DevMedia - Versão para impressão

Java no Google App Engine

Cicero Augusto Wollmann Zandoná
Oito anos de experiência com desenvolvimento em Java/Java EE, possui mestrado em Métodos Numéricos para Engenharia e certificação SCJP. Atualmente trabalha como desenvolvedor/analista no HSBC Global Technology Brasil.

De que se trata o artigo:

Introdução prática do Google App Engine, o serviço de computação em nuvem do Google. São apresentados os prós e os contras do serviço e um exemplo completo.

Para que serve:

Com o Google App Engine é possível hospedar aplicações Java Web de forma simples e sem custo inicial. O serviço inclui funcionalidades de autenticação de usuários, persistência com JPA ou JDO, envio de e-mails com Java Mail e Cache com JCache.

Em que situação o tema é útil:

É mais uma opção para o desenvolvedor que deseja colocar suas aplicações na Internet. Conforme a aplicação é requisitada o Google App Engine inicia novas máquinas virtuais, de modo que a aplicação possa escalar para grandes volumes de acesso. O desenvolvedor tem também a possibilidade de monitorar o uso da aplicação através de uma interface web.

Google App Engine:

O Google App Engine é uma promissora plataforma para hospedagem de aplicações Java. As suas vantagens são muitas, mas também existem limitações que o desenvolvedor precisa levar em conta antes de decidir pela sua adoção. Alguns recursos da plataforma Java não estão disponíveis, como uso de Threads e manipulação de arquivos. Em contrapartida, o serviço oferece recursos interessantes, como as APIs de autenticação e de tratamento de imagens.

Mesmo com a escalabilidade virtualmente ilimitada, deve-se tomar cuidado com o mal uso de recursos, como transferência de dados e uso excessivo da base de dados, que podem ser

cobrados proporcionalmente ao seu uso.
Em abril de 2008 o Google lançou seu serviço de hospedagem de aplicações web, batizado de Google App Engine, ou GAE. O grande atrativo desse serviço é a possibilidade de utilizar a enorme infra-estrutura do Google. No GAE, à medida que uma aplicação é requisitada, novas máquinas virtuais são iniciadas, provendo escalabilidade de forma dinâmica, um estilo de solução chamado de Computação em Nuvem, ou Cloud Computing.
O lançamento teve bastante repercussão em certos grupos, mas na comunidade Java pouco se falou sobre ele, por um simples motivo: Python era a única linguagem suportada. Desde o começo o Google prometeu que outras linguagens iriam ser suportadas no futuro, mas manteve o suspense sobre quais seriam.

Agora, exatamente um ano depois do lançamento, o suporte a Java foi incluído.
Se fosse apenas um serviço de hospedagem comum, não seria necessário um artigo específico para o GAE, mas se trata de um serviço com características únicas. Dessa forma, é importante conhecer bem suas vantagens e limitações antes de tomar a decisão de utilizá-lo. A idéia deste artigo é fazer uma apresentação imparcial dos principais aspectos deste serviço, e por fim, apresentar um exemplo completo.

Vantagens e Limitações

As vantagens que mais têm atraído usuários são a gratuidade inicial e a ausência da necessidade de configurar o servidor. Uma vez desenvolvida a aplicação, é feito o deploy, e ela está acessível ao público, sem complicações. Por estes fatores, o GAE tem um grande potencial para alterar o cenário atual de desenvolvimento web. Todo o programador que já tentou hospedar um site em Java sabe que os custos de hospedagem são bem mais salgados quando o provedor precisa rodar uma JVM. Esse inclusive era um fator que fazia o Java ser descartado em projetos para clientes pequenos, que preferiam stacks “baratas”, como LAMP (abreviação de Linux, Apache, MySQL e PHP/Perl/Python).
Outra vantagem óbvia é que não há necessidade de se preocupar com escalabilidade, o GAE vai automaticamente criar novas instâncias da JVM, de forma orgânica, conforme a demanda aumenta.

Nota

Escalar de forma orgânica significa de forma gradativa, como acontece em uma situação real. Por isso, é importante avisar que se você pretende executar algum teste de performance no GAE, configure a sua ferramenta para ter um período razoável de “ramp-up” de requisições.

