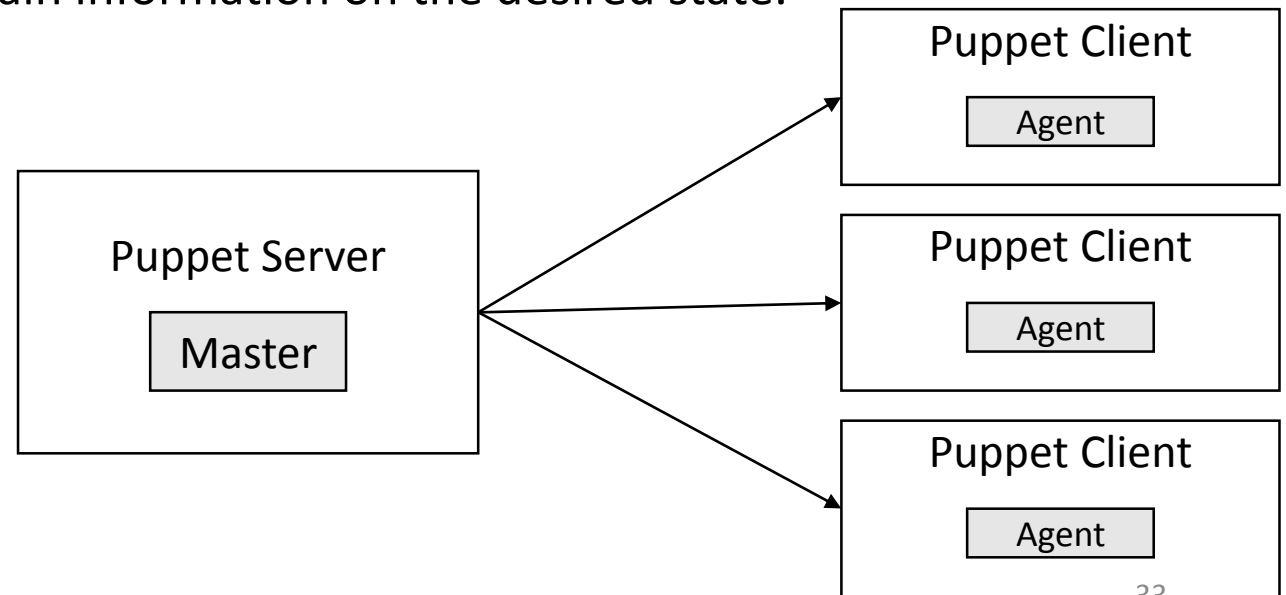
```
Knife ssh <nimbus-IP-address> 'sudo chef-client' -m -x ubuntu -i mykeypair.pem
```

```
Knife ssh <supervisor-IP-address> 'sudo chef-client' -m -x ubuntu -i mykeypair.pem
```

```
Knife ssh <zookeeper-IP-address> 'sudo chef-client' -m -x ubuntu -I mykeypair.pem
```


Puppet

- Puppet is also a configuration management tool that can be used to manage configurations on a variety of platforms.
- It is usually deployed in a client-server model. Server runs Puppet Master and client runs the Puppet Agents.
- Puppet Master maintains the configuration information for the clients.
- Puppet Agents connect to the master to obtain information on the desired state.
- Uses declarative modeling language.



Puppet – Key Concepts

Resources: It is a fundamental unit of configuration. Similar resources are grouped together into resources types.

Resource Abstraction Layer (RAL): consist of high-level modules (types) and platform-specific implementations (providers).

Module: Consists of multiple files containing the class definitions.

Puppet
Key Concepts



Class: Define a collection of resources which are managed together as a single unit.

Manifests: Puppet programs.

Multi-tier Deployment with Puppet (1/6)

Directory Structure

- Creating a multi-tier deployment comprising of haproxy, load balancer, Django application server and MongoDB database server will be learned in this case study.
- First of all, create a puppet module with the following directory structure and the files on the Puppet master node:

```
-/etc
|-puppet
  |-modules
    |-threetierdeployment
      |-manifests
      |-init.pp
      |-haproxy.pp
      |-django.pp
      |-mongodb.pp
      |-templates
      |-haproxy.cfg.erb
```

Multi-tier Deployment with Puppet (2/6)

haproxy class configuration

```
class haproxy(  
  $global_options = {  
    'chroot' => '/var/lib/haproxy',  
    'pidfile' => '/var/run/haproxy.pid',  
    'maxconn' => '4000',  
    'user' => 'haproxy',  
    'group' => 'haproxy',  
    'daemon' => '',  
    'stats' => 'socket /var/lib/haproxy/stats'  
  },  
  $defaults_options = {  
    'log' => 'global',  
    'stats' => 'enable',  
    'option' => 'redispatch',  
    'retries' => '3',  
    'timeout' => [  
      'http-request 10s',  
      'queue 1m',  
      'connect 10s',  
      'client 1m',  
      'server 1m',  
      'check 10s',  
    ],  
    'maxconn' => '8000'  
  },  
)  
{  
  package { 'haproxy':  
    ensure => present,  
  }  
  file { 'haproxy-config':  
    path => '/etc/haproxy/haproxy.cfg',  
    content => template('/etc/puppet/modules/threetierdeployment/haproxy.cfg.erb'),  
  }  
}
```

On the left is the haproxy class which contains a package resource definition for installing haproxy and file resource for configuration file (haproxy.cfg)

Multi-tier Deployment with Puppet (3/6)

Puppet Configuration File

On the left is the template for the configuration file.

