Using Google cloud services for bio-informatics
Berry Kriesels
Google and genomics?...

• It makes sense
  • Specialised in large volumes of data
  • Infrastructure in place
  • Know how to make ‘complex’ tasks “easy”
  • Serves high demand clients (Netflix...)
Today...

• Omnigen and Google
• Why should you bother?
• Case examples
• The future
• Your own implementation?
Omnigen and google
Google genomics in Europe I

- Google genomics team
  - Largely based in the US
  - Needed localised experts

- Omnigen
  - Familiar with the biological challenges involved
  - Needed a scalable solution
Google genomics in Europe II

- Collaboration
  - Hey! Lets team up!
  - Bi-weekly tech meetings
  - Knowledge crossover
- Implementations
  - Omnigen implements solutions
  - Google offers backend support
Why should you bother?
A few (familiar?) examples

• Bio-informatician life...
  • Should be doing research, projects and analysis
    o More questions than answers
    o Time is limited
  • Should not... be doing sysadmin tasks
    o Keep track of disks and servers
    o Hey could you help me with this IT problem (helpdesk...)
  • I will go for a coffee while my scripts are running
    o Coffee is good!
    o Being confronted with an error after running for 36 hours is bad!
A few (familiar?) examples

- Infrastructure
  - Yearly struggle for enough disk space or cpu power
  - Huge amounts of budget (over)allocated for hardware, bandwidth, disk space
- Results in
  - Over budgeting infrastructure just in case...
  - Use of academic budget not for academic challenges...
Getting things right

- Time consuming
  - Bio-informaticians doing a IT job
- Who is responsible?
  - Phd student, bio-informaticians?
- Making it easy is hard!
  - Results in messy solutions and legacy issues
- We just want it to work for us
  - In the end scientists just want results
Possible solution

- Google genomics
  - Public workflows
  - Active academic community
- Google cloud services
  - Easy to use
  - Scalable
  - You only pay for what you use
  - No more IT failures
Case examples
Using public datasets

- 1000 genomes data
  - Directly accessible using BigQuery in Google console
    - No setting up
    - No downloading data
Using public datasets

```sql
# Compute the Ti/Tv ratio for variants within genomic region windows.
SELECT
  reference_name,
  window * 100000 AS window_start,
  transitions,
  transversions,
  transitions/transversions AS titv,
  num variants_in_window,
FROM (
  SELECT
    reference_name, window,
    SUM(mutation IN ('A->G', 'G->A', 'C->T', 'T->C')) AS transitions,
    COUNT(mutation) AS num variants_in_window
  FROM [genomics-public-data:platinum_genomes.variants]
  WHERE
```

<table>
<thead>
<tr>
<th>Row</th>
<th>reference_name</th>
<th>window_start</th>
<th>transitions</th>
<th>transversions</th>
<th>titv</th>
<th>num variants_in_window</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>chr1</td>
<td>0</td>
<td>293</td>
<td>198</td>
<td>1.4797979797979798</td>
<td>491</td>
</tr>
<tr>
<td>2</td>
<td>chr1</td>
<td>100000</td>
<td>147</td>
<td>76</td>
<td>1.9342105263157894</td>
<td>223</td>
</tr>
<tr>
<td>3</td>
<td>chr1</td>
<td>200000</td>
<td>64</td>
<td>61</td>
<td>1.0491803278688525</td>
<td>125</td>
</tr>
<tr>
<td>4</td>
<td>chr1</td>
<td>300000</td>
<td>2</td>
<td>11</td>
<td>0.18181818181818182</td>
<td>13</td>
</tr>
<tr>
<td>5</td>
<td>chr1</td>
<td>400000</td>
<td>9</td>
<td>6</td>
<td>0.12415177288235294</td>
<td>12</td>
</tr>
</tbody>
</table>
Using public datasets

```
# Count the homozygous and heterozygous variants for each sample across the
# entirety of the 1,000 Genomes dataset.
SELECT
    sample_id,
    SUM(IF(0 = first_allele AND 0 = second_allele, 1, 0)) AS hom_RA_count,
    SUM(IF(first_allele = second_allele AND first_allele > 0, 1, 0)) AS hom_AA_count,
    SUM(IF((first_allele != second_allele OR second_allele IS NULL)
          AND (first_allele > 0 OR second_allele > 0), 1, 0)) AS het_RA_count
FROM
    [genomics-public-data:1000_genomes.variants]
WHERE
    reference_name != 'Y' AND reference_name != 'N'
GROUP BY
    sample_id
ORDER BY
    sample_id
```

