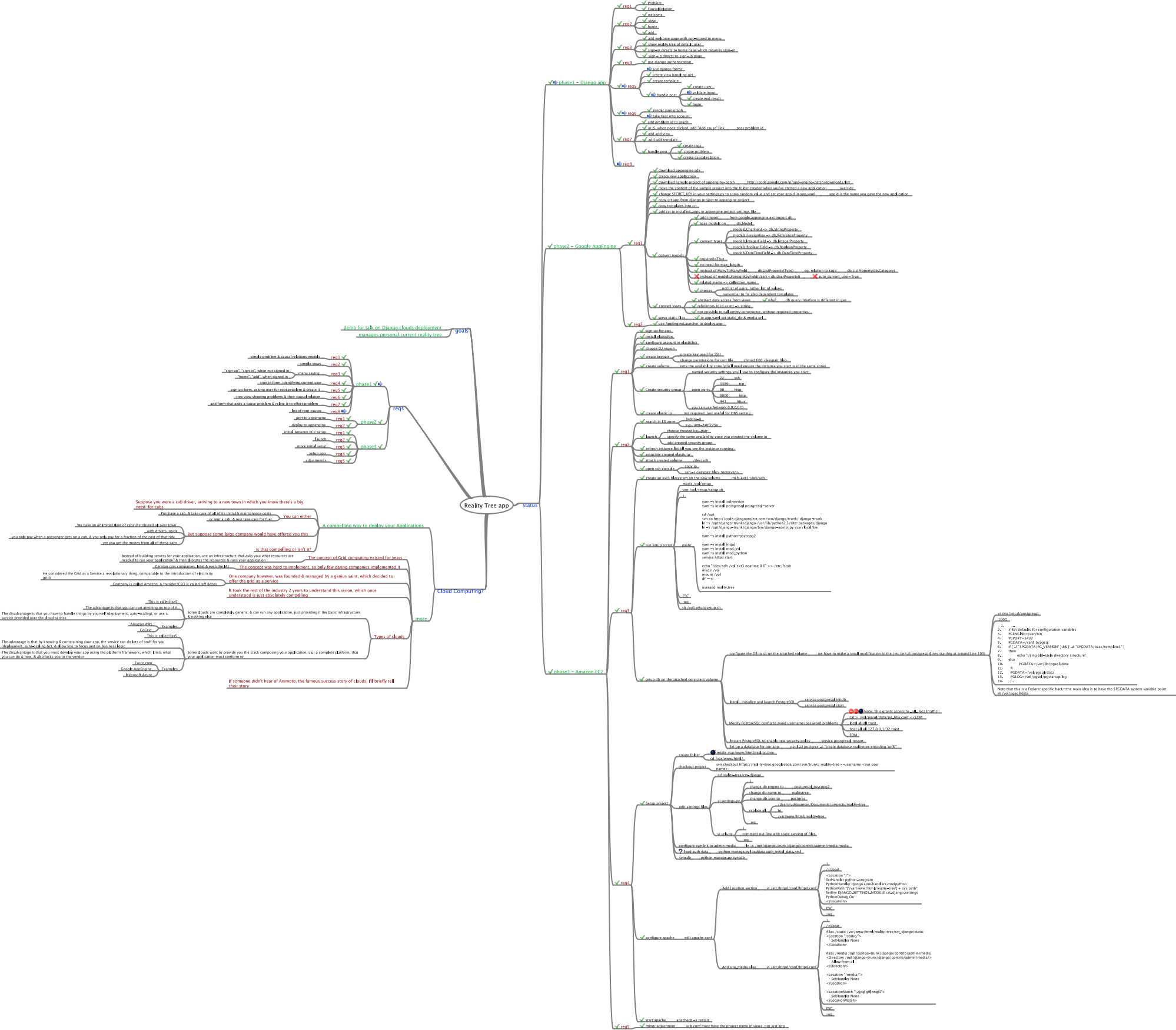
Reality Tree app
goals
demo for talk on Django clouds deployment
manages personal current reality tree
reqs
phase1
simple problem & causal relations models
req1
simple views
req2
"sign up", "sign in", when not signed in
"home", "add", when signed in
menu saying
req3
sign in form, identifying current user
req4
sign up form, asking user for root problem & create it
req5
tree view showing problems & their causal relation
req6
add form that adds a cause problem & relate it to effect problem
req7
list of root causes
req8
phase2
port to appengine
req1
deploy to appengine
req2
phase3
initial Amazon EC2 setup
req1
launch
req2
more initial setup
req3
setup app
req4
adjustments
req5
status
phase1 – Django app
req1
Problem
CausalRelation
welcome
view
home
add
add welcome page with not-signed in menu
show reality tree of default user
sign-in directs to home page which requires sign-in
sign-up directs to sign-up page
req2
req3
req4
use django authentication
req5
use django forms
create view handling get
create template
handle post
create user
validate input
create and result
login
req6
render json graph
take tags into account
add problem id to graph
in JS, when node clicked, add "Add cause" link
pass problem id
req7
add add view
add add template
handle post
create tags
create problem
create causal relation
req8
phase2 – Google AppEngine
req1
download appengine sdk
create new application
download sample project of appengine-patch
http://code.google.com/p/app-engine-patch/downloads/list
move the content of the sample project into the folder created when you've started a new application
overide
change SECRET_KEY in your settings.py to some random value and set your appid in app.yaml
appid is the name you gave the new application
copy crt app from django project to appengine project
copy templates into crt
add crt to installed apps in appengine project settings file
convert models
add import
from google.appengine.ext import db
base models on
db.Model
convert types
models.CharField –> db.StringProperty
models.ForeignKey –> db.ReferenceProperty
models.IntegerField –> db.IntegerProperty
models.BooleanField –> db.BooleanProperty
models.DateTimeField –> db.DateTimeProperty
required=True
no need for max_length
instead of ManyToManyField
db.ListProperty(Type)
eg. relation to tags:
db.ListProperty(db.Category)
instead of models.ForeignKey(AbstractUser) –> db.UserProperty()
auto_current_user=True
related_name –> collection_name
choices
