Pivotal GemFire Workshop

Wes Williams

Pivotal Advisory Data Engineer

2017-10-30 9:18 PM

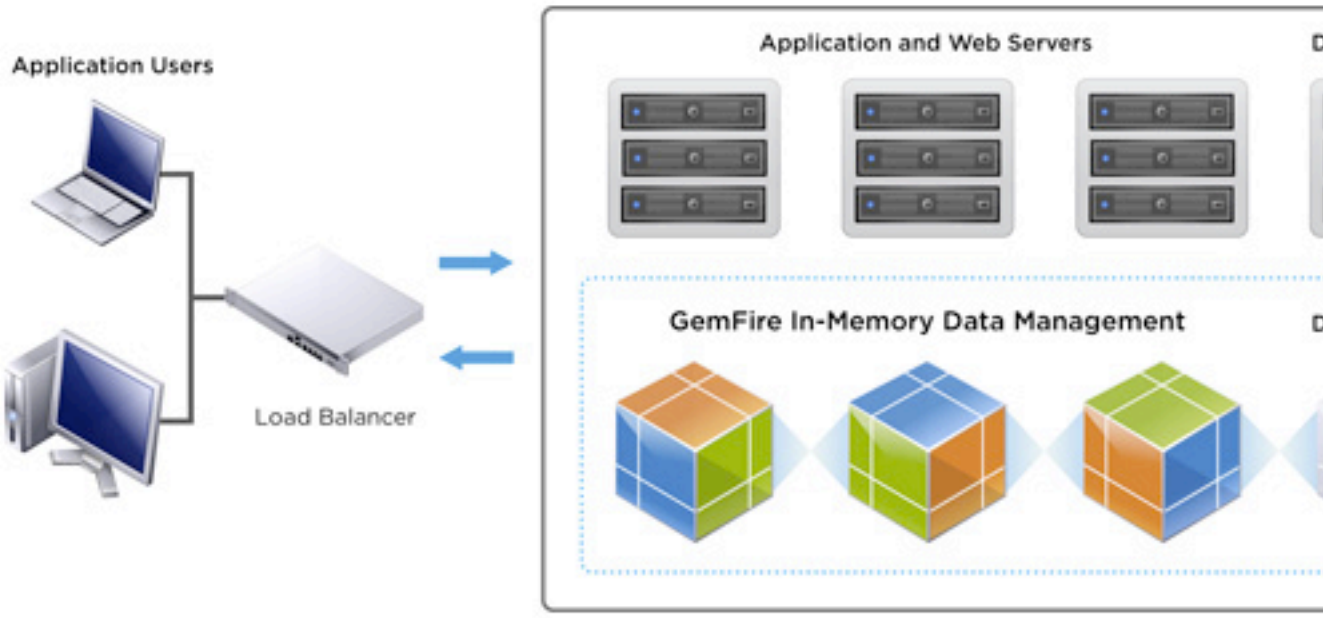
This workshop has a single focus. It keeps learning GemFire as simple as possible while teaching best practices with a real world style right from the beginning.

The goal is for you to quickly acquire the mindset of Thinking in GemFire and keep your confidence high.

It deliberately avoids the approach of introducing samples written with varying styles. This tutorial introduces deployment with a style that you would use in the real world. You will find that everything here works, given a stable environment.

The lessons build on one another. You will quickly discover how enjoyable developing with GemFire can be as your productivity increases as a result of this workshop.

I encourage you to complete the workshop examples before deviating too far so that you acquire the GemFire mindset. You will then be a more confident and independent learner.



Contents

Introduction

Pivotal GemFire is a distributed data management system providing dynamic scalability, high performance and persistence. It blends advanced techniques like replication, partitioning, data-aware routing and continuous querying.

1. Getting Started

1.1 Prerequisites

1.1.1 Install a Java IDE to use as we will run as clients from an IDE. The examples here use STS, Spring Tool Suite, BUT you could easily use Eclipse or other IDE.

1.1.2 Install JDK 1.8 on your file system. GemFire requires the JDK (and not just a JRE) to run gfsh commands)

```
Wes:workshop wwilliams$ java -version
java version "1.8.0_40"
Java(TM) SE Runtime Environment (build 1.8.0_40-b26)
Java HotSpot(TM) 64-Bit Server VM (build 25.20-b23, mixed mode)
```

1.2 Installing GemFire

1.2.1 Download GemFire. This workshop uses GemFire 9.1 from network.pivotal.io

1.2.2 Follow the instructions to install.

1.2.3 Your install should now look like this:

```
wwilliams@Wes:/usr/local/Cellar/gemfire/9.1.0/libexec $ ls -ll
total 0
drwxr-xr-x   3 wwilliams  admin   102 21 Jul 13:25 bin
drwxr-xr-x   5 wwilliams  admin   170 11 Jul 18:09 config
drwxr-xr-x   7 wwilliams  admin   238 11 Jul 18:01 docs
drwxr-xr-x  18 wwilliams  admin   612 11 Jul 18:09 javadoc
drwxr-xr-x  69 wwilliams  admin  2346 11 Jul 18:09 lib
drwxr-xr-x   5 wwilliams  admin   170 11 Jul 18:09 tools
Wes:Pivotal_GemFire_810_b50625_Linux wwilliams$
```

1.2.4 For those of us who run both GemFire 8 and GemFire 9 on the same machine, create a symbolic link to GemFire's **gfsh** in /usr/local/bin and name it **gfsh**:

```
wwilliams@Wes:/usr/local/bin $ ln -s
/some/dir/gemfire/9.1.0/libexec/bin/gfsh gfsh
```

1.3 Setting Up Your Environment

1.3.1 Download files from github to your file system. This workshop uses directory location at: ~/gf91/workshop.

GemFire 9.1.0 Workshop

```
wwilliams@Wes:~/gf91 $ unzip ~/Downloads/workshop.zip -d .
```

or

```
wwilliams@Wes:~/gf91/workshop $ git clone https://github.com/Pivotal-
Data-Engineering/gemfire9-workshop.git
```

1.3.2 Navigate to ~/gf91/workshop

1.3.3 Update your PATH to include GemFire and the Java JDK in **setenvironment.sh**. For example:

```
# PUT THIS INTO YOUR ~/.bash_profile since GemFire will use the
variables from there

export GEMFIRE_HOME=/usr/local/Cellar/gemfire/9.1.0/libexec
export GEMFIRE=$GEMFIRE_HOME

export JAVA_HOME=$(/usr/libexec/java_home -v 1.7)

# Note: the above on OSX resolves to:
#export
JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk1.7.0_71.jdk/Contents/Ho
me

export GF_JAVA=$JAVA_HOME/bin/java
```

```
# this gets the IP of the machine for some gemfire bind addresses
IP=$(ifconfig en0 | awk '/inet /{print substr($2,1)}' | tail -n1)
HOSTNAME=localhost
```

1.3.4 Open 2 separate terminal windows.

1.3.5 Edit **./config/gemfire.properties** to include the correct directory where extracted for the following property

```
deploy-working-dir=/Users/wwilliams/gf91/workshop/deploy
```

Note: You must use Java syntax here. You cannot use OS shortcuts like ~.

1.3.6 Proceed to the "**workshop**" directory and execute the **package.sh** script. It will copy your **gemfire.properties** to future chapters.

```
wwilliams@Wes:~/gf91/workshop $ . package.sh
```

2 Start GemFire Cluster

2.1 Start 1 locator and 2 server nodes using the provided script. Output should be as shown below.

```
 wwilliams@Wes:~/gf91/workshop/chapter2 $. startall.sh
```

The locator starts:

```
gfsh>start locator --name=locator --properties-
file=config/locator.properties --bind-address=localhost --port=10334 --
J=-Xms256m --J=-Xmx256m --
classpath=/usr/local/Cellar/gemfire/9.1.0/libexec/locator-
dependencies.jar
Starting a Geode Locator in
/Users/wwilliams/gf91/workshop/chapter2/locator...
......
Locator in /Users/wwilliams/gf91/workshop/chapter2/locator on
localhost[10334] as locator is currently online.
Process ID: 16944
Uptime: 7 seconds
Geode Version: 9.1.0
Java Version: 1.8.0_40
Log File: /Users/wwilliams/gf91/workshop/chapter2/locator/locator.log
JVM Arguments: -
DgemfirePropertyFile=/Users/wwilliams/gf91/workshop/chapter2/config/loc
ator.properties -Dgemfire.enable-cluster-configuration=true -
Dgemfire.load-cluster-configuration-from-dir=false -Xms256m -Xmx256m -
Dgemfire.launcher.registerSignalHandlers=true -Djava.awt.headless=true
-Dsun.rmi.dgc.server.gcInterval=9223372036854775806
Class-Path: /usr/local/Cellar/gemfire/9.1.0/libexec/lib/geode-core-
9.1.0.jar:/usr/local/Cellar/gemfire/9.1.0/libexec/locator-
dependencies.jar:/usr/local/Cellar/gemfire/9.1.0/libexec/lib/geode-
dependencies.jar

Successfully connected to: JMX Manager [host=192.168.0.10, port=1099]
```

```
Cluster configuration service is up and running.
```

Note: if you are using localhost, your machine network may require you to explicitly map your ip address to localhost in /etc/hosts. Also, if the machine on which you are running has multi-NIC cards, you will need to substitute localhost with an explicit IP address.

Server1:

```
gfsh>start server --name=server1 --locators=localhost[10334] --J=-
Xms512m --J=-Xmx512m --
classpath=/usr/local/Cellar/gemfire/9.1.0/libexec/lib/geode-
dependencies.jar:../../projects/Domain/target/Domain-1.jar --cache-xml-
file=config/cache.xml --properties-file=config/gemfire.properties
Starting a Geode Server in
/Users/wwilliams/gf91/workshop/chapter2/server1...
...
Server in /Users/wwilliams/gf91/workshop/chapter2/server1 on
192.168.0.10[40404] as server1 is currently online.
Process ID: 16947
Uptime: 2 seconds
Geode Version: 9.1.0
Java Version: 1.8.0_40
Log File: /Users/wwilliams/gf91/workshop/chapter2/server1/server1.log
JVM Arguments: -
DgemfirePropertyFile=/Users/wwilliams/gf91/workshop/chapter2/config/gem
fire.properties -Dgemfire.locators=localhost[10334] -Dgemfire.use-
cluster-configuration=true -Dgemfire.cache-xml-
file=/Users/wwilliams/gf91/workshop/chapter2/config/cache.xml -
Dgemfire.start-dev-rest-api=false -Xms512m -Xmx512m -
XX:OnOutOfMemoryError=kill -KILL %p -
Dgemfire.launcher.registerSignalHandlers=true -Djava.awt.headless=true
-Dsun.rmi.dgc.server.gcInterval=9223372036854775806
Class-Path: /usr/local/Cellar/gemfire/9.1.0/libexec/lib/geode-core-
9.1.0.jar:/usr/local/Cellar/gemfire/9.1.0/libexec/lib/geode-
dependencies.jar:../../projects/Domain/target/Domain-
1.jar:/usr/local/Cellar/gemfire/9.1.0/libexec/lib/geode-
dependencies.jar
```

Notice in the log that GemFire automatically placed its jars in the classpath.
If your server does not start, reread the previous note about updating /etc/hosts.

Server2:

```
gfsh>start server --name=server2 --locators=localhost[10334] --J=-
Xms512m --J=-Xmx512m --
classpath=/usr/local/Cellar/gemfire/9.1.0/libexec/lib/geode-
dependencies.jar:../../projects/Domain/target/Domain-1.jar --cache-xml-
file=config/cache.xml --properties-file=config/gemfire.properties --
server-port=40405
Starting a Geode Server in
/Users/wwilliams/gf91/workshop/chapter2/server2...
...
Server in /Users/wwilliams/gf91/workshop/chapter2/server2 on
192.168.0.10[40405] as server2 is currently online.
Process ID: 16950
```

```
Uptime: 2 seconds
Geode Version: 9.1.0
Java Version: 1.8.0_40
Log File: /Users/wwilliams/gf91/workshop/chapter2/server2/server2.log
JVM Arguments: -
DgemfirePropertyFile=/Users/wwilliams/gf91/workshop/chapter2/config/gem
fire.properties -Dgemfire.locators=localhost[10334] -Dgemfire.use-
cluster-configuration=true -Dgemfire.cache-xml-
file=/Users/wwilliams/gf91/workshop/chapter2/config/cache.xml -
Dgemfire.start-dev-rest-api=false -Xms512m -Xmx512m -
XX:OnOutOfMemoryError=kill -KILL %p -
Dgemfire.launcher.registerSignalHandlers=true -Djava.awt.headless=true
-Dsun.rmi.dgc.server.gcInterval=9223372036854775806
Class-Path: /usr/local/Cellar/gemfire/9.1.0/libexec/lib/geode-core-
9.1.0.jar:/usr/local/Cellar/gemfire/9.1.0/libexec/lib/geode-
dependencies.jar:../../projects/Domain/target/Domain-
1.jar:/usr/local/Cellar/gemfire/9.1.0/libexec/lib/geode-
dependencies.jar
```

Note that you do not need to specify the cache.xml or gemfire.properties in subsequent servers that reference the same locator as a previous server that used the same locator. The properties and cache attributes are "inherited" from the locator. Individual attributes can be overridden, as we will do later in this workshop.
This behavior is courtesy of the Cluster Configuration Service.

List members:

```
gfsh>list members
  Name    | Id
--------- | ---------------------------------------------
server2   | localhost(server2:94110)<v5>:1594
server1   | localhost(server1:93944)<v4>:39072
locator1  | localhost(locator1:93816:locator)<v0>:61304
```

List regions:

```
gfsh>list regions
List of regions
---------------
departments
employees
```

2.2 Keep the cluster running and proceed to Chapter 3

3 Add Data Manually

3.1 Change directory to chapter3. Start 1 locator and 2 server nodes using the provided script as shown in Chapter 2.

```
wwilliams@Wes:~/gf8/workshop/chapter3 $. startall.sh
```

3.2 Change to the **data** directory shown below.

```
wwilliams@Wes:~/gf91/workshop/chapter3/data $
```

3.3 Run the script "**populate.sh**" and it will load sample department and employee data into the defined regions.

```
wwilliams@Wes:~/gf91/workshop/chapter3/data $ . populate.sh
    _________________________     __
   / _____/ ______/ ______/ /____/ /
  / /  __/ /___  /_____  / _____  /
 / /__/ / ____/  _____/ / /    / /
/______/_/      /______/_/    /_/    9.1.0

Monitor and Manage Pivotal GemFire
gfsh>
gfsh>connect --locator=localhost[10334]
Connecting to Locator at [host=localhost, port=10334] ..
Connecting to Manager at [host=192.168.0.10, port=1099] ..
Successfully connected to: [host=192.168.0.10, port=1099]

gfsh>
gfsh>run --file=dept-data
1. Executing - put --key=10 --value=('deptno':10,'name':'ACCOUNTING') -
-value-class=io.pivotal.domain.Department --region=departments

Result      : true
Key Class   : java.lang.String
Key         : 10
Value Class : io.pivotal.domain.Department

Value
------
<NULL>
```

3.4 Return to the workshop directory and run "**verify.sh**" as shown below.

```
wwilliams@Wes:~/gf91/workshop/chapter3 $ . verify.sh
    _________________________     __
   / _____/ ______/ ______/ /____/ /
  / /  __/ /___  /_____  / _____  /
 / /__/ / ____/  _____/ / /    / /
/______/_/      /______/_/    /_/    9.1.0

Monitor and Manage GemFire
gfsh>connect --locator=localhost[10334];
Connecting to Locator at [host=localhost, port=10334] ..
Connecting to Manager at [host=localhost, port=1099] ..
Successfully connected to: [host=localhost, port=1099]

gfsh>list members;
  Name   | Id
-------- | ---------------------------------------------
server2  | localhost(server2:94110)<v5>:1594
server1  | localhost(server1:93944)<v4>:39072
```

```
locator1 | localhost(locator1:93816:locator)<v0>:61304

gfsh>list regions;
List of regions
---------------
departments
employees

gfsh>query --query="select * from /departments";

Result     : true
startCount : 0
endCount   : 20
Rows       : 4

deptno | name
------ | ----------
40     | OPERATIONS
10     | ACCOUNTING
30     | SALES
20     | RESEARCH

NEXT_STEP_NAME : END

gfsh>query --query="select * from /employees";

Result     : true
startCount : 0
endCount   : 20
Rows       : 13

empno | deptno |   name   | job
----- | ------ | -------- | ---------
7380  | 40     | BLACK    | CLERK
7373  | 40     | SIENA    | CLERK
7377  | 20     | ADAM     | CLERK
7370  | 10     | APPLES   | MANAGER
7381  | 40     | BROWN    | SALESMAN
7379  | 10     | FRANK    | CLERK
7375  | 30     | ROB      | CLERK
7371  | 10     | WILLIAMS | SALESMAN
7374  | 10     | LUCAS    | SALESMAN
7378  | 20     | SALLY    | MANAGER
7372  | 30     | LUCIA    | PRESIDENT
7376  | 20     | ADRIAN   | CLERK
7369  | 20     | SMITH    | CLERK

NEXT_STEP_NAME : END

....
```

Note that there is now data in the regions

3.5 Shut down the cluster by running the script **stopall.sh** as shown below.

```
wwilliams@Wes:~/gf91/workshop/chapter3 $ . stopall.sh
    _________________________     __
   / _____/ _____/ _____/ /____/ /
  / /  __/ /___  /_____  /_____  /
 / /__/ / ____/  _____/ / /     / /
/_____/_/      /_____/_/     /_/    9.1.0

Monitor and Manage Pivotal GemFire
gfsh>
gfsh>connect --locator=localhost[10334]
Connecting to Locator at [host=localhost, port=10334] ..
Connecting to Manager at [host=192.168.0.10, port=1099] ..
Successfully connected to: [host=192.168.0.10, port=1099]

gfsh>
gfsh>stop server --name=server2
Stopping Cache Server running in
/Users/wwilliams/gf91/workshop/chapter2/server2 on 192.168.0.10[40405]
as server2...
Process ID: 16950
Log File: /Users/wwilliams/gf91/workshop/chapter2/server2/server2.log
....
gfsh>
gfsh>shutdown --include-locators=true –time-out=60
 As a lot of data in memory will be lost, including possibly events in
queues, do you really want to shutdown the entire distributed system?
(Y/n): Y
succeeded in shutting down
gfsh>
gfsh>exit;
Exiting...
```

Note: Shutdown is the preferred way to stop a cluster. I bring down the servers in sequence here to prepare for more advanced cases involving recovery to be covered later.

