

Lazy AWS

A.k.a Unemployed DevOps Professionals



Problems

My Problems

=

Your Problems

you've heard this before

Magical problem-fixing tool!





What is the ☁️?

Datacenter with buttons?

Buzzword?

Something different?

Not total hype



≠

Datacenter

Ask Experts

pbuckley> Cache all the things

pbuckley> Consider the flow of a request thru the system. Client to api, api to db. This basic flow has at least three points of fail networks, HTML5 application cache, etc)

pbuckley> Consider a fall-back static object cache that enables a safe mode level of functionality in case of DB failure

pbuckley> Design your system so that you pay for failure without giving the appearance of being down or unavailable

pbuckley> Not all of the user experience require sync with the back-end

pbuckley> Fast and flexible

pbuckley> No upfront investment

pbuckley> Better precise cost control over the resources you NEED

pbuckley> This is not your father's datacenter

pbuckley> You don't control the network, you ask nicely

pbuckley> You can't buy an extra power supply

pbuckley> You can't buy a diesel generator

pbuckley> You can't build a crazy raid storage device (Think Hitachi USPv 5 chassis SAN, yeh nope)

pbuckley> If you try to compensate in the same ways as you would in a datacenter, your costs will run out of control

pbuckley> AWS can and does fail

pbuckley> April 21, 2011 June 29, 2012 October 22, 2012 December 24, 2012 aws outages (software bugs, human error, thunderstorms)

pbuckley> Embrace failure. Your instance will randomly fail. API's will become unavailable. Your strengths are different then in a datacenter

pbuckley> Define what your availability requirements are going to be and design your solution around it.

pbuckley> Consider the flow of a request thru the system. Client to api, api to db. This basic flow has at least three points of fail

pbuckley> Many small nodes versus a few larger nodes. A larger pool of small nodes that has a single node fail will take away less %

pbuckley> CAP Theorem lives. Choose eventual consistency and availability.

pbuckley> Some gotchas with the subsystems

pbuckley> S3 is awesome. Not fast, but speed remains consistent when many nodes are hitting it. Compress everything. Use lzo for hadoop

pbuckley> Autoscale is beautiful, not only because of what it does, but due to the architectural constraints it places on you. Your b

pbuckley> Elastic load balancers take a little while to warm up. They don't support web sockets without tcp pass through. Though othe

pbuckley> Global load balancing is your friend (DNS, personal plug for Dynect). Don't depend on the uptime of a single availability zone

pbuckley> Use identity manager roles for access. Give least privilege. Setup accounts by app and user

pbuckley> Don't try to send email from an instance. Many of them have been blacklisted by RBLs. Instead use simple email service.

pbuckley> Cloudformation can be cool, templating language is finicky and doesn't support comments. Investigate use of fog (ruby) or b

pbuckley> ELB -> ELB x N Too support cool things like having the ability to remove pools of machines for canary deploys

pbuckley> Manage costs!

pbuckley> Elastic map reduce + Spot instances or redshift is typically cheaper and requires less engineering hours then running your

pbuckley> Prices and features vary by region

pbuckley> Age out files stored in S3 to glacier. They are still visible thru the S3 api.

pbuckley> Investigate instance marketplace, potential to buy instances to get upfront savings without going with reserved instances.

pbuckley> Reserved instances have a higher upfront cost but will eventually be cheaper if the instance capacity is used over n time.

pbuckley> Maximize IOPS, use tricks like compression to reduce the amount of data that needs to be read off disk

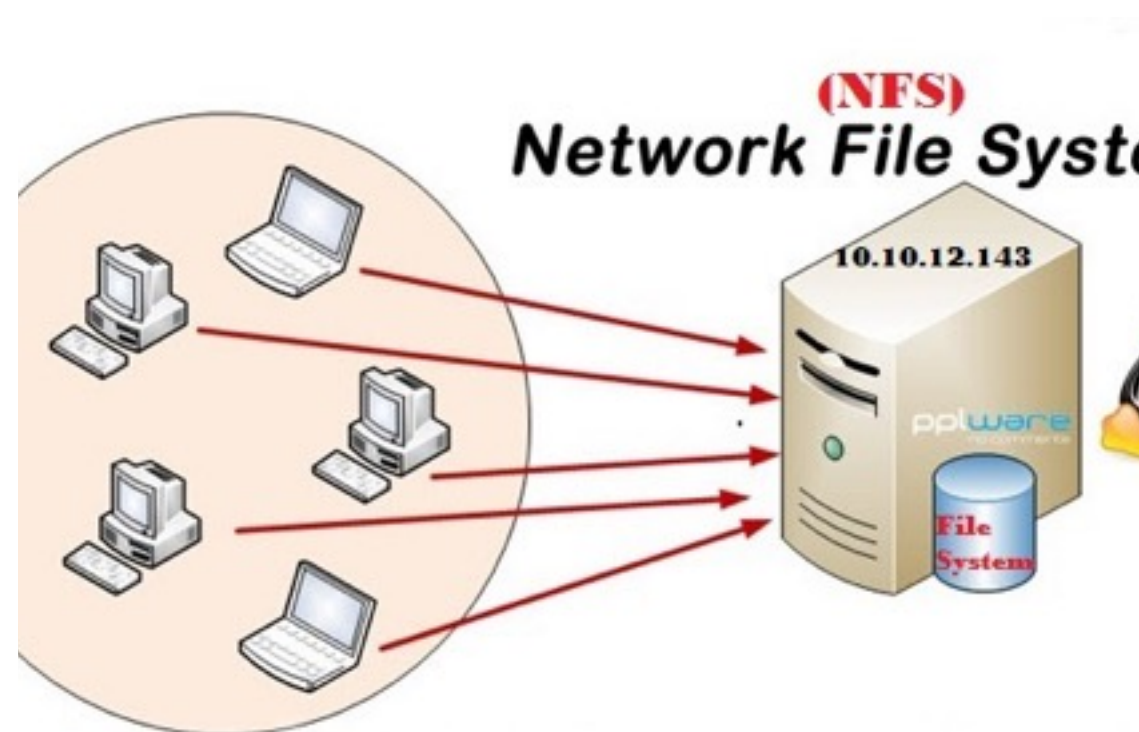
pbuckley> Use m1.xlarge or c1.xlarge or ssds and raid 0 ephemeral drives for cheaper fast IOPS

