Google App Engine for Java: Groovy Baby!

Peter Sönnergren Lind, Devoteam
@peter_lind

Patrick Chanezon, Google
Developer Advocate, @chanezon

Guillaume Laforge, SpringSource
Groovy guy, @glaforge

JFokus 2010

Stockholm

Agenda

- What does it take to host a Java Web application?
- Introducing Google App Engine for Java
- App Engine Review for Java
  - A Complete Java development stack
- Demos
- Limitations, Issues
- Roadmap
- Scala, Clojure, JRuby
- Groovy
- Gaelyk
- Questions

Google

What does it take (*for a Java developer)* to host a Web application?

Doing it yourself in Java *is still* complex!

Hosting *with Java* still means hidden costs

- Idle capacity
- Software patches & upgrades
- License fees
- IT staff server wrangling
- Traffic & utilization forecasting
- Upgrades

Google App Engine – *now with Java!*

And it's still…

- Easy to **build**
- Easy to **maintain**
- Easy to **scale**

Google

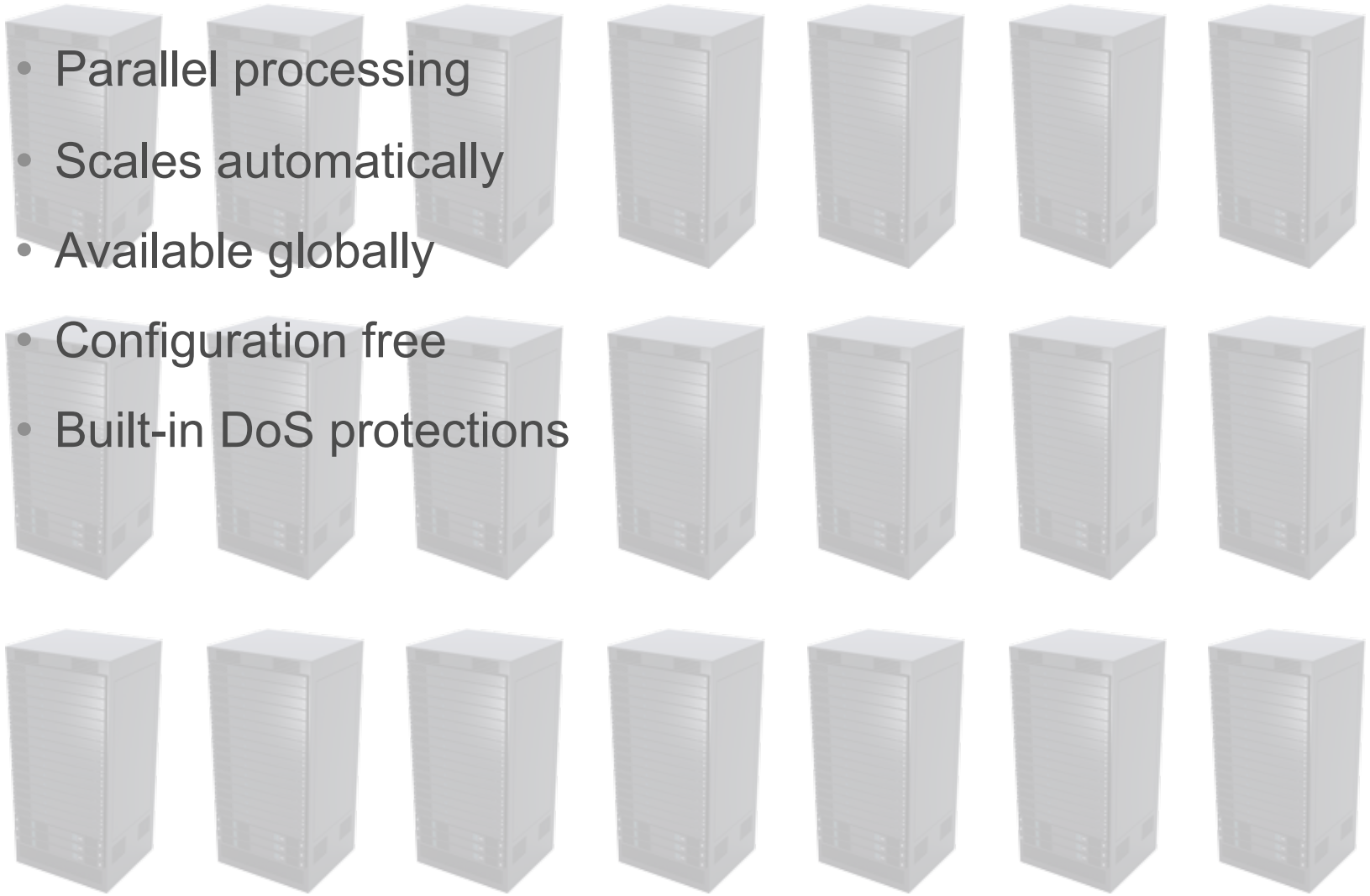
All the same services for Java App Engine as with Python

18+ months in review

Apr 2008	Python launch
May 2008	Memcache, Images API
Jul 2008	Logs export
Aug 2008	Batch write/delete
Oct 2008	HTTPS support
Dec 2008	Status dashboard, quota details
Feb 2009	Billing, larger files
Apr 2009	Java launch, DB import, cron support, SDC
May 2009	Key-only queries
Jun 2009	Task queues
Aug 2009	Kindless queries
Sep 2009	XMPP
Oct 2009	Incoming Email

