Chronos vs Aurora, Bistro Comparison

Monday, June 12, 2017 10:15 AM

Overall Comparison

Features: Facebook Bistro has the most important features that we are looking for (Custom Resources, Job Queue, Priority, Cron), but it has other major issues, including possible stability issues, that might take a long time to fix. And the issue, "job configuration is too complex", can only be fixed by redesign or re-architecture, which is not feasible, given our resources. As a pre-release software without documentation, those issues can only be discovered after in-depth investigation, as we have.

Apache Chronos and Aurora both lack the custom resource feature, but it should be possible for us to add those features in either one of them. However, it seems harder with Aurora, as it is much more complex: I have found the implementation locations with Chronos, but I am not sure where to implement it in Aurora yet. Chronos and Aurora have the other features that we need.

Ease of Use: Apache Chronos is a simple Cron scheduler replacement, and it is simple and easy to use by design. In contrast, Apache Aurora is much more ambitious and complex -- it aims to run long-running processes similar to Marathon, in addition to Cron schedules--essentially combining the functionalities of Marathon and Chronos. Chronos has low learning curve, but Aurora has high learning curve.

Production Quality: Chronos is production quality v3.0 (or v2.5 in the case of Yelp's fork), but Aurora is not yet v1.0, so we may need to have a much larger certification process with Aurora, and there might potentially be other issues in Aurora that we might yet discover--as it is pre-release for a reason.

REST API: Chronos operation is mainly based on REST API. Aurora's REST API feature has been pending for about two years, even though other parts are actively being developed. And there is no indication when it will be completed. REST API is probably better for us, as it facilitates easier remote job operations. With Aurora's command line interface (CLI), we need to investigate whether we can create jobs remotely (from slave nodes where Tellus would be running).

Comparison Table

Note: Yelp's Chronos is a fork of Apache Chronos.

Feature	Chronos	Aurora	Bistro
Version	Apache Chronos: 3.0.3; Yelp Chronos: 2.5.0	0.17	0.x
Job queue	Yes	Yes	Yes
Priority	Yes	Yes	Yes
Resource sharing	Yes	Yes	Yes
Custom resources	No [2 man-weeks]	No [3-4 man-weeks; more complex] [1]	Yes
Cron schedule	Apache Chronos supports ISO8601 but not crontab syntax (i.e. no multiple days of week),	Yes (crontab syntax)	Yes

	but Yelp's has support for both [2]		
REST API	Yes	No [Pending for about 2 years AURORA-987]	Yes, except Create jobs
Command line interface (CLI)	No	Yes [Q: But can Tellus call it from slave?]	No
Web UI	Yes [but Yelp's is older v2.5 Web UI]	Yes (but read-only; no job create/delete)	No
Job configuration database	Zookeeper	LevelDB	Flat file [> 1,2 man-month to fix; complex]
Job configuration complexity, learning curve	Easy	Complex	Too complex [fix requires redesign/re-architecture]
Remote execution on Windows	Yes (Mesos agent w/ Docker containers--need test verification)	Yes (Mesos agent w/ Docker containers--need test verification)	No
Docker container	Yes	Yes	No
Other issues	• Web UI does not auto refresh [v2.5; fixed in v3.0] • Web UI is "quirky" [fixed in v3.0]	• Web UI does not auto refresh • Web UI does not show job next run time in summary list (but does so in details; in contrast, Chronos shows it in summary list) • Aurora Web UI link to job stdout is broken [workaround in Mesos UI] • *"num_cpus must be greater than 0.0"*; This error happens with v0.19 development snapshot, but not release 0.17.	• No initial worker set ID consensus (stability issue?) • Bad file descriptor

[1]Not sure where to implement it in Aurora, yet (much more complex than Chronos)
[2]Apache's implementation of Chronos does not support crontab syntax (multiple days of week), but Yelp's fork of Chronos added crontab syntax support, and it appears to be working

Kubernetes vs. Mesos vs. Nomad Evaluation

Wednesday, October 18, 2017 8:49 AM

[Update] Added Nomad.

Evaluation Results (Preliminary)

I have completed a brief evaluation of Kubernetes vs. Mesos vs. Nomad. Evaluation comparison between Kubernetes and Mesos produced a virtual tie, each with its own advantages and disadvantages. Nomad is designed for ease of use, and that is favorable to us, but its lack of custom resources does not satisfy our requirements (though that feature is planned).

Advantages of Kubernetes:

- Kubernetes seems to be more popular than Mesos and Nomad.
- In terms of load balancing, Kubernetes is probably slightly "better" as it is integrated with built-in tools, while Mesos and Nomad need separate applications for that.
- Kubernetes supports dynamically adding resources, but Mesos is static, and requires worker machine restart upon resource change. However, dynamic resource reconfiguration is planned for Mesos.

The disadvantage of Kubernetes is that it is known to be very difficult to deploy and operate (at least in multi-node cluster configuration.) Also, Mesos seems more scalable than Kuberntes, roughly twice as much (details below).

Mesos is slightly easier than Kubernetes, but Nomad is the easiest of the three. However, Nomad is relatively immaturity at version 0.7, while Mesos and Kubernetes are mature products.