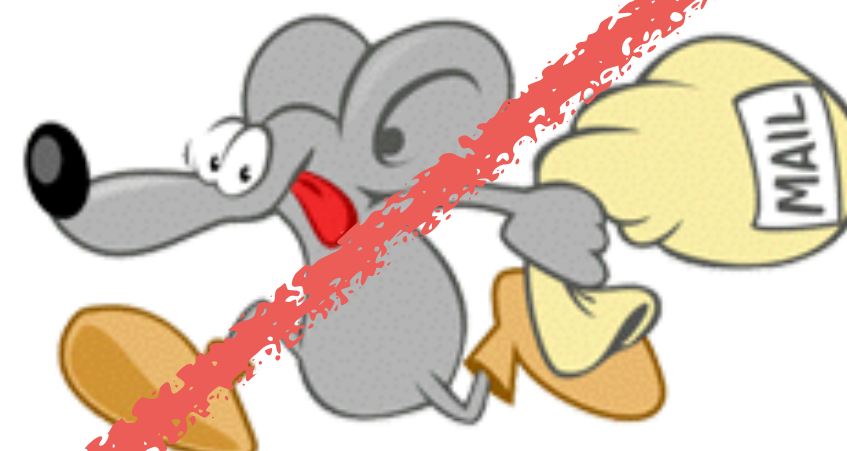
pbuckley> EBS optimized, provisioned iops for more expensive but "promised" IOPS. Increases the number of failure points. EBS has a h

pbuckley> AWS is a nickel and dime economy, watch out or it will take your leg.