Same Distributed web hosting platform

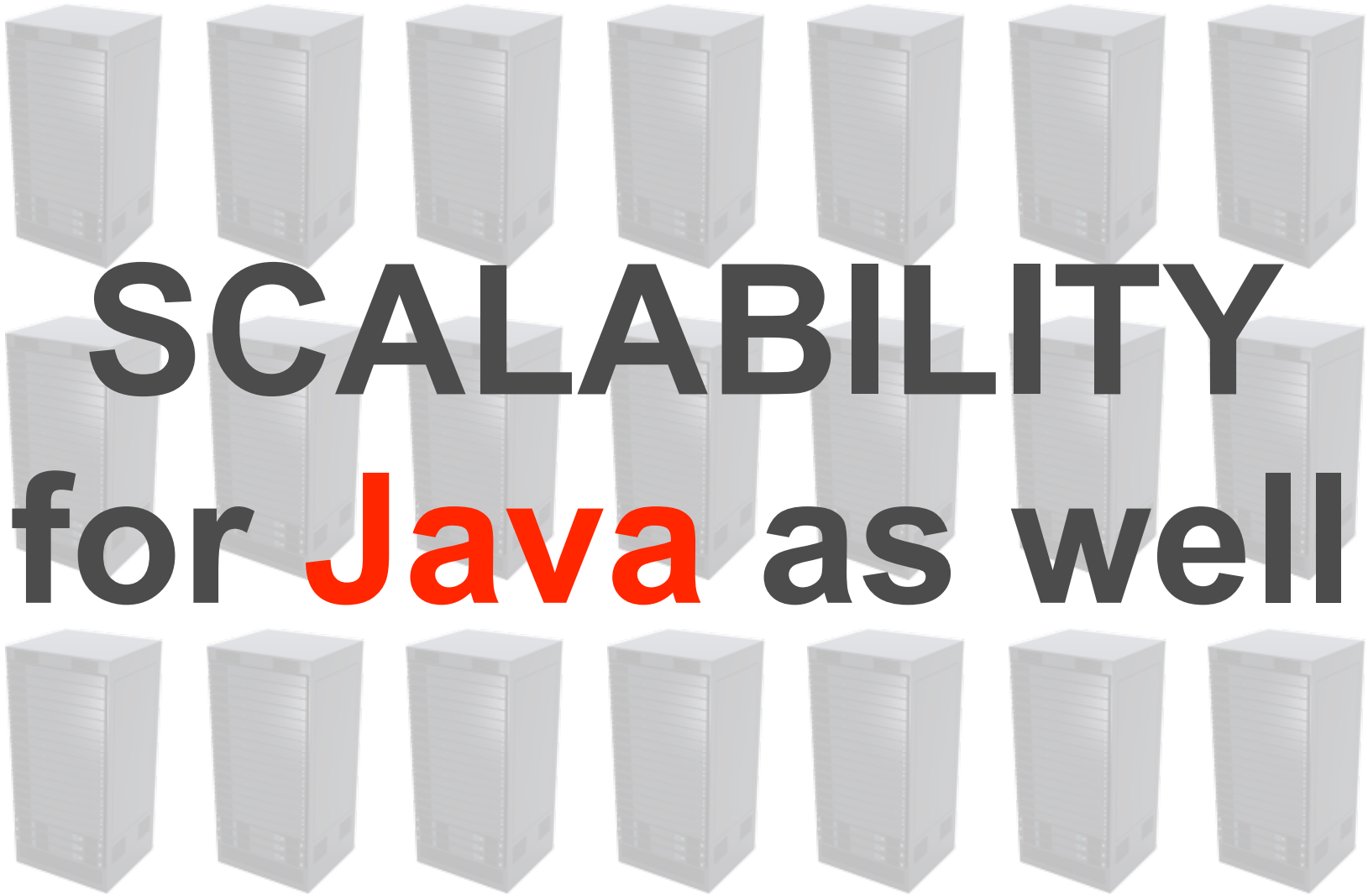
Same Distributed web hosting platform



- Parallel processing
- Scales automatically
- Available globally
- Configuration free
- Built-in DoS protections