Global

```
<% @global_options.each do |key,val| -%>
<% if val.is_a?(Array) -%>
<% val.each do |item| -%>
  <%= key %> <%= item %>
<% end -%>
<% else -%>
  <%= key %> <%= val %>
<% end -%>
<% end -%>
```

defaults

```
<% @defaults_options.each do |key,val| -%>
<% if val.is_a?(Array) -%>
<% val.each do |item| -%>
  <%= key %> <%= item %>
<% end -%>
<% else -%>
  <%= key %> <%= val %>
<% end -%>
<% end -%>
```

Multi-tier Deployment with Puppet (4/6)

Django class configuration

Below is the Django class that contains package resources for python-pip provider and Django.

```
class django{
  package { 'python-pip':
    ensure => installed,
  }
  package { 'django':
    ensure => installed,
    provider => 'pip',
    require => Package["python-pip"],
  }
}
```

Multi-tier Deployment with Puppet (5/6)

Database Configuration

Below is the Mongodb class that commands for setting up MongoDB.

```
class mongodb{
  exec { "cmd1":
    command => "/usr/bin/apt-key adv -keyserver
    keyserver.ubuntu.com --recv 7F0CEB10",
  }

  exec { "cmd2":
    command => "/bin/echo'deb http://downloads-distro.mongodb.org/repo/ubuntu-upstart dist 10gen' $>>$
    /etc/apt/sources.list.d/10gen.list",
    require => Exec["cmd1"],
  }

  exec { "cmd3":
    command => "/usr/bin/apt-get update",
    require => Exec["cmd2"],
  }

  exec { "cmd4":
    command => "/usr/bin/apt-get install mongodb-10gen",
    require => Exec["cmd3"],
  }
}
```

Multi-tier Deployment with Puppet (6/6)

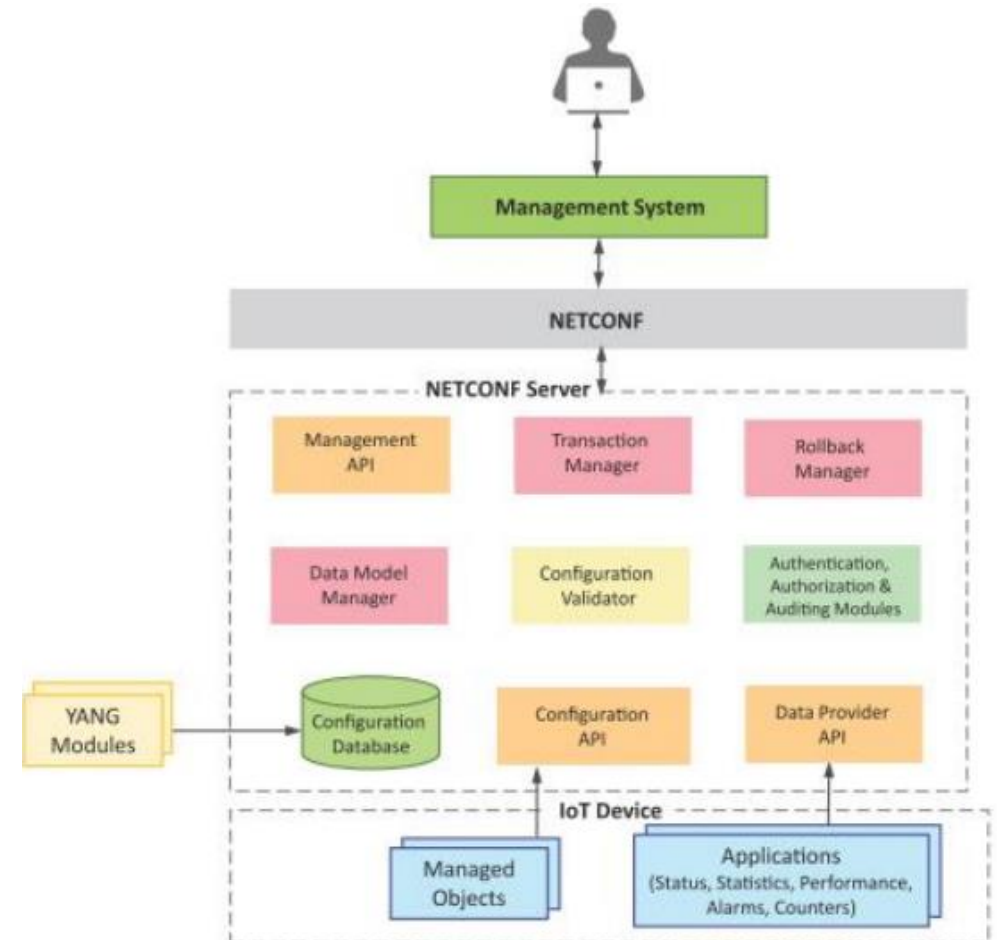
Running Puppet Agent

To apply the Puppet module on the clients nodes, run the Puppet agent on each client node as follows:

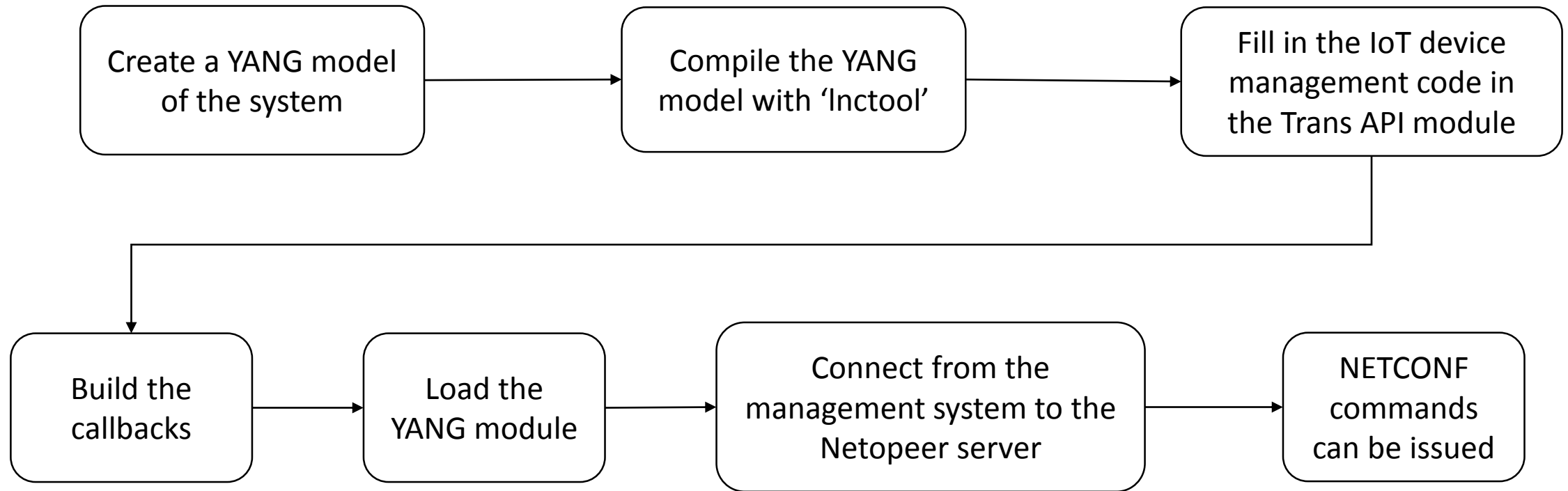
```
sudo puppet agent -t
```


IoT System Management with NETCONF-YANG

- Management System
- Management API
- Transaction Manager
- Rollback Manager
- Data Model Manager
- Configuration Validator
- Configuration Database
- Configuration API
- Data Provider API



Steps for IoT device Management with NETCONF-YANG

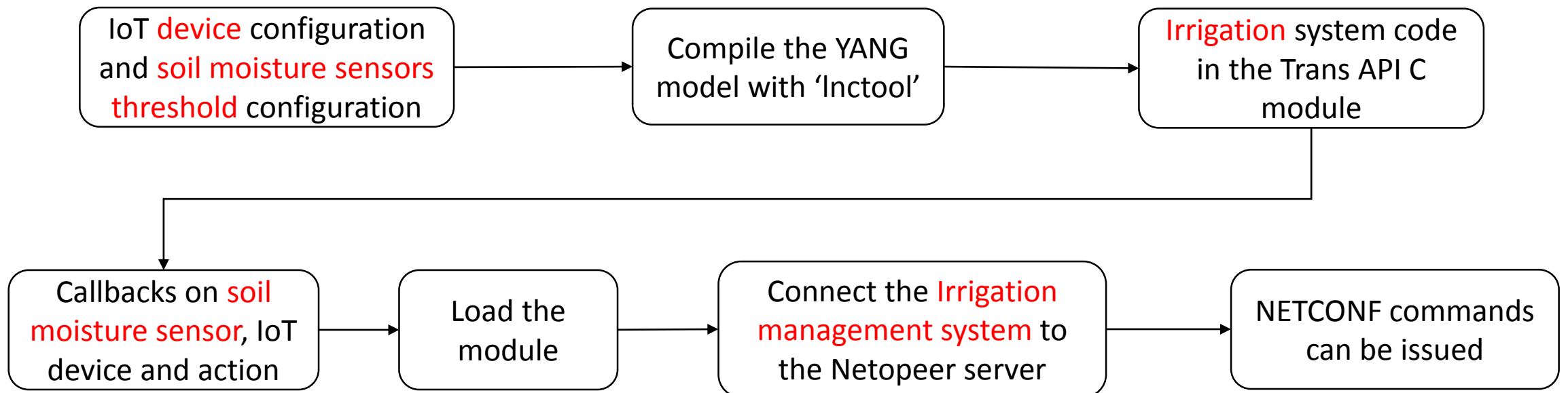


NETCONF-YANG Case Study

- Managing Smart Irrigation IoT System
- Managing Home Intrusion Detection IoT System