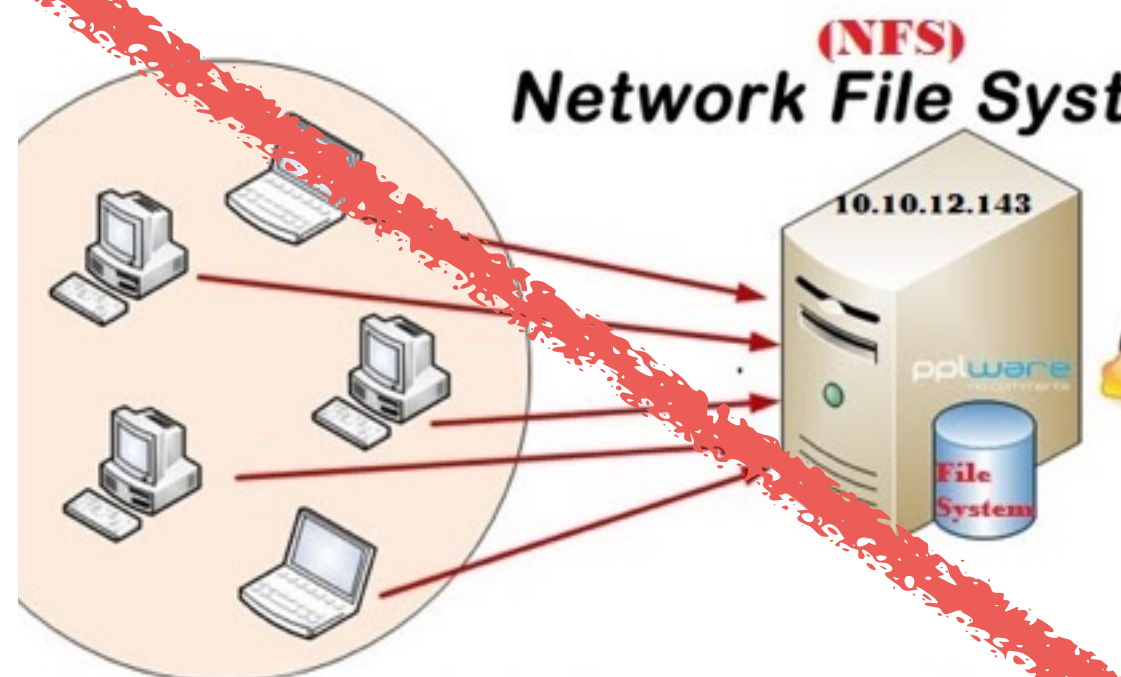


POSTFIX

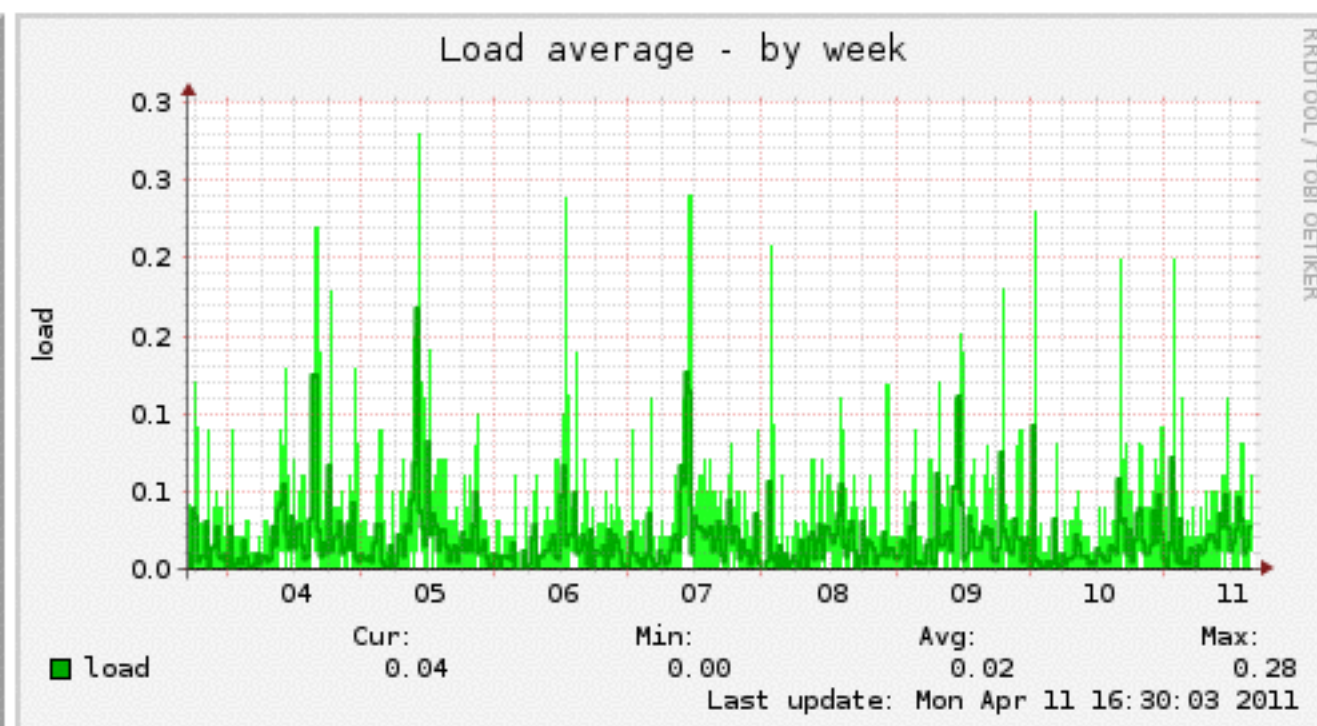
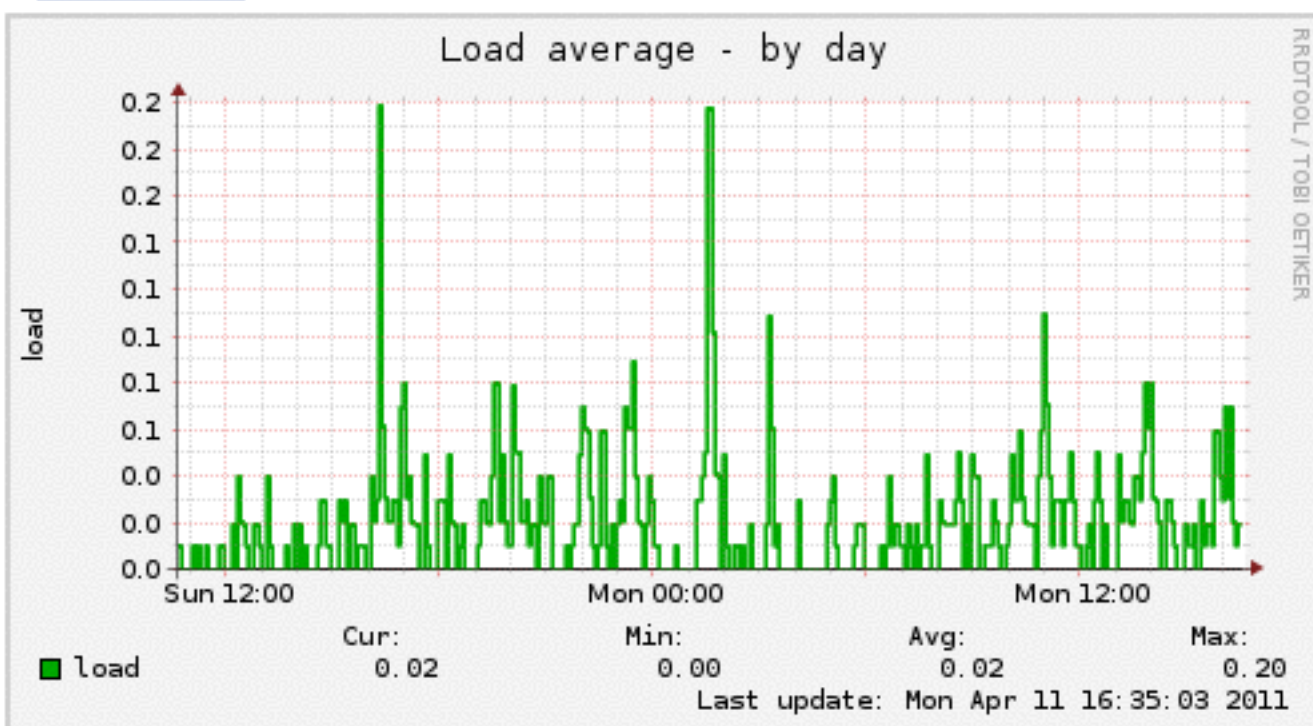