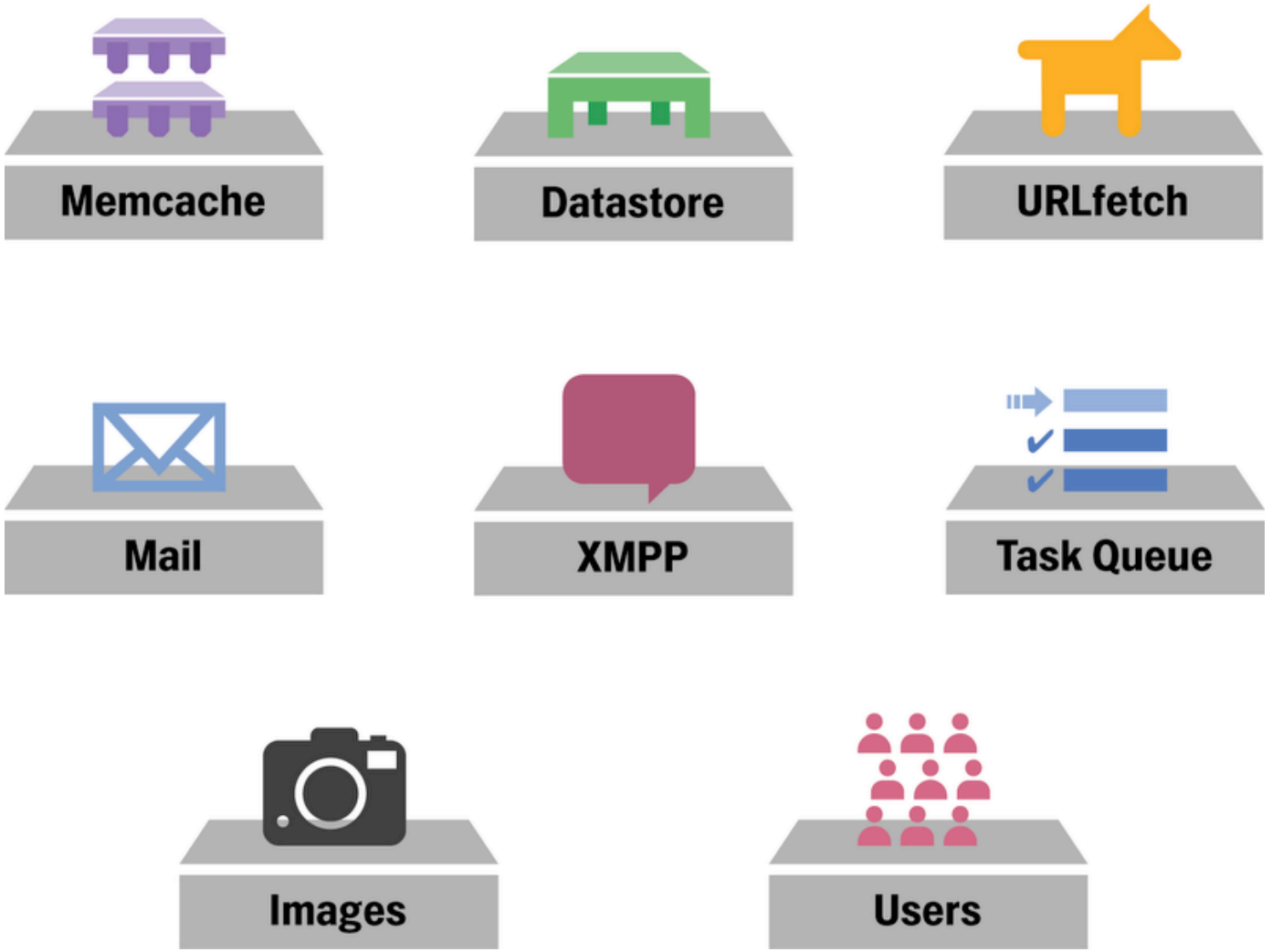
Same Distributed web hosting platform

Google

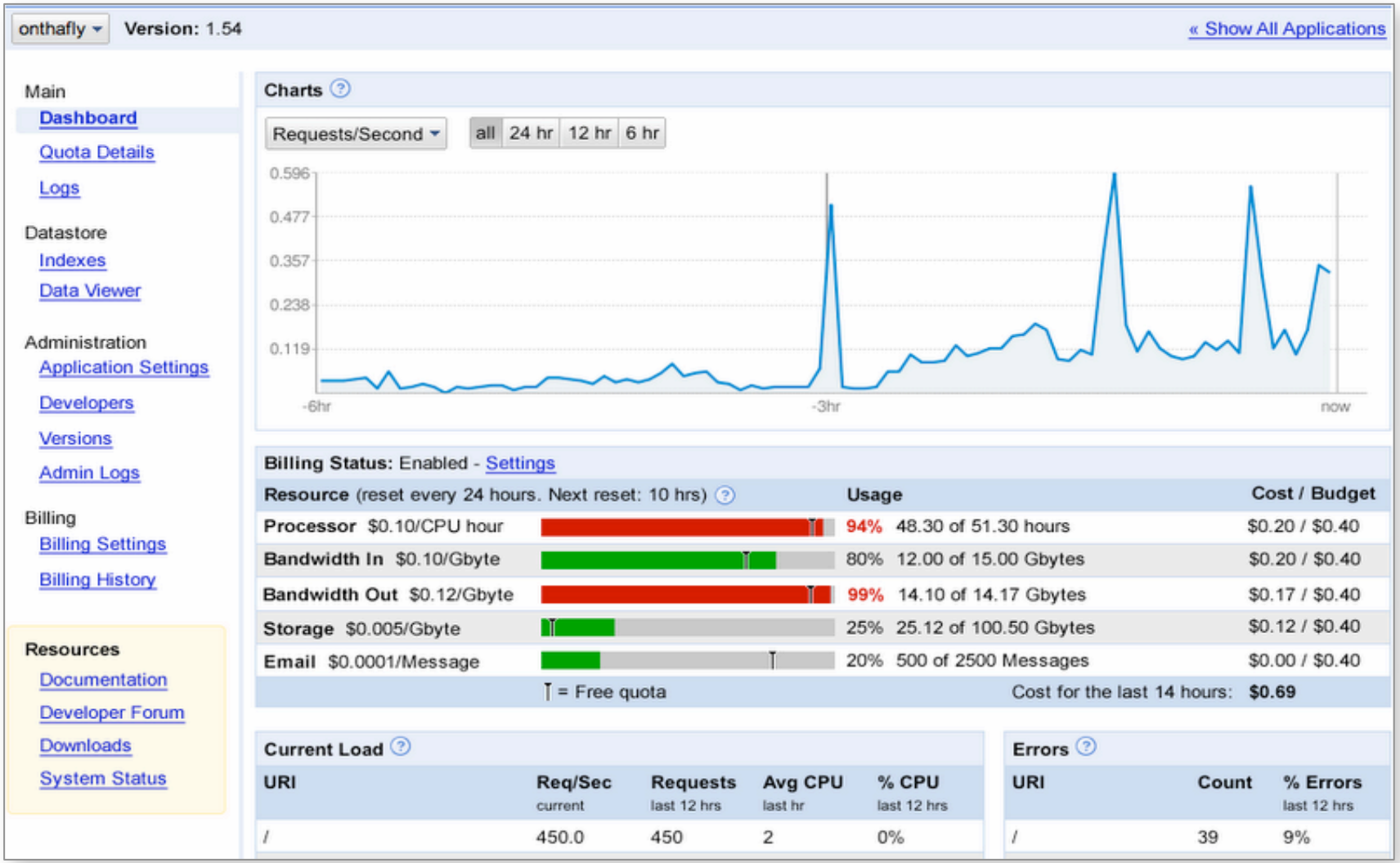


SCALABILITY for Java as well

Same Specialized services



Same App Engine Dashboard



Same initial *free* use policy as Python

- ~5M pageviews/month
- 6.5 CPU hrs/day
- 1 GB storage
- 650K URL Fetch calls
- 2,000 recipients emailed
- 1 GB/day bandwidth
- N tasks

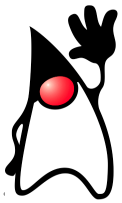
Google

App Engine and Java Standards

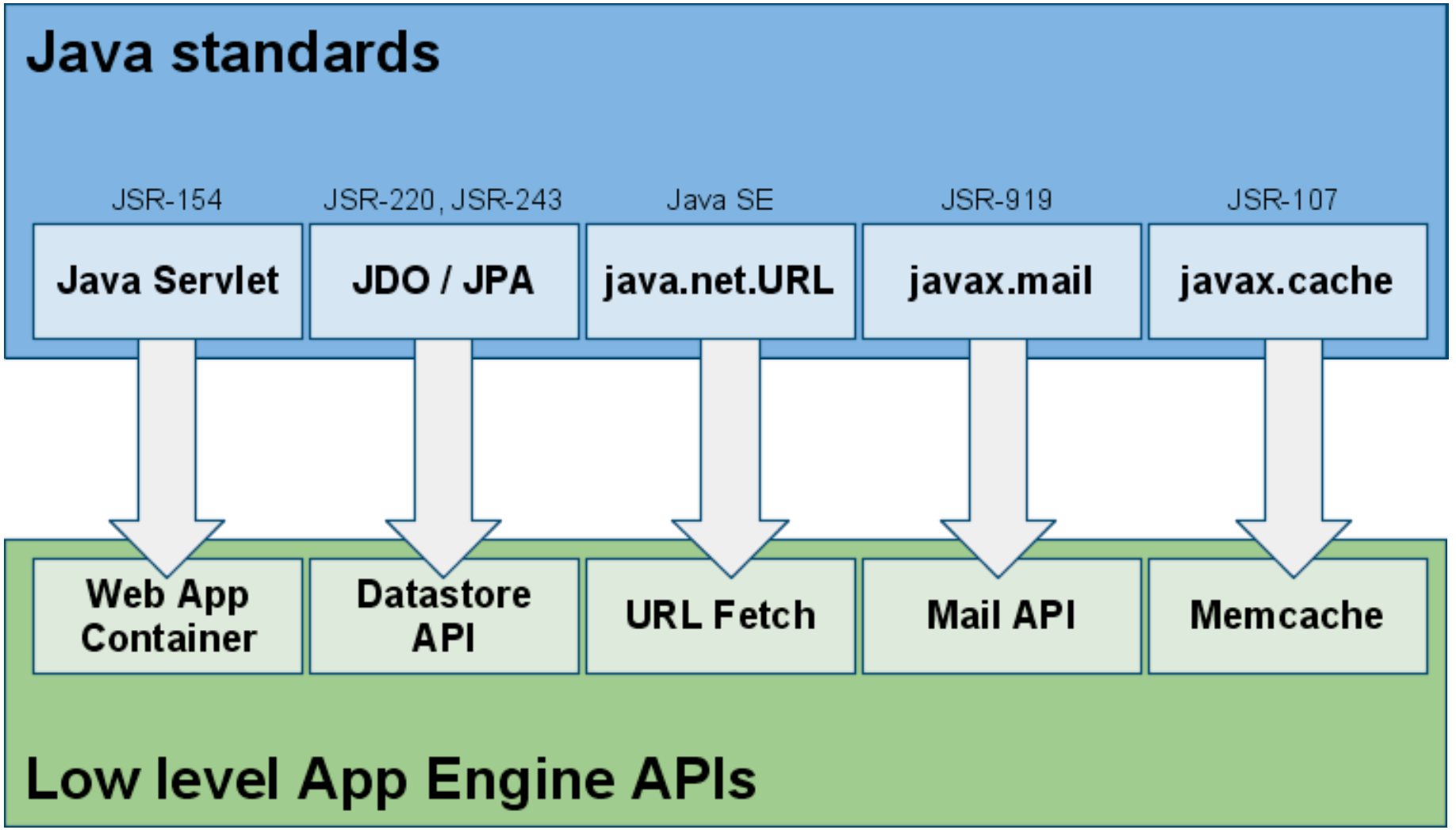
Based on Java Standards

Google

Based on Java Standards



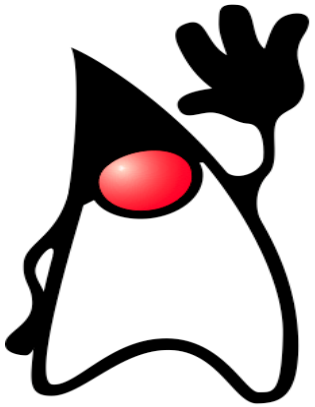
Google



Extended Language support through JVM

Google

- Java
- Scala
- JRuby (Ruby)
- Groovy
- Quercus (PHP)
- Rhino (JavaScript)
- Jython (Python)



