**Kubernetes on AWS**

Connect to Putty – using the ppk key from aws.

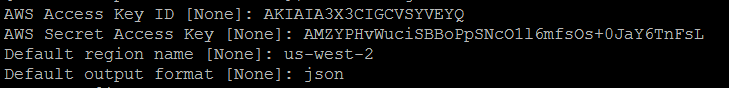
The below two commands downloads kubernetes and displays it on your putty server:

**export KUBERNETES\_PROVIDER=aws; wget -q -O - https://get.k8s.io | bash**

**AWS Configure**

Run the command:

aws configure



Check the current path using the command

Echo $PATH

Current PATH should be something like this:

PATH=/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/sbin:/opt/aws/bin:/home/ec2-user/.local/bin:/home/ec2-user/bin

Next, run the command:

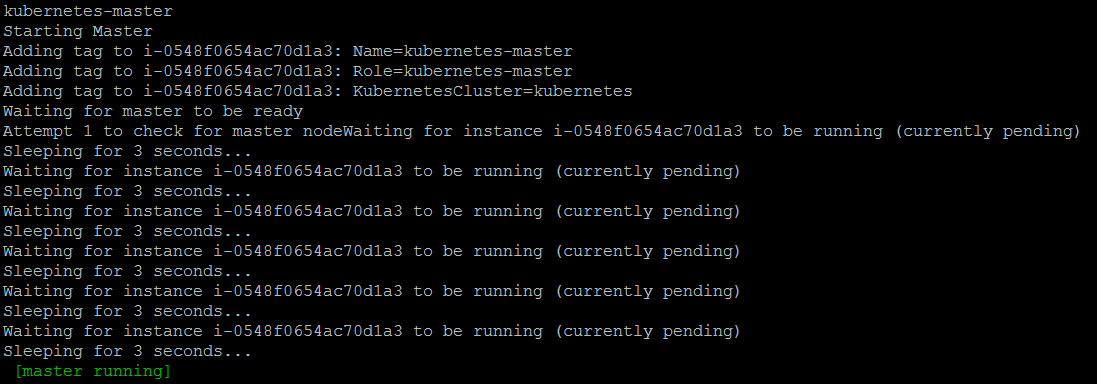
**export KUBERNETES\_PROVIDER=aws; curl -sS https://get.k8s.io | bash**

The above command takes around 5-10 minutes to get installed:

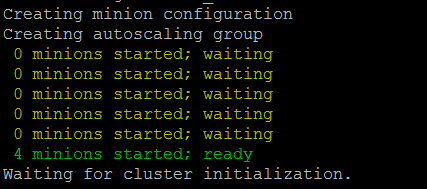
This command calls  [cluster/kube-up.sh](http://releases.k8s.io/master/cluster/kube-up.sh) which in turn calls [cluster/aws/util.sh](http://releases.k8s.io/master/cluster/aws/util.sh) using [cluster/aws/config-default.sh](http://releases.k8s.io/master/cluster/aws/config-default.sh)

The cluster will be up in some time and the master and the minion will get their IP addresses.

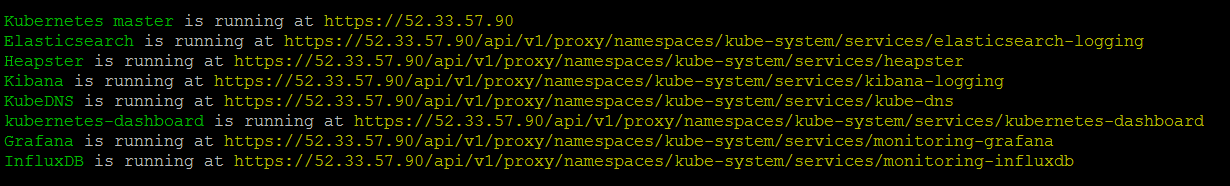
The below screenshot shows that the master is now running:



At a later stage, 4 minions started under a single master node.



Finally,



For the master, for clusters of less than 5 nodes it will use an **m3.medium**, for 6-10 nodes it will use an **m3.large**; for 11-100 nodes it will use an **m3.xlarge**.

For worker nodes, for clusters less than 50 nodes it will use a **t2.micro**, for clusters between 50 and 150 nodes it will use a **t2.small** and for clusters with greater than 150 nodes it will use a **t2.medium**.