not list of pairs, rather list of values
remember to fix also dependent templates
convert views
abstract data access from views
why?
db query interface is different in gae
references to id as int –> string
not possible to call empty constructor, without required properties
serve static files
in app.yaml set static_dir & media_url
req2
use AppEngine Launcher to deploy app
phase3 – Amazon EC2
req1
sign up for aws
install elasticfox
configure account in elasticfox
choose EU region
create keypair
private key used for SSH
change permissions for cert file
chmod 600 <keypair file>
create volume
note the availability zone (you'll need ensure the instance you start is in the same zone)
Create security group
named security settings you'll use to configure the instances you start
open ports
22 ssh
3389 tcp
80 http
8000 http
443 https
you can use Network 0.0.0.0/0
create elastic ip
not required, just useful for DNS setting
req2
search in EU zone
fedora-8...
e.g. ami-2bb2f25e
launch
choose created key-pair
specify the same availability zone you created the volume in
add created security groups
refresh instance list till you see the instance running
associate created elastic ip
attach created volume
/dev/sdh
open ssh console
copy in
ssh -i <keypair file> root@<ip>
req3
create an ext3 filesystem on the new volume
mkfs.ext3 /dev/sdh
mkdir /vol/setup
vim /vol/setup/setup.sh
i
yum -y install subversion
yum -y install postgresql postgresql-server
cd /opt
svn co http://code.djangoproject.com/svn/django/trunk/ django-trunk
ln -s /opt/django-trunk/django /usr/lib/python2.5/site-packages/django
ln -s /opt/django-trunk/django/bin/django-admin.py /usr/local/bin
yum -y install python-psycopg2
yum -y install httpd
yum -y install mod_ssl
yum -y install mod_python
service httpd start
echo "/dev/sdh /vol ext3 noatime 0 0" >> /etc/fstab
mkdir /vol
mount /vol
df —h
useradd reality_tree
ESC
wq
:
sh /vol/setup/setup.sh
run setup script
paste
setup db on the attached persistent volume
configure the DB to sit on the attached volume
we have to make a small modification to the /etc/init.d/postgresql (lines starting at around line 109)
vi /etc/init.d/postgresql
109G...
1. ...
2. # Set defaults for configuration variables
3. PGENGINE=/usr/bin
4. PGPORT=5432
5. PGDATA=/var/lib/pgsql
6. if [-z "$PGDATA/PG_VERSION"] && [-d "$PGDATA/base/template1"]
7. then
8. echo "Using old-style directory structure"
9. else
10. PGDATA=/var/lib/pgsql/data
11. fi
12. PGDATA=/vol/pgsql/data
13. PGLOG=/vol/pgsql/pgstartup.log
14. ...
