Introduction - Fleets of Ephemeral VMs

- The **DoC private IaaS cloud** gives you the ability to:
- Create **cloud instances** (VMs to you and me).
- Based on **virtual machine templates** or **ISOs**.
- As **long-lived** or **short-lived (ephemeral)** as you like.
- This tutorial shows a way of using **ephemeral VMs**:
- Pick some parallel program - we've chosen **ElasticSearch**.
- Create a **Gold** VM with ElasticSearch installed, set to run and discover other ElasticSearch nodes automatically.
- Create a **template** from the Gold VM.
- Create a **fleet** of identically configured VMs by cloning the ElasticSearch template.
- Do all this **purely to run an experiment**.
- Then **destroy** the fleet when we're done! But keep the template.

Our Example: Apache ElasticSearch

ElasticSearch (ES) is a document storage system that:

- Stores one or more **document collections** called **indexes**; think **databases**.
- Each index contains one or more **types**; think **tables**.
- Each type contains **many JSON documents** of **similar structure**; think hierarchical JSON **records**.
- **Automatically infers** the data types - with optional guidance.
- **Automatically indexes** every field in a variety of ways - eg. every word in a plain text string.
- **Elastic:** automatically stores **replicas** of each document on different ES nodes for resiliency.. and **spreads the documents out** over any number of ES nodes for scalability.
- Allows powerful ad-hoc queries.. and **search performance** apparently **scales linearly** as nodes are added.

Using ElasticSearch

- Work with ES via a **JSON-based RESTful API**.
- The API allows you to **create indexes** and **types**.
- then **add, update and delete** any number of JSON documents (assuming the nodes have space to store and index them all).
- **intelligently search what it's indexed**.
- Optionally **aggregate the results** into a frequency distribution (like **group by** in SQL terms).
- Because ES infers the **type** of each field heuristically.. it can **index** more intelligently:
- For example, suppose your documents have a "postdate" field, and ES infers that the values are dates and times.. you can search for documents posted within a **specific range of date-times**.

ElasticSearch: in the DoC Cloud?

- ES can run on a **single machine**.
- or a **cluster** of machines on the same local network, with the same **ES cluster name**.
- Nodes discover each other automatically by **network broadcasts**.. elect a master.. and cope with nodes disappearing.

To create an ES cluster via the DoC cloud:

- Create a **customized VM** called the **ES Gold VM**.. "Gold" means the perfect, hand-crafted VM from which we clone.
- Install **Java and ES** on it.
- Set a particular **ES cluster name**.
- configure ES **to run on boot**.
- Then make a **template** from the Gold VM.

- Then we **clone many VMs** from that template.
- If we got it right.. on startup they'll discover each other and **form a cluster**.
- Then we throw any number of JSON documents at **any of the VMs** via the API.. (or in parallel **at all of the VMs**).
- ES automatically stores, replicates and indexes the JSON documents for us.
- Then we can search in a variety of ways.
- Then add extra documents, or delete or update existing documents, then search again, as often as we like.
- Sounds complex - days of careful work and programming?
- No! We can do this experiment - from scratch - in *an hour*.
- Are you ready? Start your stopwatch and let's go then!

Create the ES Gold VM

- Log into **cloudstack.doc.ic.ac.uk/client**..
- Use your college username and password, and use "imperial" as the domain.
- Create a Cloud Instance (VM) called "ElasticGold":
  - Select **Instances** and then **Add Instance**.
  - Select **From Template**, then **Ubuntu 12.04 (non CSG) 64-bit**.
  - Note down the username guest/password combination.
  - Choose **Local storage, 1GB RAM, 1Ghz**.
  - Choose **No data disk**.
  - Set the name to "ElasticGold", and the group to "ElasticSearch".
  - Now **Launch VM**.
  - Determine the VM's IP address - click on **NICs**. Suppose it's 146.169.44.200.
  - Start the **VM Console**.
  - Login as guest.