Table 1. Comparison Highlight Summary

Category	Kubernetes	Mesos	Nomad
Scalability	High	Higher than Kubernetes	Winner (theoretically)
Load balancing	Winner (integrated)		
Resource sharing	Winner (dynamic resource definition)	Dynamic resource reconfiguration is planned (MESOS-1739)	No custom resources yet -- SHOW STOPPER
Cron schedule	Tie	Tie	Tie
Job queue	Tie	Tie	Tie
Priority	Winner (integer type)	2 levels (medium, high with Chronos)	Tie with Winner
Ease of use	Third Place	Second Place	Winner
Popularity	Winner		

Evaluation Criteria

1. Features or characteristics must be confirmed via either,
    a. Testing
    b. or clear documentation with examples, in the case that it is not tested

Evaluation Result Details

As we didn't want to spend too much time with Kubernetes, as another group will produce that system, we reduced the scope to evaluation mostly by documentation, plus a little testing as time permits. I was only able to test Kubernetes with scaling and load balancing in single-node configuration (with *minikube*), given the one week available, as multi-node cluster installation is very difficult with *kubeadm*. The rest of the Kubernetes evaluation were from documentation.

Mesos, with Marathon and Chronos, was more fully tested in previous proof-of-concept (RPD:30237198).

The Nomad results are essentially completely from documentation.

Notations: Advantages are indicated by green color, and disadvantages are indicated by red color.

Job Scheduler Evaluation Results

Table 2 shows a job scheduler comparison between Kubernetes, Mesos with Chronos, and Nomad.

Table 2. Job Scheduler Comparison Summary

Item	Kubernetes	Mesos + Chronos	Nomad
Cron scheduler (Linux "crontab" syntax)	Yes (built-in) *[untested]*	Yes (separate Chronos scheduler with Yelp's enhancement)	Yes (built-in)
Compute resource sharing (CPU, memory, disk)	Yes (cpu, memory) *[untested]*	Yes (cpu, memory, disk)	Yes
Custom resources	Yes, dynamically via REST API *[untested]*	Yes, after our enhancement; dynamic resource reconfiguration is planned	No (pending Feature #1081)
Job queue	Yes *[untested]*	Yes	Yes
Priority	Yes (v1.8.1+) *[untested]*	Yes	Yes
Job scalability	High	Higher than Kubernetes (more jobs per node with non-container jobs)	Higher than Mesos (theoretically)

Job Scheduler Features

As a distributed computing platform, Kubernetes has an advantage that it dynamically configures resources via REST API. Mesos currently loads resource configuration statically upon agent (worker node) startup, but has plan for dynamic resource reconfiguration (MESOS-1739).

Kubernetes provides a built-in job scheduler, but Mesos enables you to create a separate scheduler, and Chronos is such a scheduler. Other scheduler-related features are similar, including cron schedule, job queue, and priority (though Kubernetes priority is integer, while Chronos has only two levels (high and medium).

Nomad also has integrated Cron scheduler. It has all the features that we require, except that custom resources feature is pending.

Cluster Management and Load Balancing Comparison

Table 3 below is summary for comparison as load balancer with Kubernetes vs. Mesos (with Marathon, Marathon-lb, and HAProxy) vs. Nomad (with HAProxy and DNSMasq).

Table 3. Cluster Management and Load Balancer Comparison Summary

Item	Kubernetes	Mesos + Marathon + Marathon-lb + HAProxy	Nomad
Containers (Docker, etc.)	Yes	Yes	Yes
Container mount volume	Yes *[untested]*	Yes	Yes
Fetcher (transfer files to worker machine)	Yes, by built-in command *[untested]*	Yes (by job configuration)	Yes (job config.)
Can run app/job *without* containers	No	Yes	Yes
Constraints (restrict which nodes to run on)	Yes	Yes	Yes
REST API	Yes *[untested]*	Yes	Yes
High availability (HA)	Yes *[untested]*	Yes *[untested]*	Yes
Scalability	High [untested]	Higher than Kubernetes	Higher than Mesos (Theoretically)
Load balancing	Yes (using built-in kube-proxy) *[untested]*	Yes, using external Marathon-lb, HAProxy *[untested]*	Yes (via HAProxy, DNSMasq)
Service discovery/DNS (map service to IP address)	Yes, using e.g. CoreDNS *[untested]*	Yes, using e.g. Mesos-DNS *[untested]*	Yes (via Consul)
Runs application/services	Yes, built-in	Yes, using Marathon	Yes
Rolling upgrade (application/service)	Yes *[untested]*	Yes *[untested]*	Yes
Data store	etcd (faster than Zookeeper)	Zookeeper[a]	Built-in
Support for Linux	Yes	Yes	Yes
Support for Windows	Yes, nodes only	Yes, nodes only	Yes
Cluster deployment and operation difficulty	Medium with single-node minikube, but *very high* with multi-node kubeadm	Medium to high	Easiest
Maturity	Very mature	Very mature	Young

[a]Mesos will also support etcd in future version

Setup Comparison

Kubernetes is known to be very difficult to deploy and operate multi-node Kubernetes cluster (via kubeadm).

