Criando um cluster na AWS EMR para rodar Spark

Preliminares

- Você vai precisar de um key-pair, se não tem faça um. Guarde bem a chave privada.
- Crie um bucket no S3, e dentro dele um diretório logs
- Crie um arquivo *cluster_config.json* com o seguinte conteúdo:

```
[
   {
      "Classification": "spark",
      "Properties": {
         "maximizeResourceAllocation": "true"
      }
   },
   {
      "Classification": "spark-env",
      "Configurations": [
         {
            "Classification": "export",
            "Properties": {
               "PYSPARK_PYTHON": "/usr/bin/python3"
            }
         }
      ]
   }
]
```

Esta configuração será necessária para permitir que o Spark utilize todos os nós do cluster, e use Python 3. Caso contrário será aplicado um limite bastante conservador no número máximo de nós utilizado. Saiba mais aqui: http://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-spark-configure.html

- O AWS EMR nos permitirá subir um cluster padrão. Algumas bibliotecas do Python estarão faltando, e devem ser instaladas na fase de *bootstrap* (mais sobre isso depois). Crie um arquivo *instalar_dependências.sh* com o seguinte conteúdo:

```
#!/bin/bash
sudo -H python3 -m pip install --upgrade pip
sudo -H python3 -m pip install --upgrade warcio
sudo -H python3 -m pip install --upgrade boto3
```

Você pode incluir qualquer atividade de inicialização aqui. Este arquivo será executado em todos os nós do cluster. Aqui eu resolvi instalar algumas bibliotecas do Python que serão úteis para trabalhar com os arquivos do Common Crawl.

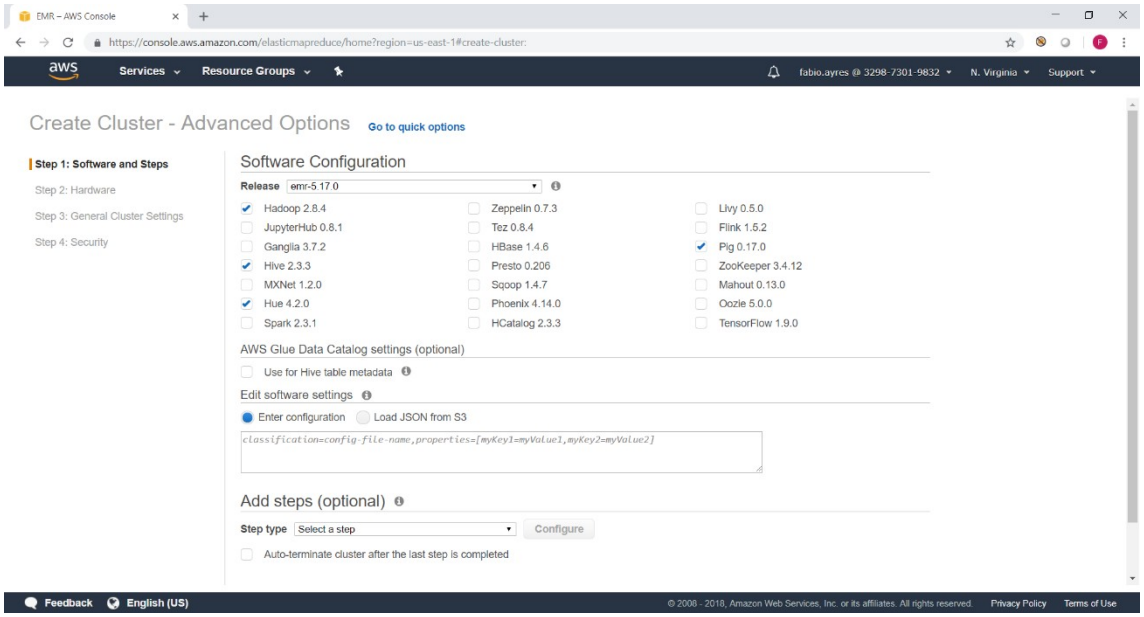
- Coloque os arquivos *cluster_config.json* e *instalar_dependencias.sh* no seu bucket. Pode ser na raiz mesmo.