Configure the Gold VM: Setting up Java and ES

- Set the password - via `passwd guest` - to something more relevant to this application, eg BouncyCastle.
- From your desktop machine, ssh into the VM and **check that the new password works**:

```
ssh guest@146.169.44.200
```

- Become **root in the ssh session**, then do the rest of this setup in that ssh session:

```
sudo bash
```

- Install **a JDK**, plus iftop:

```
apt-get update
apt-get dist-upgrade
# Reboot may be required after the above
apt-get install python-software-properties iftop
apt-get install openjdk-7-jdk
```

- Then **freshen up** all the packages:

```
apt-get autoremove
apt-get upgrade
```


Configure the Gold VM: continued

- Install the **ES Debian package**:

```
wget -O - http://packages.elasticsearch.org/GPG-KEY-elasticsearch | apt-key add -
echo 'deb http://packages.elasticsearch.org/elasticsearch/1.0/debian stable main' >
    /etc/apt/sources.list.d/elasticsearch.list
apt-get update
apt-get install elasticsearch
```

- Make ES start on boot:

```
update-rc.d elasticsearch defaults 95 10
```

- Set the cluster name to something like "BouncyBunny" by:

```
echo "cluster.name: BouncyBunny" >> /etc/elasticsearch/elasticsearch.yml
```

- Finally, tidy things up and halt the machine:

```
/etc/init.d/elasticsearch stop
rm -rf /var/lib/elasticsearch/*
rm -rf /var/log/elasticsearch/*
sync
shutdown -h now
```

- In the GUI, select **Stop Instance**; wait for it to finish.
- All the above should have taken approx 10 minutes to do.

Create a template, clone from it

- Select **View Volumes**, select the **root disk**, then **Create Template**.
- Name the template "elasticsearchnode" and set the description to:

  "BouncyBunny elastic search node, user guest, password BouncyCastle".
- Wait for the template to be created - this might take 2 minutes.
- Clone a new VM from your "elasticsearchnode" template:
  - Select **Instances** and then **Add Instance**.
  - Select **From Template**, then **My Templates**, then **elasticsearchnode**.
  - Choose **Local storage, 1GB RAM, 1Ghz**.
  - Choose **No data disk**.
  - Set the name to "elastic1", and the group to the "ElasticSearch" group.
  - **Launch VM**, wait for it to start.
  - Determine the IP address - click on **NICs**. eg 146.169.44.201.

Check the ES Node

- From your machine, ssh into your new Cloned VM:

```
ssh guest@146.169.44.201
```

- Carry on using your ssh session.
- Check that ES is running:

```
ps auxww|grep -i elastic
tail /var/log/elasticsearch/BouncyBunny.log
```

- If it isn't, you'll need to check the logs and carefully check that every single change you tried to make in the Gold VM is reflected here in the first node VM.
- When you find a difference, destroy the VM and the template, alter the Gold VM, take the template again, and try creating your first node again.
- If you got it right first time, well done!
- This probably took about 20 minutes. Iteration and debugging takes extra time, of course.

Test ES on a single node

- What can we do with a one-node ES system?
- As a basic test, use Curl to insert our first JSON document (type this all on one line):

```
curl -XPUT http://146.169.44.201:9200/twitter/tweets/1
  -d '{"username": "dunc", "message": "this is a tweet",
       "postdate": "20140225T11:55:00"}'
```

  Note that:
  - The "-d" argument is the JSON document to store.
  - "twitter" is the name of our first index, created implicitly.
  - "tweets" is the name of our first type, created implicitly.
  - "1" is the internal document id (within "twitter/tweets") of the document we want to store.
- the response is:

```
{"_index":"twitter","_type":"tweets","_id":"1","_version":1,"created":true}
```

- Insert a second fake tweet as document 2:

```
curl -XPUT http://146.169.44.201:9200/twitter/tweets/2
   -d '{"username": "dunc", "message": "this is another tweet",
        "postdate": "20140226T12:55:00"}'
```


More Tests

- Retrieve our first document via:

```
curl -XGET http://146.169.44.201:9200/twitter/tweets/1
```

- the response is:

```
{"_index":"twitter","_type":"tweets","_id":"1","_version":2,"found":true,
 "_source" : {"username": "dunc", "message": "this is a tweet",
 "postdate": "20140225T11:55:00"}}
```