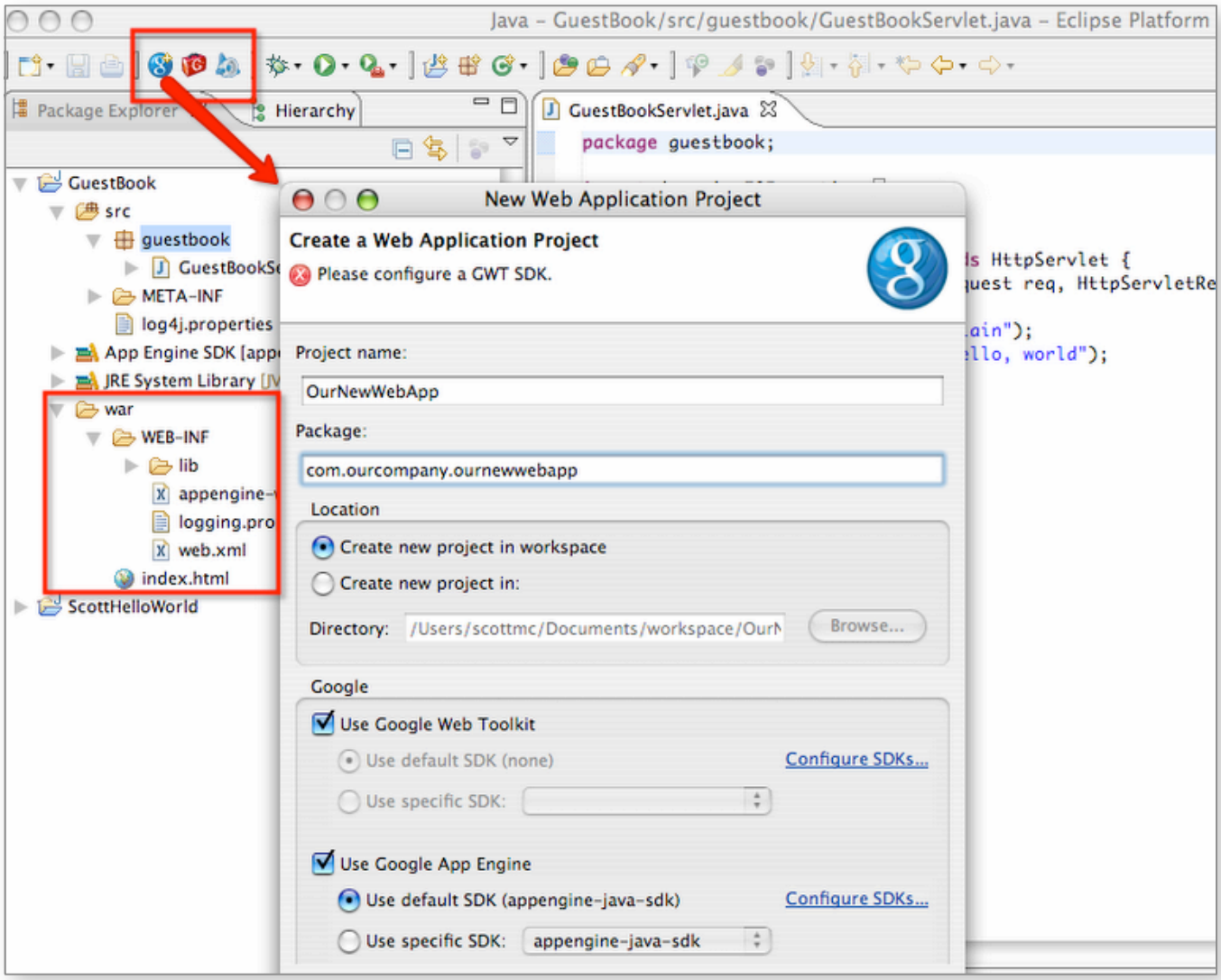
Duke, the Java mascot

Google

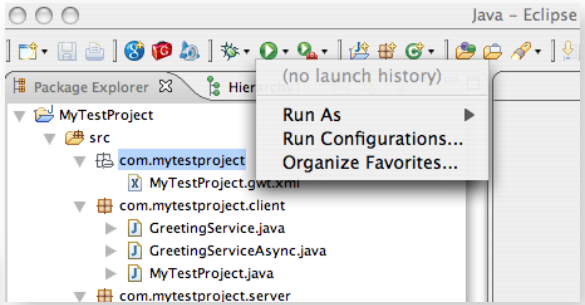
Development Tools for Java App Engine

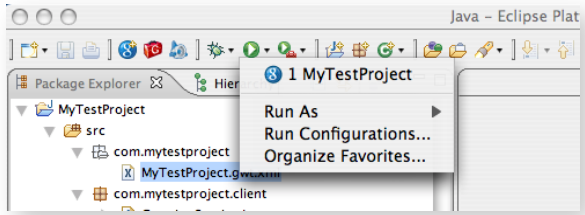
Google's Complete Java Development Stack