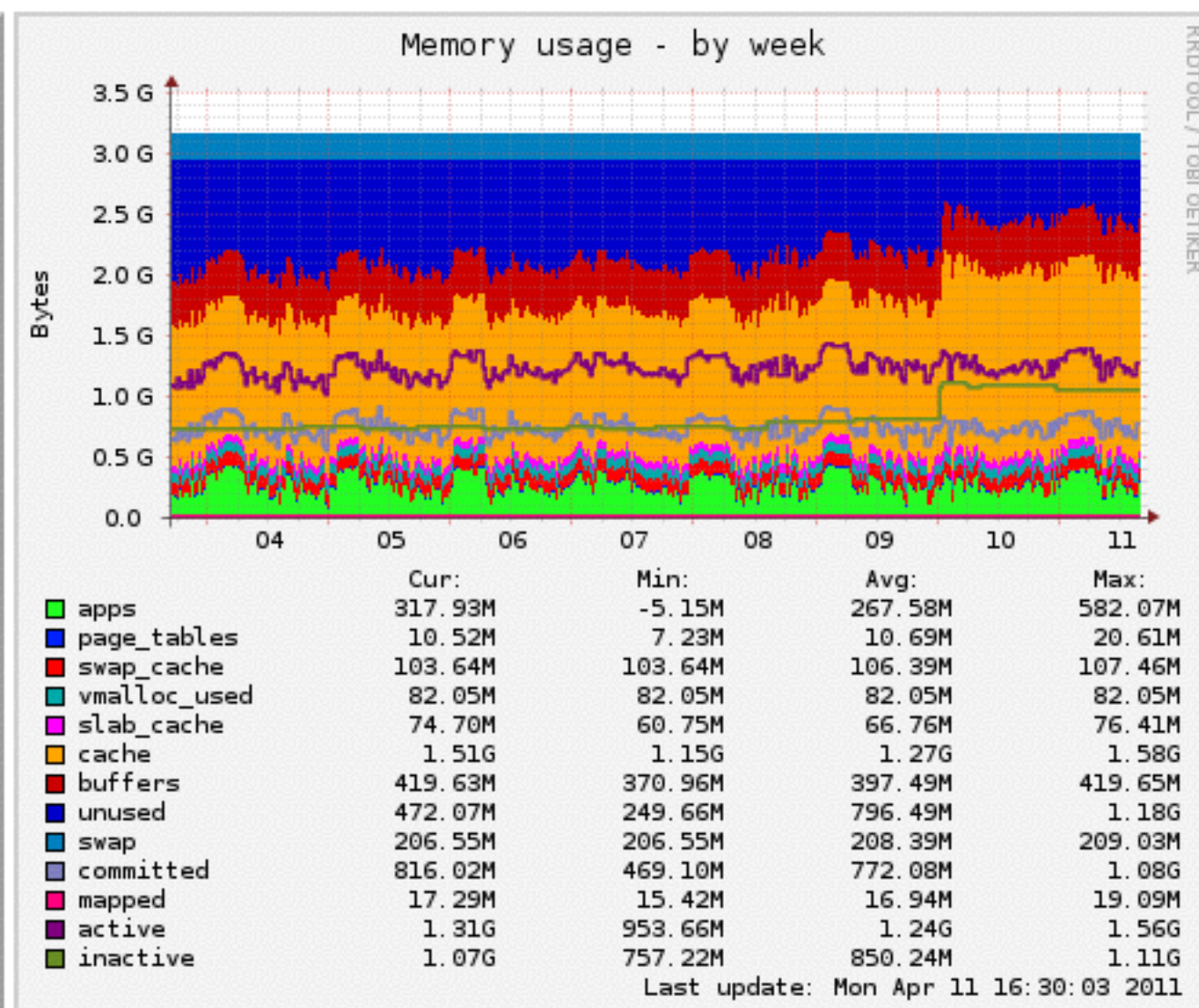
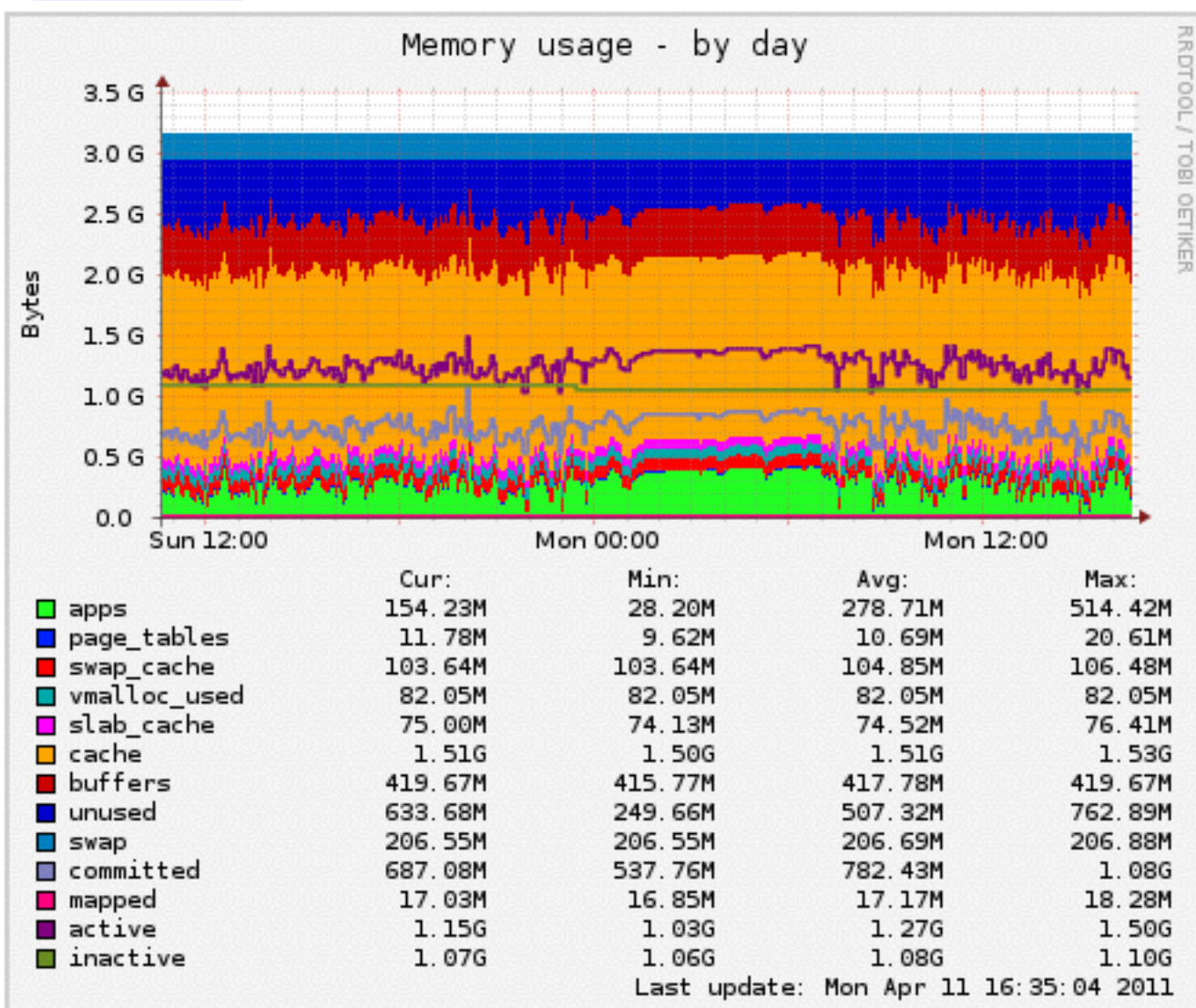
POSTFIX