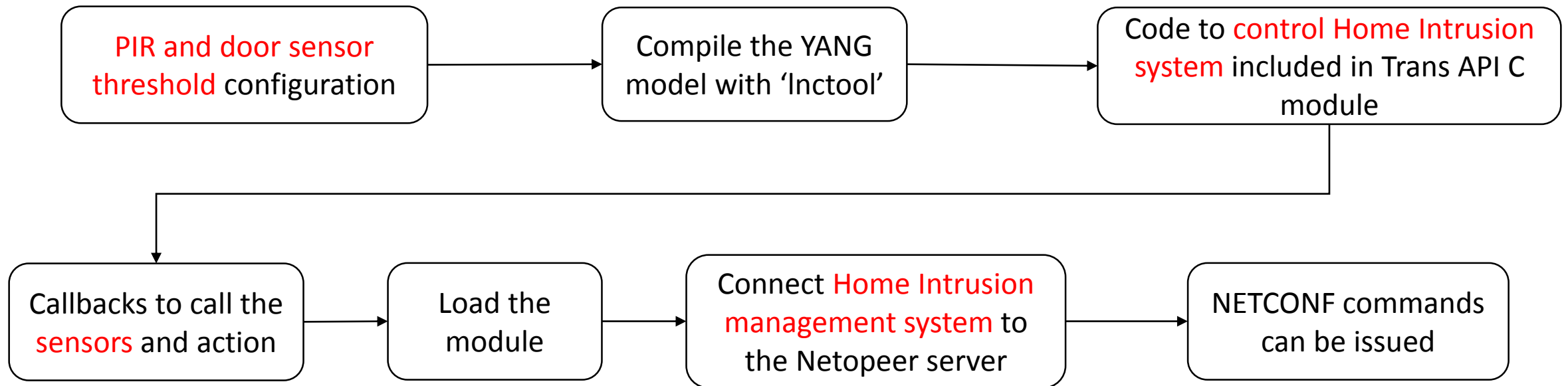
Smart Irrigation IoT System

- The flow behind this case study:
 - This case study uses an IoT device and soil moisture sensors.
 - The sensor is used to measure the amount of moisture in the soil
 - If the level of moist above threshold, it will release the flow of water through the irrigation pipes.



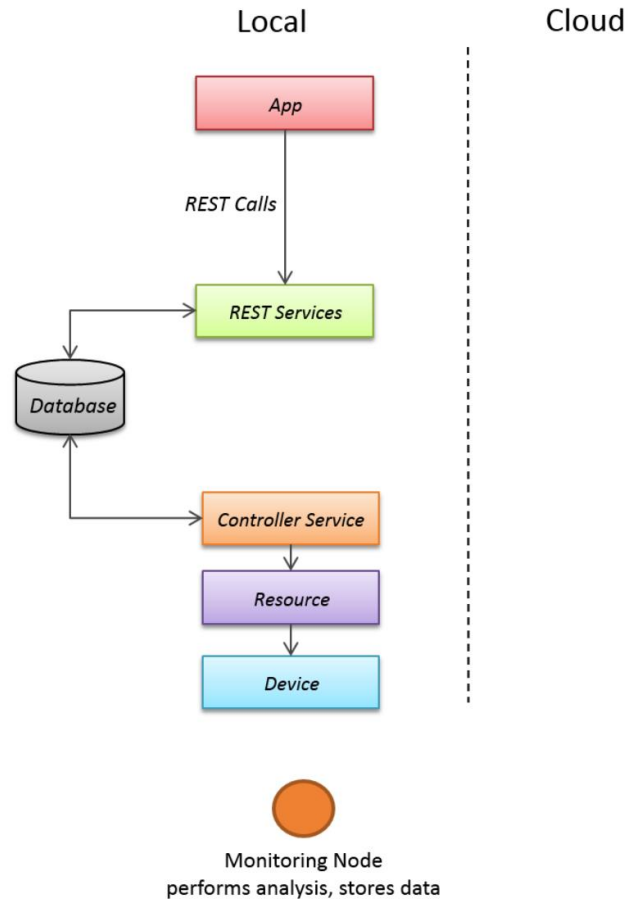
Home Intrusion Detection IoT System

- The purpose is to detect intrusion using sensors.
 - PIR sensor and door sensor.
 - Will raised alerts if intrusion detected.