4 Create A New Region

There are 3 ways to add a new Region in GemFire. We use option 1 below.

1. cache.xml (Requires a restart if adding a new region to a running cluster)
2. **gfsh** (GemFire SHell) in this workshop
3. Programmatically through the GemFire API

Do not use **GemFire** to dynamically create regions at this point in your learning since that involves more knowledge of the Cluster Configuration Service. We will add a new region by bringing down the grid and changing cache.xml.

4.1 Restart the cluster by executing the **./startall.sh** script as already demonstrated in "Start GemFire Cluster"

```
wwilliams@Wes:~/gf8/workshop/chapter2 $ . startall.sh
```

4.2 After the grid starts, navigate to chapter4 and enter the GemFire shell by starting **gfsh**

```
wwilliams@Wes:~/gf91/workshop/chapter4 $ gfsh
    _________________________     __
   / _____/ _____/ _____/ /____/ /
  / /  __/ /___  /_____  / _____  /
 / /__/ / _____/  _____/ / /    / /
/______/_/       /______/_/    /_/    v9.1

Monitor and Manage GemFire
gfsh>connect --locator=localhost[10334]
Connecting to Locator at [host=localhost, port=10334] ..
Connecting to Manager at [host=localhost, port=1099] ..
Successfully connected to: [host=localhost, port=1099]
```

4.3 Connect to the cluster by entering **connect**:

```
gfsh>connect --locator=localhost[10334]
Connecting to Locator at [host=localhost, port=10334] ..
Connecting to Manager at [host=localhost, port=1099] ..
Successfully connected to: [host=localhost, port=1099]
gfsh>
```

4.4 Create a new partitioned region called "**people**" by connecting to gfsh and executing the following command:

```
gfsh>create region --name=people --type=PARTITION
Member  | Status
------- | ------------------------------------
server2 | Region "/people" created on "server2"
server1 | Region "/people" created on "server1"

gfsh>
```

Notice that the region was created on both cache servers.

4.5 Describe the new region that you created as shown below.

```
gfsh>describe region --name=people
..........................................................
Name            : people
Data Policy     : partition
Hosting Members : server2
                  server1

Non-Default Attributes Shared By Hosting Members

 Type  |    Name     | Value
------ | ----------- | ---------
Region | size        | 0
       | data-policy | PARTITION
```

4.6 Exit gfsh

```
gfsh>exit
Exiting...
```

4.7 Keep the cluster alive. Do not stop it. Proceed to Chapter 5.

5 Deploy A New JAR File into the Cluster

At this point you have defined a region but the Person Java class is undefined in the cluster. We are going to dynamically deploy a new JAR file that defines new Person objects without restarting the cluster. The jar contains a domain object that is a simple POJO describing Person objects. This is a powerful GemFire feature.

5.1 The JAR file we are deploying contains a new Domain Model class called Person; shown below.

```
package io.pivotal.domain;

public class Person
{
    private int id;
    private String name;

    public Person()
    {
    }

    public Person(int id, String name) {
        this.id = id;
        this.name = name;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return "io.pivotal.domain.Person{" +
                "id=" + id +
                ", name='" + name + '\'' +
                '}';
    }
}
```

5.2 Connect to the cluster via **gfsh** from the root workshop directory.

```
wwilliams@Wes:~/gf91/workshop $ gfsh
    _________________________     __
```

```
    /  _____/ _____/ _____/ /____/ /
   / /   __/ /___   /_____   _____  /
  / /__/ /  ____/  _____/ / /     / /
 /______/_/       /______/_/     /_/    9.1.0

Monitor and Manage Pivotal GemFire
```

```
gfsh>connect --locator=localhost[10334]
Connecting to Locator at [host=localhost, port=10334] ..
Connecting to Manager at [host=localhost, port=1099] ..
Successfully connected to: [host=localhost, port=1099]
```

5.3 Deploy " **PeopleDomain-1.jar.**" It is included in the lib directory.

```
gfsh>deploy --jar=lib/PeopleDomain-1.jar
Member  |    Deployed JAR    | Deployed JAR Location
------- | ------------------ | ------------------------------------------
------------------------
server1 | PeopleDomain-1.jar |
/Users/wwilliams/gf91/workshop/deploy/vf.gf#PeopleDomain-1.jar#1
server2 | PeopleDomain-1.jar |
/Users/wwilliams/gf91/workshop/deploy/vf.gf#PeopleDomain-1.jar#1
```

Note: the code is in the **projects** directory. If you wish to create the domain.jar yourself, navigate to `projects/PeopleDomain/target/src/main/java/` and execute **jar cvf `PeopleDomain-0.0.1-SNAPSHOT.jar` * or execute a mvn build package in PeopleDomain (mvn package).**

5.4 Deploy "**GemServer-1.jar.**" It is included in the lib directory.

```
gfsh>deploy --jar=lib/GemServer-1.jar
Member  |    Deployed JAR    | Deployed JAR Location
------- | ------------------ | ------------------------------------------
------------------------
server1 | GemServer-1.jar    |
/Users/wwilliams/gf91/workshop/deploy/vf.gf# GemServer-1.jar#1
server2 | GemServer-1.jar    |
/Users/wwilliams/gf91/workshop/deploy/vf.gf# GemServer-1.jar#1
```

Note: this deployment is needed for Chapter7 but we are deploying it here for convenience.

5.5 You can view deployed JAR files by typing **list deployed**

```
gfsh>list deployed
Member  |               JAR               | JAR Location
------- | ------------------------------- | ---------------------------
-----------------------------------------------
server1 | PeopleDomain-1.jar              |
/Users/wwilliams/gf91/workshop/deploy/vf.gf#PeopleDomain-1.jar#1
server2 | PeopleDomain-1.jar              |
/Users/wwilliams/gf91/workshop/deploy/vf.gf#PeopleDomain-1.jar#1
```

5.6 Keep the cluster alive. Do not stop it. Proceed to Chapter 6.

6 Bulk Insert Domain Model Objects into GemFire

In the example below we will load Person.java POJO into the "/people" region using the region.putAll method which should be used when bulk loading into the cluster.

6.1 Open the java class "**BulkInsertPerson.java**" in the package "**io.pivotal.app.insert**". This will insert Person objects into our cluster as a collection rather than inserting objects one-by-one.

The java code here is as follows.

BulkInsertPerson.java

```
package io.pivotal.app.insert;

import io.pivotal.domain.Person;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Random;
import java.util.Scanner;
import java.util.logging.Level;
import java.util.logging.Logger;

import com.gemstone.gemfire.cache.Region;
import com.gemstone.gemfire.cache.client.ClientCache;
import com.gemstone.gemfire.cache.client.ClientCacheFactory;

public class BulkInsertPerson
{

    private Logger logger =
Logger.getLogger(this.getClass().getSimpleName());
    private ClientCache cache = null;
    private final int BATCH_SIZE = 10000;
    private final int SAMPLE_SIZE = 100000;

    Random r = new Random(System.currentTimeMillis());


    private BulkInsertPerson ()
    {
        ClientCacheFactory ccf = new ClientCacheFactory();
        ccf.set("cache-xml-file", "config/query-client.xml");
        cache = ccf.create();
    }

    public void run()
    {
        Region<String,Person> peopleRegion = cache.getRegion("people");
        Map<String, Person> buffer = new HashMap<String,
Person>();
        Scanner scanner = new java.util.Scanner(System.in);

        long start = System.currentTimeMillis();
```

```
        for (int i = 1; i <= SAMPLE_SIZE; i++)
        {
            int firstNameIx = r.nextInt(firstNames.size());
            int lastNameIx = r.nextInt(lastNames.size());
            // place person into temp buffer
            buffer.put(String.valueOf(i), new Person(i,
String.format("%s %s", firstNames.get(firstNameIx),
lastNames.get(lastNameIx))));

            if ((i % BATCH_SIZE) == 0)
            {
                // ready to insert a batch into our region
                peopleRegion.putAll(buffer);
                buffer.clear();

                System.out.println("Inserted " + BATCH_SIZE + "
records");
                System.out.println("Press any key to insert more...");
                scanner.nextLine();
            }
        }

        // there may be existing records to flush so this takes care of
it
        if (!buffer.isEmpty())
        {
            peopleRegion.putAll(buffer);
            buffer.clear();
        }

        long end = System.currentTimeMillis() - start;

        float elapsedTimeSec = end/1000F;

        logger.log (Level.INFO, String.format("Elapsed time in seconds
%f", elapsedTimeSec));
    }

    public static void main(String[] args)
    {
        BulkInsertPerson test = new BulkInsertPerson();
        test.run();
    }
```

6.2 Run the java class and verify output as shown below. The client app uses **config/query-client.xml** to locate and connect to the cluster.

Linux:

```
$ java -cp $GEMFIRE/lib/geode-dependencies.jar:lib/ClientApp-
1.jar:lib/PeopleDomain-1.jar io.pivotal.app.insert.BulkInsertPerson
```

Note: if you get an error that the client cannot find the locator then update localhost to the machine's IP in config/query-client.xml

Windows:

```
java -cp lib\ClientApp-1.jar;lib\PeopleDomain-
1.jar;%GEMFIRE%\lib\geode-dependencies.jar
io.pivotal.app.insert.BulkInsertPerson
```

```
Inserted 10000 records
Inserted 10000 records
Inserted 10000 records
Inserted 10000 records
Inserted 10000 records
Inserted 10000 records
Inserted 10000 records
Inserted 10000 records
Inserted 10000 records
Inserted 10000 records
Jan 19, 2015 8:37:57 PM io.pivotal.app.insert.BulkInsertPerson run
[info 2015/01/19 20:37:59.706 EST  <main> tid=0x1] (tid=1 msgId=0)
Elapsed time in seconds 1.993000
```

6.3 Get a count of the number of objects in the region, which should be 100,000 entries.

```
gfsh>query --query="select * from /people limit 10";

Result     : true
startCount : 0
endCount   : 20
Rows       : 10

      name       | id
---------------- | -----
Hyatt Morales    | 75963
Yetta Craig      | 10209
Baxter Yates     | 37511
Igor Tyler       | 74535
Mikayla Faulkner | 37702
Allistair Durham | 41179
Macaulay Nichols | 77226
Steel Duffy      | 16848
Jordan Hull      | 63568
```

```
Brielle Crosby   | 26424

NEXT_STEP_NAME : END

gfsh>query --query="select count(*) from /people";

Result     : true
startCount : 0
endCount   : 20
Rows       : 1

Result
------
100000

NEXT_STEP_NAME : END
```

6.4 Another way to get the region count is to tell gfsh to describe the region

```
gfsh>describe region --name=people
..........................................................
Name            : people
Data Policy     : partition
Hosting Members : server2
                  server1

Non-Default Attributes Shared By Hosting Members

 Type  |    Name     | Value
------ | ----------- | ---------
Region | size        | 100000
       | data-policy | PARTITION
```

6.5 Shut down the cluster since we will make configuration changes in the next chapter.

```
wwilliams@Wes:~/gf91/workshop/chapter6 $ . stopall.sh
```

7 Using the REST API

The REST API is very useful to inspect and update the contents of the cache interactively.

7.1 Verify that the locator is the jmx manager by inspecting the locator's **locator.properties** file. This is found in the chapter7/config directory.

```
jmx-manager=true
jmx-manager-start=true
```

7.2 Enable the REST API by starting ***one*** server with the following properties in bold. Change the startall.sh script.

```
gfsh>start server --name=server1 --J=-Dgemfire.start-dev-rest-api=true
--J=-Dgemfire.http-service-bind-address=xxx.xxx.x.xxx --J=-
Dgemfire.http-service-port=7075
```

Note: for multi NIC systems: you may need to add http-service-bind-address=xxx.xx.xxx.xxx in IP format if your multi NIC network requires it. If you don't have a multi NIC system then you can just use the hostname for the bind address

7.3 Start the cluster:

```
wwilliams@Wes:~/gf91/workshop/chapter7 $ . startall.sh
```

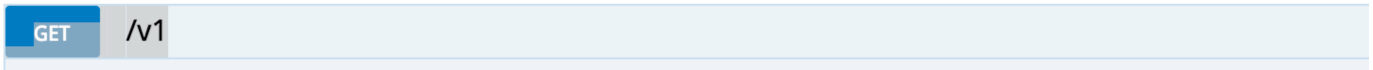
7.4 Populate data into the grid by running the script **data/populate.sh**

```
wwilliams@Wes:~/gf91/workshop/chapter7 $ cd data
wwilliams@Wes:~/gf91/workshop/chapter7/data $ . populate.sh
```

7.5 Access the browser Swagger UI by entering the following URL:
http://xxx.xxx.x.xxx:7075/gemfire-api/docs/index.html

Note that your browser may require you to enter localhost in IP format. Make sure that the port number in the URL matches the port in the start-server http-service-port when you started the cache server.

7.6 List all Regions. Select "**Region**" and then "**list all resources (Regions)**" in the right margin



7.7 Select the "Try it Out" button

Response Body


```
{
  "regions": [
    {
      "name": "employees",
      "type": "PARTITION",
      "key-constraint": "java.lang.String",
      "value-constraint": "io.pivotal.domain.Employee"
    },
    {
      "name": "departments",
      "type": "PARTITION",
      "key-constraint": "java.lang.String",
      "value-constraint": "io.pivotal.domain.Department"
    },
    {
      "name": "people",
      "type": "PARTITION",
      "key-constraint": null,
      "value-constraint": null
    }
```

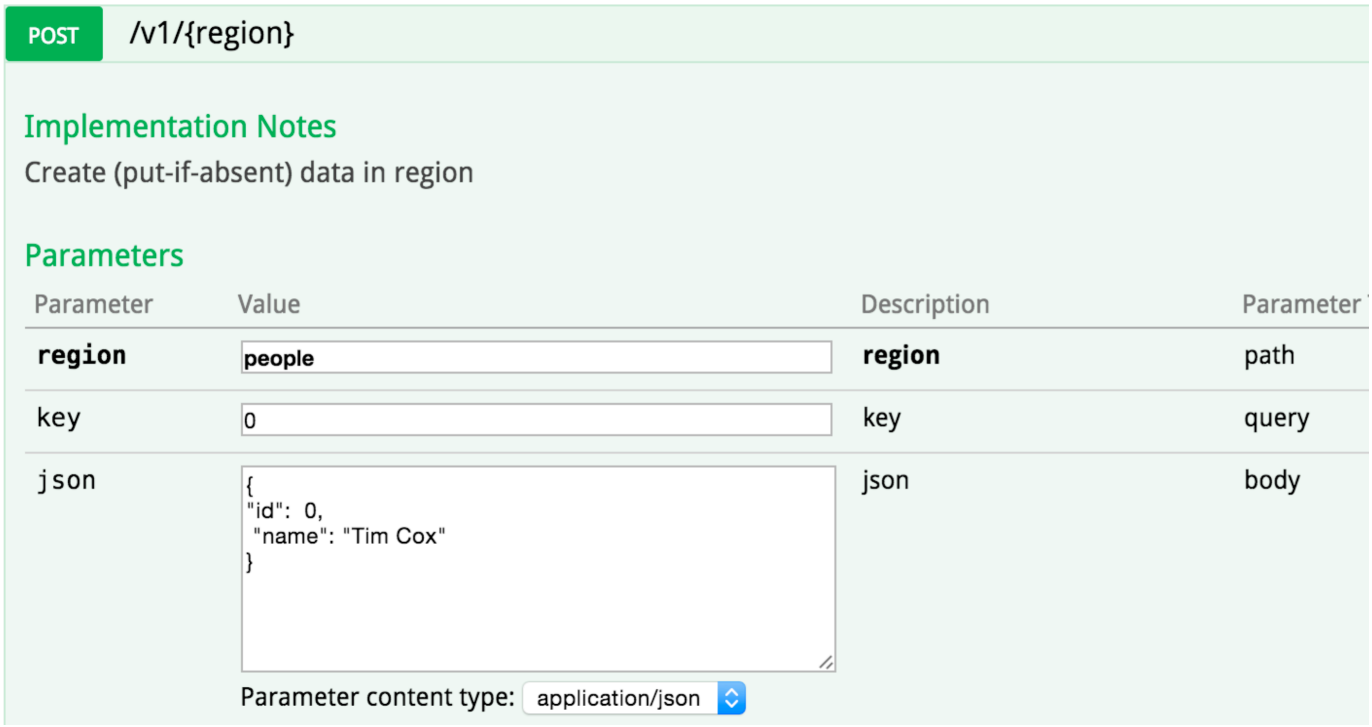

7.8 List all of the entries in the **departments** region by selecting "**read all data for region**" in the right margin.

