넘치지도, 부족하지도 않는 kubernetes 다루기

당신이 모르면 당하는 3가지 장애
Kubernetes의 Secret Recipe

Hello! I am

Junho.son

junho.son@linecorp.com

Work at **Line**

First met with Kubernetes.

순풍에 돗 단듯 잘 나가던 kubernetes

But, Some while…

Objectitive

incidents teach you
how to build a reliable system

제가 겪었던 등에 땀나는 상황을 여러분들이 겪지않고,

미리 대응할 수 있도록 공유하는 자리입니다.

만약 여러분들이 모두 겪은 일이라면...

Objectitive

아주 좋은 클러스터를 가지고 계시군요...👍

Index

오늘은 이야기 하고 싶은 것들:

- 모르면 무조건 당하는 kubernetes 장애
- 신뢰할 수 있는 kubernetes를 만들기
- 운영 자동화를 위한 다음 계획

First, Talk about:

- ETCD mvcc: database space exceeded
- Invalid master/kubelet certification
- Slow scheduling and DNS failure

ETCD database space exceeded

잘 동작하던 kubernetes 에서 아래의 에러가 발생한다.

😭

```
$ kc apply -f deployment/app.yaml
etcdserver: mvcc: database space exceeded

$ kc scale --current-replicas=2 --replicas=3 deployment/nucleo-flask-sample -n
etcdserver: mvcc: database space exceeded
```

ETCD database space exceeded

증상

- ETCD read only
- k8s object들의 update(변경, 생성 등)이 불가능해짐

원인

- ETCD keyspace는 key/value와 함께 그것의 **revision history**를 저장한다.
  - hisotry가 증가함 -> crd, deployment의 빈번한 변경
- 성능 저하 및 저장공간 고갈을 피하기 위해 주기적인 **compaction**과 **defragment**이 필요

ETCD database space exceeded

Solutions

- step1. etcdctl로 compaction과 defragment를 함

```
export PEERS="http://10.127.111.53:2379,http://10.127.114.99:2379,http://10.12
export ETCDCTL_API=3

# compaction
$ rev=$(etcdctl  endpoint status --write-out="json" | egrep -o '"revision":[0-
$ etcdctl compact $rev
compacted revision 518729

# size 안 줄었음
| http://10.127.111.53:2379 | 750655cd04e8a0f0 | 3.2.15 | 2.1 GB | false | 4 |
| http://10.127.114.99:2379 | 77c94c09da32ea5a | 3.2.15 | 2.2 GB | true | 4 |
| http://10.127.114.96:2379 | 7fa913847ab8f9f5 | 3.2.15 | 2.1 GB | false | 4 |

# defragment(resize high watermark)
$ etcdctl --endpoints ${PEERS} defrag
Finished defragmenting etcd member[http://10.127.111.53:2379]
```

ETCD database space exceeded

Solutions

- step2. etcdctl로 alarm disarm을 해야 함

```
etcdctl --endpoints ${PEERS} alarm list
memberID:8631513766031452762 alarm:NOSPACE
memberID:8432521691336843504 alarm:NOSPACE

etcdctl --endpoints ${PEERS} alarm disarm
memberID:8631513766031452762 alarm:NOSPACE
memberID:8432521691336843504 alarm:NOSPACE
```

ETCD database space exceeded

Lessons Learned

kube-apiserver compaction options(default: 5m)

```
--etcd-compaction-interval duration     Default: 5m0s
The interval of compaction requests. \
If 0, the compaction request from apiserver is disabled.
```

ETCD database space exceeded

Lessons Learned

scheduled defrag job

```
#!/bin/bash
# start with random sleep
sleep $((RANDOM % 10 + 1))

CMD=$(which etcdctl)
ENDPOINT=$(cat /etc/etcd/etcd.conf  | grep "ETCD_ADVERTISE_CLIENT_URLS" | cut
LOG="/var/log/etcd-defrag-$(date +"%Y%m%d")"

ETCDCTL_API=3 ${CMD} --endpoints http://127.0.0.1:2379 \
  defrag --command-timeout=60s 1>>${LOG} 2>&1
ETCDCTL_API=3 ${CMD} --endpoints http://127.0.0.1:2379 \
  endpoint status 1>>${LOG}

# delete log
find /var/log/etcd-defrag-* -maxdepth 1 -type f -ctime +7 -delete
```

Ref: defrag using cronjob

ETCD database space exceeded

Lessons Learned

adjust etcd db size

```
$ cat /etc/etcd/etcd.conf
...
ETCD_QUOTA_BACKEND_BYTES="4294967296"
...
```

```
--quota-backend-bytes
    Raise alarms when backend size exceeds the given quota (0 defaults to low
    default: 0
    env variable: ETCD_QUOTA_BACKEND_BYTES
```