Criando o cluster

- Entre no console web da AWS
- Entre no serviço EMR

- Services -> Analytics/EMR
- Create Cluster -> Advanced options

Você deve ter chegado na seguinte tela:



Step 1: Software and Steps

Software Configuration: mantenha a versão mais recente do emr. Marque Spark, Zeppelin, Hadoop, Ganglia, e remova todos os outros. Dica: procure saber sobre os outros também, entrevistadores andam perguntando sobre isso.

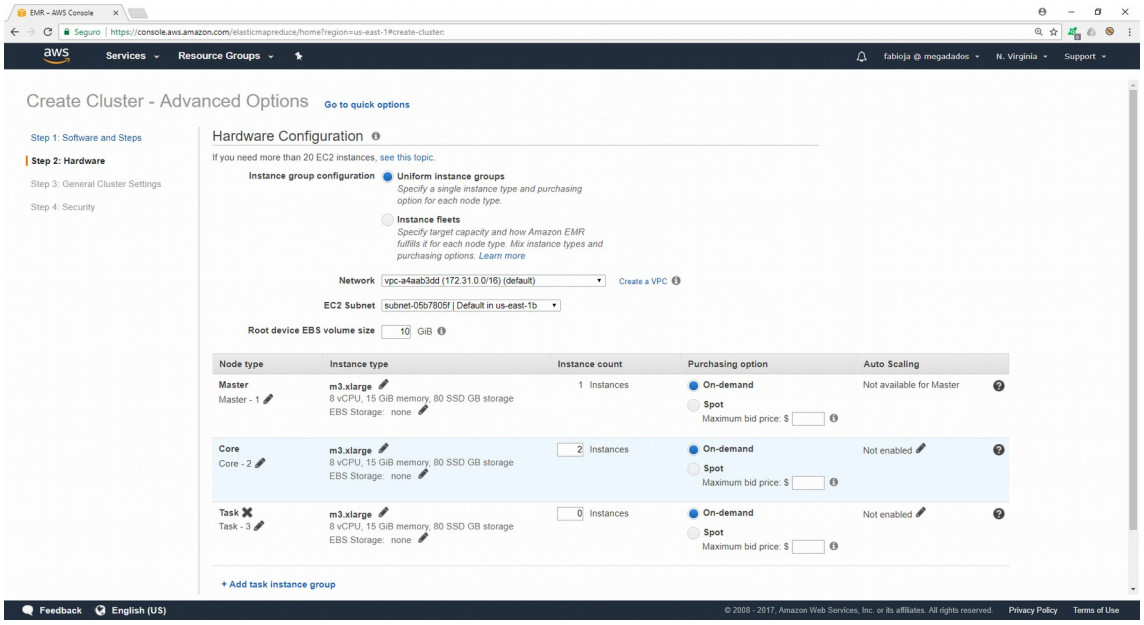
Edit software settings: Escolha “Load JSON from S3 e selecione seu *config_cluster.json* .

Add steps: Por enquanto não vamos mexer aqui, pois vamos usar o Spark interativamente via Zeppelin. Mas em breve vamos experimentar com esta opção. É aqui que você pode especificar um arquivo Python a ser executado stand-alone, e também pode indicar que o cluster deve ser terminado quando o arquivo estiver finalizado. Isso é o jeito correto de rodar *scripts batch* com Spark.

Next

Step 2: Hardware

Chegamos na seguinte tela:



Aqui podemos configurar nosso hardware. As principais decisões arquiteturais neste momento são:

- Que máquinas usar?
- O que são “Master”, “Core” e “Task”? Quantas máquinas alocar para cada grupo?
- O que são “on-demand” e “spot”?

Para isso vamos entender o que cada grupo significa. Como vimos anteriormente, um cluster Spark tem um programa máster, que coordena a atividade dos programas executores. Spark é um sistema resiliente: se um executor cai por qualquer razão o sistema inicial um novo executor, que receberá as tarefas do executor que terminou indevidamente. O nó “Master” vai rodar o programa máster, e os nós “Core” e “Task” vão rodar executores.