In comparison, Mesos multi-node setup is easier than Kubernetes. I was able to test Mesos in multi-node configuration, in a previous Chronos/Mesos proof of concept project.

Nomad is designed for ease of use. It consists of only one single binary for both master and nodes, and it has built-in cluster management database. Its ease of use and support for Windows (both master and nodes) are favorable to us.

Architecture and Approach Comparison

Kubernetes and Mesos solve many of the same problems but take different approaches. Kubernetes is a container management platform, and requires containers for running all application and jobs. Mesos is a more general "distributed operating system", managing not just containers, but also general applications, services, and jobs.

Kubernetes is more opinionated. Mesos is more flexible.

Kubernetes runs services with built-in components, but Mesos requires a separate application ("*scheduler*" in Mesos terminology) like Marathon to run applications/services.

Nomad is designed to be simple to use. It consists of one single binary. It has integrated cluster management, scheduler, and executor.

Scalability Comparison

Kubernetes is of the opinion that everything must run inside containers, but Mesos does not. Containers incur overhead. With Mesos, we ran more (roughly twice as many) jobs without containers than with containers. Containers are not needed for some types of testing jobs, such as Selenium Grid tests (but containers may be needed for isolated clean environment). Kuburnetes is less scalable per node than Mesos: From Mesos testing results, we have observed that the number of concurrent jobs drops with Docker containers (approximately by half).

With the release of version 1.6, Kubernetes scales to 5,000-node clusters (worker machines). Mesos' two-tier architecture (with Marathon) makes it very scalable. According to Mesosphere, Mesos and Marathon clusters have been scaled to 10,000 nodes. So, Mesos is more scalable than Kubernetes.

Nomad is (theoretically) more scalable than Mesos.

Load Balancing

Kubernetes is more integrated in load balancing, with built-in application/service container management and proxy; Mesos requires external applications: Marathon, Marathon-lb, and HAProxy. Nomad requires external HAProxy and DNSMasq.

In terms of service discovery, they all use external DNS applications (which is a "toss-up".)

Popularity

Kubernetes seems more popular than Mesos, according to some article. Nomad is probably the least popular currently, given its pre-release status.

Reference

- https://mesosphere.com/blog/scaling-mesos-at-apple-bloomberg-netflix-and-more/
- https://www.loomsystems.com/blog/single-post/2017/06/19/kubernetes-vs-docker-swarm-vs-

apache-mesos-container-orchestration-comparison

- https://kubernetes.io/docs/concepts/configuration/pod-priority-preemption/
- https://www.slideshare.net/SreenivasMakam/service-discovery-using-etcd-consul-and-kubernetes
- https://www.stratoscale.com/blog/kubernetes/kubernetes-vs-mesos-architects-perspective/
- https://platform9.com/blog/kubernetes-vs-mesos-marathon/
- https://mesosphere.com/blog/scaling-mesos-at-apple-bloomberg-netflix-and-more/
- Nomad vs. Kubernetes: https://www.nomadproject.io/intro/vs/kubernetes.html
- Nomad vs. Mesos (with Aurora, Marathon): https://www.nomadproject.io/intro/vs/mesos.html

Chronos RPD Description

Friday, September 29, 2017 7:18 AM

This project will evaluate or produce a job scheduler as replacement for STAF Cron, Tellus RestfulSchedulingService (priority-based scheduler), and Tellus JobQueue.

The primary motivation is a fix for the Tellus Scheduler design issue that is primary cause of "ran out of VM" issue. For example: The Scheduler schedules job1 immediately followed by job2, based on estimated duration of jobs. Each job allocates VM. If job1 runs longer than expected, it overlaps job2, occupying the VM that job2 needs.

As a replacement, the job scheduler would need to replace their functionalities:

- Resource sharing and custom resources
- Job queue
- Priority
- Cron schedule

Secondly, we also need to scale. The number of jobs have increase several fold in the past few years. A single server is starting to hit CPU and memory limits.

High availability clustering (fault tolerance) is bonus feature.

We have selected Chronos after the evaluations described in the Evaluation section below.

As summary, this is what Chronos does for us:

- Most importantly, Chronos fixes Tellus Scheduler design issue that is primary cause of "ran out of VM" issue

Item	Current Tellus/STAFCron	Chronos
Data resource sharing and custom resources (e.g. FPE VMs)	Yes, **but design issue which is primary cause of "ran out of VM" issue**	Yes (fixes Tellus design issue)
Job queue	Yes, **but canceled the queuing part due to latency and race condition concerns**	Yes
Priority	Yes	Yes
Cron schedule	Yes	Yes
CPU, memory resource sharing (helps solve "out of CPU, memory" issue)	No	Yes
Scaling (horizontally)	No	Yes
High availability cluster (fault tolerance)	No	Yes

Evaluations

Requirements:

1. Resource sharing and custom resources (for example, FocalPoint VMs)
2. Job priority
3. Cron schedule (UNIX Crontab or equivalent)
4. Scaling (horizontally)

5. High availability cluster (preferred)

We have evaluated several job schedulers, including the ones listed here, as well as commercial products:

Apache Chronos https://mesos.github.io/chronos/
Apache Aurora http://aurora.apache.org/
Facebook Bistro https://github.com/facebook/bistro
Netflix Fenzo https://github.com/Netflix/Fenzo
Quartz Scheduler http://www.quartz-scheduler.org/
SOS JobScheduler https://www.sos-berlin.com/jobscheduler

Bistro appeared to be the only product that had all of the "replacement" features that we wanted, on paper: custom resources, job queue, priority, and Cron schedule. However, it had other (complexity and reliability) issues that made it unsuitable for us.