ETCD database space exceeded

Lessons Learned

Monitoring

```
Each etcd server exports metrics under the /metrics path on its client port an
# prometheus scrap options
scrape_configs:
  - job_name: etcd
    static_configs:
    - targets: ['10.127.157.129:2379','10.127.157.133:2379','10.127.157.137:23
```

```
# prometheus query
etcd_debugging_mvcc_db_total_size_in_bytes
etcd_debugging_mvcc_keys_total
etcd_debugging_mvcc_slow_watcher_total
...
```

Ref: prometheus scrap

ETCD database space exceeded

Lessons Learned

Backup

- etcd Backup

```
#!/bin/bash
BACKUP_DIR="/var/backup/etcd"
ENDPOINT=http://127.0.0.1:2379
BACKUP_FILE="${BACKUP_DIR}/$(date +'%y%m%d').db"

mkdir -p ${BACKUP_DIR}
ETCDCTL_API=3 etcdctl --endpoints=${ENDPOINT} snapshot save "${BACKUP_FILE}"

cd ${BACKUP_DIR} && etcdctl backup --data-dir /var/lib/etcd/default.etcd

sleep 1

# move member dir
mv ${BACKUP_DIR}/member ${BACKUP_DIR}/member-$(date +'%y%m%d')

# cleanup backup older than 7 days
```

Invalid master/kubelet certification

어느날 갑자기 kubectl로 조회가 불가능 하고,

node가 하나 둘씩 NotReady가 되기 시작한다.

```
$ kc get pods
Unable to connect to the server: x509: certificate is valid

# apiserver log
E0709 10:55:24.437376       1 authentication.go:62] Unable to authenticate the
  [x509: certificate has expired or is not yet valid, x509: certificate has ex
```

Invalid master/kubelet certification

증상

- kube-apiserver 접근 불가능
- kubelet NotReady status

원인

- master component/kubelet config의 인증서 만료
- kubeadm은 master component cert는 1년짜리를 만듬
- kubeadm upgrade를 사용하지 않고 upgrade를 함(180일 보다 작으면 upgrade시 갱신)

Invalid master/kubelet certification

Solutions

kubeadm init phase certs [master component]

```
## apiserver --cert-dir로 기존 ca cert로 생성해야 함(안하면 새롭게 ca cert생성
kubeadm init phase certs all \
  --apiserver-advertise-address 10.20.192.31 \
  --apiserver-cert-extra-sans gonz-dev-caravan.devdev.com \
  --service-cidr 172.28.0.0/15 --cert-dir /root/test_pki/pki
```

위에서 갱신한 내용으로 kubelet admin.conf 변경/배포

```
ansible-playbook -i inventory/service-dev/host renew_cert_node.yml
```

Invalid master/kubelet certification

Lessons Learned

맘편한게 짱! *master component certification*은,

- 100년짜리 certification을 만들자 😅
- kubeadm을 이용하여 자주 upgrade를 하자(권장).

그렇다면 kubelet은? 🤔

certificate - kubernetes the hard way
upgrade using kubeadm

Invalid master/kubelet certification

Lessons Learned

kubelet rotate-certification 기능 이용

- kubelet에 `--rotate-certificates=true` 옵션 추가
- kubelet feature-gateway에 RotateKubeletServerCertificate=true
- RotateKubeletServerCertificate v1.12에서 Beta

```
$ cat /etc/systemd/system/kubelet.service.d/10-kubeadm.conf
[Service]
...
Environment="KUBELET_CERTIFICATE_ARGS=--rotate-certificates=true --cert-dir=/v
Environment="KUBELET_FEATURE_GATE_ARGS=--feature-gates=RotateKubeletClientCert
```

Invalid master/kubelet certification

Lessons Learned

kubelet rotate-certification

```
## 과정은 말로 설명... 필요?
ls -l /var/lib/kubelet/pki/
total 32
-rw------- 1 root root 1183 Jun 25 18:05 kubelet-client-2018-06-25-18-05-21.pe
-rw------- 1 root root 1183 Jun 25 18:46 kubelet-client-2018-06-25-18-46-17.pe
...
lrwxrwxrwx 1 root root   59 Jun 25 18:46 kubelet-client-current.pem -> /var/li

# kubelet log
Jun 26 17:04:27 junho003-k8s-dev-jp2v-dev kubelet[12771]: I0626 17:04:27.20670

# kubelet config
$ cat /etc/kubernetes/kubelet.conf
...
users:
- name: default-auth
```

Invalid master/kubelet certification

Lessons Learned

이런거 관리하기 너무 힘들다...
kops(on aws), kubespray 사용하시면...
편할수 있습니다.