Response Body


```
{
  "departments": [
    {
      "deptno": 40,
      "name": "OPERATIONS"
    },
    {
      "deptno": 30,
      "name": "SALES"
    },
    {
      "deptno": 20,
      "name": "RESEARCH"
    },
    {
      "deptno": 10,
      "name": "ACCOUNTING"
    }
  ]
}
```


7.9 Add an entry to the **people** region by expanding the **POST /v1/{region}** endpoint, or “Create entry” link

7.10 Post a Person to the people region with a key of 0:



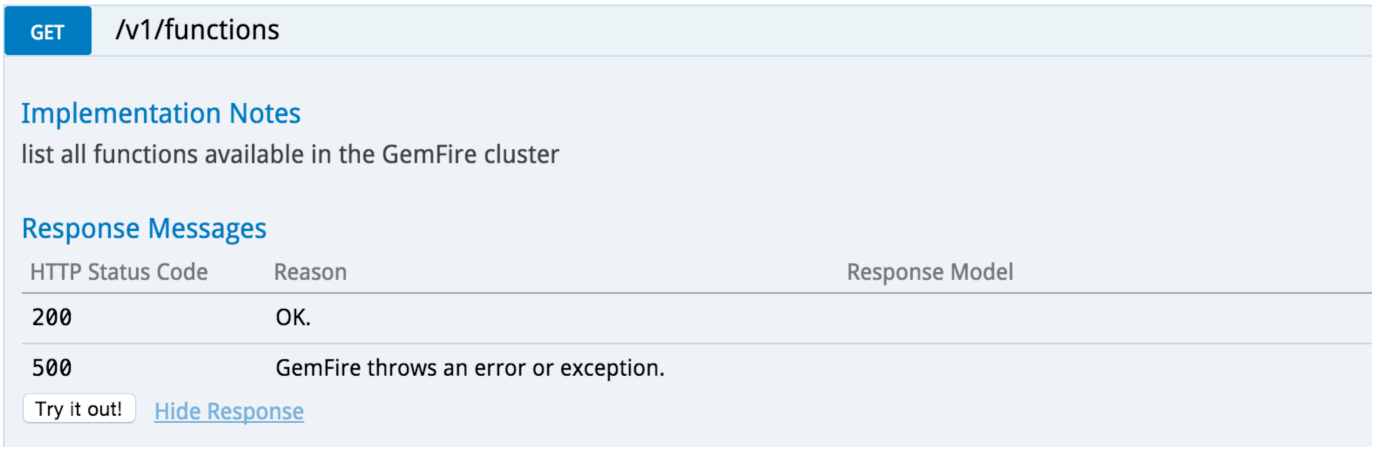
Note the Response Headers return the response from the server.

7.11 List all entries for the orders region by selecting **GET/v1/{region}.** The Response Body returns:

Response Body

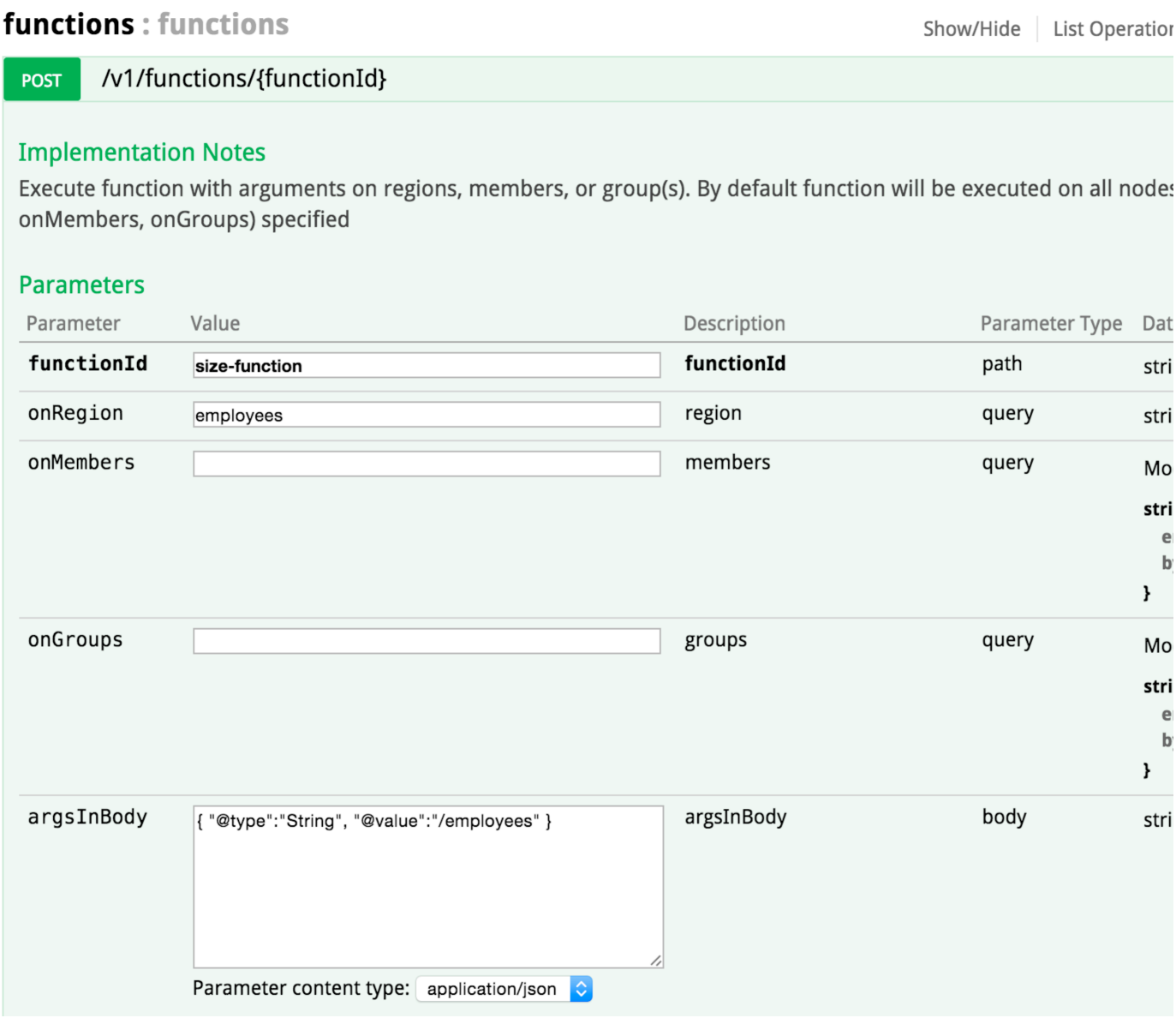

```
{
  "people": [
    {
      "id": 0,
      "name": "Tim Cox"
    }
  ]
}
```


7.12 Select the “Functions” feature and select “list all functions”



And the response is one function, the size-function

7.13 Execute the Size-Function on the **employees** region by entering “employees” as an argument of type “String in the **argsInBody** field.



The REST API returns the count of 13:

7.14 Keep the cluster running for Chapter 8.

8 Querying

In the following examples we will query the "**/employees**" region using a java client as well as **GemFire**.

8.1 Querying data from a Java Client using OQL

8.1.2 Open the java class **QueryEmployees.java** in the package **io.pivotal.app.query**.

QueryEmployees.java

```
package io.pivotal.app.query;

import com.gemstone.gemfire.cache.Region;
import com.gemstone.gemfire.cache.client.ClientCache;
import com.gemstone.gemfire.cache.client.ClientCacheFactory;
import com.gemstone.gemfire.cache.query.Query;
import com.gemstone.gemfire.cache.query.QueryService;
import com.gemstone.gemfire.cache.query.SelectResults;
import io.pivotal.domain.Employee;

import java.util.Collection;
import java.util.Iterator;
import java.util.logging.Level;
import java.util.logging.Logger;

public class QueryEmployees
{
    private ClientCache cache = null;
    private Logger logger =
Logger.getLogger(this.getClass().getSimpleName());

    public QueryEmployees()
    {
        ClientCacheFactory ccf = new ClientCacheFactory();
        ccf.set("cache-xml-file", "config/query-client.xml");
        cache = ccf.create();
    }

    public void run() throws Exception
    {
        Region<Integer,Employee> employees =
cache.getRegion("employees");

        QueryService queryService = cache.getQueryService();

        Query query = queryService.newQuery("SELECT * FROM /employees
where deptno = 10");
        logger.log (Level.INFO, "\nExecuting query:\n\t" +
query.getQueryString());

        Object result = query.execute();

        Collection<?> collection = ((SelectResults<?>)result).asList();
        logger.log (Level.INFO, String.format("%s Employees in
department 10", collection.size()));

        Iterator<?> iter = collection.iterator();

        while (iter.hasNext())
        {
            Employee emp = (Employee) iter.next();
            System.out.println(emp.toString());
        }
    }

    /**
     * @param args
     */
```

```
    public static void main(String[] args)
    {
        QueryEmployees test = new QueryEmployees();

        try
        {
            test.run();
        }
        catch (Exception e)
        {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

8.1.3 Run the java class and verify output:

Linux:

```
 $ java -classpath lib/Domain-1.jar:lib/ClientApp-
1.jar:$GEMFIRE/lib/geode-dependencies.jar
io.pivotal.app.query.QueryEmployees
```

Note: if you get an error that the client cannot find the locator then update localhost to the machine's IP in config/query-client.xml

Windows:

```
 > java -cp lib\ClientApp-1.jar;lib\Domain-1.jar;%GEMFIRE%\lib\geode-
dependencies.jar io.pivotal.app.query.QueryEmployees
```

```
Jan 19, 2015 8:19:05 PM io.pivotal.app.query.QueryEmployees run
INFO:
Executing query:
    SELECT * FROM /employees where deptno = 10
Jan 19, 2015 8:19:05 PM io.pivotal.app.query.QueryEmployees run
INFO: 4 Employees in department 10
Employee [empno=7374, name=LUCAS, job=SALESMAN, deptno=10]
Employee [empno=7371, name=WILLIAMS, job=SALESMAN, deptno=10]
Employee [empno=7370, name=APPLES, job=MANAGER, deptno=10]
Employee [empno=7379, name=FRANK, job=CLERK, deptno=10]
```

8.2 Querying all region data

In this example we get all entries by querying all Employees on the server using the keys. This is not recommended for very large data sets, where you should use OQL to query the region rather then as shown below. This example just shows a fast way to get all keys from server and iterate the set of region data.

8.2.1 Open the java class "**QueryAllEmployees.java**" in the package "io.pivotal.app.query".

The java code is as follows.

QueryAllEmployees.java

```
package io.pivotal.app.query;

import com.gemstone.gemfire.cache.Region;
import com.gemstone.gemfire.cache.client.ClientCache;
import com.gemstone.gemfire.cache.client.ClientCacheFactory;
import io.pivotal.app.domain.Employee;

import java.util.Map;
import java.util.Set;
import java.util.logging.Level;
import java.util.logging.Logger;

public class QueryAllEmployees
{
    private ClientCache cache = null;
    private Logger logger =
Logger.getLogger(this.getClass().getSimpleName());

    public QueryAllEmployees()
    {
        ClientCacheFactory ccf = new ClientCacheFactory();
        ccf.set("cache-xml-file", "config/query-client.xml");
        cache = ccf.create();
    }

    public void run() throws Exception
    {
        Region<String,Employee> employees =
cache.getRegion("employees");

        Set<String> keysOnServer = employees.keySetOnServer();

        logger.log (Level.INFO, "\nEmployees size:\n\t" +
keysOnServer.size());

        Map<String, Employee> countriesMap =
employees.getAll(keysOnServer);
        Set<Map.Entry<String, Employee>> entries =
countriesMap.entrySet();

        for (Map.Entry<String,Employee> entry: entries)
        {
            System.out.println(
                    String.format("Key %s, Value %s", entry.getKey(),
entry.getValue()));
        }

    }

    /**
     * @param args
     */
    public static void main(String[] args)
    {
        QueryAllEmployees test = new QueryAllEmployees();

        try
        {
```

```
            test.run();
        }
        catch (Exception e)
        {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }

    }

}
```

8.2.2 Run the java class and verify output as follows.

Linux:

```
$ java -cp lib/Domain-1.jar:lib/ClientApp-1.jar:$GEMFIRE/lib/geode-
dependencies.jar io.pivotal.app.query.QueryAllEmployees
```

Windows:

```
> java -cp lib\ClientApp-1.jar;lib\PeopleDomain-
1.jar;%GEMFIRE%\lib\geode-dependencies.jar
io.pivotal.app.query.QueryAllEmployees
```

```
Jan 19, 2015 8:44:39 PM io.pivotal.app.query.QueryAllEmployees run
INFO:
Employees size: 13
Key 7381, Value Employee [empno=7381, name=BROWN, job=SALESMAN,
deptno=40]
Key 7380, Value Employee [empno=7380, name=BLACK, job=CLERK, deptno=40]
Key 7370, Value Employee [empno=7370, name=APPLES, job=MANAGER,
deptno=10]
Key 7372, Value Employee [empno=7372, name=LUCIA, job=PRESIDENT,
deptno=30]
Key 7371, Value Employee [empno=7371, name=WILLIAMS, job=SALESMAN,
deptno=10]
Key 7374, Value Employee [empno=7374, name=LUCAS, job=SALESMAN,
deptno=10]
Key 7373, Value Employee [empno=7373, name=SIENA, job=CLERK, deptno=40]
Key 7376, Value Employee [empno=7376, name=ADRIAN, job=CLERK,
deptno=20]
Key 7375, Value Employee [empno=7375, name=ROB, job=CLERK, deptno=30]
Key 7369, Value Employee [empno=7369, name=SMITH, job=CLERK, deptno=20]
Key 7378, Value Employee [empno=7378, name=SALLY, job=MANAGER,
deptno=20]
Key 7377, Value Employee [empno=7377, name=ADAM, job=CLERK, deptno=20]
Key 7379, Value Employee [empno=7379, name=FRANK, job=CLERK, deptno=10]
```

8.3 Querying a Region by key

The fastest way to retrieve data is by using it's key. In this example we do that.

8.3.1 Open the java class "**QueryEmployeeByKey.java**" in the package "io.pivotal.app.query".

The java code is as follows.