Apache Chronos originally lacked Crontab schedule and custom resource features. However, Yelp's fork implemented the Crontab schedule feature, and we added the custom resource feature. It had only two levels of priority: high and medium, which seemed sufficient to us. So, our implementation of Chronos has all the required features.

Other highlights:

- Netflix Fenzo has compute resource awareness (CPU, memory) but not data resource awareness (custom data resources such as FPE VM, browser versions.) And it is a library, not an application.
- SOS JobScheduler has priority, but its resource awareness seems to be limited to exclusive access locking of file or database; it is also a more traditional job scheduler with no consideration for parallel computing which is good for future proofing.

Job Scheduler Evaluation Summary:

Product	Cron	Job Queue	CPU, Memory Resource Sharing	Custom Resources	Priority	Horizontal Scaling	High Availability
Bistro	Yes	Yes	Yes	Yes	Yes	Yes	No
Apache Chronos	Yes (with Yelp's enhancement)	Yes	Yes	Yes (after our enhancement)	Yes (only high/medium)	Yes	Yes
Apache Aurora	Yes	Yes	Yes	No	Yes	Yes	Yes
Netflix Fenzo	Yes	Yes	Yes	No	No	Yes	Yes
SOS JobScheduler	Yes	Yes	No	No	Yes	No	No
Quartz Scheduler	Yes	No	No	No	No	No	No
STAF Cron*	Yes	No	No	No	No	No	No

*STAF Cron is included as a baseline.

Chronos POC Completion Report

Wednesday, October 04, 2017 10:14 AM

Chronos Proof-of-concept Completion Report

[UPDATE 10/5/2017]

The Chronos Proof-of-concept project has completed successfully. It evaluated job schedulers and produced a Chronos job scheduler as replacement for STAF Cron, Tellus RestfulSchedulingService (priority-based scheduler), and Tellus JobQueue. We enhanced Chronos, so that Chronos has the required functionalities. We also performed testing including performance testing, and the testing passed.

The primary motivation is a fix for the Tellus Scheduler design issue that is the primary cause of "ran out of VM" issue. Example problem scenario: The Scheduler schedules job1 immediately followed by job2, based on estimated duration of jobs. Each job allocates VM. If job1 runs longer than expected, it overlaps job2, occupying the VM that job2 needs.

Secondly, we also need to scale. The number of jobs have increase several fold in the past few years. A single server is starting to hit CPU and memory limits. We need to scale out to multiple servers.

We have selected Chronos after the evaluations described in the Evaluation section below.

Chronos Benefits

As summary, this is what Chronos does for us:

- Most importantly, Chronos fixes Tellus Scheduler design issue that is primary cause of "ran out of VM" issue

Item	Current Tellus/STAFCron	Chronos
Data resource sharing and custom resources (e.g. FPE VMs)	Yes, **but design issue which is primary cause of "ran out of VM" issue**	Yes (fixes Tellus scheduler design issue)
Job queue	Yes, **but canceled the queuing part due to latency and race condition concerns, and fell back to daily batch prioritization**	Yes
Priority	Yes	Yes
Cron schedule	Yes	Yes
CPU, memory resource sharing (helps solve "out of CPU, memory" issue)	No	Yes
Scaling (horizontally)	No	Yes
High availability cluster (fault tolerance)	No	Yes

Job Scheduler Evaluations

Evaluation Criteria

As a replacement, the job scheduler would need to replace current functionalities:

1. Custom resources

2. Job queue
3. Priority
4. Cron schedule

Requirements for Evaluation:

1. Resource sharing and custom resources (for example, FocalPoint VMs)
2. Job queue
3. Job priority
4. Cron schedule (UNIX Crontab or equivalent)
5. Scaling (horizontally)
6. Load balancing (for Web applications and services)
7. High availability cluster (preferred)

Evaluation Subjects

We have evaluated several job schedulers, including the ones listed here, as well as commercial products such as JAMS:

Apache Chronos https://mesos.github.io/chronos/
Apache Aurora http://aurora.apache.org/
Facebook Bistro https://github.com/facebook/bistro
Netflix Fenzo https://github.com/Netflix/Fenzo
Quartz Scheduler http://www.quartz-scheduler.org/
SOS JobScheduler https://www.sos-berlin.com/jobscheduler

Evaluation Results

Bistro appeared to be the only product that had all of the "replacement" features that we wanted, on paper: custom resources, job queue, priority, and Cron schedule. However, it had other issues that made it unsuitable for us: overly complicated job configuration, instability (worker node connection errors) under load, and it uses flat file for job database (which is not performant under load).