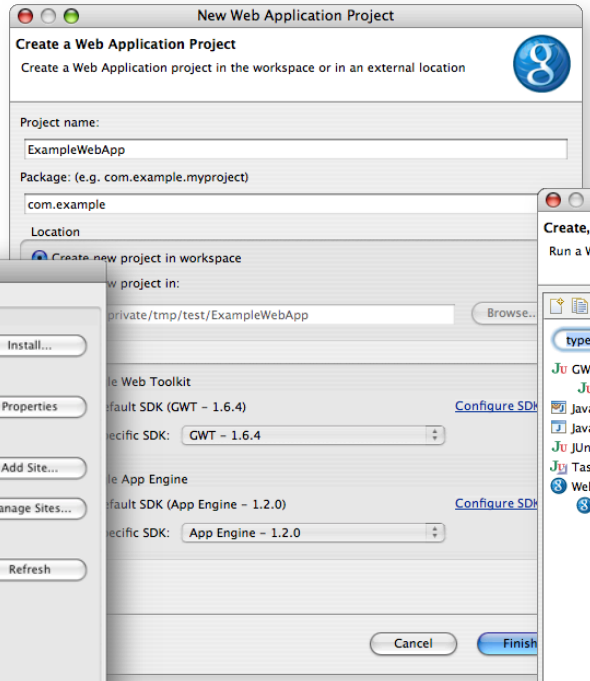
Google Plugin for Eclipse

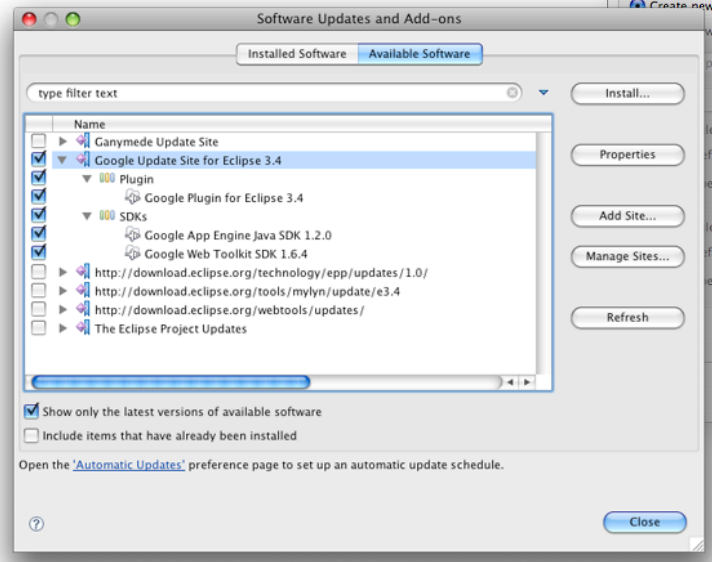


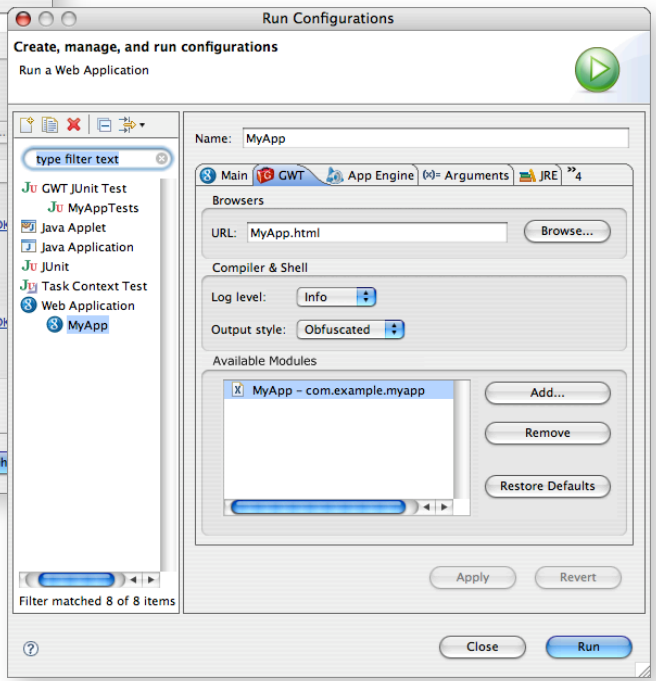
Google Plugin for Eclipse







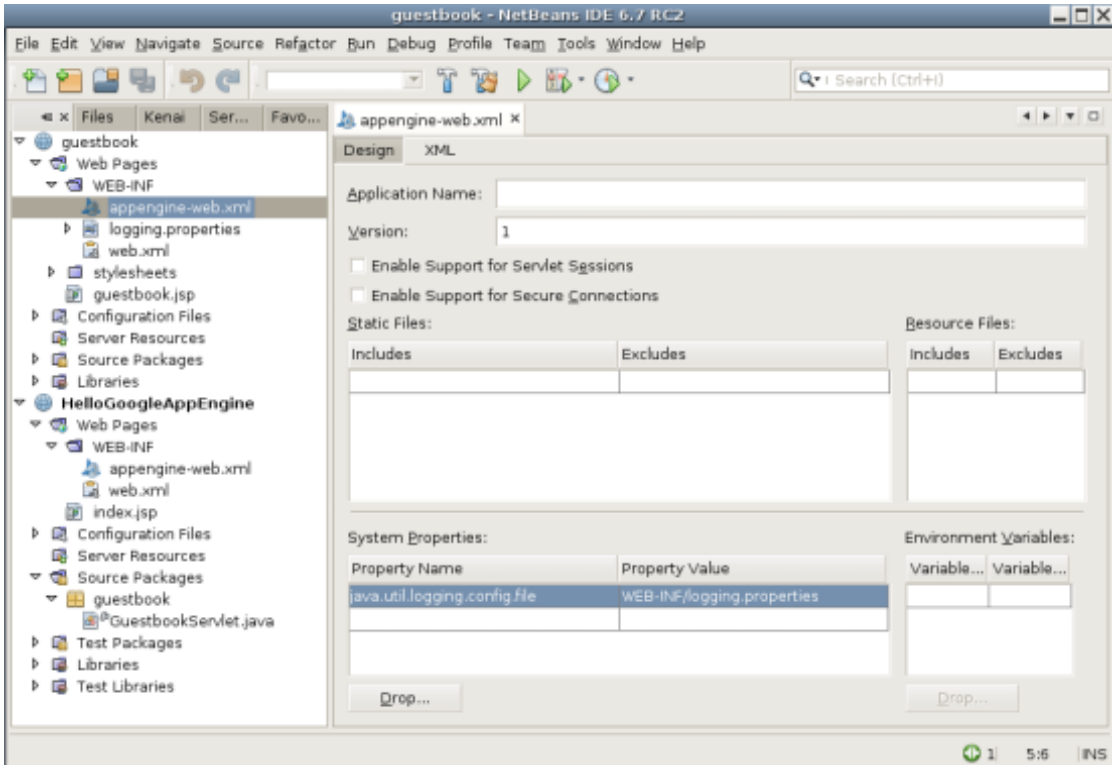




NeatBeans Plugin for Java App Engine

Google

http://kenai.com/projects/nbappengine/



Limitations, Issues

- No naked domains
- No SSL for domains
- First request for scripted languages is slow
- No MapReduce
- NoSQL
- No Sockets
- No Long running processes
- No Threads
- no javax.image APIs
- No Jax-WS
- Limit # of File on Apps

Roadmap

- Storing/serving large files
- Mapping operations across data sets
- Datastore cursors
- Notification alerts for application exceptions
- Datastore dump and restore facility

Tips

- Nick Johnson's deferred lib article
  - http://code.google.com/appengine/articles/deferred.html
  - Python but applies to Java
  - catch DeadlineExceededError and spawn a new task

Scala

- Modern language on top of VM (Java & .NET)
  - Statically typed
  - Traits -> Mixin style composition
  - More functional style, but not bigot about it: closures
  - Concurrency: Actors
  - Type inference: less verbose than Java
  - Type System
- Programming Scala, Wample & Payne, O'Reilly
- People to follow: Jonas Boner, @debasishg, Dave Pollack, Alex Payne