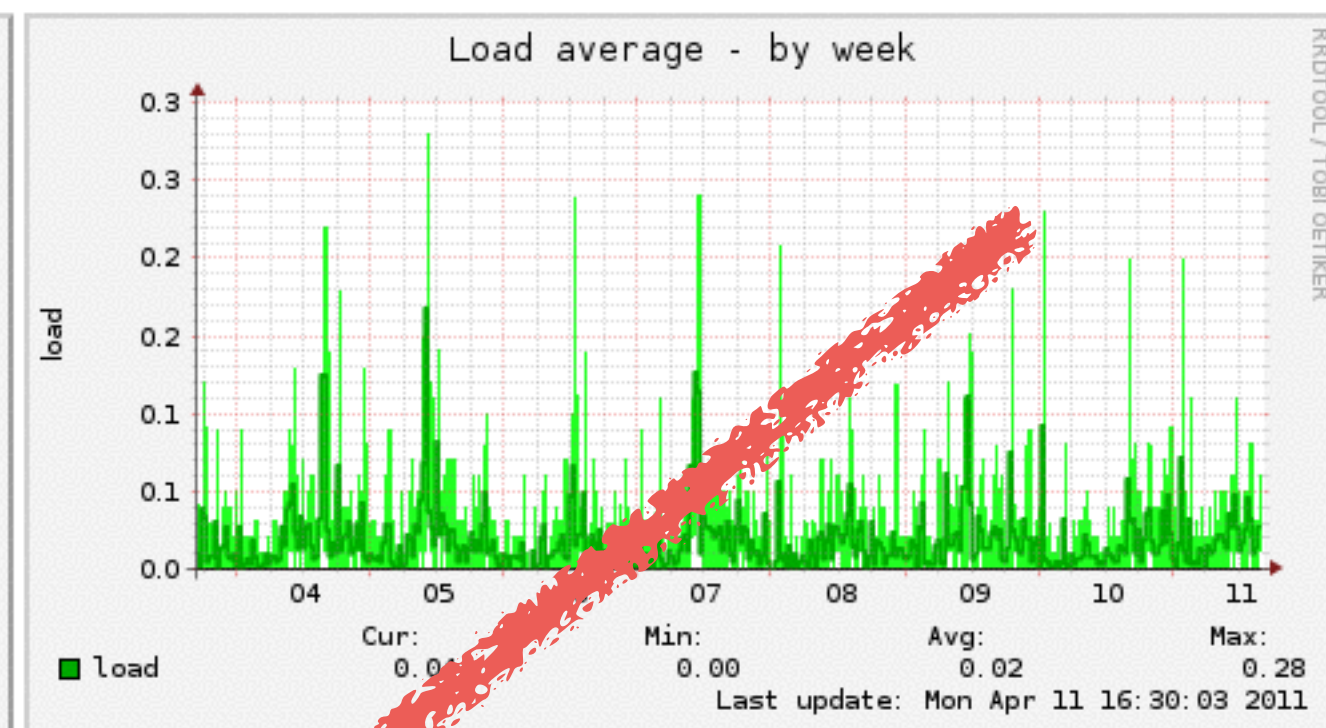
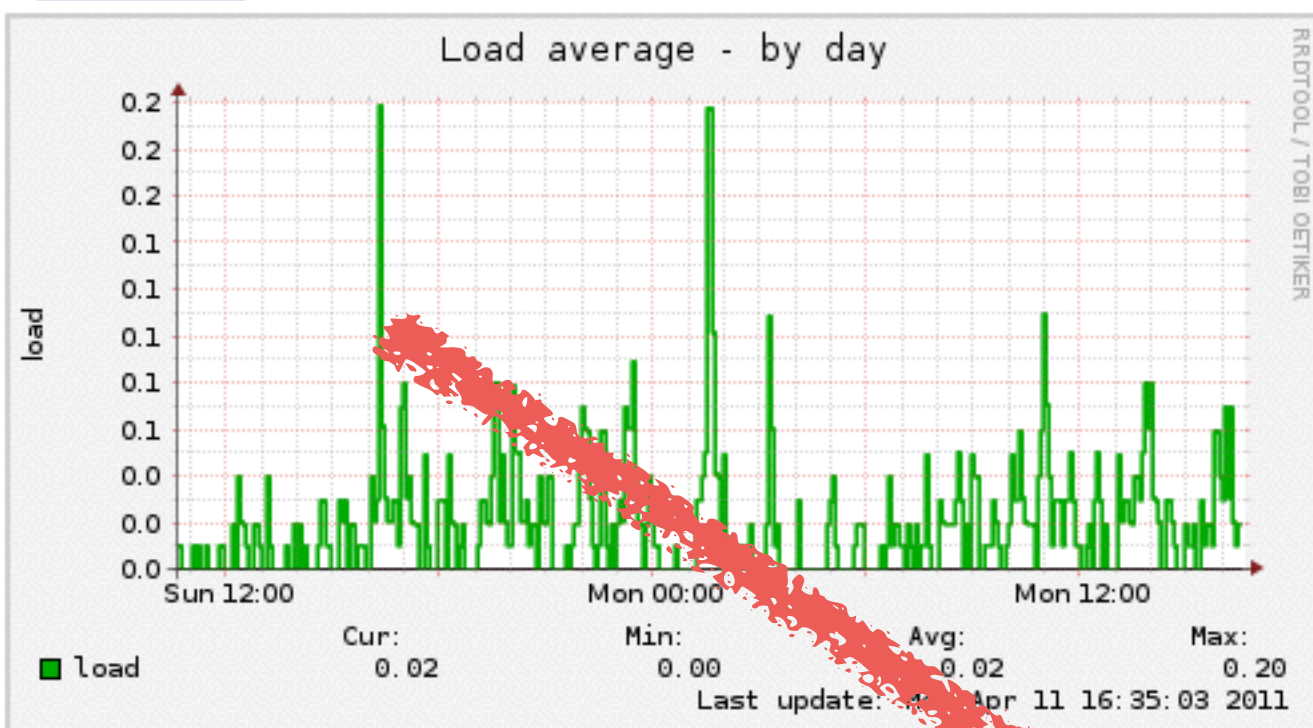
Load average