A diferença entre nós Core e Task é o tipo de instancia que iremos usar em cada grupo:

- Para os nós Core usaremos instâncias “on-demand”, que custam caro mas que não podem ser derrubadas ao gosto da Amazon.
- Para os nós Task usaremos instâncias spot.

As instâncias spot custam barato, dependendo da situação do cluster. De fato, o preço das instâncias spot é decidido em um processo de leilão, onde você determina seu lance. Se o seu lance for superior aos lances sendo praticados naquele momento, você recebe a máquina. Veja os preços atuais das instâncias spot pretendidas neste cluster: passe o mouse em cima do ícone ao lado da caixa de texto do seu lance. Este valor é uma dica, e mostra os valores atuais que tem sido bem-sucedidos, basta indicar o preço atual ou algo ligeiramente superior para ter a máquina.

Compare os valores spot com os preços de instâncias on-demand: https://calculator.s3.amazonaws.com/index.html

O problema com instâncias spot é que se o preço do mercado (que é flutuante) ultrapassar o seu valor de lance, a Amazon irá derrubar sua máquina (sem mais nem menos!) para outros usarem! Logo, não use instâncias spot para rodar servidores. Mas para rodar tarefas Spark (que já foi construído para tolerar máquinas caindo), tudo bem.

Agora podemos explicar o uso das instancias Core: se as spot caírem, as Core seguram o tranco (em marcha lenta, claro) até a situação se reestabelecer!

Para nosso cluster de teste não vamos precisar nem mesmo das instâncias Core, então coloque zero lá. No final, nosso "cluster" terá uma única máquina Master, que será um modelo respeitável: uma m4.xlarge.

Next

Step 3: General Cluster Settings

- Mude o diretório de logs para aquele diretório que você criou no S3
- Bootstrap actions -> Add bootstrap action -> Custom action -> Configure and Add
    - Selecione o arquivo *instalar_dependencias.sh* que você colocou no S3

Next

Step 4: Security

- Escolha o EC2 pair que você criou antes.

O EC2 pair permitirá a você logar no nó master. Por padrão a Amazon bloqueia o acesso externo a todos os nós, exceto à porta 22 (SSH) no nó master. Esse nó tem duas interfaces de rede: uma externa (que você vai usar para conectar a ele) e uma interna (para se conectar aos demais nós do cluster).

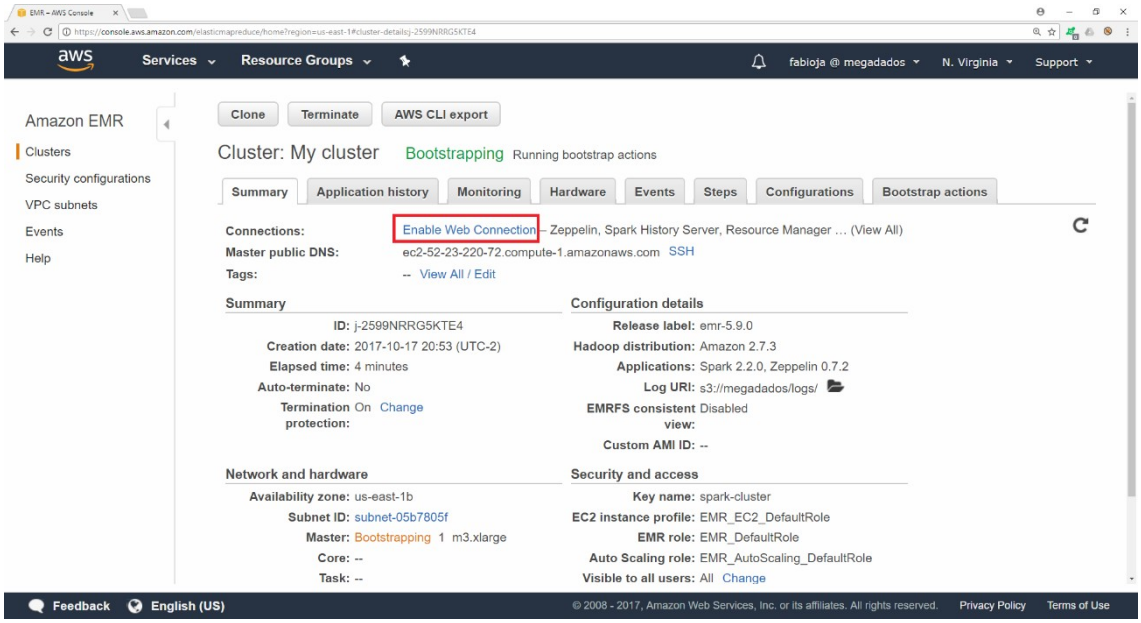
Create cluster! Agora é esperar...