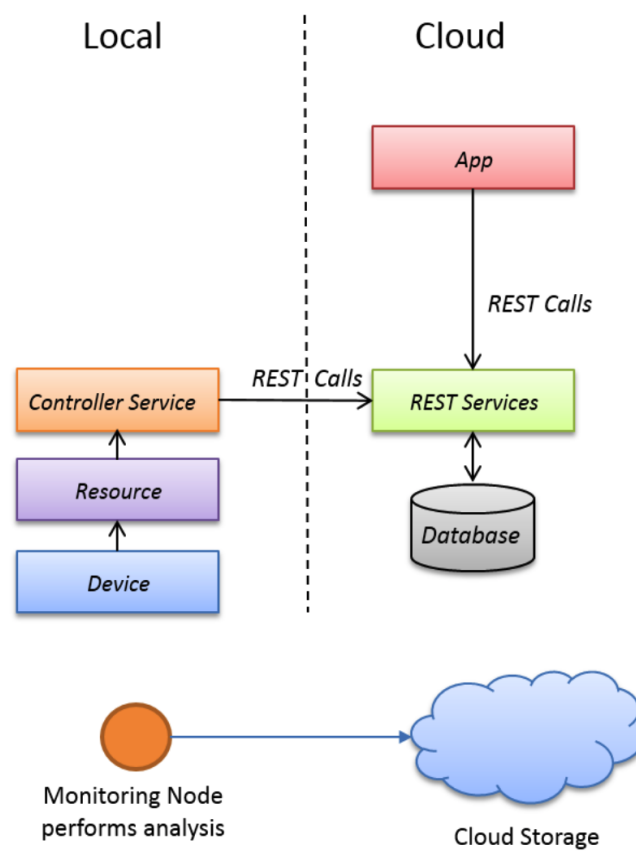
IoT Code Generator Levels (1/2)

Level-1



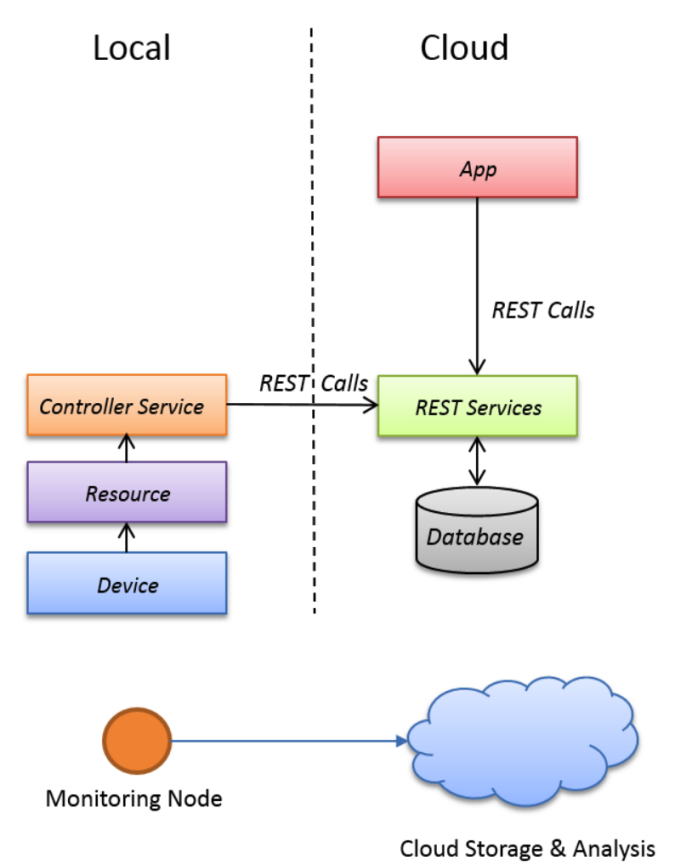
In this level there is a single node that stores data, performs analysis and hosts the app.

Level-2



In this level there is a single node that performs local analysis. Data is stored in the cloud and app is cloud-based.