```
package io.pivotal.app.query;

import com.gemstone.gemfire.cache.Region;
import com.gemstone.gemfire.cache.client.ClientCache;
import com.gemstone.gemfire.cache.client.ClientCacheFactory;
import io.pivotal.app.domain.Employee;

import java.util.logging.Level;
import java.util.logging.Logger;

public class QueryEmployeeByKey
{
    private ClientCache cache = null;
    private Logger logger =
Logger.getLogger(this.getClass().getSimpleName());

    public QueryEmployeeByKey()
    {
        ClientCacheFactory ccf = new ClientCacheFactory();
        ccf.set("cache-xml-file", "config/query-client.xml");
        cache = ccf.create();
    }

    public void run() throws Exception
    {
        Region<Integer,Employee> employees =
cache.getRegion("employees");

        String empKey = "7370";
        Employee emp = employees.get(empKey);

        logger.log (Level.INFO, emp.toString());

    }

    /**
     * @param args
     */
    public static void main(String[] args)
    {
        QueryEmployeeByKey test = new QueryEmployeeByKey();

        try
        {
            test.run();
        }
        catch (Exception e)
        {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

8.3.2 Run the java class and verify output as follows.

Linux:

```
$ java -classpath lib/Domain-1.jar:lib/ClientApp-
1.jar:$GEMFIRE/lib/geode-dependencies.jar
io.pivotal.app.query.QueryEmployeeByKey
```

Windows:

```
> java -cp lib\ClientApp-1.jar;lib\PeopleDomain-
1.jar;%GEMFIRE%\lib\geode-dependencies.jar
io.pivotal.app.query.QueryEmployeesByKey
```

```
Jan 19, 2015 8:49:05 PM io.pivotal.app.query.QueryEmployeeByKey run
INFO: Employee [empno=7370, name=APPLES, job=MANAGER, deptno=10]
```

8.4 Query with gfsh

8.4.1 Connect to the cluster and query it as shown below.

```
[Mon Oct 21 20:12:43 wwilliams@:~/gf91/workshop $ gfsh
    _____________________________     __
   / _____/ _______/ _______/ /____/ /
  / /  __/ /___   /_____   / _____  /
 / /__/ / ____/   _____/ / /     / /
/______/_/       /_______/_/     /_/    v9.1

Monitor and Manage GemFire
gfsh>connect
Connecting to Locator at [host=localhost, port=10334] ..
Connecting to Manager at [host=wes.local, port=1099] ..
 Successfully connected to: [host=wes.local, port=1099]

 gfsh>query --query="select * from /employees where deptno = 10";

 Result     : true
 startCount : 0
 endCount   : 20
 Rows       : 4

 empno | deptno |   name   | job
 ----- | ------ | -------- | --------
 7370  | 10     | APPLES   | MANAGER
 7371  | 10     | WILLIAMS | SALESMAN
 7374  | 10     | LUCAS    | SALESMAN
 7379  | 10     | FRANK    | CLERK

 NEXT_STEP_NAME : END
```

8.4.2 Do a **join** between the employee partitioned region and the departments replicated region

```
gfsh>query --query="select e as employee, d.name as department from
/employees e, /departments d where e.deptno = d.deptno"
```

```
                              employee                                 |
department
---------------------------------------------------------------------- | -------
---
```

```
{"empno":7374,"deptno":10,"name":"LUCAS","job":"SALESMAN"}     |
ACCOUNTING
{"empno":7375,"deptno":30,"name":"ROB","job":"CLERK"}          | SALES
{"empno":7370,"deptno":10,"name":"APPLES","job":"MANAGER"}     |
ACCOUNTING
{"empno":7381,"deptno":40,"name":"BROWN","job":"SALESMAN"}     |
OPERATIONS
{"empno":7372,"deptno":30,"name":"LUCIA","job":"PRESIDENT"}    | SALES
{"empno":7371,"deptno":10,"name":"WILLIAMS","job":"SALESMAN"} |
ACCOUNTING
{"empno":7377,"deptno":20,"name":"ADAM","job":"CLERK"}         |
RESEARCH
{"empno":7369,"deptno":20,"name":"SMITH","job":"CLERK"}        |
RESEARCH
{"empno":7378,"deptno":20,"name":"SALLY","job":"MANAGER"}      |
RESEARCH
{"empno":7376,"deptno":20,"name":"ADRIAN","job":"CLERK"}       |
RESEARCH
{"empno":7380,"deptno":40,"name":"BLACK","job":"CLERK"}        |
OPERATIONS
{"empno":7379,"deptno":10,"name":"FRANK","job":"CLERK"}        |
ACCOUNTING
{"empno":7373,"deptno":40,"name":"SIENA","job":"CLERK"}        |
OPERATIONS
```

8.4.3 Keep the cluster running for the next lesson.

9 Start Pulse (Monitoring Web Console)

GemFire Pulse is a Web Application that provides a graphical dashboard for monitoring vital, real-time health and performance of GemFire clusters, members, and regions.

Use Pulse to examine total memory, CPU, and disk space used by members, uptime statistics, client connections, WAN connections, and critical notifications. Pulse communicates with a GemFire JMX manager to provide a complete view of your GemFire deployment. You can drill down from a high-level cluster view to examine individual members and even regions within a member, to filter the type of information and level of detail.

By default, GemFire Pulse runs in a Tomcat server container that is embedded in a GemFire JMX manager node. You can optionally deploy Pulse to a Web application server of your choice, so that the tool runs independently of your GemFire clusters. Hosting Pulse on an application server also enables you to use SSL for accessing the application.

9.1 Connect to the cluster as shown below.

```
gfsh>connect --locator=localhost[10334]
Connecting to Locator at [host=localhost, port=10334] ..
Connecting to Manager at [host=wes.local, port=1099] ..
Successfully connected to: [host=wes.local, port=1099]
```

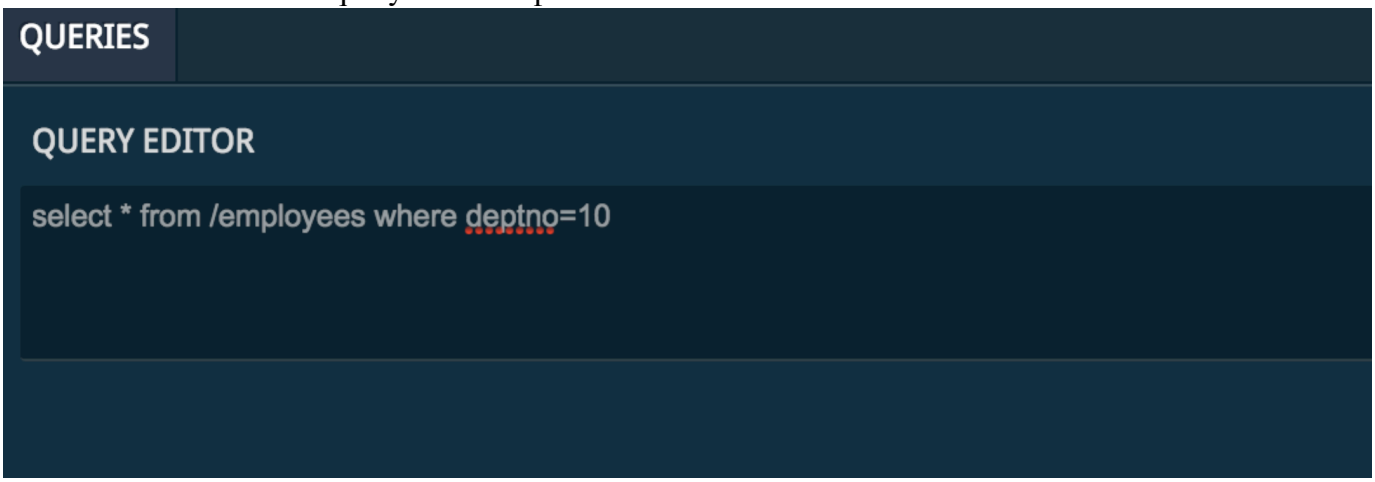
9.2 Start pulse and it should invoke a browser for you, if not then use the URL provided below. Note that the port number is 7070. You can override this port in the locator's gemfire.properties file.

```
gfsh>start pulse;
Launched Geode Pulse

Pulse URL : http://localhost:8083/pulse/
```

9.3 Connect using a username = **admin** and password = **admin**. Pulse will display its overview page.

9.4 Click on Data Browser to view all regions.

9.5 Execute the same query from Chapter 8 here in the Data Browser:



And the answer:

9.6 Keep the cluster running

10 Start JConsole (Graphical Monitoring Tool for Java)

You can browse all the GemFire MBeans in your distributed system by using JConsole. To view GemFire MBeans through JConsole, perform the following steps:

10.1 Connect to the cluster if you are not already connected

```
gfsh>connect --locator=localhost[10334]
```

10.2 Start JConsole

```
gfsh>start jconsole;
Running JDK JConsole
```

Note: You may be prompted to connect using insecure connection, that is fine as we have not enabled security.

10.3 Keep the cluster running

11 Create an Index

The GemFire `QueryService` API provides methods to create, list and remove the index. You can also use **`gfsh`** command-line interface to create, list and remove indexes, and use cache.xml to create an index. In the example below we use **gfsh** to create an index on the "**/employees**" region.

11.1 Connect to the cluster

```
gfsh>connect --locator=localhost[10334]
```

11.2 List what indexes we have in place right now, there should be none

```
gfsh>list indexes;
No Indexes Found
```

11.3 Create an index as follows then list the indexes

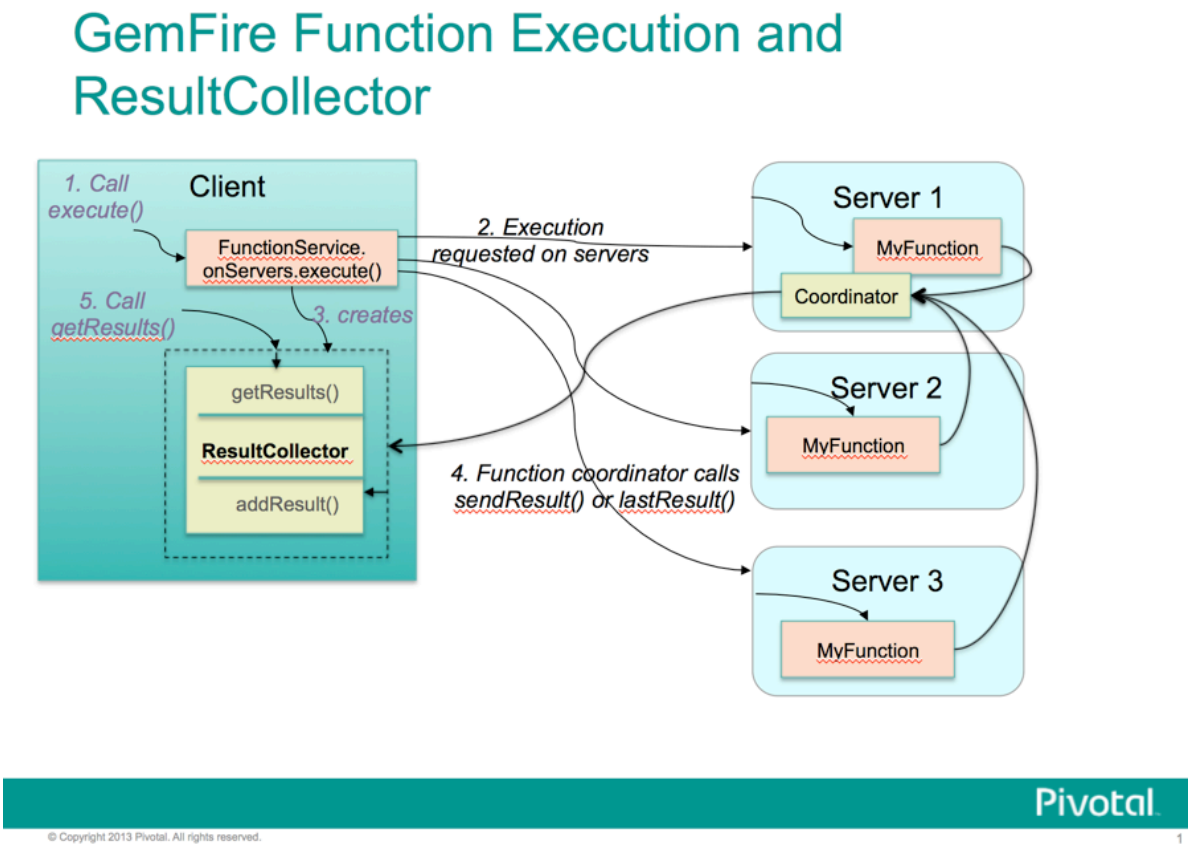
```
gfsh>create index --name=employees_deptno_idx --expression=deptno --
region=/employees
Index successfully created with following details
Name       : employees_deptno_idx
Expression : deptno
RegionPath : /employees
Members which contain the index
1. localhost(server1:4020)<v1>:43264
2. localhost(server2:4021)<v2>:19156

gfsh>list indexes
Member Name |             Member ID             | Region Path |
Name         | Type  | Indexed Expression | From Clause
------------ | --------------------------------- | ------------ | -------
-------------- | ----- | ------------------ | ------------
server1      | localhost(server1:4020)<v1>:43264 | /employees   |
employees_deptno_idx | RANGE | deptno              | /employees
server2      | localhost(server2:4021)<v2>:19156 | /employees   |
employees_deptno_idx | RANGE | deptno              | /employees
```

Note: Indexes exist in memory and need to be considered when sizing the cluster

11.4 Keep the database running for Chapter 12.

12 Functions

Using the GemFire function execution service, you can execute application functions on a single server member, in parallel on a subset of server members, or in parallel on all server members of a distributed system.



12.1 Creating the function

12.1.1 Open the java class "SizeFunction.java" in the package "io.pivotal.app.functions".

The java code is as follows.

```
package io.pivotal.app.functions;

import java.util.Properties;

import org.apache.geode.cache.CacheFactory;
import org.apache.geode.cache.GemFireCache;
import org.apache.geode.cache.Region;
import org.apache.geode.cache.execute.Function;
import org.apache.geode.cache.execute.FunctionContext;
```

```
public class SizeFunction extends Function implements Declarable
{
    private static final long serialVersionUID = 1L;

    public static final String ID = "size-function";

    private GemFireCache cache;

    public SizeFunction()
    {
        this.cache = CacheFactory.getAnyInstance();
    }

    public void execute(FunctionContext context)
    {
        String regionName = (String) context.getArguments();
        Region region = this.cache.getRegion(regionName);
        System.out.println("Getting size of region " +
region.getFullPath());
        context.getResultSender().lastResult(region.size());
    }

    public String getId()
    {
        return ID;
    }

    public void init(Properties properties) {
    }
}
```

12.2 Deploying to the GemFire Distributed system

We are going to dynamically deploy this function to the cluster using **gfsh**. **gfsh** will automatically register functions in the system from what it finds in the JAR files we upload.

12.2.1 Connect to the cluster after re-launching GemFire in the Chapter12 directory

```
gfsh>connect --locator=localhost[10334]
```

12.2.2 Deploy the "**GemServer-1.jar**" file in the lib directory for chapter 12. You will need to use the directory you have installed this workshop in.

```
gfsh>deploy --jar=lib/GemServer-1.jar
Member  |  Deployed JAR   | Deployed JAR Location
------- | --------------- | ----------------------------------------
------------------
server1 | GemServer-1.jar |
/Users/wwilliams/gf91/workshop/deploy/vf.gf#GemServer-1.jar#1
server2 | GemServer-1.jar |
/Users/wwilliams/gf91/workshop/deploy/vf.gf#GemServer-1.jar#1
```

12.2.3 You can view deployed functions as follows

```
gfsh>list functions
Member  | Function
------- | -------------
server1 | size-function
server2 | size-function
```

12.3 Executing the function from a client

12.3.1 Open the java class "ExecuteSizeFunction.java" in the package "io.pivotal.app.functions".

The java code here is as follows.

```
package io.pivotal.app.functions;

import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;

import com.gemstone.gemfire.cache.client.ClientCache;
import com.gemstone.gemfire.cache.client.ClientCacheFactory;
import com.gemstone.gemfire.cache.execute.Execution;
import com.gemstone.gemfire.cache.execute.FunctionService;
import com.gemstone.gemfire.cache.execute.ResultCollector;

public class ExecuteSizeFunction
{
    private Logger logger =
Logger.getLogger(this.getClass().getSimpleName());
    private ClientCache cache = null;

    public ExecuteSizeFunction()
    {
        ClientCacheFactory ccf = new ClientCacheFactory();
        ccf.set("cache-xml-file", "config/query-client.xml");
        cache = ccf.create();
    }

    public void run(String regionName)
    {
        Execution execution =
FunctionService.onServer(this.cache).withArgs("/" + regionName);

        ResultCollector collector = execution.execute(SizeFunction.ID);

        logger.log (Level.INFO,
                String.format("Region %s contains %s entries",
                        "/" + regionName, ((List)
collector.getResult()).get(0)));
    }

    public static void main(String[] args)
    {
        ExecuteSizeFunction test = new ExecuteSizeFunction();
        test.run("employees");
        test.run("departments");
    }
}
```

12.3.2 Run the Java Class and verify output as follows.

Linux:

```
$ java -classpath lib/ClientApp-1.jar:$GEMFIRE/lib/geode-
dependencies.jar io.pivotal.app.function.ExecuteSizeFunction
```

Windows:

```
> java -cp lib\ClientApp-1.jar;%GEMFIRE%\lib\geode-dependencies.jar
io.pivotal.app.function.ExecuteSizeFunction
```

Note: GemServer-1.jar would normally need to be in the classpath but since you deployed the jar into the cluster, GemFire already knows about it.

```
Mar 14, 2015 5:39:15 PM io.pivotal.app.function.ExecuteSizeFunction run
INFO: Region /employees contains 13 entries
Mar 14, 2015 5:39:15 PM io.pivotal.app.function.ExecuteSizeFunction run
INFO: Region /departments contains 4 entries
```

12.3.3 Keep the database running for Chapter 13.

Further reading: Function Best Practices

13 Verify Data Member Locations

Each cache server in the GemFire cluster knows the location of the primary host of each region entry. We can list those entry locations as we will now do.