Slow scheduling and DNS failure

large cluster를 만들고(node 1000),
서비스 배포(nginx - replica 5000)시
pod의 scheduling이 너무 느리다
세월아 네월아...😅

Slow scheduling and DNS failure

증상

- 1000개 노드의 클러스터에서 deployment 배포시 pod scheduling이 느리다.

원인

- pod이 scheduling될때 filtering과 ranking(scoring)
- 모든 노드(1000)를 filtering하고 ranking을 계산하다 보니 시간이 오래걸림

Ref

scheduler algorithm

scheduler details

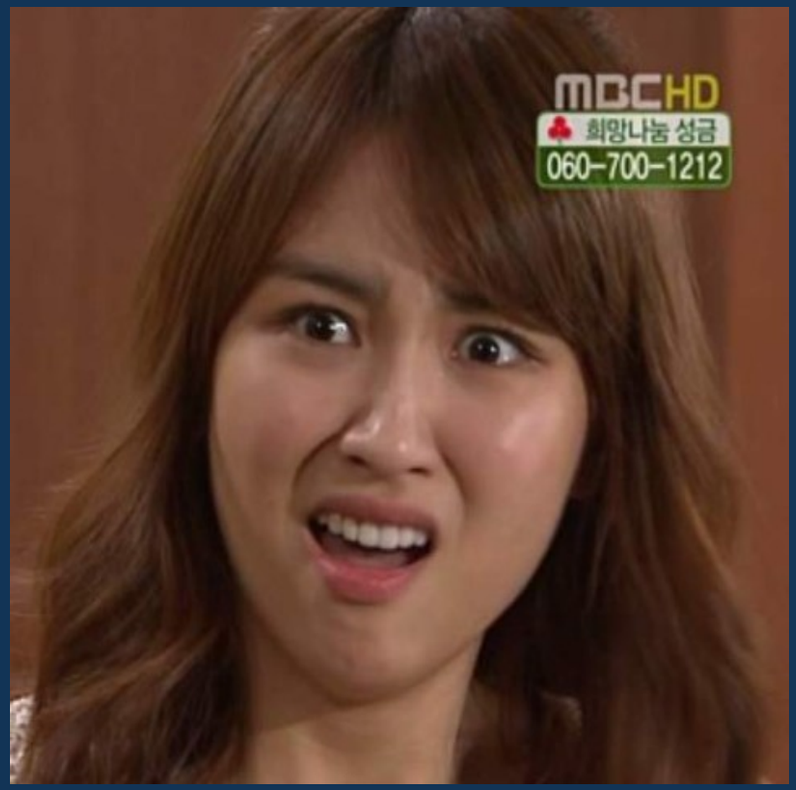
Slow scheduling and DNS failure

Solutions

kube-scheduler에서
percentageOfNodeToScore 옵션 조정

```
# cat schedulerconfig.yaml
...
percentageOfNodesToScore: 10
...
```