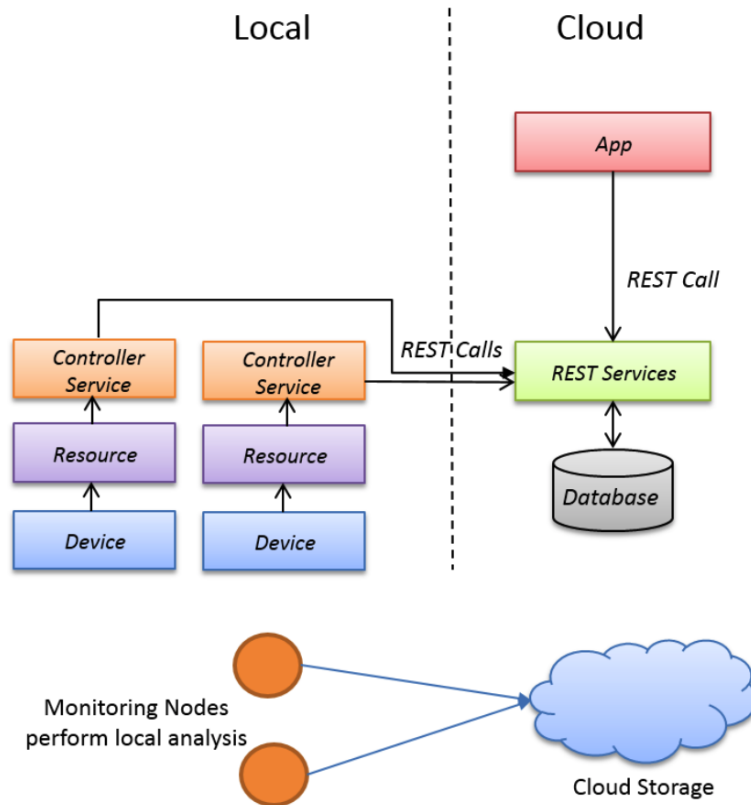
Level-3



In this level there is a single node. Data is stored and analyzed in the cloud and app is cloud-based.⁴⁶

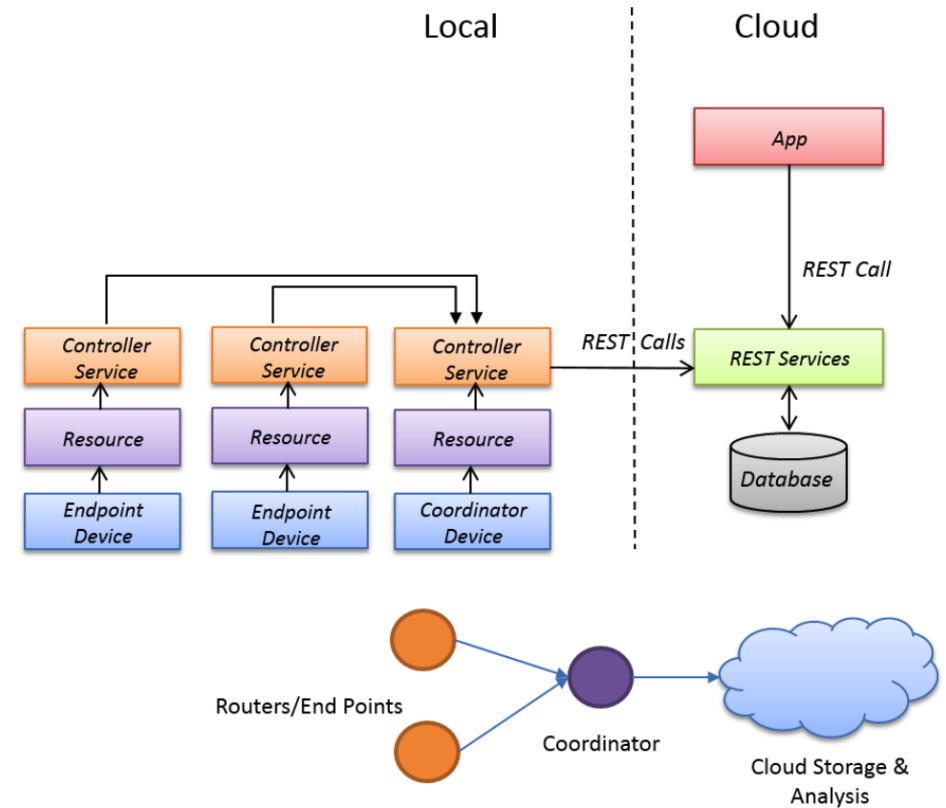
IoT Code Generator Levels (2/2)

Level-4



In this level there are multiple nodes that perform local analysis.
Data is stored in the cloud and app is cloud-based.

Level-5



In this level there are multiple nodes. Data is stored and analyzed
in the cloud and app is cloud-based.

IoT Code Generator (1/7)

Choose Board

To begin with, the user selects an IoT level for the system for which the code is to be generated. The next step is to select an IoT device as shown below.

The screenshot displays the 'IoT Code Generator' interface at 'Level-1' for 'Generate Controller & App Code'. On the left, a vertical list of four steps is shown: 'Step 1 Choose Board' (highlighted in orange), 'Step 2 Select Sensors' (green), 'Step 3 Configure Storage' (green), and 'Step 4 Generate Code' (grey). The main content area is titled 'Choose Board' and contains a 'Choose Board:' label above a dropdown menu with 'Raspberry Pi' selected. At the bottom right of the main area are three buttons: 'Previous' (disabled), 'Next' (active), and 'Finish' (disabled).

IoT Code Generator (2/7)

Select Sensors

Select the sensors that will be used.

IoT Code Generator

Level-1

Generate Controller & App Code

1 Step 1
Choose Board

2 Step 2
Select Sensors

3 Step 3
Configure Storage

4 Step 4
Generate Code

Select Sensors

Select Sensors:

DHT22 - Temperature & Humidity Sensor

BMP085 - Pressure & Temperature Sensor

LDR - Light Sensor

Ultrasonic Sensor

PIR Sensor

Magnetic Door Sensor

Soil Moisture Sensor

MICS-2710 - NO2 Concentration Sensor

MICS-5525 - CO Concentration Sensor

HRLV-MaxSonar - Long Range Ultrasonic Sensor

Hold down the Ctrl button to select multiple options.