Scala/Lift

- Scala's Web Framework
  - Seaside's highly granular sessions and security
  - Rails fast flash-to-bang
  - Django's "more than just CRUD is included"
  - Wicket's designer-friendly templating style (see Lift View First)
- Lift in Appengine
  - Not all works: no Threads -> No Actors
- David Pollack & al, Lift Getting started
  - http://liftweb.net/docs/getting_started/mod_master.html#x1-50001.3
- ymnk's Books example
  - git://gist.github.com/98561.git

Scala/Lift

Author.scala

```
package net.liftweb.example.model

import javax.persistence._
import com.google.appengine.api.datastore.Key

@Entity
class Author {
  @Id
  @GeneratedValue(){val strategy = GenerationType.IDENTITY}
  var id : Key = _

  @Column{val nullable = false}
  var name : String = ""

  @OneToMany{val mappedBy = "author", val targetEntity=classOf[Book]}
  var books : java.util.List[Book] = _
}
```

JRuby

Google

- Ola Bini's work on JRuby so that it works on appengine
  - Many presos and blogs about rails on GAE (IO talk)
  - Bumble / Datastore, BeeU / User service
- Ryan Brown appengine-apis gem
  - http://code.google.com/p/appengine-jruby/
  - Rack::AppEngine task
  - See their Railsconf talk
- Rails works
- Sinatra framework
  - sudo gem install sinatra
- Sam Ruby Wave Robot in JRuby/Sinatra
  - http://intertwingly.net/blog/2009/06/07/Google-Wave-Robot-w-Sinatra-JRuby-AppEngine
  - Wave client lib

JRuby

Google

- Don’t forget to replace net/http by the Google version
  - Based on UrlFetch instead of Sockets
- Curt Thompson of Best Buy’s Giftag.com
  - http://googleappengine.blogspot.com/2009/02/best-buys-giftag-on-app-engine.html
  - Real world use case

Guestbook in JRuby

Google

guestbook.rb

```
require 'sinatra'
require 'dm-core'

# Configure DataMapper to use the App Engine datastore
DataMapper.setup(:default, "appengine://auto")

# Create your model class
class Shout
  include DataMapper::Resource

  property :id, Serial
  property :message, Text
end

# Make sure our template can use <%=h
helpers do
  include Rack::Utils
  alias_method :h, :escape_html
end
```

Guestbook in JRuby

Google

```
get '/' do
  # Just list all the shouts
  @shouts = Shout.all
  erb :index
end

post '/' do
  # Create a new shout and redirect back to the list.
  shout = Shout.create(:message => params[:message])
  redirect '/'
end
```

views/index.erb

```
<% @shouts.each do |shout| %>
<p>Someone wrote, <q><%=h shout.message %></q></p>
<% end %>

<form method=post>
<textarea name="message"></textarea>
<input type=submit value=Shout>
</form>
```

Clojure

Google

- Lisp dialect on top of Java VM
- Programming Clojure, Halloway, PragProg book
- John Hume appengine-clj lib
  - git://github.com/duelinmarkers/appengine-clj.git
  - users, datastore, test-utils

```
(require '[appengine-clj.datastore :as ds])
(ds/create {:kind "Person" :name "Jimmmy" :age 25})
=> {:kind "Person" :key #<Key Person(1138)> :name "Jimmmy" :age
25}
```

- Compojure web framework (inspired by Sinatra)
  - git://github.com/weavejester/compojure.git
  - http://elhumidor.blogspot.com/2009/04/clojure-on-google-appengine.html

Guestbook in Clojure/Compojure

Google

```
(ns guestbook.servlet
  ... you no longer need to import UserServiceFactory ...
  (:require
    [appengine-clj.users :as users]))

(defroutes guestbook-app
  (GET "/"
    (let [user-info (request :appengine-clj/user-info)
          user (user-info :user)]
      (html
        [:h1 "Hello, " (if user (.getNickname user) "World") "!"]
        [:p (link-to (.createLoginURL (user-info :user-service) "/")
"sign in")]
        [:p (link-to (.createLogoutURL (user-info :user-service) "/")
"sign out")]))))

(defservice (users/wrap-with-user-info guestbook-app))
```

Tools: Emacs or Counterclokcwise

Google

- Counterclockwide Eclipse plugin
  - http://code.google.com/p/counterclockwise/
- Emacs: clojure-mode, slime, swank
- Setup tools:
  - http://riddell.us/tutorial/slime_swank/slime_swank.html
- Interactive Programming with Clojure, Compojure, Google App Engine and Emacs
  - http://www.hackers-with-attitude.com/2009/08/intertactive-programming-with-clojure.html
  - Edit, C-c C-c, shift-reload
- Jeff Foster Wave Robot in Clojure
  - http://www.fatvat.co.uk/2009/07/google-wave-and-clojure.html

Wave Robot in Clojure

Google

Jeff Foster http://www.fatvat.co.uk/2009/07/google-wave-and-clojure.html

```
(ns uk.co.fatvat.wave.parrot
  (:import [com.google.wave.api RobotMessageBundle EventType])
  (:gen-class :extends com.google.wave.api.AbstractRobotServlet))

(defn- add-blip
  [wavelet message]
  (.append (.getDocument (.appendBlip wavelet)) message))

(defn -processEvents
  [_ bundle]
  (let [wavelet (.getWavelet bundle)]
    (when (.wasSelfAdded bundle)
      (add-blip wavelet "Greetings.  I'm alive!"))
    (let [participant-changed-events (filter
                                        (fn [e] (= (.getType e)
(EventType/WAVELET_PARTICIPANTS_CHANGED)))
                                        (.getEvents bundle))]
      (doseq [event participant-changed-events]
        (add-blip wavelet "Hey! It's me!")))))
```

Demos

Peter Sönnergren Lind, Pockus
@peter_lind

http://github.com/peterlind

Groovy/Gaelyk

Google


Gaelyk
http://gaelyk.appspot.com
Gaelyk – a lightweight Groovy toolkit for Google App Engine Java
http://gaelyk.appspot.com/tutorial/
Google
jeu3 mer19 jeu9 dim28 Gmail Reader DZone Twitter GAE reddit VMW Stock EUR/USD
Gaelyk
Groovy
Home Tutorial Download Community
Tutorial
The goal of this tutorial is to quickly get you started with using Gaelyk to let you write and deploy your Groovy applications on Google App Engine. We'll assume you have already downloaded and installed the Google App Engine SDK of your machine. If you haven't, please do so by reading the instructions from Google.
The easiest way to get setup rapidly is to download the template project from the download section. It provides a ready-to-go project with the right configuration files pre-filled and an appropriate directory layout:
web.xml precondigured with the Gaelyk servlets
appengine-web.xml with the right settings predefined (static file directive)
a sample Groovlet and template
the needed JARs (Groovy, Gaelyk and Google App Engine SDK)
You can browse the JavaDoc of the classes composing Gaelyk.
Setting up your project
Directory layout
We'll follow the directory layout proposed by the Gaelyk template project:

Gaelyk

- Gaelyk is a **lightweight Groovy toolkit** on top of the Google App Engine Java SDK

- Gaelyk builds on Groovy's servlet support
  - **Groovlets**: Groovy scripts instead of raw servlets!
  - **Groovy templates**: JSP-like template engine
  - Both allow for a clean separation of views and logic