Slow scheduling and DNS failure

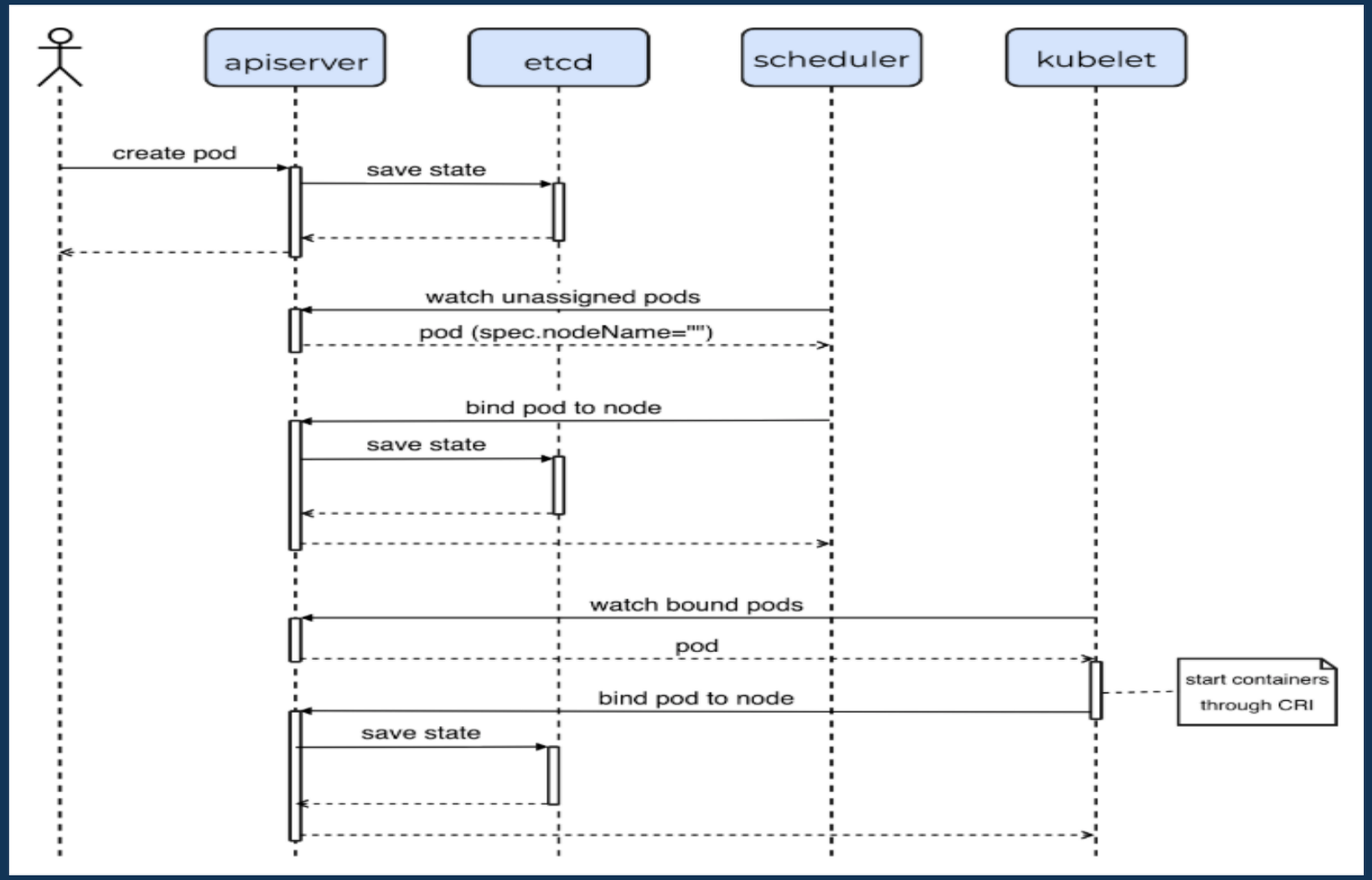


뭐요?

Slow scheduling and DNS failure

Lessons Learned

k8s pod scheduling



Slow scheduling and DNS failure

Lessons Learned

Scheduler Performance Tuning

- percentageOfNodesToScore - k8s v1.12에 추가된 기능
- 이전에는 무조건 모든 노드를 filtering, ranking함
  - 모든 노드의 feasibilty를 확인하는 - filtering
  - feasibility가 충족한 노드들에 대해서 score를 매기는 - ranking

Slow scheduling and DNS failure

Lessons Learned

Scheduler Performance Tuning

- percentageOfNodesToScore를 적용하면, filtering시 모든 노드의 feasibilty 대상으로 하지 않고,
- percentageOfNodesToScore(%)갯수만큼 filtering이 되면, 해당 노드들을 점수를 매겨 ranking한다

Ref: scheduler perf tuning

Slow scheduling and DNS failure

그런데 v1.14 부터
이 클러스터 사이즈를 이용하여 이 값을 계산하는 공식
이 추가 됐다 하네요...

Slow scheduling and DNS failure

당신의 DNS는 안녕하십니까?

보통 `dnsPolicy: clusterFirst` 설정

만약 db를 사용하는 java spring service 들이 동시에 100개 이상 배포된다면?
-> dns pod들이 죽어나가면서, service pod들이 CrashloopbackOff…
-> 난 봤어… 그 잔인한 모습을… 서로가 서로를 죽이는… 그 모습은 마치

Slow scheduling and DNS failure

Slow scheduling and DNS failure

증상

- dns도 죽고, service pod도 죽고 우리 모두 죽자…

원인

- java process에서 동시에 kube-dns에 쿼리함
- 5 dot 미만(default)은 cache(dnsmasq) 한번 조회하고 upstream host /etc/resolve.conf 조회
- 150개의 java pod(sidecar 3)이 동시에 5개 pod kube-dns 쿼리

Slow scheduling and DNS failure

Solutions

걱정하지 마시고 cluster-proportional-autoscaler 한대 들이세요...