13.1 Open the java class "**VerifyDataLocations.java**" in the package "io.pivotal.app" in the Client App.

The java code is as follows.

```
package io.pivotal.app.;

import java.util.Map;
import java.util.Set;

import com.gemstone.gemfire.cache.Cache;
import com.gemstone.gemfire.cache.CacheFactory;
import com.gemstone.gemfire.cache.Region;
import com.gemstone.gemfire.cache.partition.PartitionRegionHelper;
import com.gemstone.gemfire.distributed.DistributedMember;
import io.pivotal.app.domain.Employee;

public class VerifyDataLocations
{
    private Cache cache = null;

    public VerifyDataLocations()
    {
        CacheFactory cf = new CacheFactory();
        cf.set("cache-xml-file", "config/datalocations-cache-no-
storage.xml");
        cf.set("locators", "localhost[10334]");
        cache = cf.create();
    }

    public void run() throws InterruptedException
    {
        Region<String,Employee> exampleRegion =
cache.getRegion("employees");
        System.out.println("Employees region size = " +
exampleRegion.size());

        Set<Map.Entry<String, Employee>> entries =
exampleRegion.entrySet();

        for (Map.Entry entry: entries)
        {

            DistributedMember member =
                    PartitionRegionHelper.getPrimaryMemberForKey(exampl
eRegion, (String) entry.getKey());
            System.out.println
                    (String.format("\"Primary Member [Host=%s, Id=%s -
Key=%s, Value=%s]",
                            member.getHost(), member.getId(),
entry.getKey(), entry.getValue()));
        }

        cache.close();
    }

    /**
     * @param args
     * @throws InterruptedException
     */
    public static void main(String[] args) throws InterruptedException
    {
        // TODO Auto-generated method stub
```

```
        VerifyDataLocations test = new VerifyDataLocations();
        test.run();
        System.exit(1);
    }
}
```

13.2 In order to view where data exists on each server for a partitioned region we must connect as a PEER member and this member we specify won't be storing any region data as shown below.

datalocations-cache-no-storage.xml

```
<?xml version="1.0"?>
<!DOCTYPE cache PUBLIC
        "-//GemStone Systems, Inc.//GemFire Declarative Caching
9.1//EN"
        "http://www.gemstone.com/dtd/cache8_1.dtd">

<cache>    <pdx read-serialized="true">
        <pdx-serializer>
            <class-
name>com.gemstone.gemfire.pdx.ReflectionBasedAutoSerializer</class-
name>
            <parameter name="classes">
                <string>io.pivotal.app.domain.*</string>
            </parameter>
        </pdx-serializer>
    </pdx>

    <region name="employees" >
        <region-attributes data-policy="partition" >
            <partition-attributes local-max-memory="0" redundant-
copies="1" total-num-buckets="11"/>
            <subscription-attributes interest-policy="all"/>
            <eviction-attributes>
                <lru-heap-percentage action="overflow-to-disk"/>
            </eviction-attributes>
        </region-attributes>
    </region>
</cache>
```

13.3 Run the Java Class and verify output.

Linux:

```
$ java -classpath lib/Domain-1.jar:lib/ClientApp-
1.jar:$GEMFIRE/lib/geode-dependencies.jar
io.pivotal.app.VerifyDataLocations
```

Note: the program opens datalocations-cache-no-storage.xml internally. If the program reports that it cannot find a locator, then change the hostname to an IP in the code.

Windows:

```
> java -cp lib\ClientApp-1.jar;lib\PeopleDomain-
1.jar;%GEMFIRE%\lib\geode-dependencies.jar
io.pivotal.app.VerifyDataLocations
```

```
Employees region size = 13
"Primary Member [Host=localhost, Id=localhost(server1:4615)<v1>:19973 -
Key=7371, Value=PDX[2,Employee]{deptno=10, empno=7371, job=SALESMAN,
name=WILLIAMS}]
"Primary Member [Host=localhost, Id=localhost(server2:4616)<v2>:17535 -
Key=7372, Value=PDX[2,Employee]{deptno=30, empno=7372, job=PRESIDENT,
name=LUCIA}]
"Primary Member [Host=localhost, Id=localhost(server2:4616)<v2>:17535 -
Key=7373, Value=PDX[2,Employee]{deptno=40, empno=7373, job=CLERK,
name=SIENA}]
"Primary Member [Host=localhost, Id=localhost(server1:4615)<v1>:19973 -
Key=7374, Value=PDX[2,Employee]{deptno=10, empno=7374, job=SALESMAN,
name=LUCAS}]
"Primary Member [Host=localhost, Id=localhost(server1:4615)<v1>:19973 -
Key=7375, Value=PDX[2,Employee]{deptno=30, empno=7375, job=CLERK,
name=ROB}]
"Primary Member [Host=localhost, Id=localhost(server2:4616)<v2>:17535 -
Key=7376, Value=PDX[2,Employee]{deptno=20, empno=7376, job=CLERK,
name=ADRIAN}]
"Primary Member [Host=localhost, Id=localhost(server1:4615)<v1>:19973 -
Key=7377, Value=PDX[2,Employee]{deptno=20, empno=7377, job=CLERK,
name=ADAM}]
"Primary Member [Host=localhost, Id=localhost(server2:4616)<v2>:17535 -
Key=7378, Value=PDX[2,Employee]{deptno=20, empno=7378, job=MANAGER,
name=SALLY}]
"Primary Member [Host=localhost, Id=localhost(server2:4616)<v2>:17535 -
Key=7379, Value=PDX[2,Employee]{deptno=10, empno=7379, job=CLERK,
name=FRANK}]
"Primary Member [Host=localhost, Id=localhost(server2:4616)<v2>:17535 -
Key=7380, Value=PDX[2,Employee]{deptno=40, empno=7380, job=CLERK,
name=BLACK}]
"Primary Member [Host=localhost, Id=localhost(server1:4615)<v1>:19973 -
Key=7381, Value=PDX[2,Employee]{deptno=40, empno=7381, job=SALESMAN,
name=BROWN}]
"Primary Member [Host=localhost, Id=localhost(server1:4615)<v1>:19973 -
Key=7369, Value=PDX[2,Employee]{deptno=20, empno=7369, job=CLERK,
name=SMITH}]
"Primary Member [Host=localhost, Id=localhost(server1:4615)<v1>:19973 -
Key=7370, Value=PDX[2,Employee]{deptno=10, empno=7370, job=MANAGER,
name=APPLES}]
```

Note the distribution of the above entries. Seven Employee entries are on server1 and 6 are on server2. This is a balanced expectation of a region spread across two nodes.

13.4 Keep the database running for Chapter 14.

14 Add a New Cache Server

In the following section we show how we would add a third cache server to our GemFire system. This cache server will be a peer of the other cache servers already running. For this reason and in this context it is known as a PEER as opposed to a client member. You can dynamically add peers to the system while it is running. The other peers and the client automatically discover the new peers. The new peers automatically receive a copy

of any replicated regions that the others create. However, partitioned regions do not automatically redistribute data to the new peer unless you explicitly instruct GemFire to rebalance the partitioned region.

14.1 Start up the cluster if it is not already running from Chapter 13.

```
Wed Jan 19 09:11:40 wwilliams@:~/gf91/workshop/chapter14 $ .
startall.sh
```

14.2 Populate the cluster if it is not already populate from a previous lesson.

```
Wed Jan 19 09:11:40 wwilliams@:~/gf91/workshop/chapter14 $ cd data
Wed Jan 19 09:11:40 wwilliams@:~/gf91/workshop/chapter14/data $ .
populate.sh
Wed Jan 19 09:11:40 wwilliams@:~/gf91/workshop/chapter14 $ cd ..
```

14.3 Make a new directory in your terminal window where the workshop files exist called "**server3**" as shown below.

```
Wed Jan 19 09:11:40 wwilliams@:~/gf91/workshop/chapter14 $ mkdir
server3
```

14.4 Open a separate terminal window. Connect to **gfsh** and start a new cache server member. You can execute the script start_server3.sh in chapter14 or you can copy/ paste the launch for server2 from the **gfsh** echo and change the name from server2 to server3, along with the port number, as follows:

```
gfsh>connect

gfsh> start server --name=server3 --locators=localhost[10334] --J=-
Xms512m --J=-Xmx512m --classpath=../../projects/Domain/target/Domain-
1.jar --cache-xml-file=config/cache.xml --properties-
file=config/gemfire.properties --server-port=40406

Starting a GemFire Server in
/Users/wwilliams/gf91/workshop/chapter13/server3...
.....
Server in /Users/wwilliams/gf91/workshop/chapter13/server3 on
localhost[40406] as server3 is currently online.
Process ID: 4839
Uptime: 2 seconds
GemFire Version: 9.1
Java Version: 1.8.0_40
Log File: /Users/wwilliams/gf91/workshop/chapter13/server3/server3.log
JVM Arguments: -
DGemfirePropertyFile=/Users/wwilliams/gf91/workshop/chapter13/config/ge
mfire.properties
-Dgemfire.cache-xml-
file=/Users/wwilliams/gf91/workshop/chapter13/config/cache.xml -
Dgemfire.locators=localhost[10334]
-Dgemfire.use-cluster-configuration=true -Xms512m -Xmx512m -
XX:OnOutOfMemoryError=kill -KILL %p -
Dgemfire.launcher.registerSignalHandlers=true
-Djava.awt.headless=true -
Dsun.rmi.dgc.server.gcInterval=9223372036854775806
Class-Path:../lib/ClientApp-1.jar:../lib/Domain-
1.jar:/usr/local/Cellar/GemFire/9.1/libexec/lib/geode-dependencies.jar
```

14.5 Verify through the GemFire shell that server3 is now added

```
gfsh>list members
  Name    | Id
--------- | ---------------------------------------
server2   | localhost(server2:4934)<v2>:23225
locator1  | localhost(locator1:4932:locator)<v0>:38259
server1   | localhost(server1:4933)<v1>:52223
server3   | localhost(server3:4963)<v3>:54649

gfsh>list regions
List of regions
---------------
departments
employees
people
```

14.6 Re-run the **VerifyDataLocations** from the previous lesson. Note that no entry is yet on your new cache server.

```
Employees region size = 13
"Primary Member [Host=localhost, Id=localhost(server1:4933)<v1>:52223 -
Key=7371, Value=PDX[2,Employee]{deptno=10, empno=7371, job=SALESMAN,
name=WILLIAMS}]
"Primary Member [Host=localhost, Id=localhost(server1:4933)<v1>:52223 -
Key=7372, Value=PDX[2,Employee]{deptno=30, empno=7372, job=PRESIDENT,
name=LUCIA}]
"Primary Member [Host=localhost, Id=localhost(server2:4934)<v2>:23225 -
Key=7373, Value=PDX[2,Employee]{deptno=40, empno=7373, job=CLERK,
name=SIENA}]
"Primary Member [Host=localhost, Id=localhost(server1:4933)<v1>:52223 -
Key=7374, Value=PDX[2,Employee]{deptno=10, empno=7374, job=SALESMAN,
name=LUCAS}]
"Primary Member [Host=localhost, Id=localhost(server2:4934)<v2>:23225 -
Key=7375, Value=PDX[2,Employee]{deptno=30, empno=7375, job=CLERK,
name=ROB}]
"Primary Member [Host=localhost, Id=localhost(server2:4934)<v2>:23225 -
Key=7376, Value=PDX[2,Employee]{deptno=20, empno=7376, job=CLERK,
name=ADRIAN}]
"Primary Member [Host=localhost, Id=localhost(server1:4933)<v1>:52223 -
Key=7377, Value=PDX[2,Employee]{deptno=20, empno=7377, job=CLERK,
name=ADAM}]
"Primary Member [Host=localhost, Id=localhost(server1:4933)<v1>:52223 -
Key=7378, Value=PDX[2,Employee]{deptno=20, empno=7378, job=MANAGER,
name=SALLY}]
"Primary Member [Host=localhost, Id=localhost(server2:4934)<v2>:23225 -
Key=7379, Value=PDX[2,Employee]{deptno=10, empno=7379, job=CLERK,
name=FRANK}]
"Primary Member [Host=localhost, Id=localhost(server2:4934)<v2>:23225 -
Key=7380, Value=PDX[2,Employee]{deptno=40, empno=7380, job=CLERK,
name=BLACK}]
"Primary Member [Host=localhost, Id=localhost(server1:4933)<v1>:52223 -
Key=7381, Value=PDX[2,Employee]{deptno=40, empno=7381, job=SALESMAN,
name=BROWN}]
"Primary Member [Host=localhost, Id=localhost(server2:4934)<v2>:23225 -
Key=7369, Value=PDX[2,Employee]{deptno=20, empno=7369, job=CLERK,
name=SMITH}]
```

```
"Primary Member [Host=localhost, Id=localhost(server2:4934)<v2>:23225 -
Key=7370, Value=PDX[2,Employee]{deptno=10, empno=7370, job=MANAGER,
name=APPLES}]
```

Note: Six Employee objects are on server 1 and 7 Employee objects are on server 2. None are on server3.

14.7 Issue a rebalance command to the employees region.

```
gfsh>rebalance --include-region=employees

Rebalanced partition regions  /employees

                                        Rebalanced Stats
| Value
-------------------------------------------------------------------------
----------------------- | -----
Total bytes in all redundant bucket copies created during this
rebalance                          | 0
Total time (in milliseconds) spent creating redundant bucket copies
during this rebalance       | 0
Total number of redundant copies created during this rebalance
| 0
Total bytes in buckets moved during this rebalance
| 474
Total time (in milliseconds) spent moving buckets during this rebalance
| 176
Total number of buckets moved during this rebalance
| 8
Total time (in milliseconds) spent switching the primary state of
buckets during this rebalance | 0
Total primaries transferred during this rebalance
| 0
Total time (in milliseconds) for this rebalance
| 188
```

14.8 Re-run the **VerifyDataLocations** from the previous lesson. Note that the entries are distributed across all 3 cache servers.

```
Employees region size = 13
"Primary Member [Host=localhost, Id=localhost(server1:4933)<v1>:52223 -
Key=7371, Value=PDX[2,Employee]{deptno=10, empno=7371, job=SALESMAN,
name=WILLIAMS}]
"Primary Member [Host=localhost, Id=localhost(server1:4933)<v1>:52223 -
Key=7372, Value=PDX[2,Employee]{deptno=30, empno=7372, job=PRESIDENT,
name=LUCIA}]
"Primary Member [Host=localhost, Id=localhost(server2:4934)<v2>:23225 -
Key=7373, Value=PDX[2,Employee]{deptno=40, empno=7373, job=CLERK,
name=SIENA}]
"Primary Member [Host=localhost, Id=localhost(server1:4933)<v1>:52223 -
Key=7374, Value=PDX[2,Employee]{deptno=10, empno=7374, job=SALESMAN,
name=LUCAS}]
"Primary Member [Host=localhost, Id=localhost(server2:4934)<v2>:23225 -
Key=7375, Value=PDX[2,Employee]{deptno=30, empno=7375, job=CLERK,
name=ROB}]
"Primary Member [Host=localhost, Id=localhost(server3:4963)<v3>:54649 -
Key=7376, Value=PDX[2,Employee]{deptno=20, empno=7376, job=CLERK,
name=ADRIAN}]
```

```
"Primary Member [Host=localhost, Id=localhost(server1:4933)<v1>:52223 -
Key=7377, Value=PDX[2,Employee]{deptno=20, empno=7377, job=CLERK,
name=ADAM}]
"Primary Member [Host=localhost, Id=localhost(server3:4963)<v3>:54649 -
Key=7378, Value=PDX[2,Employee]{deptno=20, empno=7378, job=MANAGER,
name=SALLY}]
"Primary Member [Host=localhost, Id=localhost(server2:4934)<v2>:23225 -
Key=7379, Value=PDX[2,Employee]{deptno=10, empno=7379, job=CLERK,
name=FRANK}]
"Primary Member [Host=localhost, Id=localhost(server2:4934)<v2>:23225 -
Key=7380, Value=PDX[2,Employee]{deptno=40, empno=7380, job=CLERK,
name=BLACK}]
"Primary Member [Host=localhost, Id=localhost(server3:4963)<v3>:54649 -
Key=7381, Value=PDX[2,Employee]{deptno=40, empno=7381, job=SALESMAN,
name=BROWN}]
"Primary Member [Host=localhost, Id=localhost(server2:4934)<v2>:23225 -
Key=7369, Value=PDX[2,Employee]{deptno=20, empno=7369, job=CLERK,
name=SMITH}]
"Primary Member [Host=localhost, Id=localhost(server2:4934)<v2>:23225 -
Key=7370, Value=PDX[2,Employee]{deptno=10, empno=7370, job=MANAGER,
name=APPLES}]
```

Note: Four Employee objects are on server1, 6 on server 2 and 3 on server3.