Apache Chronos originally lacked these features: a) Crontab schedule syntax, and b) custom resource. However, Yelp's fork implemented the Crontab schedule feature, and we (FactSet QA Automation) added the custom resource feature. It has only two levels of priority: high and medium, which seems sufficient to us. So, our implementation of Chronos has all the required features.

Other Evaluation highlights:

- Apache Aurora is feature rich, but not yet v1.0, and REST API is not available yet
- Netflix Fenzo has compute resource awareness (CPU, memory) but not data resource awareness (custom data resources such as FPE VM, browser versions.) And it is a library, not an application.
- SOS JobScheduler has priority, but its resource awareness seems to be limited to exclusive access locking of file or database; it is also a more traditional job scheduler with no consideration for parallel computing which is good for future proofing.

Job Scheduler Evaluation Result Summary:

Product	Cron	Job Queue	CPU, Memory Resource Sharing	Custom Resources	Priority	Horizontal Scaling	High Availability	RPC
Apache Chronos	Yes (with Yelp's	Yes	Yes	Yes (after FactSet	Yes (only high/medi	Yes	Yes	Yes (REST)

	enhancement)			enhancement)	um)			
Bistro	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes (REST)
Apache Aurora	Yes	Yes	Yes	No	Yes	Yes	Yes	No
Netflix Fenzo	Yes	Yes	Yes	No	No	Yes	Yes	No
SOS JobScheduler	Yes	Yes	No	No	Yes	No	No	Yes
Quartz Scheduler	Yes	No	No	No	No	No	No	No (in progress)
STAF Cron*	Yes	No	No	No	No	No	No	Yes

*STAF Cron is included as a baseline.

Chronos Test Results Summary

Category	Test Result
Crontab schedule syntax feature	Pass
Custom resources	Pass
Constraints (where to run jobs)	Pass
REST API	Pass
Multiple agent ("slave") machines	Pass
Run job in Docker container	Pass
Fetcher (file transfer to agent)	Pass
Selenium test in Chronos job	Pass
Load test 1000 jobs (single, dual agents)	Pass
Stress test 5000 jobs	Pass, except UI issue
Running Chronos in Marathon	Pass

Load Test Highlights

In load test, we submitted 1000 jobs to run at the same time, and they ran to completion successfully. About 276 jobs actually ran concurrently per agent machine, and the rest of the jobs waited in hold queue, until resources became available. CPU and memory consumptions were reasonable (about 1% CPU and 55 MB RAM per job with Selenium.)

The Chronos machine consumed near 100% CPU for a few minutes while jobs are being submitted in large volume. Which might indicate that Chronos should be on a separate machine. The CPU utilization is otherwise controlled by resource sharing and do not reach 100%.

Stress Test Highlights

We also performed stress test with running 5000 jobs, and the jobs ran with no significant delay or

slowness. The REST API is slower at about 1-2 seconds for creating job; listing jobs took about 6 seconds while jobs are being submitted, but 1-2 seconds afterwards. That performance seems acceptable.

However, the UI becomes unresponsive, as it tries to list 5000 jobs on one single page. It is partly a design issue: it probably should paginate. It is also partly due to zookeeper performance -- it has significant slowdown under load (Chronos uses Zookeeper for job database). That is the only defect found. A workaround is to use REST API, instead of UI.

Apache Mesos has plans to replace Zookeeper by etcd, which supposedly is designed for high performance, but it is unclear if Chronos plans to switch to it as well.

Resource Consumption Statistics Summary

Approximate statistics (Job statistics are based on Protractor Selenium job.)

Job CPU	1%
Job Memory	55 MB
Job Disk	400 MB
Chronos CPU	2.5%
Chronos Memory	304 MB
Mesos Master CPU	0.3%
Mesos Master Memory	33 MB
Mesos Agent CPU	0.3%
Mesos Agent Memory	208 MB

Future Notes

Chronos will be productionized with the upcoming Tellus 2.0. We tentatively plan to deploy Chronos as Marathon task (instead of Linux service).

A related but separate project is Tellus load balancing for Tellus 2.0 (for Tellus Web application, not Chronos itself). For that, we need to requisition permanent Linux development machines and evaluate and test load balancing solutions:

- RPD:32625440 Request for Linux machine
- RPD:34729846 Evaluate Mesos + Marathon + Marathon-LB + HAProxy for Tellus load balancing
- RPD:34621075 Evaluate Kubernetes for Tellus scaling, load balancing

Also, far in the future, we hope to use Mesos for Windows as agents, when it is available. That would allow us to replace STAF STAX (and therefore all of STAF).

Mesos/Chronos Machine Requirements (Tentative)

Development machines:

- Mesos master+agent machine (1x) (this includes Zookeeper)
- Mesos agent machine (1x)
- Chronos machine (1x)

Production machines (Non-High-availability Configuration):

- Mesos master machine (1x) (this includes Zookeeper)
- Mesos agent machines (2x)
- Chronos machine (1x)

[FUTURE, Post Tellus 2.0] High-availability Production Cluster:

- Mesos master machines cluster (3x)
- Zookeeper machines cluster (3x)
- agent machines (2x)
- Chronos machine (1x)

Minimum System Requirements:

(For all machines)
CPU: 4 cores
Memory: 8 GB
Disk: 200 GB
OS: RHEL7

Note: In future HA configuration, Zookeeper as separate machines might have lower system requirements.