Provisionando o cluster

Observe o processo de provisionamento do cluster. EMR vai criar as máquinas requeridas (apenas uma neste nosso caso), que passarão pelos seguintes estágios:

- Provisioning: a máquina está sendo reservada pela Amazon
- Bootstrapping: a máquina foi provisionada e está em processo de boot. A essa altura vocês já estão bem familiarizados com esse processo lá na disciplina de Computação em Núvem!
- Waiting: o cluster está funcionando e aguarda comandos!

Eventualmente, durante a fase de bootstrapping, vai aparecer um link "Enable Web Connection":

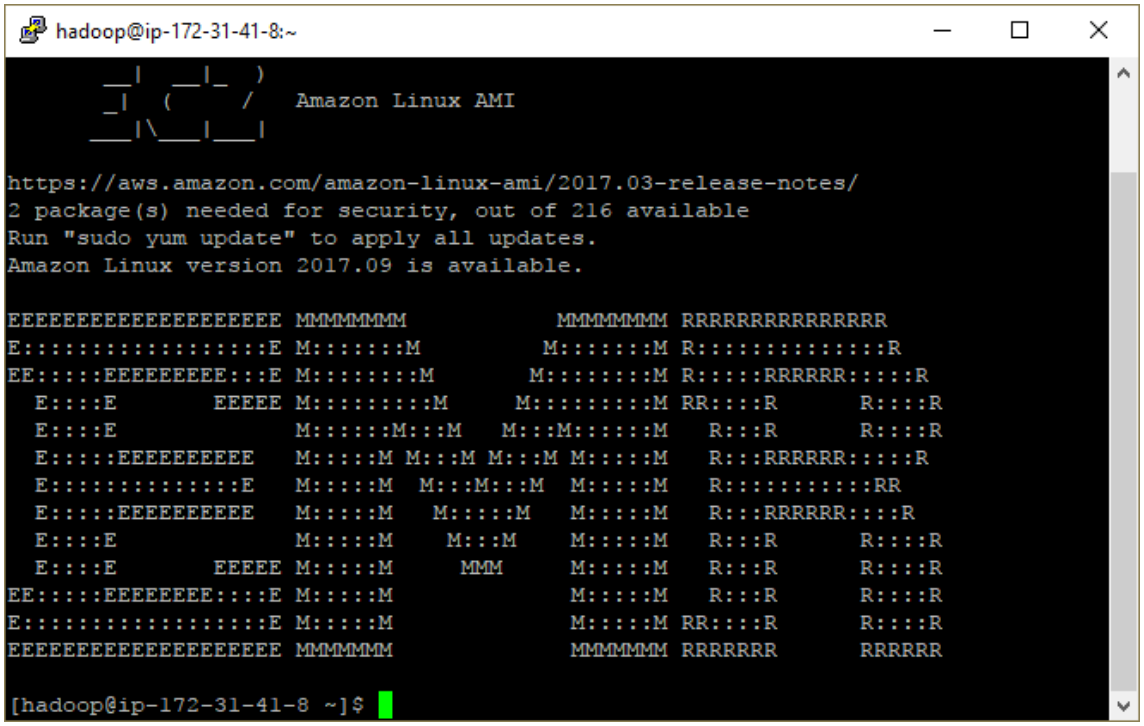


O que acontece: como dito antes, a Amazon bloqueia todos os acesso ao cluster, exceto via SSH. Como fazer para acessar outras portas do cluster, como a porta 8890 que é onde o Apache Zeppelin (o “Jupyter Notebook” do projeto Apache) estará rodando?