Previous

Next

Finish

IoT Code Generator (3/7)

Select Database

Storage option is selected as shown below.

IoT Code Generator

Level-1

Generate Controller & App Code

1 Step 1
Choose Board

2 Step 2
Select Sensors

3 Step 3
Configure Storage

4 Step 4
Generate Code

Configure Storage

Local Storage

✓ None

SQLite

MySQL

Previous

Next

Finish

50

IoT Code Generator (4/7)

Configure Database

Storage option is selected and configured as shown below.

IoT Code Generator

Level-1

Generate Controller & App Code

1 Step 1
Choose Board

2 Step 2
Select Sensors

3 Step 3
Configure Storage

4 Step 4
Generate Code

Configure SQLite Storage

Database Host

Database User

Database Name

Database Password

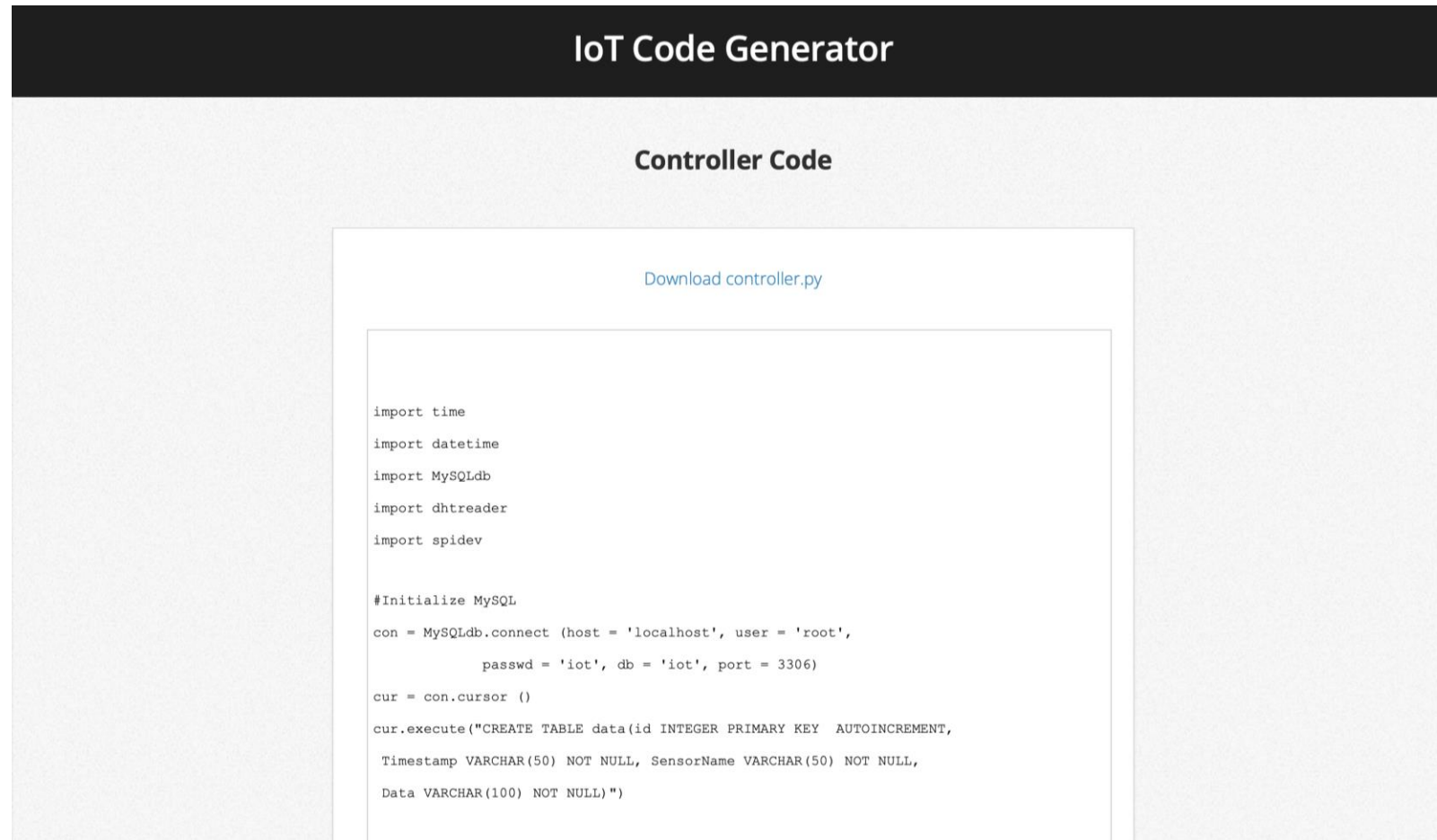
When using REST services to access the data in MySQL database, set the database table names in the generated controller code same as the tables generated by the REST framework for the corresponding services. Also, use the same database host, name, user and password for the REST framework as in the controller.