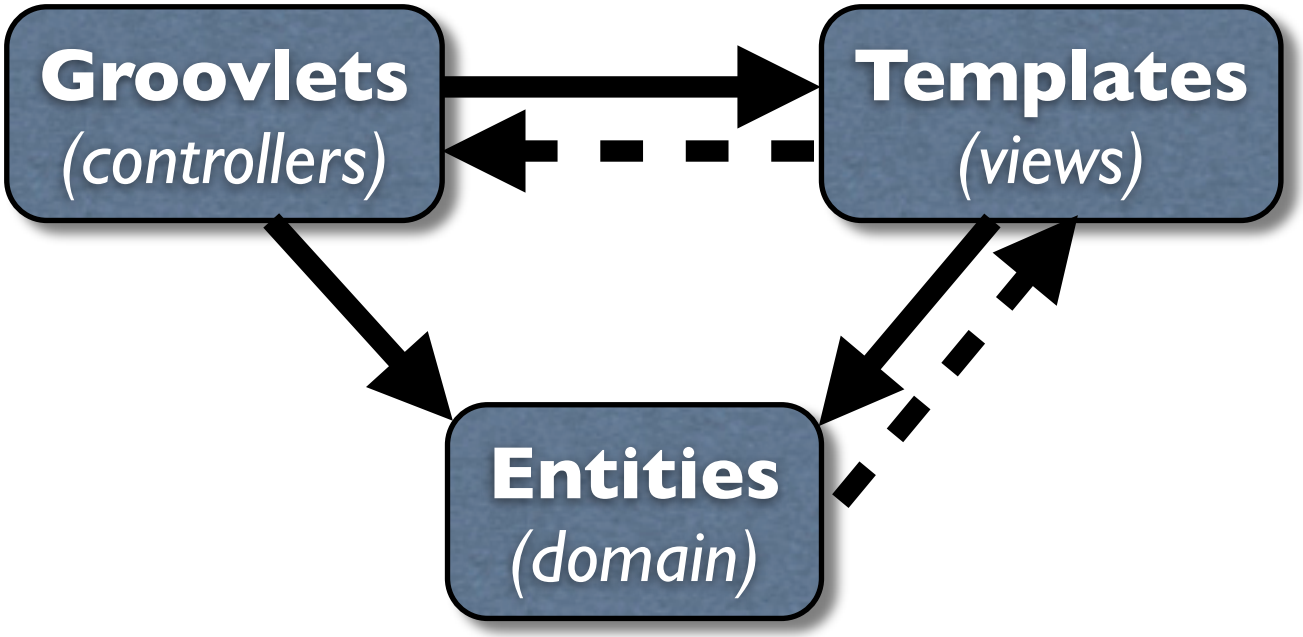
- Gaelyk provides several **enhancements** around the GAE Java SDK to make life easier, thanks to Groovy's dynamic nature

Why Groovy?

- **Groovy** is a **dynamic language** for the JVM
  - very flexible, malleable, expressive and **concise** syntax
  - easy to learn for Java developers
    - deriving from the Java 5 grammar
  - provides powerful APIs to simplify the life of developers
    - possibility to **dynamically enrich existing APIs**
  - support for **Groovlets** and its own **template engine**

- We worked with the Google App Engine Java team before the official launch of the platform, to ensure Groovy would run well on this new environment

MVC: Groovlets and templates




First steps...
Go to http://gaelyk.appspot.com
Download the template project
Put your first Groovlet in /WEB-INF/groovy
And your templates at the root
And you're ready to go!
Launch dev_appserver.sh
Go to http://localhost:8080/

The web.xml

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee" version="2.5">
  <servlet>
    <servlet-name>GroovletServlet</servlet-name>
    <servlet-class>groovyx.gaelyk.GaelykServlet</servlet-class>
  </servlet>
    <servlet>
    <servlet-name>TemplateServlet</servlet-name>
    <servlet-class>groovyx.gaelyk.GaelykTemplateServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>GroovletServlet</servlet-name>
    <url-pattern>*.groovy</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>TemplateServlet</servlet-name>
    <url-pattern>*.gtpl</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>index.gtpl</welcome-file>
  </welcome-file-list>
</web-app>
```


A groovlet

- Instead of writing full-blown servlets, just write Groovy scripts (aka Groovlets)

```
def numbers = [1, 2, 3, 4]
def now = new Date()

html.html {
    body {
        numbers.each { number -> p number }
        p now
    }
}
```


A template

```
<html>
    <body>
        <p><%
            def message = "Hello World!"
            print message %>
        </p>
        <p><%= message %></p>
        <p>${message}</p>
        <ul>
        <% 3.times { %>
            <li>${message}</li>
        <% } %>
        </ul>
    </body>
</html>
```


Shortcuts

- **Variables available**
  - request / response
  - context / applicaiton
  - session
  - params
  - header
  - out / sout / html
- **Methods available**
  - include / forward
  - print / println
- **Google services**
  - datastore
  - memcache
  - urlFetch
  - mail
  - userService
  - user
  - defaultQueue
  - queues
  - xmpp


Groovy sugar!

Sending emails with Gaelyk

```
mail.send to: 'foobar@gmail.com',
     from: 'other@gmail.com',
     subject: 'Hello World',
     htmlBody: '<bold>Hello</bold>'
```


...compared to Java

```
Properties props = new Properties();
Session session = Session.getDefaultInstance(props, null);

String msgBody = "...";

try {
    Message msg = new MimeMessage(session);
    msg.setFrom(new InternetAddress("admin@example.com", "Admin"));
    msg.addRecipient(Message.RecipientType.TO,
                     new InternetAddress("user@example.com", "Mr. User"));
    msg.setSubject("Your Example.com account has been activated");
    msg.setText(msgBody);
    Transport.send(msg);
} catch (AddressException e) {}
} catch (MessagingException e) {}
```


Accessing the datastore

- Direct interaction with the low-level datastore API

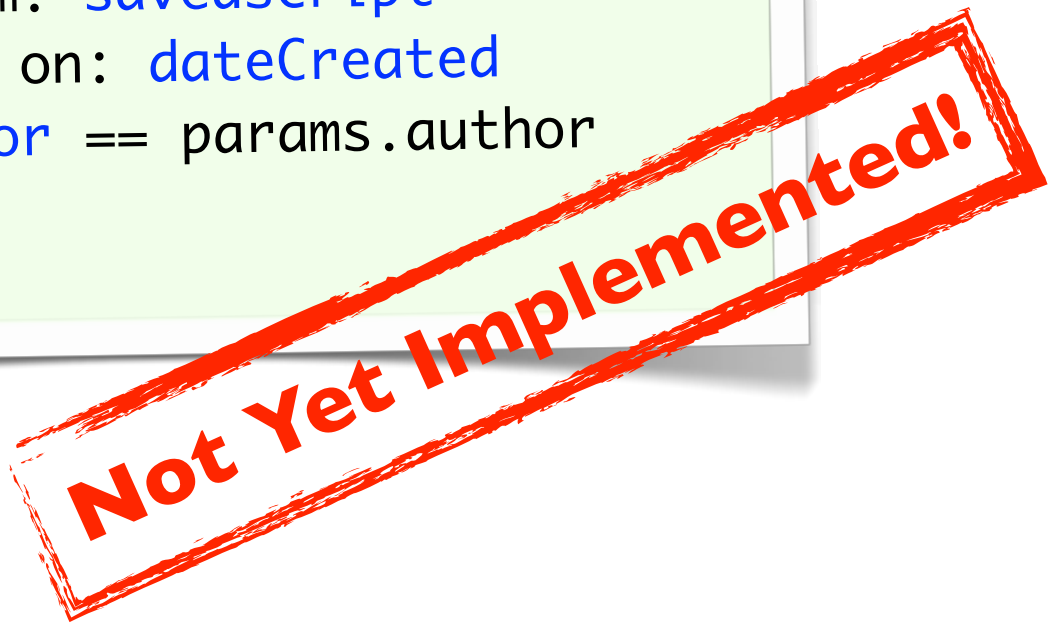
```
import com.google.appengine.api.datastore.Entity