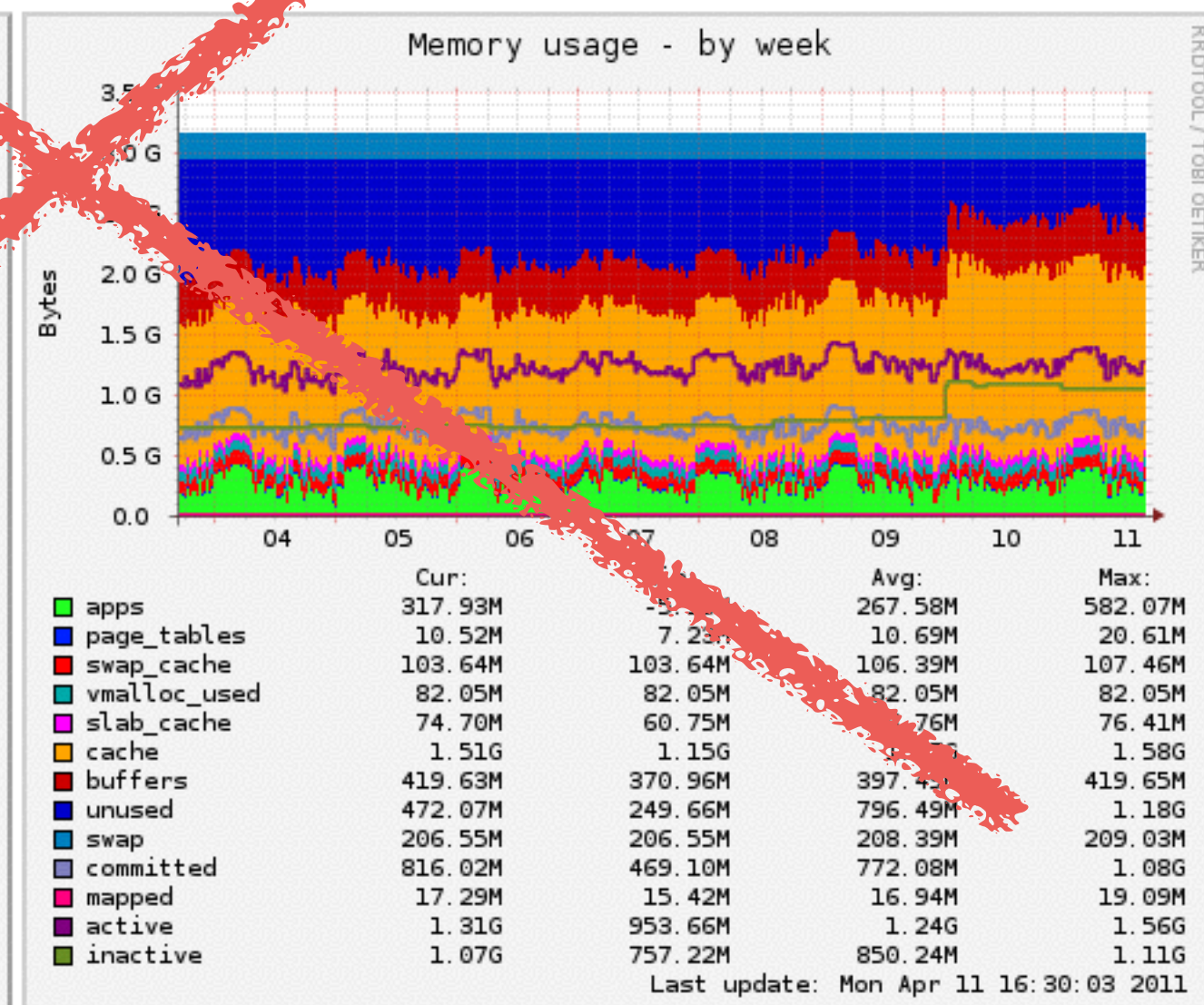
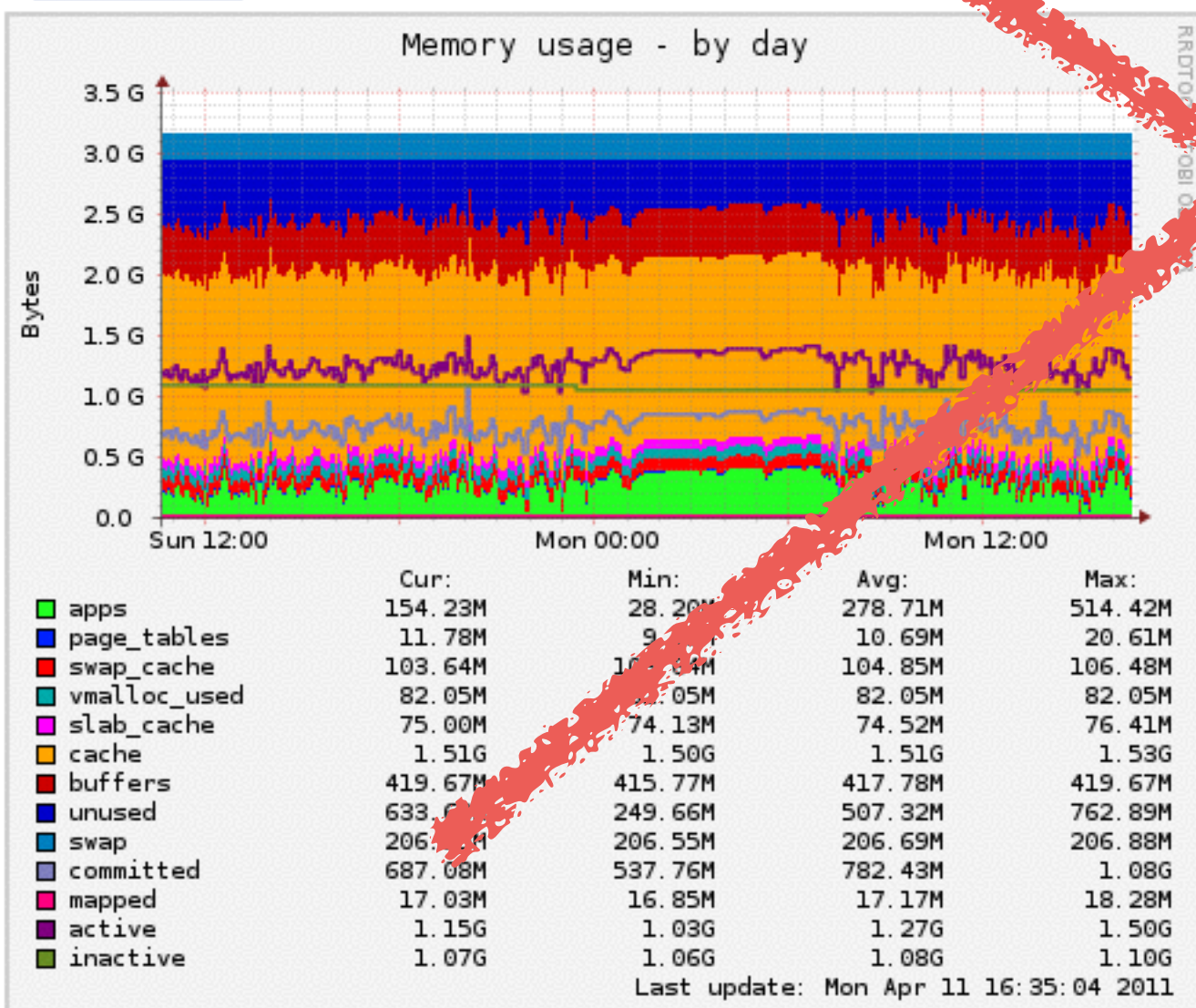
Memory usage



Load average



Memory usage







“B- B- But I need my
configuration
management engine!!”



“B- B- But I need my
configuration
management engine!!”



DevOps Hipsters

Simply your life

Why make things complicated?

Simplify!