Query complete (9.6s elapsed, 1.0 TB processed)

<table>
<thead>
<tr>
<th>Row</th>
<th>sample_id</th>
<th>hom_RA_count</th>
<th>hom_AA_count</th>
<th>het_RA_count</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>HG00096</td>
<td>34506994</td>
<td>1475804</td>
<td>2365816</td>
</tr>
<tr>
<td>2</td>
<td>HG00097</td>
<td>35857671</td>
<td>1448123</td>
<td>2400962</td>
</tr>
</tbody>
</table>
Using public datasets

annotateVariants.R

```r
require(GoogleGenomics)
require(VariantAnnotation)
require(BSgenome.Hsapiens.UCSC.hg19)
require(TxDb.Hsapiens.UCSC.hg19.knownGene)

variants <- getVariants(datasetId="10473108253681171589", chromosome="17",
start="41196311", end="41277499")
granges <- variantsToGRanges(variants)

txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
codingVariants <- locateVariants(ranges, txdb, CodingVariants()

coding <- predictCoding(rep("granges", elementlengths(grangesSALT)), txdb,
seqSource="Hsapiens", varAllele=unlist(grangesSALT, use.names=FALSE))
```

16:1 (Top Level)
I have my own data and pipeline

• Use a Google VM engine
  • Directly scalable
  • Easy to set up via
    o Commandline
    o Google web console
  • Realtime add / remove
    o Diskspace etc
Launch with Google Cloud SDK

gcloud compute instances create my-instance --zone eu-central1-d

Connect seconds later

gcloud compute ssh my-instance --zone eu-central1-d

Copy a file

gcloud compute copy-files ~/files* my-instance:~/remote-destination --zone eu-central1-d
Compute engine VMs

- Consistent high performance
- Start up in seconds
- All data encrypted at rest and in flight
- Pay only for the minutes you use

<table>
<thead>
<tr>
<th>Instance type</th>
<th>Virtual Cores</th>
<th>Memory</th>
<th>GCEU</th>
</tr>
</thead>
<tbody>
<tr>
<td>n1-standard-1</td>
<td>1</td>
<td>3.75GB</td>
<td>2.75</td>
</tr>
<tr>
<td>n1-standard-2</td>
<td>2</td>
<td>7.5GB</td>
<td>5.50</td>
</tr>
<tr>
<td>n1-standard-4</td>
<td>4</td>
<td>15GB</td>
<td>11</td>
</tr>
<tr>
<td>n1-standard-8</td>
<td>8</td>
<td>30GB</td>
<td>22</td>
</tr>
<tr>
<td>n1-standard-16</td>
<td>16</td>
<td>60GB</td>
<td>44</td>
</tr>
<tr>
<td>n1-standard-32Beta</td>
<td>32</td>
<td>120GB</td>
<td>88</td>
</tr>
</tbody>
</table>

High Memory
Machines for tasks that require more memory relative to virtual cores

<table>
<thead>
<tr>
<th>Instance type</th>
<th>Virtual Cores</th>
<th>Memory</th>
<th>GCEU</th>
</tr>
</thead>
<tbody>
<tr>
<td>n1-highmem-2</td>
<td>2</td>
<td>13GB</td>
<td>5.50</td>
</tr>
<tr>
<td>n1-highmem-4</td>
<td>4</td>
<td>26GB</td>
<td>11</td>
</tr>
<tr>
<td>n1-highmem-8</td>
<td>8</td>
<td>52GB</td>
<td>22</td>
</tr>
<tr>
<td>n1-highmem-16</td>
<td>16</td>
<td>104GB</td>
<td>44</td>
</tr>
<tr>
<td>n1-highmem-32Beta</td>
<td>32</td>
<td>208GB</td>
<td>88</td>
</tr>
</tbody>
</table>

High CPU
Machines for tasks that require more virtual cores relative to memory

<table>
<thead>
<tr>
<th>Instance type</th>
<th>Virtual Cores</th>
<th>Memory</th>
<th>GCEU</th>
</tr>
</thead>
<tbody>
<tr>
<td>n1-highcpu-2</td>
<td>2</td>
<td>1.80GB</td>
<td>5.50</td>
</tr>
<tr>
<td>n1-highcpu-4</td>
<td>4</td>
<td>3.60GB</td>
<td>11</td>
</tr>
<tr>
<td>n1-highcpu-8</td>
<td>8</td>
<td>7.20GB</td>
<td>22</td>
</tr>
<tr>
<td>n1-highcpu-16</td>
<td>16</td>
<td>14.40GB</td>
<td>44</td>
</tr>
<tr>
<td>n1-highcpu-32Beta</td>
<td>32</td>
<td>28.80GB</td>
<td>88</td>
</tr>
</tbody>
</table>