아니면, Pod `.spec.dnsConfig`를 이용하여 ndots 옵션을 수정

Slow scheduling and DNS failure

```
...
    spec:
      containers:
      - command:
        - /cluster-proportional-autoscaler
        - --namespace=kube-system
        - --configmap=coredns-autoscaler
        - --target=deployment/coredns          # 자신의 cluster dns target
        - --default-params={"linear":{"coresPerReplica":256,"nodesPerReplica":
        - --logtostderr=true
        - --v=2
        image: k8s.gcr.io/cluster-proportional-autoscaler-amd64:1.2.0
...
```

Ref: cluster proportional autoscaler

Slow scheduling and DNS failure

default param의 coresPerReplica, nodesPerReplica
값으로 적절한 pod의 개수를 계산한 후
dns deploy 값을 조정합니다.

```
replicas = max( ceil( cores * 1/coresPerReplica ) ,
                ceil( nodes * 1/nodesPerReplica ) )
```

Ref: cluster-proportional-autoscaler

Slow scheduling and DNS failure

ndots 3개 미만이면, upstream dns를 보게 조정 (absoulute name)

```yaml
apiVersion: v1
kind: Pod
metadata:
  namespace: default
  name: example
spec:
  containers:
    - name: test
      image: nginx
  dnsConfig:
    options:
      - name: ndots
        value: "3"
```

Ref: ndots option affect your application performances

신뢰할수 있는 kubernetes를 만들기

Nginx ingress tunes

필요성

- 사용자가 local 머신(동일 IP)에서 부하테스트 진행, 특정한 nginx로 리퀘스트가 몰리게 된다.
- 이때 특정한 nginx는 이런 traffic을 backend로 보내나 backend가 이를 처리하지 못하게 되면, 순식간에 DDos 공격처럼 되어 버림
- 하나의 ip에서 동시에 들어오는 커넥션 갯수의 조정이 필요.

Nginx ingress tunes

증상

사용자가 504 gateway timeout 에러 확인
kibana를 확인해 보니 15:38~15:41사이에 ingress 한 장비에서 503(4000건) 504(15건) 발생
해당 장비에 nginx 프로세스 확인해보니 다른프로세스와 생성시간이 많이 차이나는 프로세스 확인
(reload 하지 못한 프로세스가 생김)

```
$ ps -ef | grep nginx
65534     10022 131013  0 15:37 ?        00:00:57 nginx: worker process <-
65534     11562 131013  1 15:38 ?        00:01:43 nginx: worker process <-
65534     62933 131013  0 17:41 ?        00:00:00 nginx: worker process
65534     62934 131013  0 17:41 ?        00:00:00 nginx: worker process
65534     62935 131013  0 17:41 ?        00:00:00 nginx: worker process

# 해당 프로세스  SIGTERM(kill -15) 로 죽인 후 위 에러 발생안함
```

로그 확인해 보니 부하테스트가 있었고, backend pod은 hpa로 확장 됐으나,
nginx는 그 이후 이상 프로세스가 되고 정상 동작 하지 못함

Nginx ingress tunes

동일 ip로 높은 부하가 오는 상황(DDos)을 막기 위한 nginx 튜닝 metadata.annotation에 아래의 값 조정

```
nginx.ingress.kubernetes.io/limit-connections(개수) :
  # 하나의 ip에서 허용되는 동시 연결 개수 -> limit_conn
nginx.ingress.kubernetes.io/limit-rps(개수) :
  # 매초당 주어진 ip에서 부터 허용되는 연결의 수   -> limit_req
nginx.ingress.kubernetes.io/limit-rpm(개수) :
  # 매분당 주어진 ip에서 부터 허용되는 연결의 수  -> limit_req
```

- nginx ingress rate limiting

Nginx ingress tunes

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  annotations:
  ...
    'nginx.ingress.kubernetes.io/limit-connections' : '300',
    'nginx.ingress.kubernetes.io/limit-rpm' : '3500',
    'nginx.ingress.kubernetes.io/limit-rps' : '500',
  ...
```

Pod QoS

필요성

- k8s에 올라간 app들의 reource 경합시 중요한 app의 oom 방지

Pod QoS

QoS 분류 및 설정

- Pod은 3개의 QoS class를 갖는다.
  - Guaranteed - limit/request를 모두 동일한 값으로 준 경우,
  - Burstable - request는 지정되어 있으나 limit가 없는경우, 혹은 같지가 않은 경우, limit를 지정안하면 node capacity가 지정됨
  - BestEffort - request/limit가 모두 지정 안된경우

Pod QoS

Resource

- compressible: cpu는 리소스 보장이 충족되지 않을 경우, pod이 종료되지 않는다. 일시적으로 throttring 될 뿐이다.
- incompressible: memory은 압축할수 없는 리소스이기 때문에 분류가 있다.

Pod QoS

동작방식

kubernetes는 QoS 별로 OOM_SCORE_ADJ를 다르게 설정하여 linux oom kill 우선순위를 조정합니다.