Amazon Linux

yum-s3-iam

Shell scripts/boto

CodeDeploy

AutoScaling

Udo

yum-s3-iam

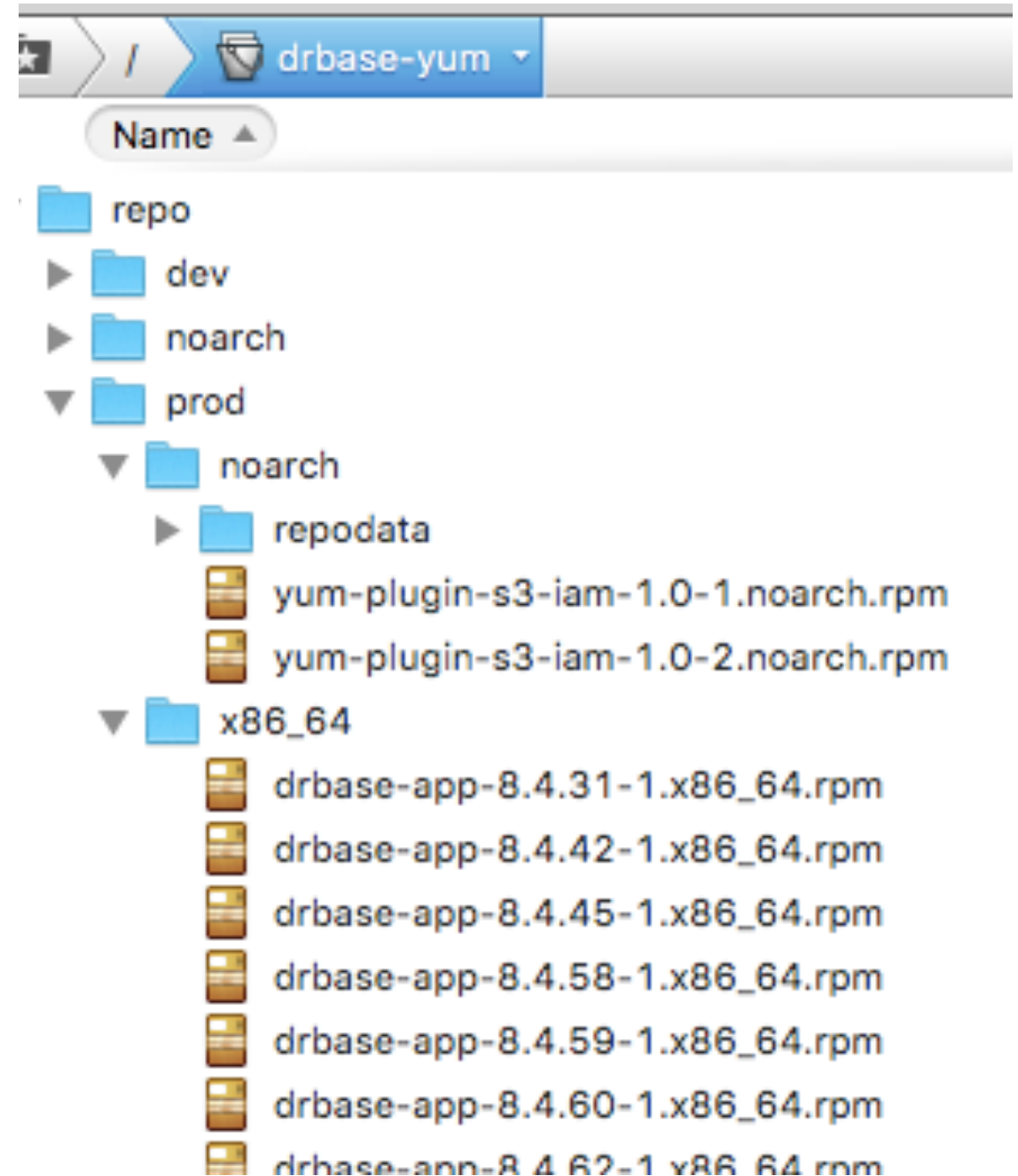
* Yum (RPM)

* S3

* IAM

* rpmbuild

* s3sync



RPM

- * Provisioning
- * Dependencies
- * Config files
- * Run commands
- * Initscripts
- * Application

```
# sync up to s3 drbase-pub/$env/repo
S3PUT=s3put --region us-west-2 -c 4
sync-yum-plugin:
    $(S3PUT) --bucket drbase-pub --prefix $(NOARCH) --key_p
sync-dev: sync-yum-plugin sumologic-download-rpm repo
    $(S3PUT) --bucket drbase-yum --prefix $(REPODIR) --key_p
sync-qa: sync-yum-plugin sumologic-download-rpm repo
    $(S3PUT) --bucket drbase-yum --prefix $(REPODIR) --key_p
sync-stage: sync-yum-plugin sumologic-download-rpm repo
    $(S3PUT) --bucket drbase-yum --prefix $(REPODIR) --key_p
sync-prod: sync-yum-plugin sumologic-download-rpm repo
    $(S3PUT) --bucket drbase-yum --prefix $(REPODIR) --key_p
```

```
# build everything a cluster needs
all-rpms: rpmprep base logger static configs app wsnotifi

# rpm builds:
base: rpmprep
    rpmbuild -bb SPECS/drbase-base.spec
logger: rpmprep
    rpmbuild -bb SPECS/drbase-papertrail.spec
    rpmbuild -bb SPECS/drbase-sumologic.spec
static: rpmprep
    rpmbuild -bb SPECS/drbase-static.spec
configs: rpmprep
    rpmbuild -bb SPECS/drbase-cfg-dev.spec
    rpmbuild -bb SPECS/drbase-cfg-qa.spec
    rpmbuild -bb SPECS/drbase-cfg-stage.spec
    rpmbuild -bb SPECS/drbase-cfg-prod.spec
    rpmbuild -bb SPECS/drbase-cfg-prod-replicated.spec
```

Summary: Static file webserver

Name: myapp-static

Version: 1.0

Release: 1

Source0: myapp-static-nginx.conf

BuildRoot: %{_tmppath}/%{name}-%
{version}-%{release}-root

Requires: nginx myapp-static-files

%description

Sets up nginx to serve static files
from disk

%prep

rm -rf \$RPM_BUILD_ROOT

%build

mkdir -p \$RPM_BUILD_ROOT/etc/nginx/
conf.d

cp %{SOURCE0} \$RPM_BUILD_ROOT/etc/
nginx/conf.d/myapp-static.conf

%install

%post

/sbin/service nginx start >/dev/null
2>&1

/sbin/chkconfig nginx on

%preun

if [\$1 -eq 0] ; then

 /sbin/service nginx stop >/dev/
null 2>&1

 /sbin/chkconfig --del nginx

fi

%clean

rm -rf \$RPM_BUILD_ROOT

%files

%attr(-,root,root)

/etc/nginx/conf.d/myapp-static.conf

make(1)