  Note that it returns the original JSON document plus various bits of meta-data.
- Perform a simple search for tweets by "dunc":

```
curl -XGET http://146.169.44.201:9200/twitter/tweets/_search\?q=username:dunc
```

- The (very long) results include:

```
{ ... "hits":[
  {"_index":"twitter","_type":"tweets","_id":"1","_score":0.5945348,
   "_source" : {"username": "dunc", "message": "this is a tweet",
                "postdate": "20140225T11:55:00"}
  },
  {"_index":"twitter","_type":"tweets","_id":"-JifysE9QGO5ngODwUnOAw", ...
   "_source" : {"username": "dunc", "message": "this is another tweet",
                "postdate": "20140226T12:55:00"}
  }
 ]
}
```

ES on multiple nodes

- Now, return to the CloudStack Web UI, and create a second VM from the ElasticSearch template.
- Check both nodes' log files and verify that they've formed a cluster. It's hard to see them replicating data because they're so quick, and our data set is so tiny.
- Rerun your search unchanged. Same results should appear.
- Determine the second node's IP address, and rerun your search query using the second node's IP address. Both nodes are up, have copies of all documents, and can simultaneously permit inserts, updates, deletions and queries.
- Make up more fake tweets and insert them. Write a program to read a file containing tweets (or make up random tweets) and invoke Curl once per tweet to insert it.
- Create a third and fourth VM from the ElasticSearch template. Rerun some tests. Then destroy one of the VMs and verify that no data has been lost.

New Example: Words, Anagrams and Anagram Sets

- Let's switch example: Consider the problem of finding **sets of words** that are anagrams of one another (like "dog" and "god").
- Specifically, our task is to find **the biggest set of anagram words** in a given wordlist, and which words form that set (or sets of the same maximum size)?
- Choose an ES index and type name: **words/anagrams**.
- Decide what JSON structure we want to store:
  - A word, and
  - That word's **signature**.
- The signature is the bag of letters contained in the word, sorted into character order. (eg. **sig('dog')='dgo'**). Words that are anagrams of one another have the **same signature**.
- In JSON format, a document describing a single word and it's signature is: `{ "word": "dog", "sig": "dgo" }`

Anagram Sets continued

- Getting bored with running Curl myself, I wrote some Perl scripts (and a module) to simplify things. Fetch them by:

```
git clone git@gitlab.doc.ic.ac.uk:dcw/elasticsearch_anagrams.git
```

  and look around.
- Edit the **Defns.pm** module and change the IP address of an ES node, contained in the `$elastic` definition near the top.
- Decide on a wordlist (eg `/usr/share/dict/words`), and insert all the words via:

```
./insert-words-and-sigs /usr/share/dict/words
```

- This takes approximately 20 minutes to complete. After this, search for words with signature "dgo":

```
curl -XGET http://146.169.44.201:9200/words/anagrams/_search\?q=sig:dgo
```


Anagram Sets continued

- Use **findanagram** to find all anagrams of a given word, this is a wrapper around the search logic:

```
./findanagram dog
./findanagram last
```

- The results of the latter are:

```
last: last salt slat lats
```

- Use **findanagrams** to find all anagrams of all words:

```
./findanagrams > /tmp/anagram_sets
```

- Use **findbiggestanagramsets** to find the biggest anagram set(s) and display their members:

```
./findbiggestanagramsets
```

- To speed up the very slow insert script, use a **bulk insert** API variant, used in the Perl script:

```
./bulkinsert /usr/share/dict/words
```

- Now it finishes in a few seconds!

Summary

What have we achived? we've:

- Created fleets of ephemeral VMs to perform experiments.
- Created a hand crafted Gold VM with your desired software and configuration on it.
- Shut the Gold VM down cleanly and made a template from it.
- Cloned a VM from the template.
- Tested that it works.
- Cloned more VMs from the template.
- Ran your experiments; gathered your results.
- Destroyed all the cloned VMs.
- Note: keep the Gold VM or template to ensure reproducibility.

Read `www.doc.ic.ac.uk/csg/services/cloud` for much more information about the DoC private cloud.