```
# 값이 1000에 가까울수록 oom kill 대상
-1000 <= OOM_SCORE_ADJ <= 1000
```

how is kernel oom score calculated

Pod QoS

BestEffort

```
OOM_SCORE_ADJ: 1000
```

- 노드의 메모리가 부족한 경우 제일 먼저 죽음 lowest priority, 이 컨테이너는 노드의 여유 메모리 만 사용 할수 있음

Pod QoS

Burstable

```
2<OOM_SCORE_ADJ<999,

min(max(2, 1000 - (1000 * memoryRequestBytes) / machineMemoryCapacityBytes), 9
```

- 최소한의 리소스를 보장한다. 하지만 가능하다면 더 많은 리소스를 사용할수 있습니다(limit - node capacity).
- 시스템메모리가 부족할 경우, 그리고 BestEffort가 없을 경우, request를 넘어선 사용량이면 이 컨테이너가 죽을 것이다.
- 현제 메모리 사용량에 따라 계산되는 방식이 다름.

Pod QoS

Guaranteed

```
OOM_SCORE_ADJ:-998
```

- 최우선순위(top priority)로 간주되며 limit를 초과하지 전까지 죽지 않는다.

kubelet, docker는 OOM_SCORE_ADJ; -999

Pod QoS

OOM_SCORE_ADJ 값을 확인하고 싶을때는...

```
docker inspect CONTAINERID | grep -i oom
cat /proc/PID/oom_score_adj
```

- resource QoS
- OOM_Killer
- OOM killer and overcommit

ETCD

ETCD Tuning

Disk나 Network상태에 따른 Tune

ETCD Tuning

필요성

SSD 서버가 없다면…?
ETCD 이중화를 위해 노드를 다른 zone에 두고 싶다면…
아래와 같은 warn이 etcd에서 자주 발생한다면…

```
2017-04-12 03:04:09.678778 W | etcdserver: \
failed to send out heartbeat on time \
(deadline exceeded for 185.938874ms)
```

disk(SSD권장) file I/O와 network latency를 고민
물리적인 교체가 당장 힘들다면 etcd tuning

ETCD Tuning

Let’s get some tune

heartbeat interval : leader가 follow에게 자신의 리더임을 알리는 주기(default: 100ms)
election timeout : follower가 leader 선출을 하기 전까지 기다리는 시간(default: 1000ms)

- leader는 heartbeat에 metadata를 함께 전송함.

ETCD Tuning

heartbeat interval

너무 작으면 자주 보내서 cpu/network resource 많이 사용
너무 크면 leader fail을 늦게 감지

Guide: member들 간 평균 round-trip time(using ping) 평균의 max

election timeout

Guide: rrt의 최소 10배 이상

ETCD Tuning

```
$ cat /etc/etcd/etcd.conf
[member]
...
ETCD_HEARTBEAT_INTERVAL="500"
ETCD_ELECTION_TIMEOUT="3000"
```

ETCD Tuning

만약 disk의 metric을 보려면...

```
## disk
etcd_disk_wal_fsync_duration_seconds < 10ms
etcd_disk_backend_commit_duration_seconds < 25ms
```

Service Lifecycle 관리

Initial delay

필요성

server process에게 시작할 시간을 주자

Initial delay

java spring process의 경우 server port가 LISTEN 이어도
server initializing 하는 시간이(DB 접속 등) 필요하다.

Initial delay

```
$ cat nucleo/app.yml
...
# Initial Delay Second(default: 30s)
# If you need to more than 30 sec to initialize your server process.
initial_delay: 60

# Health check path(default: /)
# if you don't want to send health checking request on root path(/).
# Important! App have to exist a path you specify.
health_path: /healthz
```

Initial delay

```
$ kc get pod \
 nucleo-flask-sample-371dac7-default-68f5674455-jnh6n -n junho-son -o yaml
...
    livenessProbe:
      failureThreshold: 3
      httpGet:
        path: /healthz
        port: 8080
        scheme: HTTP
      periodSeconds: 10
      initialDelaySeconds: 60
...
```

History 관리

필요성

- 이전 버전 replicaset, 실패한 job pod, helm release 등 history성 목록들이 남아있음
- 안지우면 etcd, api list 시 문제가 생긴다.

History 관리

- deployment

```
spec.revisionHistory
```

- batchjob

```
spec.[activeDeadlineSeconds | backoffLimit]
```

- cronjob

```
spec.[successfulJobsHistoryLimit(3) | failedJobsHistoryLimit(1)]
```

- helm

```
helm init --history-max 10
```

History 관리

적용

```
## deploy revisionHistory
$ kc get deployment -n junho-son meister-f778746-default -o yaml
apiVersion: extensions/v1beta1
kind: Deployment
...
spec:
  progressDeadlineSeconds: 600
  replicas: 3
  revisionHistoryLimit: 3
...