Note that this is a Fedora-specific hack—the main idea is to have the $PGDATA system variable point at /vol/pgsql/data
Install, initialize and launch PostgreSQL
service postgresql initdb
service postgresql start
Modify PostgreSQL config to avoid username/password problems
Note: This grants access to all local traffic!
cat > /vol/pgsql/data/pg_hba.conf <<EOM
local all trust
host all all 127.0.0.1/32 trust
EOM
Restart PostgreSQL to enable new security policy
service postgresql restart
Set up a database for our app
psql -U postgres -c "create database realtrees encoding 'utf8'"
req4
Setup project
create folder
mkdir /var/www/html/reality-tree
cd /var/www/html/
checkout project
svn checkout https://reality-tree.googlecode.com/svn/trunk/ reality-tree --username <svn user name>
edit settings files
cd reality-tree/crt-django
vi settings.py
i
change db engine to
postgresql_psycopg2
change db name to
realtrees
change db user to
postgres
/Users/adibaumann/Documents/projects/reality-tree
replace all
to
/var/www/html/reality-tree
ESC
wq
vi urls.py
i
comment out line with static serving of files
ESC
wq
configure symlink to admin media
ln -s /opt/django-trunk/django/contrib/admin/media media
load auth data
python manage.py loaddata auth_initial_data.xml
syncdb
python manage.py syncdb
configure apache
edit apache conf
Add Location section
vi /etc/httpd/conf/httpd.conf
i
/ c$cgat
<Location "/">
SetHandler python-program
PythonHandler django.core.handlers.modpython
PythonPath "['/var/www/html/reality-tree'] + sys.path"
SetEnv DJANGO_SETTINGS_MODULE crt_django.settings
PythonDebug On
</Location>
ESC
wq
Add site_media alias
vi /etc/httpd/conf/httpd.conf
i
/ c$cgat
Alias /static /var/www/html/reality-tree/crt_django/static
<Location "/static/">
SetHandler None
</Location>
Alias /media /opt/django-trunk/django/contrib/admin/media
<Directory /opt/django-trunk/django/contrib/admin/media/>
Allow from all
</Directory>
<Location "/media/">
SetHandler None
</Location>
<LocationMatch "\.(jpg|gif|png)$">
SetHandler None
</LocationMatch>
ESC
wq
start apache
apachectl -k restart
req5
minor adjustment
urls.conf must have the project name in views, not just app
Cloud Computing?
status
more
A compelling way to deploy your Applications
Suppose you were a cab driver, arriving to a new town in which you know there's a big need for cabs
You can either
Purchase a cab, & take care of all of its initial & maintenance costs
or rent a cab, & just take care for fuel
But suppose some large company would have offered you this
We have an unlimited fleet of cabs distributed all over town
with drivers inside
you only pay when a passenger gets on a cab, & you only pay for a fraction of the cost of that ride
and you get the money from all of these cabs
is that compelling or isn't it?
The concept of Grid computing existed for years
Instead of building servers for your application, use an infrastructure that asks you: what resources are needed to run your application? & then allocates the resources & runs your application
The concept was hard to implement, so only few daring companies implemented it
German cars companies tried & even the US
One company however, was founded & managed by a genius saint, which decided to offer the grid as a service
He considered the Grid as a Service a revolutionary thing, comparable to the introduction of electricity grids
Company is called Amazon, & founder/CEO is called Jeff Bezos
It took the rest of the industry 2 years to understand this vision, which once understood is just absolutely compelling
Types of clouds
Some clouds are completely generic, & can run any application, just providing it the basic infrastructure & nothing else
This is called IaaS
The advantage is that you can run anything on top of it
The disadvantage is that you have to handle things by yourself (deployment, auto-scaling), or use a service provided over the cloud service
Examples
Amazon AWS
GoGrid
Some clouds want to provide you the stack composing your application, i.e., a complete platform, that your application must conform to
This is called PaaS
The advantage is that by knowing & constraining your app, the service can do lots of stuff for you (deployment, auto-scaling &c), & allow you to focus just on business logic
The disadvantage is that you must develop your app using the platform framework, which limits what you can do & how, & also locks you to the vendor
Examples
Force.com
Google AppEngine
Microsoft Azure
If someone didn't hear of Animoto, the famous success story of clouds, I'll briefly tell their story