[Previous](#) [Next](#) [Finish](#)

IoT Code Generator (5/7)

Generated Codes (1/2)

The code generator generates the controller code and app code.



The screenshot displays the 'IoT Code Generator' application. Under the 'Controller Code' section, there is a blue link labeled 'Download controller.py'. Below this link, a code editor shows the following Python code:

```
import time
import datetime
import MySQLdb
import dhtreader
import spidev

#Initialize MySQL
con = MySQLdb.connect (host = 'localhost', user = 'root',
                        passwd = 'iot', db = 'iot', port = 3306)
cur = con.cursor ()
cur.execute("CREATE TABLE data(id INTEGER PRIMARY KEY AUTOINCREMENT,
Timestamp VARCHAR(50) NOT NULL, SensorName VARCHAR(50) NOT NULL,
Data VARCHAR(100) NOT NULL)")
```

IoT Code Generator (5/7)

Generated Controller Codes

```
import time
import datetime
import MySQLdb

#Initialize MySQL
con = MySQLdb.connect (host = 'localhost', user = 'iot', passwd = 'iot', db = 'iot', port = 3306)
cur = con.cursor ()
cur.execute("CREATE TABLE data(id INTEGER PRIMARY KEY AUTOINCREMENT, Timestamp VARCHAR(50) NOT NULL, SensorName VARCHAR(50) NOT NULL, Data VARCHAR(100) NOT NULL)")

#Controller setup function
def setupController():
    return true

#Controller main function
def runController():
    timestamp=datetime.datetime.utcnow()
    print timestamp
    return true

setupController()
while True:
    runController()
    time.sleep(10)
```

Import necessary Library

Initiate connection to DB

Create table for the data

It is the controller setup function.

Function to run your controller.

Initialize the controller

Run the controller and set delay time 10 seconds

IoT Code Generator (5/7)

Generated Codes (2/2)

The code generator generates the controller code and app code.

IoT Code Generator

App Code

[Download views.py](#)

```
from django.shortcuts import render_to_response
from django.template import RequestContext
import requests
import json

def home(request):

    #Get dht22 data (for id=1)
    r=requests.get('http://service-endpoint/dht22/1', auth=('username','password'))
    result=r.text
    output = json.loads(result)
    dht22timestamp=output['Timestamp']
    dht22data=output['Data']
```

IoT Code Generator (5/7)

Generated App Codes

```
def home(request):  
    return render_to_response('template.html', {}, context_instance=RequestContext(request))  
  
urlpatterns = patterns(  
    url(r'^home/', 'myapp.views.home'),  
)  
  
<!DOCTYPE HTML>  
<html>  
  <head>  
    <title>App</title>  
    <meta http-equiv="content-type" content="text/html; charset=utf-8" />  
  </head>  
  <body>  
    <h2>Sensor Data</h2>  
    <table width="50%">  
      <tr>  
        <td>Sensor Name</td>  
        <td>Timestamp</td>  
        <td>Data</td>  
      </tr>  
    </table>  
  </body>  
</html>
```

—————→ Display the data in
template.html format

—————→ Pattern for URL

—————→ HTML template

IoT Code Generator (6/7)

Generate Service Code (1/2)

Below are the wizards for generating the services code.

IoT Code Generator

Level-1

Generate Services Code

1 Step 1
Upload Specifications

2 Step 2
Generate Code

Upload Service Specifications

Choose the specifications file:

Choose File no file selected

Submit service specifications formatted as JSON. For example:

```
{  "service": [{    "name": "state",    "fields": { "name": "CharField" }  }]}
```

Previous

Next

Finish

IoT Code Generator (6/7)

Generate Service Code (2/2)

Below are the wizards for generating the services code.

IoT Code Generator

Level-1

Generate Services Code

1 Step 1
Upload Specifications

2 Step 2
Generate Code

Generate Code

Click finish to generate services code.

The Services Code Generator generates services code based on the Django REST Framework. To ensure that the data is updated correctly from the controller in the database, set the database table names in the generated controller code same as the tables generated by the REST framework for the corresponding services. Also, use the same database host, name, user and password in the controller as in the REST framework.

Previous

Next

Finish

IoT Code Generator (7/7)

Generate Service Code

The generated service code shown below.

IoT Code Generator

Controller Code

models.py
[Download models.py](#)

```
from django.db import models

class State(models.Model):
    name = models.CharField(max_length=50)
```

serializers.py
[Download serializers.py](#)

```
from rest_framework import serializers
```

IoT Code Generator

Advantages:

- It provides a simple code skeleton
- Do not need to build the code from scratch

Current Disadvantages:

- Only Raspberry Pi is supported
- Limited types of sensors
- Limited types of local and cloud data storage

Summary

- Chef
 - Multi-tier Deployment
 - Hadoop Cluster
 - Storm Cluster
- Puppet
 - Multi-tier Deployment
- NETCONF-YANG
 - Managing Smart Irrigation IoT System
 - Managing Home Intrusion Detection IoT System
- IoT Code Generator