Shared Core
Machines for tasks that don't require a lot of resources but do require more compute or memory than a Standard instance

<table>
<thead>
<tr>
<th>Instance type</th>
<th>Virtual Cores</th>
<th>Memory</th>
<th>GCEU</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1-micro</td>
<td>1</td>
<td>0.60GB</td>
<td>Shared CPU</td>
</tr>
<tr>
<td>g1-small</td>
<td>1</td>
<td>1.70GB</td>
<td>1.38</td>
</tr>
</tbody>
</table>
Familiar with Docker containers?

- Like virtual machines but better!
  - No more dependency issues
  - Small container (images) possible (200mb)
  - Use them as microservices
Google Supports Docker and Cluster-Based Computing

- Open-source container orchestration
- Broad industry support
- Based on Google’s internal experience with containers

- Fully managed and hosted Kubernetes
- Create and deploy clusters with one-click
# Container registry

## Container images

Images hosted in Google Container Registry Learn more

<table>
<thead>
<tr>
<th>Name</th>
<th>Location</th>
<th>Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>app-lamp</td>
<td>gcr.io</td>
<td>latest</td>
</tr>
<tr>
<td>django-python</td>
<td>eu.gcr.io</td>
<td>latest, v1, v2, v3, v4, 2 more</td>
</tr>
<tr>
<td>galaxy-bioblend</td>
<td>eu.gcr.io</td>
<td>latest</td>
</tr>
<tr>
<td>galaxy-datavolume</td>
<td>eu.gcr.io</td>
<td>latest</td>
</tr>
<tr>
<td>galaxy-emc</td>
<td>eu.gcr.io</td>
<td>latest</td>
</tr>
<tr>
<td>galaxy-emc-vn</td>
<td>eu.gcr.io</td>
<td>latest</td>
</tr>
<tr>
<td>git-sync</td>
<td>eu.gcr.io</td>
<td>latest, v1, v2</td>
</tr>
<tr>
<td>git-sync-ssh</td>
<td>eu.gcr.io</td>
<td>latest</td>
</tr>
</tbody>
</table>
Starting a service at Omnigen...

Knowledgebase
This system is used to run the web-based knowledgebase for Omnigen genomics data.

Layout of the system:
- Django webfront
- Django loadbalancing service
- MySQL database
- MySQL internal service

Starting this system will create two ReplicationControllers (RC) and two Services, each RC will create and manage a single pod which contains the required containers and links to existing storage volumes (Google Permanent Storage disks).

The two services each perform a different task:
- The MySQL service listens to the MySQL pod on the 3306 port (default MySQL port) and allows all other pods in the cluster to connect to that specific pod and MySQL database by using the service.
- The Django load balancer service is an external facing service which creates an external IP-address and links the Django pod to this IP-address. A load balancer service also scales itself up and down according to the amount of hits the IP gets.

How to start the entire system:

```
# This will start everything up
kubectl create -f knowledgebase
# Yes indeed, just the directory-name, nothing else.
```

The underlying idea of this setup is that the services are only started once, and will only link with pods that have certain labels. The ReplicationControllers will create one of these pods from a blueprint and will attempt to always keep one pod running. Updates to the pod structure (running containers, configurations) have to be implemented in the `*.yaml` files.

As long as the labels of the pods do not change the services will link to new pods without issue.
How?

Container in private repo

*.yml file
Up in ~12 seconds!

Checkout latest code
Keep track of running VMs and container clusters
The future...
Imagine..

- For each separate analysis
  - Start a VM by using a pre made *.yml
    - Get that well deserved cup of coffee
  - Custom Docker container is pulled from repo
  - Load your latest code from code repo
  - Crunch the numbers (specify as much CPU / RAM as you want)
  - Data gets stored (automated snapshots / backups)
  - Automatically spin down the machine
  - Focus on research instead of infrastructure!
I am convinced!

• I would like a solution too!
  • Omnigen can help implement your solution together with Google engineers
    • On site demo and use cases
  • Marconistraat 16, 3029AK Rotterdam
• Email: info@omnigen.nl
• www: www.omnigen.nl