Pre-packaged commands
Dependencies
Shell

Not just for compiling!

```
vagrant-update-dev: vagrant-destroy vagrant-login
vagrant-reload-dev: vagrant-destroy run
vagrant-destroy:
    vagrant destroy -f
init-dev-vagrant: init-vagrant-db init-schema init-travis
init-vagrant-db:
    sudo su - postgres --session-command="createuser $(USER) && createdb $(USER)"
init-local-db: pgsetup create-local-db init-schema
pgsetup:
    $(PGSETUP)
create-local-db:
    dropdb --if-exists "$(USER)"
    createdb
    psql < bootstrap/extensions.sql
init-schema:
    psql -c "ALTER USER \"$(USER)\" SET search_path=$(USER)"
    psql < bootstrap/mwschema.sql
    if [ -e bootstrap/updates.sql ]; then psql < bootstrap/updates.sql
populate-dev: populate-sql dev-migrations
init-travis: dep-deploy init-travis-config init-travis-db
init-travis-config:
    cp config/travis/mw_local.yml ./
init-travis-db:
    psql -U postgres travis < bootstrap/extensions.sql
populate-sql:
    psql < bootstrap/populate.sql
    psql < bootstrap/materialized_view_for_reports.sql
init-local-config:
    cp config/local/mw_local.yml ./mw_local.yml
    sed -e 's/host=localhost/host=$(PGHOST)/' mw_local.yml > mw_local.yml
    mv mw_local.yml-pghost mw_local.yml
init-vagrant-config:
    cp config/vagrant/mw_local.yml ./mw_local.yml
```


AutoScaling Groups

Not just for autoscaling

Deployment Group

qa

Deployment ID

d-RFPETA80F

Deployment Config

[CodeDeployDefault.AllAtOnce](#)

Minimum Healthy Hosts

0 of 3 instances

Revision Created

Apr 8, 2016 11:09:46 PM UTC

Description

Application revision registered by Deployment ID: d-CMGU35OYE

Filter [Status](#) ^

Instances per page 10

< Viewing 1 to 3 instances >

Instance ID	Start Time	End Time	Duration	Status	Most Recent Event	Events
i-1c62bedb	1 minute ago			In Progress	ApplicationStop	View Events
i-999b445e	1 minute ago			In Progress	ApplicationStart	View Events
i-f59c4332	1 minute ago			In Progress	ApplicationStart	View Events

CodeDeploy

Alarm:workers-outstanding

Details

History

State Details: State changed to ALARM at 2016/04/11. Reason: Threshold Crossed: 5 datapoints were greater than or equal to the threshold (100.0). The most recent datapoints: [554.0, 660.0].

Description:

Threshold: PMI \geq 100 for 5 minutes

Actions: In ALARM:

- For group **prod-worker** use policy **outstand-scale-up** (Add 2 instances)

In OK:

- For group **prod-worker** use policy **outstand-scale-down** (Remove 1 instance)

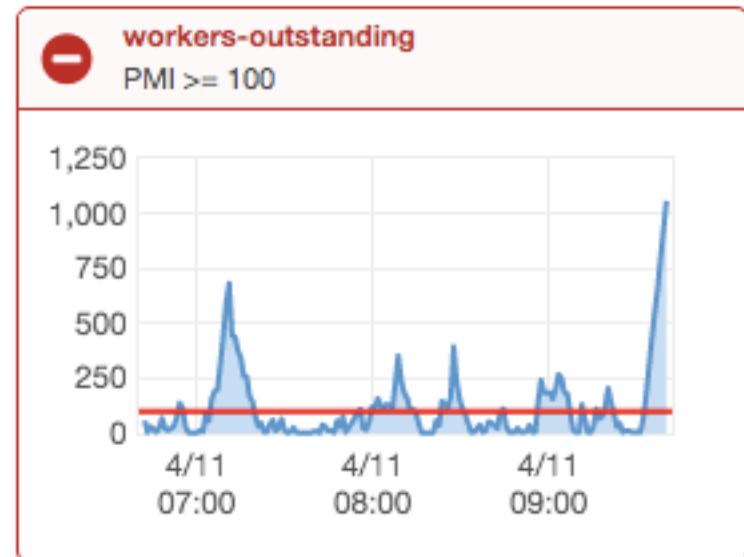
Namespace: WorkersOutstanding

Metric Name: PMI

Dimensions:

Statistic: Maximum

Period: 1 minute



CloudWatch

Custom data are sweet!
Trigger AutoScaling actions!
Trigger alerts!

UDO

Unemployed DevOps

udo.sample.yml

Configuration

Declarative config

(Okay maybe it's a bit like
configuration
management... sue me)

Just wrappers around handy
API methods

Lazy CLI interface

```
# (instances, nsgroup, ec2instanceprofile, etc in a VPC)
clusters:
  # create a "development" cluster
  dev:
    # human-readable name
    description: "Development"
    # override default
    repo_url: https://s3-us-west-2.amazonaws.com/mya
    # yum packages to install
    packages:
      - 'myapp-prov-dev'
      - 'myapp-cfg-dev'
    keypair_name: 'dev'
    subnets_cidr:
      - '10.10.0.0/16'
      - '10.20.0.0/16'
    # for instance roles
    iam_profile: 'developer'
    # optional
    tenancy: 'dedicated' # VPC 'default' or 'dedica
    # application roles for instances
    # (sub-clusters that can be managed as distinct
    roles:
      webapp:
        instance_type: 'm3.medium'
        scale_policy:
          min_size: 1
          max_size: 3
          desired: 2
        elbs:
          - 'dev-https'
        security_groups:
          - 'sg-abcdef123' # you'll want to s
        packages:
          - 'myapp-app'
          - 'myapp-app-deps'
          - 'myapp-app-core'
```

```
(db)nobhill~/dev/db (master)* $ udo deploy qa 1e2a5410636a48799ac14b988e9eeb81081f8e8f  
Deploying commit 1e2a541063 to deployment group: qa  
'Waiting for deployment...'  
..... Deployment of commit 1e2a541063  
6a48799ac14b988e9eeb81081f8e8f to deployment group: qa successful.
```



UdoBot BOT 3:21 PM

bobo@nobhill.lan

Deploying commit 1e2a541063 to deployment group: qa



UdoBot BOT 3:24 PM

bobo@nobhill.lan

Deployment of commit 1e2a5410636a48799ac14b988e9eeb81081f8e8f to deployment group:
qa successful.

CodeDeploy Integration

(And Slack )

Quick Demo

Questions?

Happy 4/20!