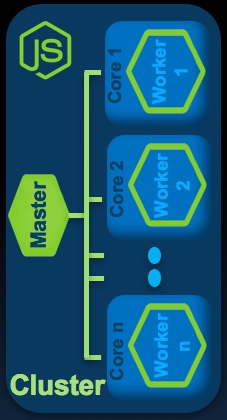
**UCG Node.js Cluster Feature**

By Dennis F. Noto

<https://github.com/dennisnotojr/UCG-Repo>

As you are aware, Node.js “out of the box” utilizes one event loop to process all node commands, while it does off-load requests to threads when blocking I/O messages occur, wouldn’t it be nice to start our own worker threads to offload the event loop.

Some time ago, Node.js developed the Cluster module that allows the ability to fork Node.js code into a Master and Child worker. I’ve been in contact with Aaron Silvas over at GoDaddy because he wrote a npm module for the Cluster module.

* <https://www.youtube.com/watch?v=ArWik4tY1O0&feature=youtu.be>
* <https://www.npmjs.com/package/cluster-service>

UCG now has this built-in. Today we have four container options that can be built from the UCGBuild.sh file on UCG-REPO

* dennisnotojr/node-red-docker-node8-nr-17-data-mapped:version-1 -
* dennisnotojr/node-red-docker-node8-nr-17-data-mapped-cluster:version-1
* dennisnotojr/node-red-docker-node8-nr-17-data-container:version-1
* dennisnotojr/node-red-docker-node8-nr-17-data-container-cluster:version-1

The cluster images contain the cluster-service. Which fully supported/runs on 1,000s of servers. The npm is solid, very few updates, maybe one a year.

When the cluster image starts, it will fork the red.js(NodeRed) on a worker thread in each CPU core automatically. The master takes in the command line args and passes to each child thread. Also, all children will bind to port 1880 and the master will distribute round-robin to each of the child without any configuration. This is the power of the cluster module.

How do you achieve this?

* First grab the NodeRed’s docker github project ***(UCG-Repo already contains a working copy)***.
* Created special docker files for each image (container in and out, cluster on or off)**(Pre-built in the UCG-Repo : node-red-docker/latest)**
* Create a special package.json that gets copied in at build time. ***(Pre-built in UCG-Repo)***
* Cluster package.json included a special cluster.js file with one line it for the cluster config.
* Changed the npm start to be the cluster.js and the worker is red.js

Cluster Service has a built-in administration service that can manage/monitor/upgrade the children workers. Here are the steps to run the service:

* First you need to exec into the container – docker exec –it <container> /bin/bash
* Second, cd over to the /usr/src/node-red/node-modules/cluster-service
* Third, run the admin tool ./cservice --accessKey ucg

Here are the commands available via the service:

cservice> help

cservice:  Running remote command: http://localhost:11987/cli?cmd=help&accessKey={ACCESS\_KEY}

cservice: { more: 'Commands (Use \'help [command\_name]\' for more details)',

  commands:  [ 'exit', 'health','help','info','proxy',a'restart','shutdown','start','upgrade','version','workers' ] }

**Performance and testing**

I have put the cluster containers through various testing on 10 – 2Cpu cluster and 1 – 16Cpu VM to bench test Node.js Clustering vs Kubernetes clustering. Below you will find the 16Cpu JMeter tests.