Admin permissions:

1. Tellus admins need to read/write files on agent (slave) machines (mesodevagent1, mesosdevagent2):
    a. /etc/mesos-slave/resources
    b. /etc/mesos-slave/attributes
2. Tellus admins may also need/request rebooting agent machines, in order for the resource/attribute changes to take effect

Resources

Chronos (FactSet) source code: https://gitlab.factset.com/app-qa-automation/chronos2x
Chronos (v3) source code (for future merge): https://gitlab.factset.com/app-qa-automation/chronos
Apache Mesos: http://mesos.apache.org/
Apache Chronos: https://mesos.github.io/chronos/

Apache Chronos Tutorial

Wednesday, June 21, 2017 9:36 AM

The best way to get started with Apache Mesos and Chronos is probably the following tutorial.

Follow https://open.mesosphere.com/advanced-course/, except that you should do this:

- Use Git Bash for working with vagrant on Windows
- Mesos-dns example doesn't seem to work (but generally it should)
- You don't need to do DCOS, Ansible, or the Advanced Usage sections, for now
- Vagrant VM is in UTC time zone, so Cron schedules should also be in UTC time

  Note: Yelp's Chronos has the same usage as Apache's Chronos, except that Yelp's REST API removed the /v1 prefix. So this info is applicable to Yelp's Chronos.

Yelp Chronos Introduction

Wednesday, June 21, 2017 8:55 AM

Yelp/chronos

Yelp Chronos Info

Yelp Chronos is Yelp's fork of Apache Chronos.
The source code is located here: Yelp Chronos

It adds support for crontab schedule syntax, using third-party Cron schedule parser, cron-utils: https://github.com/jmrozanec/cron-utils

Yelp made approximately more than 7000 lines of code changes to Chronos. And Yelp would like Apache to merge it, but there is no response, yet. It is hard to merge, because there is large amount of divergence and merge conflicts.

Note: It uses UNIX style crontab syntax, which does not support year. (In contrast, *Quartz Scheduler* extends the syntax with year.)

Yelp's fork of Chronos seems to be based on Apache Chronos version 2.5.0. Current version of Apache Chronos is 3.0.3.

The older Chronos version 2.5.0 has Web UI issues:

- It has "quirky" design; the newer version has a completely different style that is much "cleaner" and more intuitive
- Does not automatically refresh upon job changes by REST API (fixed in 3.0)
- Does not show next run time like the new version
- Updating job in Web UI has known issues (use REST API instead)

Initial testing seems to indicate that the crontab enhancement is working. The test consisted of typical/important cases:

- Every 3 minutes (*/3 * * * *)
- Specific date and time (42 14 14 6 *)
- Days of week:
    - Wednesday at specific time (42 14 * * 3)
    - Monday, Wednesday at specific time (42 14 * * 1,3)
    - Negative case (today is not specified day of week)

REST API Endpoints

Yelp removed the /v1 prefix from REST endpoints.

Create schedule-based job:

POST http://localhost:4400/scheduler/iso8601

Create dependency-based job:

POST http://localhost:4400/scheduler/dependency

Delete job:

DELETE http://localhost:4400/scheduler/job/*jobname*

List jobs:

GET http://localhost:4400/scheduler/jobs

Search job by name:

GET http://localhost:4400/scheduler/jobs/search?name=*jobname*

...