Entity entity = new Entity("person")

// subscript notation, like when accessing a map
entity['name'] = "Guillaume Laforge"

// normal property access notation
entity.age = 32

entity.save()
entity.delete()

datastore.withTransaction {
    // do stuff with your entities
    // within the transaction
}
```


Querying to be improved...

```
import com.google.appengine.api.datastore.*
import static com.google.appengine.api.datastore.FetchOptions.Builder.*

// query the scripts stored in the datastore
def query = new Query("savedscript")

// sort results by descending order of the creation date
query.addSort("dateCreated", Query.SortDirection.DESCENDING)

// filters the entities so as to return only scripts by a certain author
query.addFilter("author", Query.FilterOperator.EQUAL, params.author)

PreparedQuery preparedQuery = datastore.prepare(query)

// return only the first 10 results
def entities = preparedQuery.asList( withLimit(10) )
```


...into something groovier?

```
def entities = datastore.createQuery {
    select from: savedscript
    sort DESC, on: dateCreated
    where author == params.author
    limit 10
} as List
```



Task queue API

```
// access a configured queue using the subscript notation
queues['dailyEmailQueue']

// or using the property access notation
queues.dailyEmailQueue

// you can also access the default queue with:
queues.default
defaultQueue

// add a task to the queue
queue << [
    countdownMillis: 1000, url: "/task/dailyEmail",
    taskName: "Send daily email newsletter",
    method: 'PUT', params: [date: '20090914'],
    payload: content
]
```


Jabber / XMPP support (1/3)

- Sending instant messages

```
String recipient = "someone@gmail.com"

// check if the user is online
if (xmpp.getPresence(recipient).isAvailable()) {
    // send the message
    def status = xmpp.send(to: recipient,
                           body: "Hello, how are you?")

    // checks the message was successfully
    // delivered to all the recipients
    assert status.isSuccessful()
}
```


Jabber / XMPP support (2/3)

- Sending instant messages with an XML payload

```
String recipient = "service@gmail.com"

// check if the service is online
if (xmpp.getPresence(recipient).isAvailable()) {
    // send the message
    def status = xmpp.send(to: recipient, xml: {
        customers {
            customer(id: 1) {
                name 'Google'
            }
        }
    })

    // checks the message was successfully delivered to the service
    assert status.isSuccessful()
}
```

```
<customers>
    <customer id='1'>
        <name>Google</name>
    </customer>
</customers>
```


Jabber / XMPP support (3/3)

- Receving incoming instant messages
  - Configure the XmppServlet in web.xml
  - Add the inbound message service in appengine-web.xml

```
// get the body of the message
message.body
// get the sender Jabber ID
message.from
// get the list of recipients Jabber IDs
message.recipients

// if the message is an XML document instead of a raw string message
if (message.isXml()) {
    // get the raw XML
    message.stanza
    // get a document parsed with XmlSlurper
    message.xml
}
```


Memcache service

- Map notation access to the cache

```
class Country implements Serialzable { String name }

def countryFr = new Country(name: 'France')

// use the subscript notation to put a country object in the cache
// (you can also use non-string keys)
memcache['FR'] = countryFr

// check that a key is present in the cache
if ('FR' in memcache) {
    // use the subscript notation to get an entry from the cache using a key
    def countryFromCache = memcache['FR']
}
```

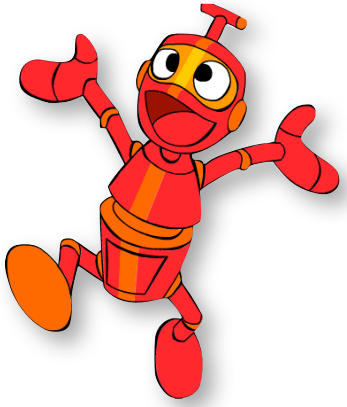

What's coming next?

- A new Gaelyk 0.3 release a couple days ago
  - Sugar for the Memcache service
  - The incoming email support in GAE SDK 1.2.6
- Add more sugar around...
  - The Datastore query system
    - SQL-like DSL, dynamic finders...
  - DSL for the URL Fetch service for accessing REST resources
- More generally...
  - Anything that'll come up in upcoming GAE SDK versions

The use case

- Collaboratively consolidated Twitter search
- Send a search query to
  - groovydemos@appspot.com
- The bot will
  - receive your message
  - store the query in the datastore
  - enqueue a task for processing the search
- The queue handler will
  - parse the Twitter search feed for the query
  - put the results into memcache
- Browse http://groovydemos.appspot.com

groovydemos@appspot.com



http://groovydemos.appspot.com

Summary

- Easy access to a cloud solution
  - Deploying Java apps, as easily as you would with PHP
- Familiar to Java folks
  - Your good old Servlet centric webapps style
- Pretty cheap
  - You need a high-trafficed website to reach the quotas
- Gaelyk provides a simplified approach to creating Servlet centric webapps in a productive manner
  - Leveraging Groovy's servlet / template support and dynamic capabilities

Thanks for your attention!

References:

http://code.google.com/appengine/

http://gaelyk.appspot.com/

http://groovy.codehaus.org/

http://grails.org/

Images used in this presentation

- Clouds
  - http://www.morguefile.com/archive/display/627059
  - http://www.morguefile.com/archive/display/625552
  - http://www.morguefile.com/archive/display/629785
- Duke ok GAE
  - http://code.google.com/images/duke-on-gae.jpg
  - http://weblogs.java.net/blog/felipegaucho/archive/ae_gwt_java.png
- Python logo : http://python.org/images/python-logo.gif
- Gaelyc cross with clouds : http://www.morguefile.com/archive/display/37889
- Speed limit : http://www.morguefile.com/archive/display/18492
- Warehouse : http://www.morguefile.com/archive/display/85628
- Snow foot steps : http://www.flickr.com/photos/robinvanmourik/2875929243/
- Sugar : http://www.flickr.com/photos/ayelie/441101223/sizes/l/

Questions

Google

Q&A

Google

Patrick Chanezon
Twitter: @chanezon
chanezon@google.com

Groovy

Guillaume Laforge
Twitter: @glaforge

Peter Sönnergren Lind, Devoteam
@peter_lind