Environment : 16 core 64GB shared IBM Softlayer server

JMeter Config : 20,000 concurrent users with a ramp up of 300 seconds

Watson Convo : 8 turns with a 30 second think time

NodeRed Settings : Log Level set to Error only, no custom logger,

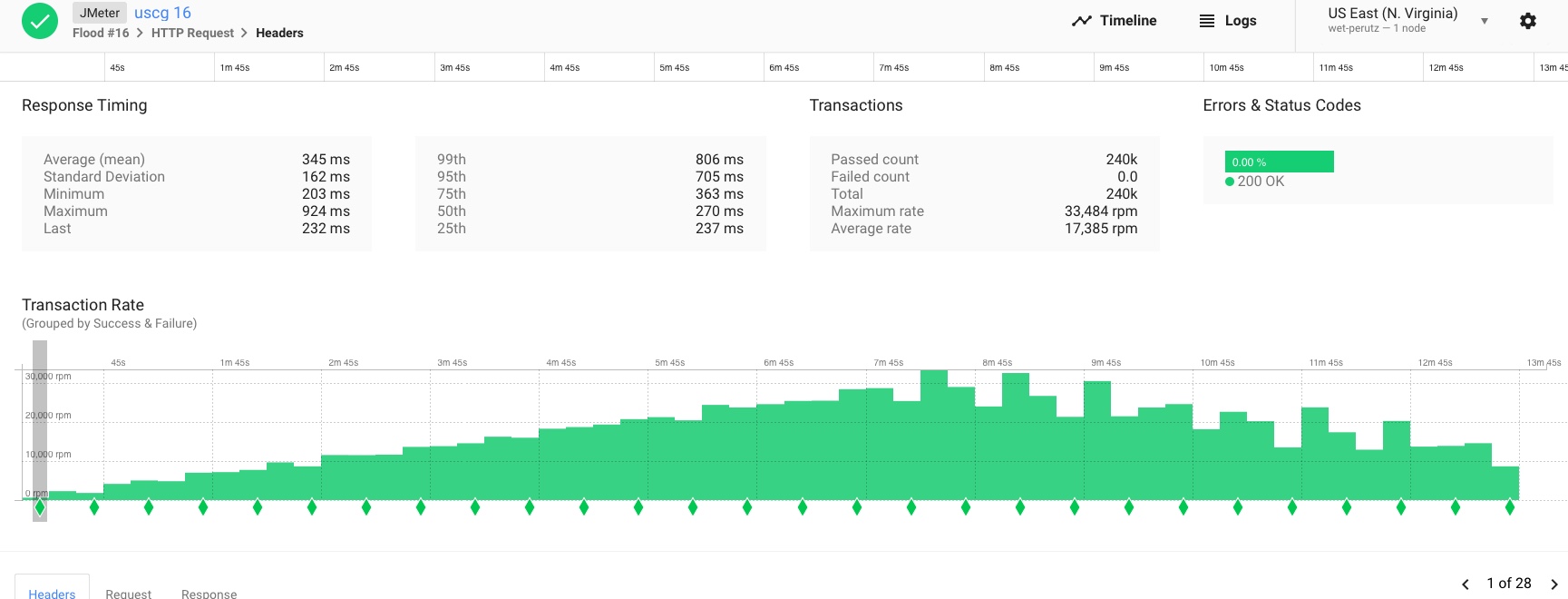
Images : dennisnotojr/node-red-docker-node8-nr-17-data-container-cluster:version-1

dennisnotojr/node-red-docker-node8-nr-17-data-container:version-1

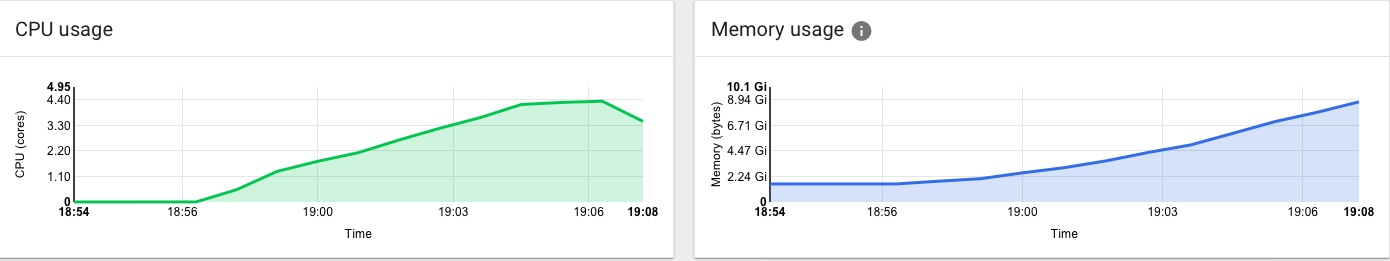
Test one – Kubernetes config with one service and one deployment. Kube literally acts as a hand-off to the VM. Node.js Master fires up 16 children, one per CPU. Node.js load balances off port 1880.