14.9 Stop server3.

```
gfsh>stop server --name=server3
```

14.10 If you prefer to always automatically rebalance whenever a new cache server joins the cluster, just add the --rebalance option to the start server command.

```
gfsh>start server --name=server3 --locators=localhost[10334] --J=-
Xms512m --J=-Xmx512m
--classpath=../lib/ClientApp-1.jar:../lib/Domain-1.jar --cache-xml-
file=config/cache.xml --properties-file=config/gemfire.properties
--server-port=40406 --rebalance
```

Note: You can hit the up arrow to repeat the previous start server command and then modify it.

15 Asyncronous Event Listener (Write Behind)

An AsyncEventListener receives callbacks for events that change region data. You can use an AsyncEventListener implementation as a write-behind cache event handler to synchronize region updates with a database or message queue. An AsyncEventListener instance is serviced by its own dedicated thread that calls back to method **processEvents**. Events are placed in an internal AsyncEventQueue, and the dedicated thread dispatches a batch of events at a time to the listener implementation.

Given the GemFire cluster is down we are going to add an Async Event Listener and region using the cache.xml file. We could use **gfsh** but this shows how we can do it in XML

The code for our Async Event Listener is defined as follows, it simple displays each event to the log file for each server.

TestAsyncEventListener.java in package "**io.pivotal.listener**"

```
package io.pivotal.listener;

import java.util.List;
import java.util.Properties;
import java.util.logging.Level;
import java.util.logging.Logger;

import com.gemstone.gemfire.cache.Declarable;
import com.gemstone.gemfire.cache.asyncqueue.AsyncEvent;
import com.gemstone.gemfire.cache.asyncqueue.AsyncEventListener;
import com.gemstone.gemfire.pdx.PdxInstance;
import io.pivotal.app.domain.Test;

public class TestAsyncEventListener implements AsyncEventListener,
Declarable
{
    private Logger logger =
Logger.getLogger(this.getClass().getSimpleName());

    public void close() {
        // TODO Auto-generated method stub

    }

    public void init(Properties arg0) {
        // TODO Auto-generated method stub

    }

    public boolean processEvents(List<AsyncEvent> entries)
    {
        logger.log (Level.INFO, String.format("TestAsyncEventListener :
Size of List<AsyncEvent> = %s", entries.size()));

        // process the events here, could write to a database etc
        for (AsyncEvent ge: entries)
        {

             PdxInstance pdxInstance = (PdxInstance)
ge.getDeserializedValue();
             Test test = (Test) pdxInstance.getObject();

             System.out.println(test.toString());
        }

        return true;
    }
}
```

15.1. Edit "**./server1/cache.xml**" to add the section in bold.

```
<?xml version="1.0"?>
<!DOCTYPE cache PUBLIC
    "-//GemStone Systems, Inc.//GemFire Declarative Caching 7.0//EN"
    "http://www.gemstone.com/dtd/cache7_0.dtd">

<cache>
    <async-event-queue id="sampleQueue" batch-size="10" batch-time-
interval="20000" parallel="true" dispatcher-threads="5">
        <async-event-listener>
            <class-
name>io.pivotal.listener.TestAsyncEventListener</class-name>
        </async-event-listener>
    </async-event-queue>
    <cache-server port="40001"/>
    <pdx read-serialized="true">
        <pdx-serializer>
            <class-
name>com.gemstone.gemfire.pdx.ReflectionBasedAutoSerializer</class-
name>
            <parameter name="classes">
                <string>io.pivotal.listener.domain..*</string>
            </parameter>
        </pdx-serializer>
    </pdx>

    <region name="asyncRegion">
        <region-attributes data-policy="partition"
                           statistics-enabled="true"
                           concurrency-level="16"
                           async-event-queue-ids="sampleQueue">
            <partition-attributes redundant-copies="1" total-num-
buckets="113"/>
        </region-attributes>
    </region>
    ......
```

15.2 Edit "**./sever2/cache.xml**" to add the section in bold

```
<?xml version="1.0"?>
<!DOCTYPE cache PUBLIC
    "-//GemStone Systems, Inc.//GemFire Declarative Caching 7.0//EN"
    "http://www.gemstone.com/dtd/cache7_0.dtd">

<cache>
    <async-event-queue id="sampleQueue" batch-size="10" batch-time-
interval="20000" parallel="true" dispatcher-threads="5">
        <async-event-listener>
            <class-
name>io.pivotal.listener.TestAsyncEventListener</class-name>
        </async-event-listener>
    </async-event-queue>

    <cache-server port="40002"/>
    <pdx read-serialized="true">
        <pdx-serializer>
            <class-
name>com.gemstone.gemfire.pdx.ReflectionBasedAutoSerializer</class-
name>
            <parameter name="classes">
                <string>io.pivotal.listener.domain..*</string>
            </parameter>
        </pdx-serializer>
    </pdx>

    <region name="asyncRegion">
        <region-attributes data-policy="partition"
                           statistics-enabled="true"
                           concurrency-level="16"
                           async-event-queue-ids="sampleQueue">
            <partition-attributes redundant-copies="1" total-num-
buckets="113"/>
        </region-attributes>
    </region>
    ....
```

15.3 Add the **asynceventlistener.jar** to the classpath in **startall.sh**

```
start server --name=server1 --cache-xml-file=./server1/cache.xml --
properties-file=./server1/gemfire.properties --
locators=localhost[10334] --J=-Xms512m --J=-Xmx512m --
classpath=../lib/ClientApp-1.jar:../lib/asynceventlistener.jar

start server --name=server2 --cache-xml-file=./server2/cache.xml --
properties-file=./server2/gemfire.properties --
locators=localhost[10334] --server-port=40405 --J=-Xms512m --J=-Xmx512m
--classpath=../lib/ ClientApp-1.jar:../lib/asynceventlistener.jar
```

15.4 Start the cluster

```
Wed Jan 19 13:27:21 wwilliams@:~/gf91/workshop $ ./startall.sh
```

15.5 Add the dept/emp data

```
Wed Jan 19 13:09:06 wwilliams@:~/gf91/workshop $ . data/populate.sh
```

15.6 Run "**verify.sh**" to ensure data still exists

```
Wed Jan 19 13:35:45 wwilliams@:~/gf91/workshop $ ./verify.sh
```

At this point we will test the Async Event Listener.

15.7 Add some data into the region "/asyncRegion" by saving the data below into a file called test-data. This will add 50 entries.

test-data

```
put --key=1 --value=('id':1,'name':'test1') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=2 --value=('id':2,'name':'test2') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=3 --value=('id':3,'name':'test3') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=4 --value=('id':4,'name':'test4') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=5 --value=('id':5,'name':'test5') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=6 --value=('id':6,'name':'test6') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=7 --value=('id':7,'name':'test7') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=8 --value=('id':8,'name':'test8') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=9 --value=('id':9,'name':'test9') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=10 --value=('id':10,'name':'test10') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=11 --value=('id':11,'name':'test11') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=12 --value=('id':12,'name':'test12') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=13 --value=('id':13,'name':'test13') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=14 --value=('id':14,'name':'test14') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=15 --value=('id':15,'name':'test15') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=16 --value=('id':16,'name':'test16') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=17 --value=('id':17,'name':'test17') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=18 --value=('id':18,'name':'test18') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=19 --value=('id':19,'name':'test19') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=20 --value=('id':20,'name':'test20') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=21 --value=('id':21,'name':'test21') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=22 --value=('id':22,'name':'test22') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=23 --value=('id':23,'name':'test23') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=24 --value=('id':24,'name':'test24') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=25 --value=('id':25,'name':'test25') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=26 --value=('id':26,'name':'test26') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=27 --value=('id':27,'name':'test27') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=28 --value=('id':28,'name':'test28') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=29 --value=('id':29,'name':'test29') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
```

```
put --key=30 --value=('id':30,'name':'test30') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=31 --value=('id':31,'name':'test31') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=32 --value=('id':32,'name':'test32') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=33 --value=('id':33,'name':'test33') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=34 --value=('id':34,'name':'test34') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=35 --value=('id':35,'name':'test35') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=36 --value=('id':36,'name':'test36') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=37 --value=('id':37,'name':'test37') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=38 --value=('id':38,'name':'test38') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=39 --value=('id':39,'name':'test39') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=40 --value=('id':40,'name':'test40') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=41 --value=('id':41,'name':'test41') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=42 --value=('id':42,'name':'test42') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=43 --value=('id':43,'name':'test43') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=44 --value=('id':44,'name':'test44') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=45 --value=('id':45,'name':'test45') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=46 --value=('id':46,'name':'test46') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=47 --value=('id':47,'name':'test47') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=48 --value=('id':48,'name':'test48') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=49 --value=('id':49,'name':'test49') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
put --key=50 --value=('id':50,'name':'test50') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion;
```

15.8 Add to region as follows. Verify that you give it the correct path to the file test-data you created above.

```
wwilliams@Wes:~/gf91/workshop$ GemFire
        _________________________     __
       / _____/ _____/ _____/ /____/ /
      / /  __/ /___  /_____  / _____  /
     / /__/ / _____/  _____/ / /    / /
    /______/_/       /______/_/    /_/    v1.0.0

Monitor and Manage GemFire
gfsh>connect --locator=localhost[10334];
Connecting to Locator at [host=localhost, port=10334] ..
Connecting to Manager at [host=localhost, port=1099] ..
Successfully connected to: [host=localhost, port=1099]
```

```
gfsh>run --file=data/test-data

....

50. Executing - put --key=50 --value=('id':50,'name':'test50') --value-
class=io.pivotal.listener.domain.Test --region=asyncRegion

Result      : true
Key Class   : java.lang.String
Key         : 50
Value Class : io.pivotal.listener.domain.Test

Value
------
<NULL>
```

15.9 Open the 2 logs files below and verify output as follows

./server1/server1.log

```
...

[info 2015/01/19 15:13:22.163 EST server1 <Pooled Waiting Message
Processor 0> tid=0x25] Initializing region
_B__AsyncEventQueue__sampleQueue__PARALLEL__GATEWAY__SENDER__QUEUE_78
Jan 19, 2015 3:13:22 PM io.pivotal.listener.TestAsyncEventListener
processEvents
INFO: TestAsyncEventListener : Size of List<AsyncEvent> = 10
Test [id=21, name=test21]
Test [id=23, name=test23]
Test [id=26, name=test26]
Test [id=27, name=test27]
Test [id=29, name=test29]
Test [id=32, name=test32]
Test [id=37, name=test37]
Test [id=35, name=test35]
Test [id=39, name=test39]
Test [id=40, name=test40]

...
```

./server2/server2.log

```
...

[info 2015/01/19 15:13:22.452 EST server2 <Function Execution
Processor1> tid=0x46] Initializing region
_B__AsyncEventQueue__sampleQueue__PARALLEL__GATEWAY__SENDER__QUEUE_109
Jan 19, 2015 3:13:42 PM io.pivotal.listener.TestAsyncEventListener
processEvents
INFO: TestAsyncEventListener : Size of List<AsyncEvent> = 5
Test [id=42, name=test42]
Test [id=43, name=test43]
Test [id=45, name=test45]
Test [id=48, name=test48]
Test [id=50, name=test50]

...
```

16 Failover

This test will involve streaming objects into the cluster and taking components down. The BulkInsertPerson has an option to stream persons slowly by pausing 1 second after every insert. Details on starting the streaming client are explained in the workflow.

16.1 Change startall script to use the cache-people.xml configuration

```
start server --name=server1 --cache-xml-file=./server1/cache.xml --
properties-file=./config/gemfire.properties --locators=localhost[10334]
--J=-Xms512m --J=-Xmx512m --classpath=../lib/ClientApp-1.jar

start server --name=server2 --cache-xml-file=./server2/cache.xml --
properties-file=./config/gemfire.properties --locators=localhost[10334]
--server-port=40405 --J=-Xms512m --J=-Xmx512m --classpath=../lib/
ClientApp-1.jar
```

Change from cache.xml to cache-people.xml in startall.sh

16.2 Start cluster

```
wwilliams@Wes:~/gf91/workshop $ . startall.sh
```

Start cluster with the /people region explicitly defined

16.3 Start Streaming Clients in cluster

The following will cause Person objects to stream into the grid starting with key 1. Each key number will be incremented by 2 (and so only even numbered keys will be generated). The program will pause 1 second after every insert.

16.3.1 Start Streaming Client1 to generate odd numbered keys

```
java -classpath lib/PeopleDomain-1.jar:lib/ClientApp-
1.jar:$GEMFIRE/lib/geode-dependencies.jar
io.pivotal.app.insert.BulkInsertPerson SLOW 1 2 1
```

SLOW 1 2 1: SLOW = Pause after each insert. 1 = start the key count at 1. 2 = increment the keys by 2. 1 = pause 1 second after each insert.

The following will cause Person objects to stream into the grid starting with key 1. Each key number will be incremented by 2 (and so only even numbered keys will be generated). The program will pause 1 second after every insert.

16.3.2 Start Streaming Client2 to generate even numbered keys

```
java -classpath lib/PeopleDomain-1.jar:lib/ClientApp-
1.jar:$GEMFIRE/lib/geode-dependencies.jar
io.pivotal.app.insert.BulkInsertPerson SLOW 0 2 1
```

SLOW 0 2 1: SLOW = Pause after each insert. 0 = start the key count at 0. 2 = increment the keys by 2. 1 = pause 1 second after each insert.

16.4 Verify that Person's are streaming into the cluster

```
gfsh>query --query="select * from /people"

Result     : true
startCount : 0
endCount   : 20
Rows       : 419

      name        | id
----------------- | ---
Ocean Larson      | 223
Lydia Morales     | 317
Ocean Morris      | 483
Caleb Kim         | 244
Joan Guerra       | 364
Kendall Peterson  | 204
Brody Conway      | 413
Timon James       | 93
Yvette Palmer     | 63
Quynn Cash        | 88
Cairo Guerrero    | 362
Karyn Warner      | 448
Amena Dudley      | 68
Lacota Vinson     | 114
Freya Perez       | 394
Jaden Mccullough  | 109
Miranda Rosa      | 98
Bo Hogan          | 91
Talon Owens       | 190
Kane Peterson     | 147
Mariko Sharp      | 176

Press n to move to next page, q to quit and p to previous page : q
NEXT_STEP_NAME : END
```

Note that there are both odd and even id's in the cluster. This shows that both clients are streaming.