A solução é o tunelamento SSH (SSH tunneling). Funciona assim: vamos abrir uma conexão SSH com a máquina remota, e vamos passar todo nosso tráfego para essa máquina remota através dessa conexão. Lá na máquina remota o servidor SSH se encarregará de entregar nossos pacotes de rede para as portas desejadas.

Clique no link e instruções aparecerão. SIGA AS INSTRUÇÕES À RISCA! É muito fácil errar nesse momento! Para usuários de Windows, você tem uma missão extra: instalar Putty e Puttygen.

- Puttygen para converter a chave privada do formato original para o formato do Putty.
- Putty para efetivamente fazer a conexão.

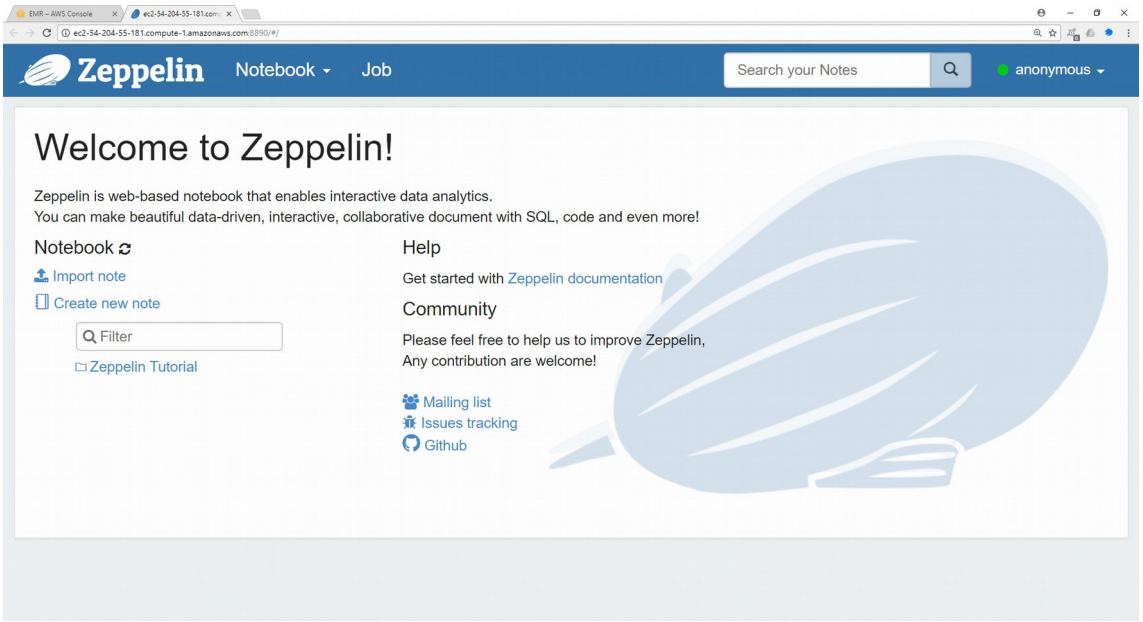


Eu tive sucesso com esse procedimento instalando o FoxyProxy Basic no Chrome. Não sei se o outro funciona bem. Se você conseguiu fazer o tunelamento com o ssh e instalou o proxy no browser, ligue o proxy.

Se você for bem sucedido, o link “Enable Web Connection” desaparecerá, e em seu lugar teremos links para as ferramentas reais.