JMeter - Test

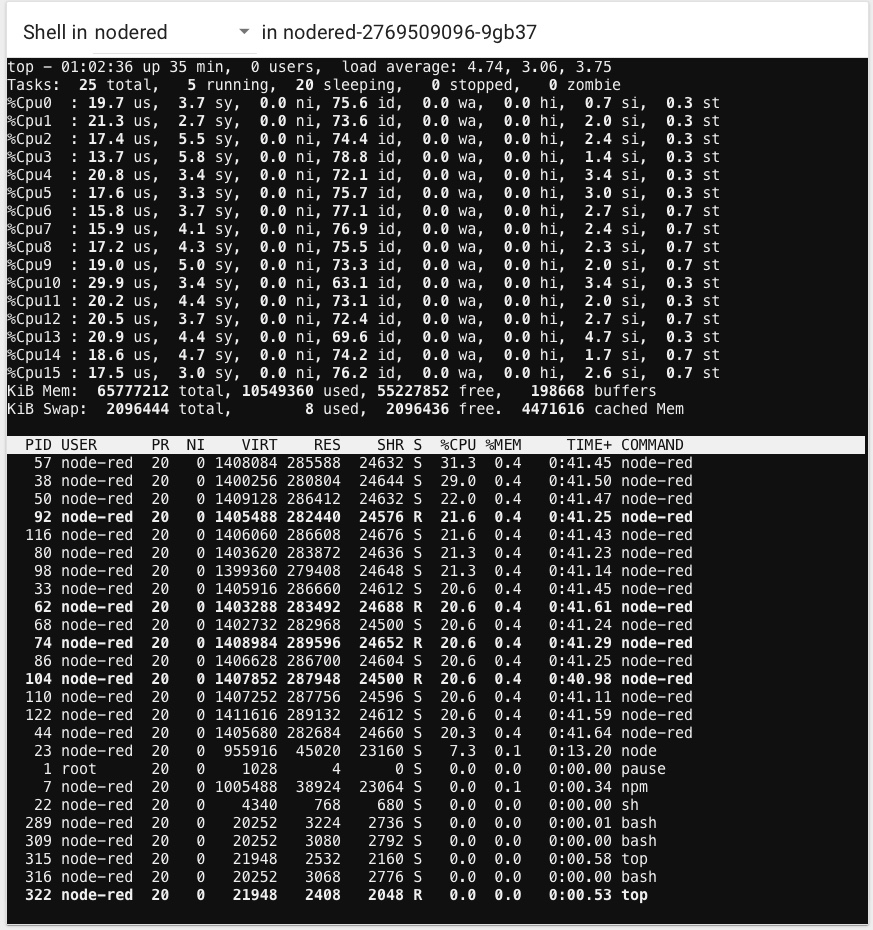




Kubernetes view of the CPU usage



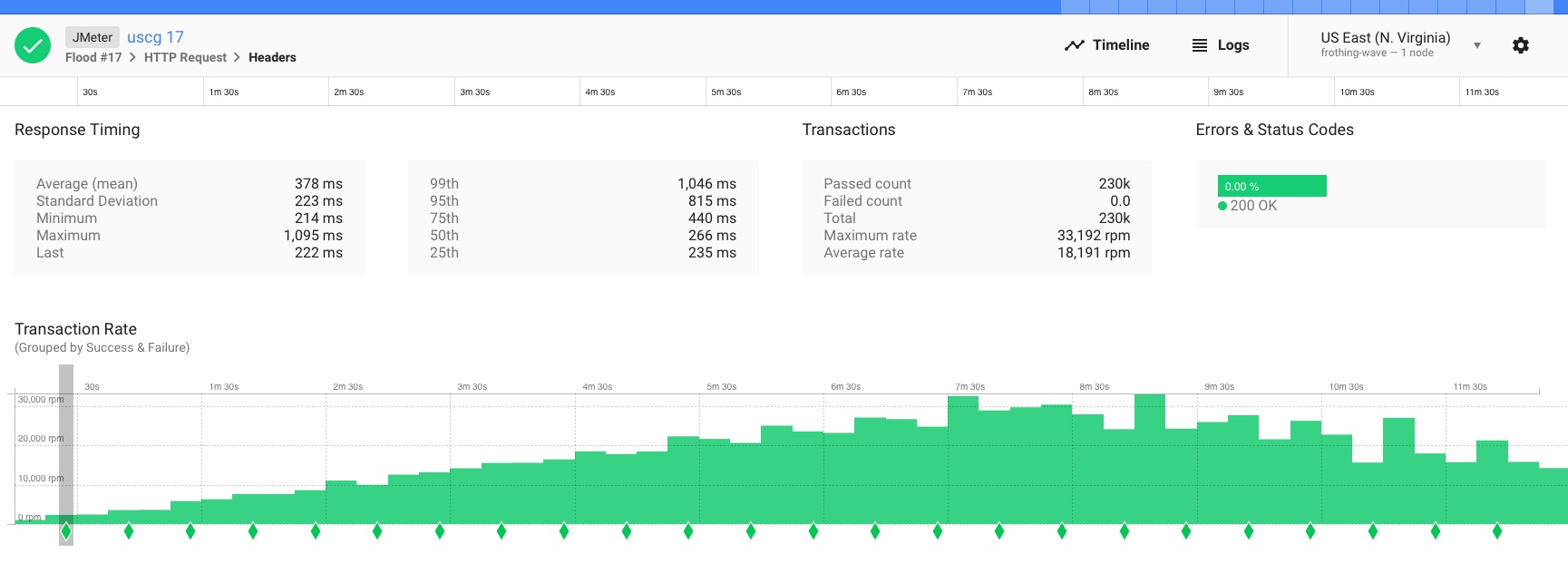
Top’s peak CPU usage



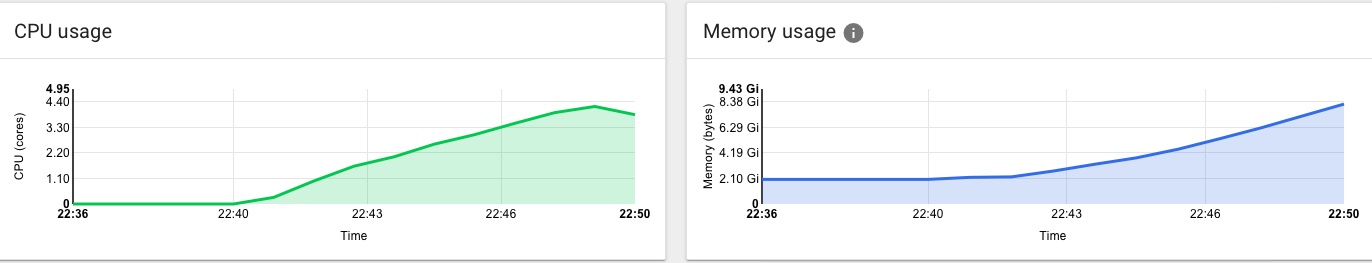
Test 2 – Kubernetes config with one service load-balancing over a deployment with 16 replicas. Kube handles the distribution of requests across call container instances via port 1880

JMeter - Test

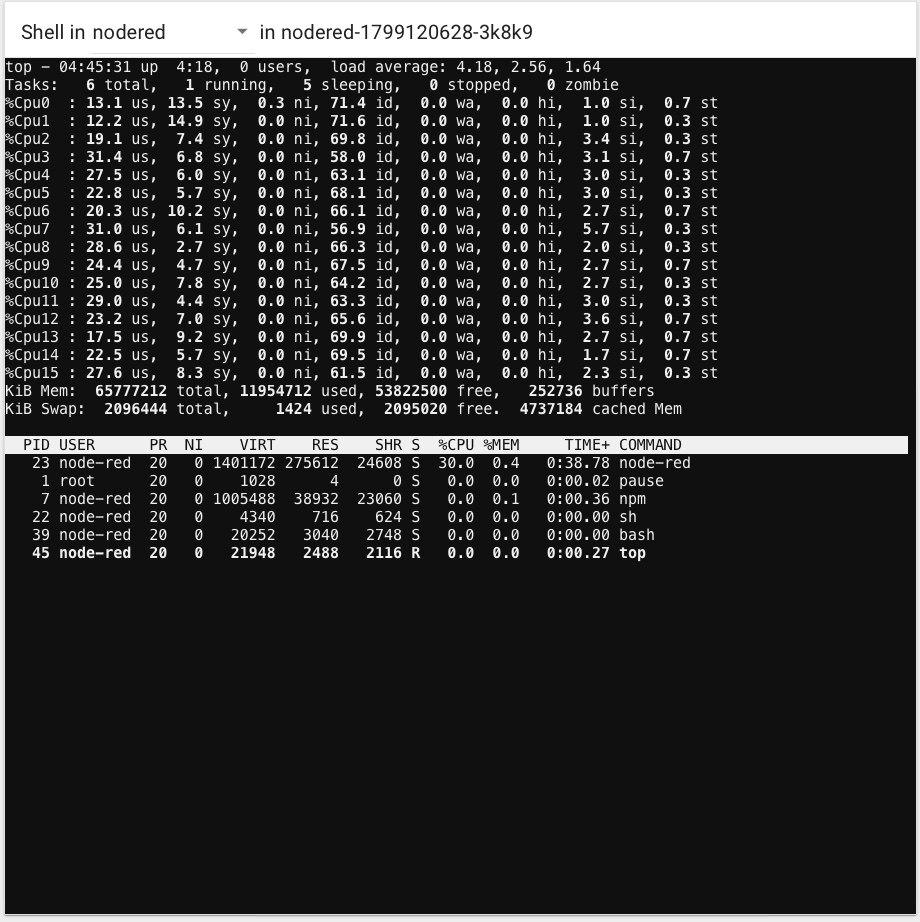




Kubernetes view of the CPU usage



Top’s peak CPU usage



Observations

Both tests achieved maximum users 20K at the 7:30 minute mark during the benchmark, the Kube load-balancing test consumed a little less CPU busy by .1 of a cpu core or avg load of .5% difference. Avg response time was in favor of Node.js clustering 345 mills versus 378. Recording a difference of 33 mills or 10%. Max response time was lower for Node.js at 924 mills versus 1095, which reflects are larger fluctuation from the mean and a higher std-dev in for the Kube load-balancing.

Node.js is a viable option to distribute virtual load over cores with a lower impact to configuration and maintenance of an application framework. Actually, a combination for Kubernetes across physical boxes (Horizontal Scaling) and Node.js Clustering(Vertical Scaling) within the VM is highly recommended. The below picture graphically depicts how UCG handles scaling now available in the current release.