16.5 Examine a key to show its distribution while cluster is up. Keep changing the key until you find an entry that is Primary on server2

```
gfsh>locate entry --region=/people --key='21'
Result          : true
Key Class       : java.lang.String
Key             : 21
Locations Found : 2


MemberName |           MemberId           |   Primary    | BucketId
---------- | ---------------------------- | ------------ | --------
server2    | wes(server2:10953)<v2>:20362 | *Primary PR* | 4
server1    | wes(server1:10939)<v1>:61558 | No           | 4
```

Note that the primary entry is on server2 and the backup is on server1

16.6 Kill server2 while both clients are streaming data

```
wwilliams@Wes:~/gf91/workshop $ ps -ef | grep server2
  501 10353     1   0  8:10AM ttys006    0:08.43
/Library/Java/JavaVirtualMachines/jdk1.8.0_40.jdk/Contents/Home/jre/bin
/java -server -classpath
../lib/clientapp.jar:../lib/asynceventlistener.jar:/Programs/pivotal/Pi
votal_GemFire_810_b50625_Linux/lib/geode-dependencies.jar -
DGemfirePropertyFile=/Users/wwilliams/gf91/workshop/server2/gemfire.pro
perties -Dgemfire.cache-xml-
file=/Users/wwilliams/gf91/workshop/server2/cache-people.xml -
Dgemfire.locators=localhost[10334] -Dgemfire.use-cluster-
configuration=true -Xms512m -Xmx512m -XX:OnOutOfMemoryError="kill -9
%p" -Dgemfire.launcher.registerSignalHandlers=true -
Djava.awt.headless=true -
Dsun.rmi.dgc.server.gcInterval=9223372036854775806
com.gemstone.gemfire.distributed.ServerLauncher start server2 --
redirect-output --server-port=40405
  501 10414   487   0  8:12AM ttys009    0:00.00 grep server2

wwilliams@Wes:~/gf91/workshop $
wwilliams@Wes:~/gf91/workshop $ kill 10353
wwilliams@Wes:~/gf91/workshop $
wwilliams@Wes:~/gf91/workshop $ ps -ef | grep server2
  501 10417   487   0  8:13AM ttys009    0:00.00 grep server2
```

16.7 Verify that Person's have been relocated to server1

```
gfsh>locate entry --region=/people --key='21'
Result          : true
Key Class       : java.lang.String
Key             : 21
Locations Found : 1


MemberName |           MemberId           |   Primary    | BucketId
---------- | ---------------------------- | ------------ | --------
server1    | wes(server1:10939)<v1>:61558 | *Primary PR* | 4
```

16.8 Restart server2

```
wwilliams@Wes:~/gf91/workshop $ gfsh start server --name=server2 --
cache-xml-file=./config/cache.xml --properties-
file=./config/gemfire.properties --locators=localhost[10334] --server-
port=40405 --J=-Xms512m --J=-Xmx512m --classpath
../../projects/PeopleDomain/target/PeopleDomain-
1.jar:../../projects/Domain/target/Domain-1.jar
```

Note that the restart is with the cache-people.xml configuration

16.9 Verify that Person objects are rebalanced in the cluster

```
gfsh>locate entry --region=/people --key='21'
Result          : true
Key Class       : java.lang.String
Key             : 21
Locations Found : 2


MemberName |           MemberId            |   Primary    | BucketId
---------- | ----------------------------- | ------------ | --------
server1    | wes(server1:10939)<v1>:61558 | No           | 4
server2    | wes(server2:10997)<v4>:45669 | *Primary PR* | 4
```

Note that the Person with key 21 has its backup restored

16.10 Verify other keys to show that some entries are primary on server2

```
gfsh>locate entry --region=/people --key='23'
Result          : true
Key Class       : java.lang.String
Key             : 23
Locations Found : 2


MemberName |           MemberId            |   Primary    | BucketId
---------- | ----------------------------- | ------------ | --------
server1    | wes(server1:10939)<v1>:61558 | *Primary PR* | 6
server2    | wes(server2:10997)<v4>:45669 | No           | 6
```

Note that the Person with key 23 is on server1, demonstrating that the entries are rebalanced

17 Region Persistence

This test will involve streaming objects into the cluster and taking servers down until the entire cluster is gone. We will then restore the cluster.

17.1 Change startall script to use the cache-people.xml configuration

```
start server --name=server1 --cache-xml-file=./server1/cache-people.xml
--properties-file=./server1/gemfire.properties --
locators=localhost[10334] --J=-Xms512m --J=-Xmx512m --
classpath=../lib/ClientApp-1:../lib/asynceventlistener.jar

start server --name=server2 --cache-xml-file=./server2/cache-people.xml
--properties-file=./server2/gemfire.properties --
locators=localhost[10334] --server-port=40405 --J=-Xms512m --J=-Xmx512m
--classpath=../lib/ClientApp-1.jar:../lib/asynceventlistener.jar
```

Change from cache.xml to cache-people.xml in startall.sh

17.2 Start cluster

```
wwilliams@Wes:~/gf91/workshop $ . startall.sh
```

Start cluster with the /people region explicitly defined

17.3 Start Streaming Clients in cluster

The following will cause Person objects to stream into the grid starting with key 1. Each key number will be incremented by 2 (and so only even numbered keys will be generated). The program will pause 1 second after every insert.

17.3.1 Start Streaming Client1 to generate odd numbered keys

```
java -classpath lib/Domain-1.jar:lib/ClientApp-
1.jar:$GEMFIRE/lib/geode-dependencies.jar
io.pivotal.app.insert.BulkInsertPerson SLOW 1 2 1
```

SLOW 1 2 1: SLOW = Pause after each insert. 1 = start the key count at 1. 2 = increment the keys by 2. 1 = pause 1 second after each insert.

The following will cause Person objects to stream into the grid starting with key 1. Each key number will be incremented by 2 (and so only even numbered keys will be generated). The program will pause 1 second after every insert.

17.4 Verify that Person's are streaming into the cluster

```
gfsh>query --query="select * from /people"

Result     : true
startCount : 0
endCount   : 20
Rows       : 114

      name         | id
------------------ | ---
Miranda Owen       | 133
Libby Le           | 119
Mariko Bradford    | 171
Lydia Craft        | 37
Giacomo Parsons    | 135
Talon Craig        | 91
Lynn Simpson       | 31
Libby Harrison     | 113
Penelope Cash      | 79
Chantale Bradford  | 9
Hyatt Dominguez    | 213
Joan Kim           | 185
Victor Russo       | 73
Jordan Warner      | 193
Joan Mason         | 55
Giacomo Landry     | 45
Lynn Bradford      | 147
Georgia Mcknight   | 197
Channing Owen      | 155
Bo Strong          | 77
Phyllis Whitney    | 81

Press n to move to next page, q to quit and p to previous page : q
NEXT_STEP_NAME : END
```

17.5 Kill server2 while the client is streaming data

```
wwilliams@Wes:~/gf91/workshop $ ps -ef | grep server2
  501 11205     1   0  9:43AM ttys008    0:08.80
/Library/Java/JavaVirtualMachines/jdk1.8.0_40.jdk/Contents/Home/jre/bin
/java -server -classpath ../lib/ClientApp-
1.jar:../lib/asynceventlistener.jar:/Programs/pivotal/Pivotal_GemFire_8
10_b50625_Linux/lib/geode-dependencies.jar -
DGemfirePropertyFile=/Users/wwilliams/gf91/workshop/server2/gemfire.pro
perties -Dgemfire.cache-xml-
file=/Users/wwilliams/gf91/workshop/server2/cache-people.xml -
Dgemfire.locators=localhost[10334] -Dgemfire.use-cluster-
configuration=true -Xms512m -Xmx512m -XX:OnOutOfMemoryError="kill -9
%p" -Dgemfire.launcher.registerSignalHandlers=true -
Djava.awt.headless=true -
Dsun.rmi.dgc.server.gcInterval=9223372036854775806
com.gemstone.gemfire.distributed.ServerLauncher start server2 --
redirect-output --server-port=40405
  501 11253   481   0  9:45AM ttys008    0:00.00 grep server2

wwilliams@Wes:~/gf91/workshop $ kill 11205
wwilliams@Wes:~/gf91/workshop $
wwilliams@Wes:~/gf91/workshop $ ps -ef | grep server2
  501 11253   481   0  9:45AM ttys008    0:00.00 grep server2
```

17.6 Kill server1 while the client is streaming data

```
wwilliams@Wes:~/gf91/workshop $ ps -ef | grep server1
  501 11191     1   0  9:43AM ttys008    0:10.75
/Library/Java/JavaVirtualMachines/jdk1.8.0_40.jdk/Contents/Home/jre/bin
/java -server -classpath ../lib/ClientApp-
1.jar:../lib/asynceventlistener.jar:/Programs/pivotal/Pivotal_GemFire_8
10_b50625_Linux/lib/geode-dependencies.jar -
DGemfirePropertyFile=/Users/wwilliams/gf91/workshop/server1/gemfire.pro
perties -Dgemfire.cache-xml-
file=/Users/wwilliams/gf91/workshop/server1/cache-people.xml -
Dgemfire.locators=localhost[10334] -Dgemfire.use-cluster-
configuration=true -Xms512m -Xmx512m -XX:OnOutOfMemoryError="kill -9
%p" -Dgemfire.launcher.registerSignalHandlers=true -
Djava.awt.headless=true -
Dsun.rmi.dgc.server.gcInterval=9223372036854775806
com.gemstone.gemfire.distributed.ServerLauncher start server1 --
redirect-output --server-port=40404
  501 11264   481   0  9:46AM ttys008    0:00.00 grep server1

wwilliams@Wes:~/gf91/workshop $ kill 11191
wwilliams@Wes:~/gf91/workshop $ ps -ef | grep server2
  501 11264   481   0  9:46AM ttys008    0:00.00 grep server1
```

17.7 Verify that the servers are down in **gfsh**:

```
gfsh>list members
  Name    | Id
--------- | ----------------------------------------
locator1 | localhost(locator1:11130:locator)<v0>:31663
```

17.8 Restart server1

```
wwilliams@Wes:~/gf91/workshop $ gfsh start server --name=server1 --
cache-xml-file=./server1/cache-people.xml --properties-
file=./server1/gemfire.properties --locators=localhost[10334] --server-
port=40404 --J=-Xms512m --J=-Xmx512m --classpath=../lib/ClientApp-
1.jar:../lib/asynceventlistener.jar
```

Important note: You must restart the servers in the same order as when they went down

17.9 Restart server2 (remember to use the cache-people.xml configuration)

```
wwilliams@Wes:~/gf91/workshop $ gfsh start server --name=server2 --
cache-xml-file=./server2/cache-people.xml --properties-
file=./server2/gemfire.properties --locators=localhost[10334] --server-
port=40405 --J=-Xms512m --J=-Xmx512m --classpath=../lib/ClientApp-
1.jar:../lib/asynceventlistener.jar
```

Important note: You must restart the servers in the same order as when they went down

17.10 Verify that Person's have been successfully restored from disk

```
gfsh>query --query="select * from /people"

Result     : true
startCount : 0
endCount   : 20
Rows       : 133

      name        | id
----------------- | ---
Macaulay Mckee    | 165
Lydia Guerrero    | 251
Giacomo Landry    | 45
Alfonso Durham    | 181
Yetta Mccullough  | 215
Caleb Foley       | 257
Baxter Valentine  | 183
Caryn Guerrero    | 221
Mariko Bradford   | 171
Joan Mason        | 55
Gail Baker        | 1
Camden Holt       | 159
Penelope Cash     | 79
Germane Pollard   | 237
Germane Morris    | 89
Bo Rosa           | 235
Germane Farmer    | 139
Victor Russo      | 73
Odette Sharp      | 227
Blossom Rosario   | 49
Daniel Russo      | 69

Press n to move to next page, q to quit and p to previous page : q
NEXT_STEP_NAME : END
```

18 Developing REST Applications

- add $GEMFIRE/tools/Extensions/gemfire-api.war to the script starting the application.

- add start-dev-rest-api=true to gemfire.properties

- add http-service-bind-address=xxx.xx.xxx.xxx to gemfire.properties (for multi NIC systems)

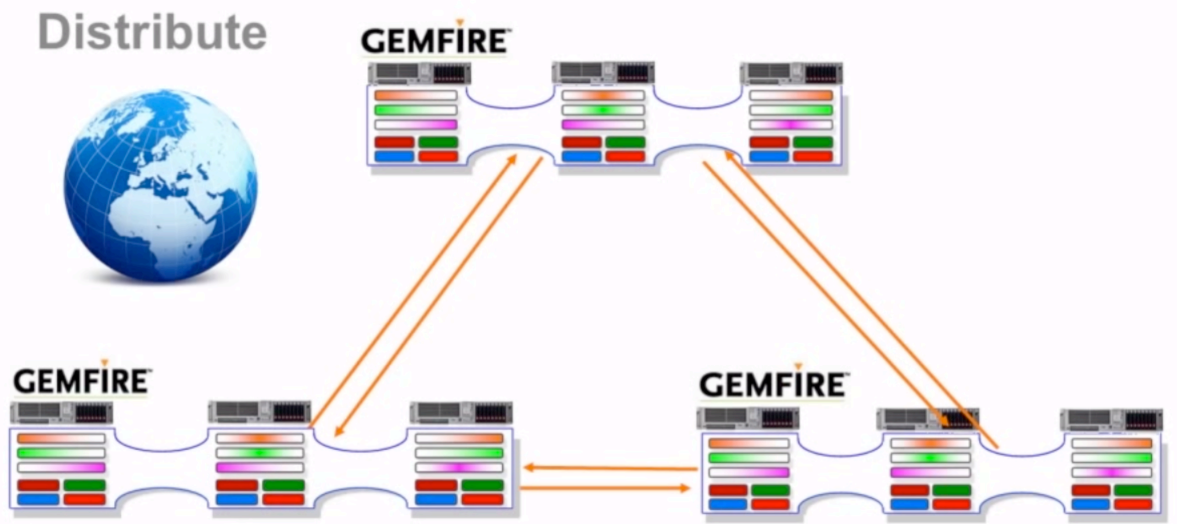
18.x Start the servers with the REST Api and JMX manager enabled.

```
gfsh>start server --name=server1 --J=-Dgemfire.start-dev-rest-api=true
\
--J=-Dgemfire.http-service-bind-address=localhost \
--J=-Dgemfire.jmx-manager=true --J=-Dgemfire.jmx-manager-start=true
```

18.x In gfsh, connect to the server running the JMX Manager.

```
gfsh>connect --jmx-manager=localhost[1099]
```

19 WAN Gateway



We will emulate bring up two different GemFire clusters and use the WAN Gateway to replicate between them. We will use an active-active topology. This means that you can write data into Data Center 1 and Data Center 2 simultaneously and they will replicate to the other data centers.

We will also enable parallel connections instead of serial connections. A serial connection uses only one connection between the data centers. One of the nodes replicates to another node in the other data centers. If you have very high throughput, think of this as pushing oatmeal through a straw. GemFire will still guarantee delivery to the other side but it will throttle the data during bursts of high activity. Parallel connections use multiple connections to replicate across the network. This allows very high throughput across the WAN. The WAN throughput scales linearly upwards when you add another node. This is a very advanced feature unique to GemFire and fully mature and tested in the world's most demanding use-cases.

Each Data Center is given a unique numeric **distributed system id**. **locator.properties** defines this id.

```
# assign number to data center 1
distributed-system-id=1
```

The remote Data Center 2 is given a distributed system id of 2:

```
# assign number to data center 2
distributed-system-id=2
```

We define a **Gateway Sender** in each data center server node. It is defined in the **cache.xml** of each server node. Here is the cache.xml **gateway-sender** for Data Center 1:

```
 <!-- Defines the persistent channel that chats and batches across the
WAN -->
 <!-- sends data when it has 50 objects or 1 second has elapsed,
whichever comes first -->
 <gateway-sender id="DC1" remote-distributed-system-id="2"
parallel="true" enable-persistence="true"
     maximum-queue-memory="50" batch-size="10" batch-time-
interval="1000"
     manual-start="false"/>
 <gateway-receiver/>
```

At this point the servers have no direct knowledge of the remote data centers. Only the locators do.

19.1 Start Locator1 and Servers in Data Center 1

```
wwilliams@Wes:~/gf91/workshop/chapter19/DataCenter1 $ . startall.sh
```

The cluster for Data Center 1 has a locator on port 10334 and servers connected to it on ports 40404 and 40405. The locator knows about Data Center 2 and will periodically poll to connect to it. The locator has this knowledge because of what we put into **locator.properties**:

```
# make the locator that there is a remote locator to which to connect
with the WAN gateway
remote-locators=localhost[10335]
```

Note that the port is 10335, the port of the *remote* locator.

19.2 Open a new terminal window. Start Locator2 and Servers in Data Center 2

```
wwilliams@Wes:~/gf91/workshop/chapter19/DataCenter2 $ . startall.sh
```

Note: if you get errors about not finding the remote locators, change the hostname to IP in the locator.properties files to make your network happy.