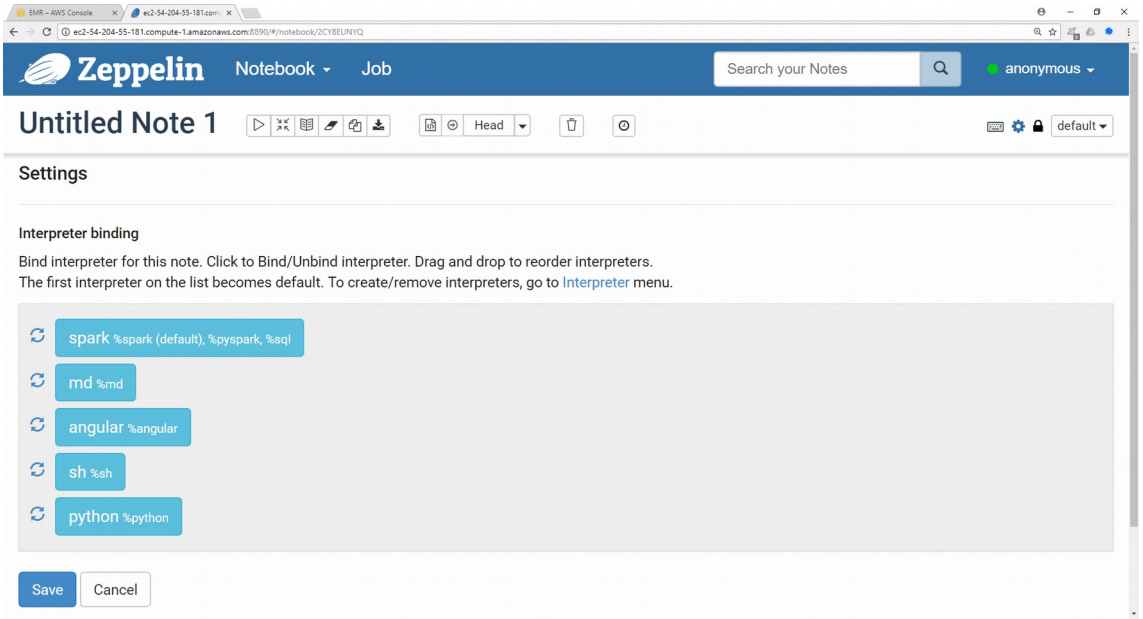
Apache Zeppelin

Clique no link para ir para o Apache Zeppelin:



Crie um novo note, com default interpreter “spark”.

O Zeppelin trabalha com várias linguagens, todas no mesmo notebook. Para isso o zeppelin possui vários interpretadores. Clique na pequena engrenagem no canto superior direito para ver alguns dos interpretadores.



Configure seu interpretados Spark para trabalhar com python3, faça o mesmo para o interpretador python.

Em seguida cancele para retornar ao notebook.

Na primeira célula, digite o seguinte código:

```
%pyspark
1 + 1
```

Rode a célula. Leva bastante tempo mesmo na primeira vez, vários processos estão sendo criados, paciência. Depois da primeira execução as coisas ficam mais rápidas.

Precisamos do magic %pyspark para informar que esta célula deve ser executada no interpretador spark, e que a linguagem a ser usada é Python. Este será o tipo padrão de célula para nós. O outro tipo costumeiro de célula será a %md, para Markdown.

O interpretador “spark” do Zeppelin é um cliente Spark. Portanto, já vem com um objeto especial: o SparkContext. Tente agora o seguinte código:

```
%pyspark
sc
```

O resultado é

```
<SparkContext master=yarn-client appName=Zeppelin>
```

O objeto sc é um SparkContext, nossa porta de entrada para o Spark!

Tente agora seu primeiro programa com Spark!

```
%pyspark
rdd = sc.parallelize(range(1000))
resultado = rdd \
   .map(lambda x: x**2) \
   .filter(lambda x: x % 2 == 0) \
   .reduce(lambda x, y: x + y)
print(resultado)
```

Note o uso de barra reversa “\” para permitir a quebra de linha em Python, pois programas em Spark tendem a ser uma única linha gigante!

Terminando o cluster

Vamos agora encerrar o cluster. O professor já subiu um cluster de tamanho suficiente para a sala toda trabalhar.

- Desligue o seu proxy no browser (“Disable Foxy Proxy”)
- Feche a conexão SSH
- Vá para o console AWS, serviço EMR, e termine o seu cluster
    - A página vai reclamar que você tem “Termination protection” ligado, basta ali mesmo desligar a proteção e terminar o cluster.
- Após o término do cluster, vá para a S3 e delete o bucket