## batch job
$ kc get job -n junho-son block-junho-son-mysql-dev-my-mysqldump-1547071200 -o
apiVersion: batch/v1
kind: Job
```

The Future of Operating

- operator sdk
- kustomize + argo
- cluster auto scaling
- admin tools(app diagnostics)

Q & A

Pod Priority and PriorityClass

필요성

- pod scheduling시 상대적 우선순위(weight)를 지정
- 중요한 서비스에 우선순위를 매겨, 먼저 스케줄링 되게 한다.
- 클러스터에 allocable resource가 부족할 경우,
- 우선순위가 높은 pod이 pending이라면, 낮은순위 pod이 preemption됨
- priorityclass별 Quota 지정가능(v1.13 beta)

Pod Priority and PriorityClass

동작방식

- pod을 노드에 스케줄링할때 사용하는 우선순위 (weight)
- PriorityClass(1.14 stable)로 pod .spec에 지정
- 스케줄링 순서 선점과 리소스 부족시 preemption/evition 순서에 영향을 준다.
- High priority가 pending되고 있으면, Low priority는 preemption될 확률이 높다.
- QoS object와 독립적이고, namespace와 별개

Pod Priority and PriorityClass

```
$ kc get pc
NAME                      VALUE        GLOBAL-DEFAULT   AGE
system-cluster-critical   2000000000   false            135d
system-node-critical      2000001000   false            135d

$ kc get pod --all-namespaces -o \
  jsonpath='{range .items[*]}{.metadata.name}{"\t"}{.spec.priorityClassName}{"
kube-apiserver-junho-devel001   system-cluster-critical
kube-apiserver-junho-devel002   system-cluster-critical
kube-apiserver-junho-devel003   system-cluster-critical
```

Pod Priority and PriorityClass

동작 방식 - 스케줄링

- scheduler는 pending pod들의 priority를 가지고 순서를 정한다.
- 높은 우선순위의 pod이 스케줄링 요구사항을 만족하면 먼저 스케줄링됨
- 만약 요구사항을 만족 못할 경우 preemption이 동작하여 낮은 우선순위를 preemption한다.
- preemption을 해도 만족 못하면 *다음 우선순위 pod*이 스케줄링된다.

Pod Priority and PriorityClass

특징

- namespace에 제약 받지 않는 object.
- name(.metadata.name)으로부터 우선순위 정수 값(.value)을 priority에 매핑하여 정의한다.(by admission controller)
- pod spec에 존재 하지 않는 priorityclass가 지정될 경우 reject(by admission controller)
- system priority class(system-node-critical, cluster)은 kube-system에서만 사용 가능하다. 참고

Pod Priority and PriorityClass

```
type PodSpec struct {
  ...
  PriorityClassName string
  Priority          *int32  // Populated by Admission Controller. Users are no
}
```

Pod Priority and PriorityClass

```
type PriorityClass struct {
  metav1.TypeMeta
  // +optional
  metav1.ObjectMeta

  // The value of this priority class. This is the actual priority that pods
  // receive when they have the above name in their pod spec.
  Value         int32
  GlobalDefault     bool
  Description       string
}
```

Pod Priority and PriorityClass

테스트

목적

- 사용자 앱은 github repo 이름으로 namespace가 나뉘는데,
- 그 namespace에 QoS별 다른 priority quota를 갖게 하자.

Pod Priority and PriorityClass

priority class 생성

```
$ kc get priorityclass
NAME                      VALUE        GLOBAL-DEFAULT   AGE
bustable                  1000         false            2s
guaranteed                900000000    false            2s
system-cluster-critical   2000000000   false            23d
system-node-critical      2000001000   false            23d
```

Pod Priority and PriorityClass

Quota 생성 및 priorityclass 지정

```
$ cat namespace-quota.yaml
---
apiVersion: v1
kind: Namespace
metadata:
  name: test-quota1
---
apiVersion: v1
kind: List
items:
- apiVersion: v1
  kind: ResourceQuota
  metadata:
    name: pods-guaranteed
    namespace: test-quota1
  spec:
```

Pod Priority and PriorityClass

deployment 배포

```
# deployment for guaranteed
$ cat test-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment-on-guaranteed
  namespace: test-quota1
  labels:
    app: nginx
spec:
  replicas: 3
  ...
    spec:
      ...
      priorityClassName: guaranteed # priority 지정
```

Pod Priority and PriorityClass

배포