Tellus in Mesos Architecture

Monday, June 19, 2017 2:31 PM

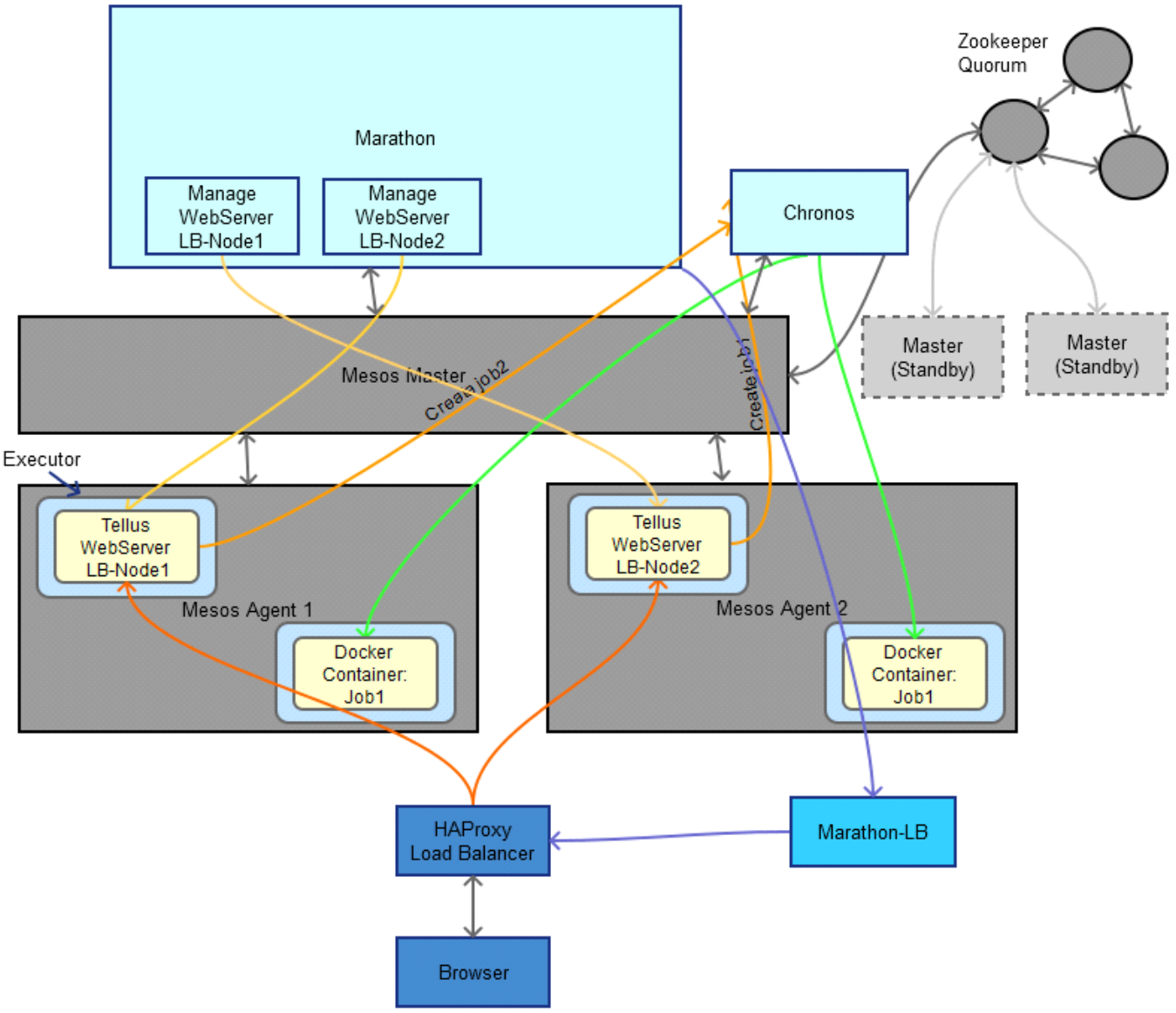

Figure 1. Tellus in Mesos

Pencil file for diagram above:

Tellus in Mesos

- Mesos depends on Zookeeper
- Zookeeper: At least one machine. For fault tolerance, run at least three.
- Mesos Master: At least one machine. For fault tolerance, run at least three with two on standby.

- Mesos agent: One or more machines.

Docker Container vs FocalPoint

- Mesos agents (slaves) need to be statically started and connected to master, so Mesos agents cannot run on FPE VM, which are dynamically started on-demand. That is, Mesos agents need to run on servers instead of FPE VM.
- As some jobs need to run on clean machines or different versions of Windows operating system, they need to run in FPE VM or Docker containers. As Chronos and Aurora have support for Docker containers but not FPE, it might be better to run jobs in Docker containers (instead of FPE), over the long term. Docker should be more lightweight than FPE VM, but its Windows support is not as mature.

FDSCloud VMs

Thursday, June 22, 2017 10:23 AM

http://fdscloud.factset.com

- Click Get Started link (on right)
- Click Get Started button (at bottom)
- Login with Windows credentials
- Click Catalog tab
- Select RHEL 7 (Dept - Quality Assurance)
- Click Request

- It will request a VM with lease of 30 days.
- It will notify you by company email.

RHEL7 Mesos Development Server Installation

Thursday, September 14, 2017 9:46 AM

Summary

- ***On Mesos master machine:*** Install and enable Zookeeper, Mesos master and agent.
- ***On Mesos agent ("slave") machine:*** Only enable Mesos agent (not Mesos master or Chronos).
- ***On Chronos machine:*** Will run Mesos agent and Chronos but not master.
  Chronos on Development Server: Developer themselves will install Chronos in user directory, and run it by Marathon
  Note: For future Chronos on Production Server: Chronos will reside in global directory and be started by Marathon (will not run as systemd service)

Java Installation

- Install OpenJDK:

```
$ sudo yum install -y java-1.8.0-openjdk-devel.x86_64
```

- To see actually Java installation location:
  `$ sudo ls -l /etc/alternatives/javac`

- Edit /etc/environment to add Java home, for example:

```
JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.141-1.b16.el7_3.x86_64
```

Docker Installation (Mesos Agent/Chronos Machine Only)

- Install Docker:

```
$ sudo yum install -y docker-ce
```

- Start Docker:

```
$ sudo systemctl start docker
```

- Enable auto startup:

```
$ sudo systemctl enable docker
```

- Verify that docker is installed correctly by running the hello-world image:

```
$ sudo docker run hello-world
```