After the kube-up.sh is completed, export the PATH variable

Add the appropriate binary folder to your **PATH** to access kubectl:

**export PATH=/home/ec2-user/kubernetes/platforms/linux/amd64:$PATH**

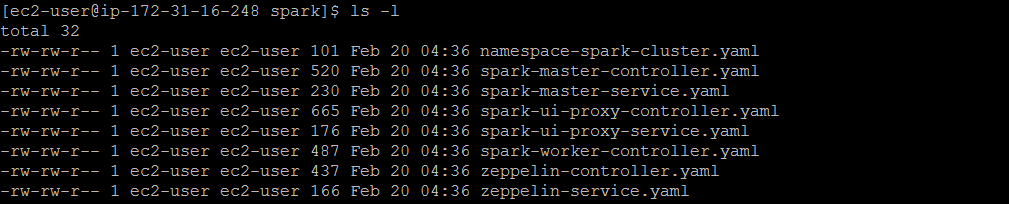
This is how kubernetes is set up on AWS.

**DEPLOYING SPARK ON KUBERNETES USING AWS**

Once Kubernetes cluster is installed and running on AWS, we can create a functional Apache Spark cluster using Kubernetes and Docker

Please follow the below steps:

1. Copy all the required yaml files in a spark folder on the server using WinSCP.



1. Create a namespace:

kubectl create -f namespace-spark-cluster.yaml

1. We can get a list of namespaces using

kubectl get namespaces

1. Create a [replication controller](https://github.com/kubernetes/kubernetes/blob/master/docs/user-guide/replication-controller.md) running the Spark Master service

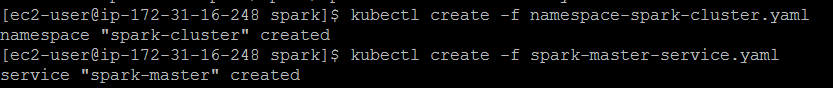
kubectl create -f spark-master-controller.yaml

1. Create a spark-master service and the cluster using the below command:

kubectl create -f spark-master-service.yaml

1. Check if Master is running and accessible:

kubectl get pods



1. Similarly, run the below commands to deploy proxy controller and a corresponding Loadbalanced service:

kubectl create -f spark-ui-proxy-controller.yaml

kubectl create -f spark-ui-proxy-service.yaml

1. After creating the service, you should eventually get a loadbalanced endpoint:

kubectl get svc spark-ui-proxy -o wide

After this The Spark UI in the above example output will be available at [http://aad59283284d611e6839606c214502b5-833417581.us-east-1.elb.amazonaws.com](http://aad59283284d611e6839606c214502b5-833417581.us-east-1.elb.amazonaws.com/)

1. Start spark workers

kubectl create -f spark-worker-controller.yaml

1. You can check if all the workers are running using the command:

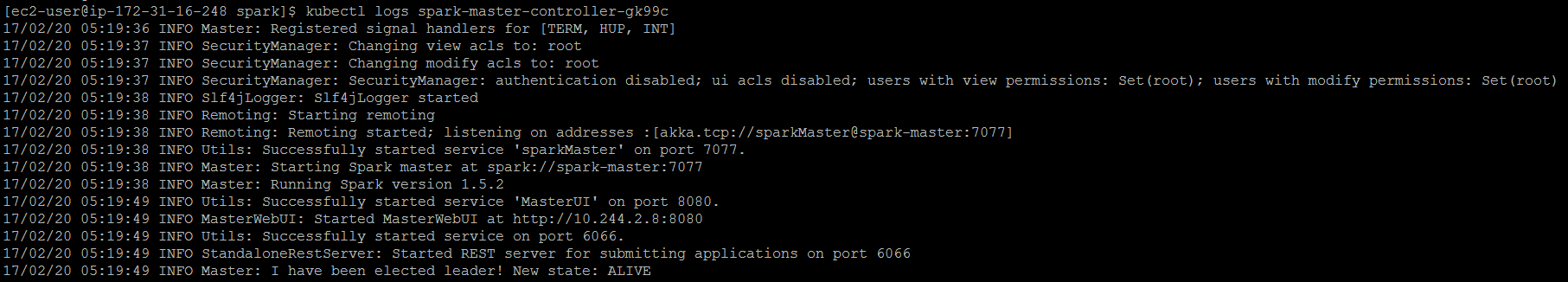
kubectl get pods



1. You can also check the status of an individual node by using the command:

kubectl logs node\_name:

Eg: kubectl logs spark-master-controller-gk99c



1. Create a zeppelin controller and a service to launch jobs on spark using the below commands:

kubectl create -f zeppelin-controller.yaml

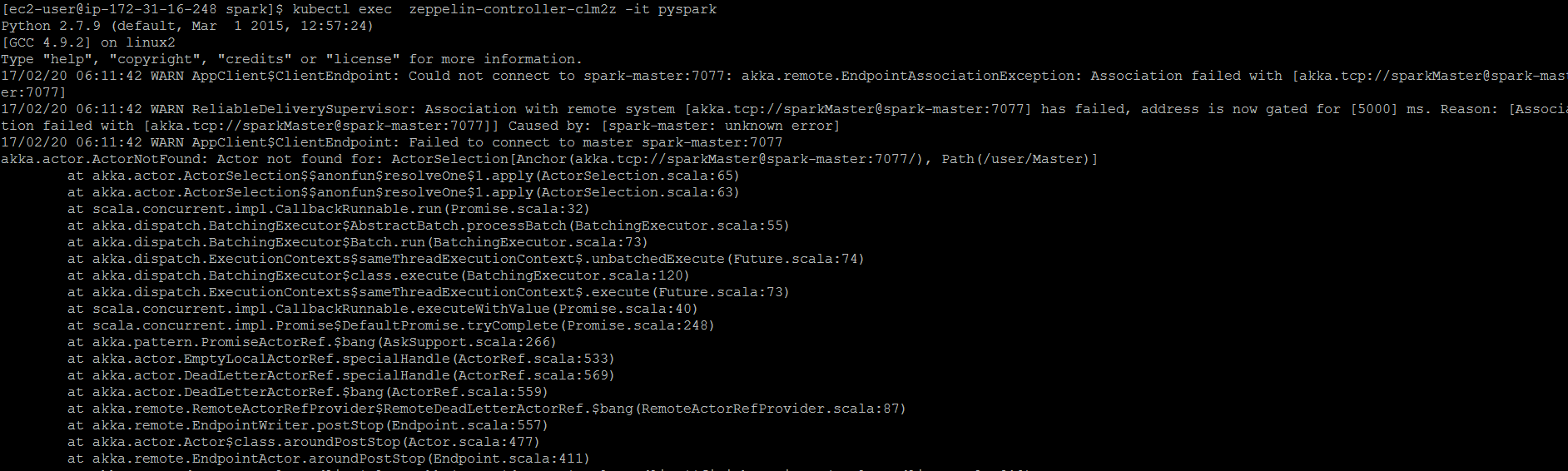
kubectl create -f zeppelin-service.yaml

1. You can check if Zeppelin is running using the command:

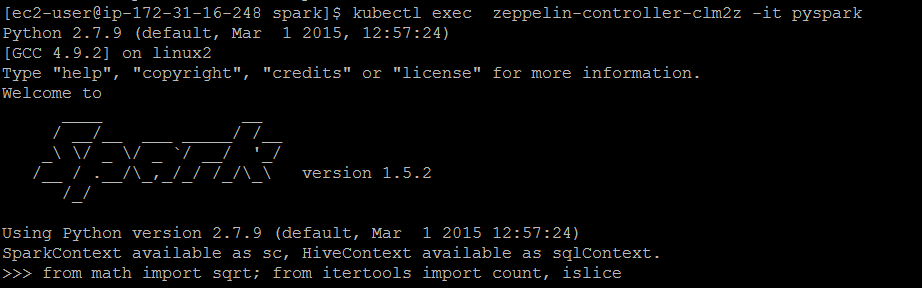
kubectl get pods -l component=zeppelin

Once Zeppelin is running successfully, we can start working with Python snippet to find the number of prime number within the first 10 million numbers.

kubectl exec zeppelin-controller-ja09s -it pyspark



The first time we got a ‘Java not configured or association failed with spark’ error. But once the connectivity fired up, we received the success message:



We can now run the python snippet to get the number of prime numbers within first 10 million numbers.

from math import sqrt; from itertools import count, islice

def isprime(n):

return n > 1 and all(n%i for i in islice(count(2), int(sqrt(n)-1)))

nums = sc.parallelize(xrange(10000000))

print nums.filter(isprime).count()