```
$ kc create -f namespace-quota.yaml
$ kc create -f test-deployment-guaranteed.yaml
$ kc create -f test-deployment-bustable.yaml
$ kc describe quota -n test-quota1
Name:       pods-bustable
Namespace:  test-quota1
Resource    Used  Hard
--------    ----  ----
cpu         3     10
memory      3Gi   10Gi
#-> bustable pod 3개 잡힘

Name:       pods-guaranteed
Namespace:  test-quota1
Resource    Used    Hard
--------    ----    ----
```

Pod Priority and PriorityClass

- Ref:
- 테스트 상세
- priority class design

Prevent Node Failure
and
Managing Resource

Kubelet GC

지긋지긋한 node disk관리... 이젠 GC로 해결

필요성

- Node에 resource 부족 대비

Ref
image collection
container collection

Kubelet GC

동작방식

image collection

- 매 5분 마다 동작
- HighThresholdPercent >= disk usage 실행
- LowThresholdPercent까지 size가 줄면 종료

container collection

- 매 1분 마다 동작
- minAge보다 오래된 (not running) container를 제거
- MaxPerPodContainer, MaxContainer까지 남겨두고 정리

Kubelet GC

어떻게 적용했나 - kubelet options

```
apiVersion: kubelet.config.k8s.io/v1beta1
...
imageGCHighThresholdPercent: 85
imageGCLowThresholdPercent: 70
imageMinimumGCAge: 2m0s
maxContainer: -1
maxPerPodContainer: 1  # Crashloopbackoff시 log -p 옵션을 위한 설정
...
```

Ref: kubelet GC

Kubelet GC

하지만, 곧 deprecated 된다고 하네요...😢

```
# 기존의 옵션이 사라지고, 이런식으로...
EvictionHard
  nodefs.available<15%
EvictionSoft
  nodefs.available<25%
  imagefs.available<25%
```

- 더 자세한건: evict options

Graceful eliminate

필요성

- ingress(nginx or haproxy)를 통한 서비스 인입구조
- sh -c “command” container cmd로 server process가 SIG TERM을 못받음
  - container termination은 pid 1로 SIG TERM보냄
- 서버 프로세스는 graceperiod 이후 SIG KILL로 죽음

Graceful eliminate

- pod 종료시 pre_stop hook을 이용해 볼까?

근데 그냥 .spec.container.cmd를 서버프로세스 실행 명령어로 바꾸면 안됨?

맞아요... 그게 제일 간단합니다.😂
그리서 우리도 그걸로 바꾸려고요.👍

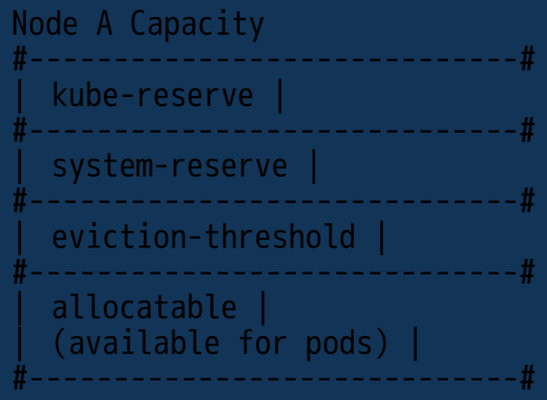
Reservce Resource for system

필요성

처음엔 system 영역의 리소스를 확보하고자 했다. 하지만...

Reservce Resource for system

동작방식



Node의 capacity는 위 처럼 구성

* kube-reserved: kubelet, container runtime, npd
* system-reserved: linux processes, sshd
* eviction-threshold: 앞에서 설명한
* allocatble: node가 pod의 스케줄링을 위한 resource

2개의 reserved와 threshold를 지정하지 않으면 전체 노드 리소스가 allocatable

Reservce Resource for system

우리의 설정

```
$ kc describe node [Nodename]
...
Allocatable:
cpu: 40
ephemeral-storage: 857955043546
hugepages-1Gi: 0
hugepages-2Mi: 0
memory: 65582228Ki
pods: 110
...

$ cat /var/lib/kubelet/config.yml
evictionHard:
imagefs.available: 15%
memory.available: 100Mi
nodefs.available: 10%
```

왜? kube-reserved, system-reserved는 하지 않죠?

Reservce Resource for system

하... 근데 이게 문제인게...

- kube-reserved, system-reserved를 지정하면, 해당 프로세스들은 Bustable하지 못함니다.
- 오히려 성능을 저하시킬수 있죠, 의미없는 evition이 발생할수도 있고
- 만약 사용하고 싶으시다면 기존의 사용량을 잘 모니터링 해야 합니다.
- 어차피 kube,system-reserved는 priority도 높고, oom_score_adj도 낮습니다.
- 다른 프로세스 또는 pod 보다, evition, oom kill될 확률이 낮죠.