Zookeeper Installation (Mesos Master Machine Only)

- Install zookeeper packages:

```
$ sudo rpm -Uvh http://archive.cloudera.com/cdh4/one-click-install/redhat/6/x86_64/cloudera-cdh-4-0.x86_64.rpm

$ sudo yum install -y zookeeper zookeeper-server
```

- To create /var/lib/zookeeper and set permissions:

```
$ sudo mkdir -p /var/lib/zookeeper
$ sudo chown -R zookeeper /var/lib/zookeeper/
```

- To initialize ZooKeeper for the first-time:

```
$ sudo service zookeeper-server init --myid=1
```

- To start ZooKeeper:

```
$ sudo service zookeeper-server start
```

- To enable Zookeeper auto startup:

```
$ sudo chkconfig zookeeper-server on
```

- Edit /etc/zookeeper/conf/zoo.cfg, and add line:

```
server.1=mesosmasterdevb01:2888:3888
```

- Use the interactive shell to test your installation:

```
$ sudo /usr/lib/zookeeper/bin/zkCli.sh
help
create /test 1
get /test
set /test 2
get /test
delete /test
quit
```

Mesos Installation

Install Mesos on Mesos master, Mesos agent, and Chronos servers.

- Install Mesos package:

```
$ sudo rpm -Uvh http://repos.mesosphere.com/el/7/noarch/RPMS/mesosphere-el-repo-7-3.noarch.rpm

$ sudo yum -y install mesos
```

Mesos Configuration

- Edit /etc/mesos/zk, and it should look like this:

```
zk://mesosmasterdevb01:2181/mesos
```

  where *<zkserver>* is zookeeper server; for single-node configuration, you can use mesosmasterdevb01.

- Edit `/etc/mesos-master/quorum`, and it should be 1:
  `1`

- In /etc/mesos-slave/executor_environment_variables file (use JAVA_HOME value above):

```
{"JAVA_HOME": "/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.141-2.b16.el7_4.x86_64"}
```

- Configure Mesos to use the Mesos containerizer with docker image support. Edit the following files as root:

  `/etc/mesos-slave/containerizers:`
  `docker,mesos`

  `/etc/mesos-slave/image_providers:`
  `docker`

  `/etc/mesos-slave/isolation:`
  `filesystem/linux,docker/runtime`

Starting Mesos

On Mesos Master Machine:

```
sudo systemctl start mesos-master
sudo systemctl start mesos-slave

sudo chkconfig mesos-master on
sudo chkconfig mesos-slave on
```

On Mesos Agent Machine

```
sudo systemctl start mesos-slave
sudo chkconfig mesos-slave on
```

Verifying Installation (on Mesos Master Machine)

Verify that Mesos is running normally:

```
MASTER=$(mesos-resolve `cat /etc/mesos/zk` 2>/dev/null)

mesos-execute --master=$MASTER --name="cluster-test" --command="sleep 5"
```

This URL should show Mesos Web UI: http://mesosmasterdevb01:5050

Firewall Configuration for Mesos

On Mesos Master and Agent Machine:

```
sudo firewall-cmd --zone=public --permanent --add-port=5050/tcp
sudo firewall-cmd --zone=public --permanent --add-port=5051/tcp
sudo firewall-cmd --zone=public --permanent --add-port=8080/tcp
```

On Mesos Master Machines Only:

```
sudo firewall-cmd --zone=public --permanent --add-port=2181/tcp
```

Ports 5000, 5051 are for Mesos, 2181 is for Zookeeper, 8080 is for Marathon.

Restart firewall to take effect:

```
$ sudo service firewalld restart
```

Marathon Installation (Mesos Master Machine Only)

- Install Marathon:
  `$ sudo yum -y install marathon`

- Marathon: http://mesosmasterdevb01:8080

Log Configuration

(All servers)
Create the file */etc/rsyslog.d/mesos.conf* with these contents:

```
if $programname == 'marathon' then {
   action(type="omfile" file="/var/log/mesos/marathon.log")
}

if $programname == 'chronos' then {
   action(type="omfile" file="/var/log/mesos/chronos.log")
}

if $programname == 'mesos-master' then {
   action(type="omfile" file="/var/log/mesos/mesos-master.log")
}

if $programname == 'mesos-slave' then {
   action(type="omfile" file="/var/log/mesos/mesos-slave.log")
}
```

Restart rsyslog to verify that this configuration works:

```
$ sudo service rsyslog restart
```

If successful, you should see new log files in /var/log/mesos/:

1. ``/var/log/mesos/marathon.log`` contains the Marathon related messages.
2. ``/var/log/mesos/chronos.log`` contains the Chronos relates messages.
3. ``/var/log/mesos/mesos-master.log`` has the Mesos master messages.
4. ``/var/log/mesos/mesos-slave.log`` has the Mesos slave messages.

Reference

- https://docs.docker.com/engine/installation/linux/docker-ce/centos/
- http://www.cloudera.com/documentation/archive/cdh/4-x/4-7-1/CDH4-Installation-Guide/cdh4ig_topic_21_3.html