The cluster for Data Center 2 has a locator on port 10335 and servers connected to it on ports 40406 and 40407. The locator knows about Data Center 1 and will connect to it. Likewise with its **locator.properties**:

```
# make the locator that there is a remote locator to which to connect
with the WAN gateway
remote-locators=localhost[10334]
```

19.3 Connect to gfsh in Data Center 1

```
gfsh> connect --locator=localhost[10334]
Connecting to Locator at [host=localhost, port=10334] ..
Connecting to Manager at [host=localhost, port=1099] ..
Successfully connected to: [host=localhost, port=1099]
```

19.4 List the members in Data Center 1. Note that it has only two servers – the two that connected to the locator on port 10334.

```
Cluster-1 gfsh>list members
  Name   | Id
-------- | ----------------------------------------
server2  | localhost(server2:89184)<v2>:13760
server1  | localhost(server1:89183)<v1>:28824
locator1 | localhost(locator1:89182:locator)<v0>:32897
```

19.5 Open a new terminal window. Connect to gfsh in Data Center 2

```
gfsh>connect --locator=localhost[10335]
Connecting to Locator at [host=localhost, port=10335] ..
Connecting to Manager at [host=localhost, port=1098] ..
Successfully connected to: [host=localhost, port=1098]
```

Note that you explicitly connect to a different port than the locator in Data Center 1, which is on port 10334

19.6 List the members in Data Center 2. Note that it has only two servers. And the two servers are different that the ones in Data Center 1.

```
Cluster-1 gfsh>list members
  Name   | Id
-------- | ----------------------------------------
server3  | localhost(server3:89616)<v1>:16033
server4  | localhost(server4:89670)<v2>:20516
locator2 | localhost(locator2:89615:locator)<v0>:53684
```

19.7 Populate Data Center 1 with Data

19.7.1 In one of the non-gfsh terminal windows, populate data into the cache using the instructions from Chapter 3 "**Add Data Manually**" (see 3.1)

19.7.2 Verify the data in Data Center 1 by querying:

```
Cluster-1 gfsh>query --query="select * from /employees"

Result     : true
startCount : 0
endCount   : 20
Rows       : 13

empno | deptno |   name   | job
```

```
-----  | ------  | --------  | ---------
7369   | 20      | SMITH     | CLERK
7378   | 20      | SALLY     | MANAGER
7381   | 40      | BROWN     | SALESMAN
7374   | 10      | LUCAS     | SALESMAN
7380   | 40      | BLACK     | CLERK
7371   | 10      | WILLIAMS  | SALESMAN
7372   | 30      | LUCIA     | PRESIDENT
7377   | 20      | ADAM      | CLERK
7373   | 40      | SIENA     | CLERK
7375   | 30      | ROB       | CLERK
7370   | 10      | APPLES    | MANAGER
7376   | 20      | ADRIAN    | CLERK
7379   | 10      | FRANK     | CLERK

NEXT_STEP_NAME : END
```

19.8 Verify that data was copied to Data Center 2 by querying in the gfsh connected to Data Center 2:

```
Cluster-2 gfsh>query --query="select * from /employees"

Result     : true
startCount : 0
endCount   : 20
Rows       : 13

empno  | deptno |   name    | job
-----  | ------ | --------  | ---------
7369   | 20     | SMITH     | CLERK
7378   | 20     | SALLY     | MANAGER
7381   | 40     | BROWN     | SALESMAN
7374   | 10     | LUCAS     | SALESMAN
7380   | 40     | BLACK     | CLERK
7371   | 10     | WILLIAMS  | SALESMAN
7372   | 30     | LUCIA     | PRESIDENT
7377   | 20     | ADAM      | CLERK
7373   | 40     | SIENA     | CLERK
7375   | 30     | ROB       | CLERK
7370   | 10     | APPLES    | MANAGER
7376   | 20     | ADRIAN    | CLERK
7379   | 10     | FRANK     | CLERK

NEXT_STEP_NAME : END
```

How did the data get into Data Center 2? You never loaded it there. It was copied via the WAN Gateway. This is proven by that fact that no Department objects were copied:

19.9 Query **Data Center 1** departments region:

```
Cluster-1 gfsh>query --query="select * from /departments"

Result     : true
startCount : 0
endCount   : 20
Rows       : 4

deptno | name
------ | ----------
10     | ACCOUNTING
```

```
40      | OPERATIONS
30      | SALES
20      | RESEARCH

NEXT_STEP_NAME : END
```

Note that it has data because you loaded data here earlier

19.10 Query **Data Center 2** departments region:

```
Cluster-2 gfsh>query --query="select * from /departments"

Result     : true
startCount : 0
endCount   : 20
Rows       : 0

NEXT_STEP_NAME : END
```

Note that it has no data because the WAN Gateway was only configured for the employees region

19.11 Shutdown Data Center 1

```
wwilliams@Wes:~/gf91/workshop/chapter19/DataCenter1 $ . stopall.sh
```

19.12 Shutdown Data Center 2

```
wwilliams@Wes:~/gf91/workshop/chapter19/DataCenter2 $ . stopall.sh
```

20 Cache Events

20.1 Add a Cache Loader

A **CacheLoader** is a GemFire event callback to your Java program when a region object is requested but is not there. You can intercept this event and lazily load the value from en external source. It is synchronous. The object that you return in the load method gets put into the region and returned to the caller. To pre-populate the region with many objects, use a client program and call region.putAll(objects);

20.1.1 To implement a **CacheLoader**, you create a Java program that implements the GemFire **CacheLoader**. You then supply code for the event in which you are interested. In this example we will write a log entry to a file:

```
public class SimpleLoader<K,V> implements CacheLoader<K,V>, Declarable
{

      /*
       * The return value gets placed into the region and returned to
the caller.
       * Load your value from your persistent layer, RDBMS, HAWQ, etc
       */
       @SuppressWarnings("unchecked")
      public V load(LoaderHelper<K,V> helper) {
```

```
            String key = (String) helper.getKey();

            if
(helper.getRegion().getName().equalsIgnoreCase("people"))
                  return (V) generatePerson(Integer.parseInt(key));
            else
                  return (V) ("LoadedValue for " + key + " from " +
this.url);
      }
```

20.1.2 Tell GemFire about the loader by installing it in the region for which you want callbacks. In this case we also pass first and last names for Person generation.

```
    <region name="people">
        <region-attributes data-policy="partition">
            <partition-attributes redundant-copies="1" total-num-
buckets="11"/>
          <cache-loader>
              <class-
name>io.pivotal.eventhandlers.SimpleLoader</class-name>
              <parameter name="hawqUrl">

<string>jdbc:postgresql://172.16.6.30:5432/gpadmin</string>
              </parameter>
              <parameter name="firstNames">

<string>Tim,Prasad,Aditi,Amey,David,Joshua,John,Louie</string>
              </parameter>
              <parameter name="lastNames">
                  <string>Coventry-
Cox,Raghakrishna,Yadav,Banrse,Saunders,Schnitzel,Plotkin,Mugnano</strin
g>
              </parameter>
          </cache-loader>
        </region-attributes>
    </region>
```

20.1.3 Go to the Chapter20 directory and start the cluster

```
wwilliams@Wes:~/gf91/workshop/chapter20 $ . startall.sh
```

20.1.4 Launch gfsh and request keys from the empty **people** region. The region will populate with random first and last names.

```
gfsh> get --region=people --key=1

id | name
-- | --------------
1  | Prasad Plotkin
```

```
gfsh> get --region=people --key=2

id | name
-- | --------------
2  | Prasad Coventry-Cox
```

```
gfsh> get --region=people --key=3
```

```
id | name
-- | --------------
3  | Louie Schnitzel
```

```
gfsh> get --region=people --key=4

id | name
-- | --------------
4  | John Yadav
```

```
gfsh> get --region=people --key=5

id | name
-- | --------------
5  | Louie Plotkin
```

20.1.5 Run a query to verify the region contents:

```
gfsh>query --query="select * from /people"

id | name
-- | -------------------
2  | Prasad Coventry-Cox
3  | Louie Schnitzel
1  | Prasad Plotkin
5  | Louie Plotkin
4  | John Yadav
```

20.2 Add a Cache Listener

A **CacheListener** is a GemFire event callback to your Java program when an object is put, removed or invalidated in a region. GemFire "listens" and notifies you. It is asynchronous and does not guarantee delivery of an event in case of a node failure.

20.2.1 To implement a **CacheListener**, you create a Java program that extends the GemFire **CacheListenerAdapter**. You then supply code for the event in which you are interested. In this example we will write a log entry to a file:

```
public class SimpleListener<K,V> extends CacheListenerAdapter<K,V>
implements Declarable {

      /*
       * This method is called asynchronously after an insert.
       * If the node fails while here, delivery is not guaranteed.
       * Good for dashboarding. Use this as a Spring XD source.
       * For guaranteed delivery use the AsyncCacheListener
       *
       */
        @Override
        public void afterCreate(EntryEvent<K,V> e) {
              String line = " Received afterCreate event for entry: " +
                        e.getKey() + ", " + e.getNewValue();
          writeToFile(line);
        }
```

20.2.2 Tell GemFire about the listener by installing it in the region for which you want callbacks:

```
    <region name="people">
        <region-attributes data-policy="partition">
            <partition-attributes redundant-copies="1" total-num-
buckets="11"/>
            <cache-listener>
                <class-
name>io.pivotal.eventhandlers.SimpleListener</class-name>
            </cache-listener>
        </region-attributes>
    </region>
```

20.2.3 Open a different terminal window, create an empty **listener.log** file in the **server1** directory and tail it

```
wwilliams@Wes:~/gf91/workshop/chapter20 $ touch server1/listener.log
wwilliams@Wes:~/gf91/workshop/chapter20 $ tail -f server1/listener.log
```

20.2.4 Start a streaming client to generate random Person objects and watch the "after create" events stream from the tailed file. Copy the command below:

```
wwilliams@Wes:~/gf91/workshop/chapter20 $ java -classpath
../projects/Domain/target/Domain-
1.jar:../projects/ClientApp/target/ClientApp-
1.jar:../projects/PeopleDomain/target/PeopleDomain-
1.jar:$GEMFIRE/lib/geode-dependencies.jar
io.pivotal.app.insert.BulkInsertPerson
```

20.3 Add a Cache Writer

A **CacheWriter** is very similar to your CacheListener except that it is called BEFORE committing the modifying operation. This gives you the opportunity to change the object or reject the operation altogether. Unlike the CacheListener, it is synchronous and threadsafe. In case of a node failure the operation is not committed.

20.3.1 To implement a **CacheWriter**, you create a Java program that extends the GemFire **CacheWriterAdapter**. You then supply code for the event in which you are interested. In this example we will replace an inbound name to a different name:

```
public class SimpleWriter<K,V> extends CacheWriterAdapter<K,V>
implements Declarable {

      /*
       * This method is called synchronously after an insert.
       * Abort the write by throwing a CacheWriterException
       *
       */
        @Override
        public void beforeCreate(EntryEvent<K,V> event) {
            Person person = (Person) event.getNewValue();
            if (person.getName().endsWith(fromName)) {
                  person.replaceLastName(toName);
            }
        }
```

20.3.2 Tell GemFire about the writer by installing it in the region for which you want callbacks. In this case, **employees**. We pass in two parameters, fromName and toName. This **SimpleWriter** will change the fromName to toName before committing it to the region.

```
        <region-attributes data-policy="partition">
            <key-constraint>java.lang.String</key-constraint>
            <value-constraint>io.pivotal.domain.Employee</value-
constraint>
            <partition-attributes redundant-copies="1" total-num-
buckets="11"/>
           <cache-writer>
               <class-
name>io.pivotal.eventhandlers.SimpleWriter</class-name>
              <parameter name="fromName">
                  <string>WILLIAMS</string>
               </parameter>
               <parameter name="toName">
                   <string>TAYLOR</string>
               </parameter>
           </cache-writer>
        </region-attributes>
    </region>
```

20.3.3 Populate the cluster with Employee objects

```
wwilliams@Wes:~/gf91/workshop/chapter20 $ . data/populate.sh
```

20.3.4 Run a query on employees and note that employee 7371 name was changed from WILLIAMS to TAYLOR.

```
gfsh>query --query="Select * from /employees"

empno | deptno |  name  | job
----- | ------ | ------ | ---------
7369  | 20     | SMITH  | CLERK
7378  | 20     | SALLY  | MANAGER
7381  | 40     | BROWN  | SALESMAN
7374  | 10     | LUCAS  | SALESMAN
7380  | 40     | BLACK  | CLERK
7372  | 30     | LUCIA  | PRESIDENT
7377  | 20     | ADAM   | CLERK
7371  | 10     | TAYLOR | SALESMAN
7373  | 40     | SIENA  | CLERK
7375  | 30     | ROB    | CLERK
7370  | 10     | APPLES | MANAGER
7376  | 20     | ADRIAN | CLERK
7379  | 10     | FRANK  | CLERK
```

20.3.5 Stop the cluster and clean up the large files

```
wwilliams@Wes:~/gf91/workshop/chapter20 $ . stopall.sh
wwilliams@Wes:~/gf91/workshop/chapter20 $ . clean.sh
```

21 Tuning

-XX:+UseParNewGC

Use same values for –Xmx and –Xms

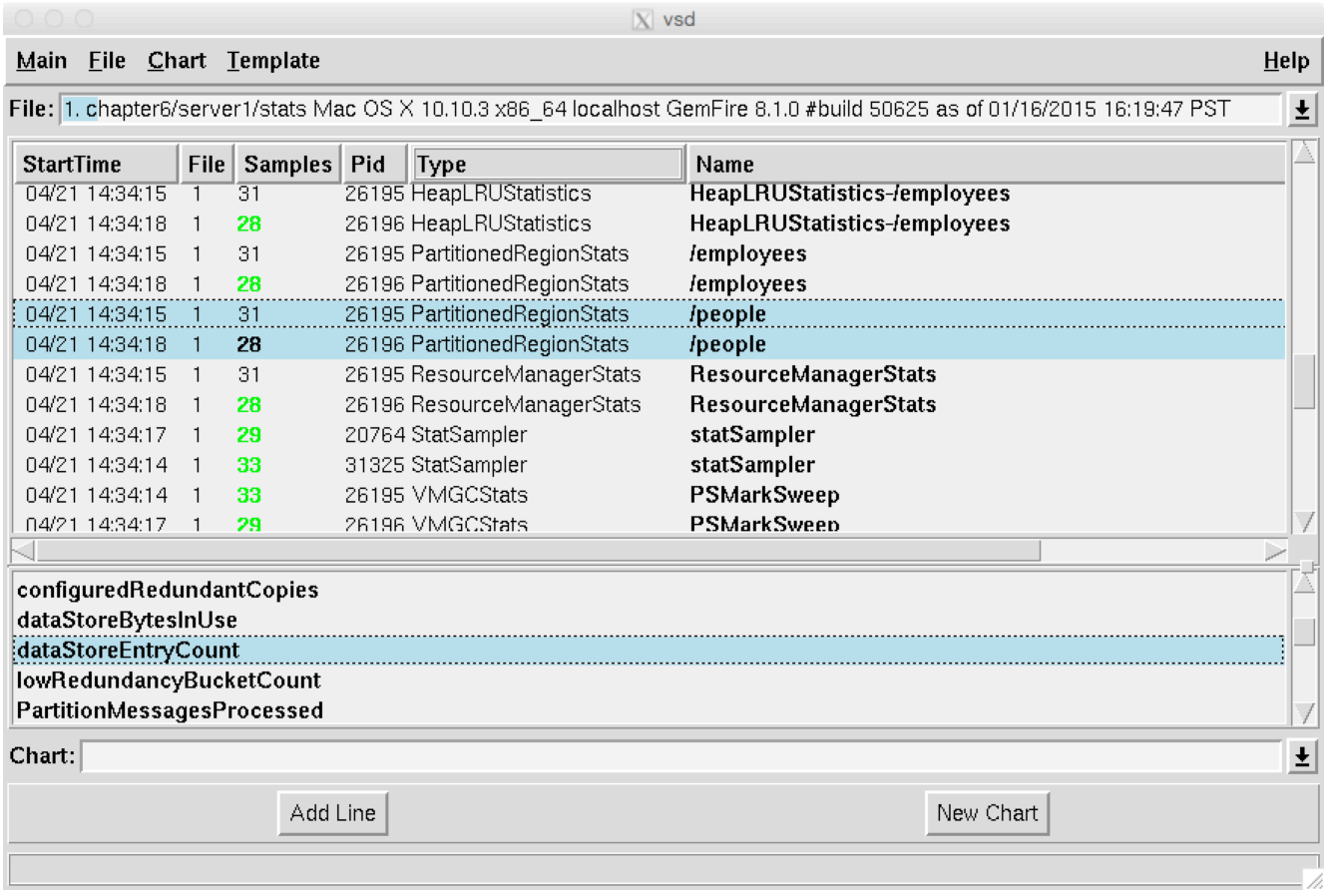
Keep –Xmn about 10-15% of –Xmx but keep –Xmn values under or equal to 4G.

In most cases the default -Xss is too large. If you are memory constrained then reduce it to something smaller.

The G1 GC has shown excellent results even though it is unsupported. Stay tuned.

22 Monitoring

22.1 How much memory is my cache consuming?



Also, Pulse “Entry Size” for the region.

More Information

The following links / guides provide detailed information on how to use Pivotal GemFire

1. GemFire 9.1 Documentation

http://GemFire.docs.pivotal.io/latest/pdf/pivotal-GemFire-ug.pdf

---

Wes Williams is a Senior Data Engineer for Data products at Pivotal, Inc.