O Google App Engine é gratuito até certo ponto, existem limites diários e por minuto, chamados de cotas. São mais de 20 tipos de cotas, que vão desde tempo de CPU e número de requisições até quantos arquivos são anexados em e-mails. Veja na seção Links, no final do artigo, o endereço para uma listagem completa das cotas. Felizmente, os limites são bem generosos para a maioria das aplicações. Segundo o Google é o suficiente para disponibilizar mais de cinco milhões de pageviews/mês (apesar de sabermos que esse número é tão relativo quanto dizer que em um iPod cabem 3.000 músicas). O que exceder o limite é cobrado proporcionalmente ao uso.

Mas o leitor não precisa ficar preocupado em se endividar caso o seu site se torne um sucesso repentino, os limites de cotas são completamente customizáveis. Isto é, se a cota estabelecida for atingida o GAE mostrará uma mensagem de “limite excedido” para o visitante voltar no dia seguinte.
A questão que deve ser vista com mais cuidado para decidir sobre o uso do serviço são as cotas máximas, que não podem ser compradas. Por exemplo, atualmente um request não pode ultrapassar o tamanho de 10 megabytes, nem durar mais que 30 segundos (se isso acontecer a thread será derrubada), e não é possível alterar estes limites, nem que o desenvolvedor pague. De certa forma estes limites impossibilitam o uso do GAE para sites que precisem hospedar grandes arquivos ou que façam processamento offline pesado.

Nota

Quando uma requisição é interrompida por exceder o limite de tempo estabelecido, uma exceção do tipo com.google.apphosting.api.DeadlineExceededException é lançada e pode ser tratada, para que o visitante não seja surpreendido com um erro HTTP 500 depois de uma longa espera.

Outro ponto que deve ser visto com cuidado é que o GAE, por questões de segurança e escalabilidade, não fornece uma JRE completa. Veja na seção Links o endereço para uma listagem com todas as classes disponíveis. As ausências mais importantes são a AWT/Swing, JDBC e classes que possam ler e escrever no sistema de arquivos. Também não é possível iniciar Threads, abrir sockets e usar Reflection para acessar membros privados de classes que não sejam da aplicação.
Então, considerando que o GAE não permite acesso a todas as classes do JDK, surgem imediatamente dúvidas se determinado framework (o preferido do leitor) vai funcionar corretamente. Como o universo de frameworks web Java é enorme, não é possível colocar uma listagem completa neste artigo, mas esta lista existe, e é mantida por usuários do GAE. Veja em Links o endereço para a página “Will it play in App Engine?”. Como regra geral, frameworks que não usem as APIs citadas acima devem funcionar normalmente, e a maioria dos frameworks MVC entra nesse grupo.

Nota

Além disso, muitos frameworks atualmente não compatíveis com o GAE logo deverão ter atualizações compatíveis. Isso é praticamente garantido pela força de mercado e popularidade do Google, cujos engenheiros também têm colaborado com os autores de frameworks populares para “corrigi-los” e possibilitar a compatibilidade com o GAE.

Para de certa forma compensar as exclusões, são oferecidas algumas APIs próprias, que tornam o serviço bem interessante. Estas APIs serão apresentadas nas seções deste artigo. Também é importante mencionar que outras linguagens feitas para a JVM também são suportadas no GAE, como Groovy, JRuby, Rhino, Scala e Jython.

Autenticação

Para autenticação, o GAE oferece uma API para o Google Accounts, o sistema de contas de usuário do Google. Isto é, se o seu (potencial) usuário possui cadastro em algum serviço do Google que necessite autenticação, como Gmail, Google Docs, Orkut e YouTube, ele poderá utilizar este mesmo login para acessar a sua aplicação, facilitando em muito o processo de cadastro.

O uso desta API é simples: os métodos createLoginURL() e createLogoutURL(), da interface UserService, criam respectivamente URLs para o login e logout do serviço. Ao clicar no link de login o usuário é redirecionado para uma página de autenticação (Figura 1) e depois mandado de volta, autenticado, para a aplicação. Mais detalhes na aplicação exemplo.
Este serviço também permite que se definam paginas administrativas, com acesso restrito aos desenvolvedores do projeto. As configurações de acesso são feitas no arquivo web.xml, utilizando a tag padrão <security-contraint>. Por exemplo, para restringir todas as páginas dentro de /admin incluiríamos o seguinte trecho:

```
<security-constraint>
<web-resourcecollection>
<url-pattern>/admin/*</url-pattern>
</web-resourcecollection>
<auth-constraint>
<role-name>admin</role-name>
</auth-constraint>
</security-constraint>
```

[abrir imagem em janela]

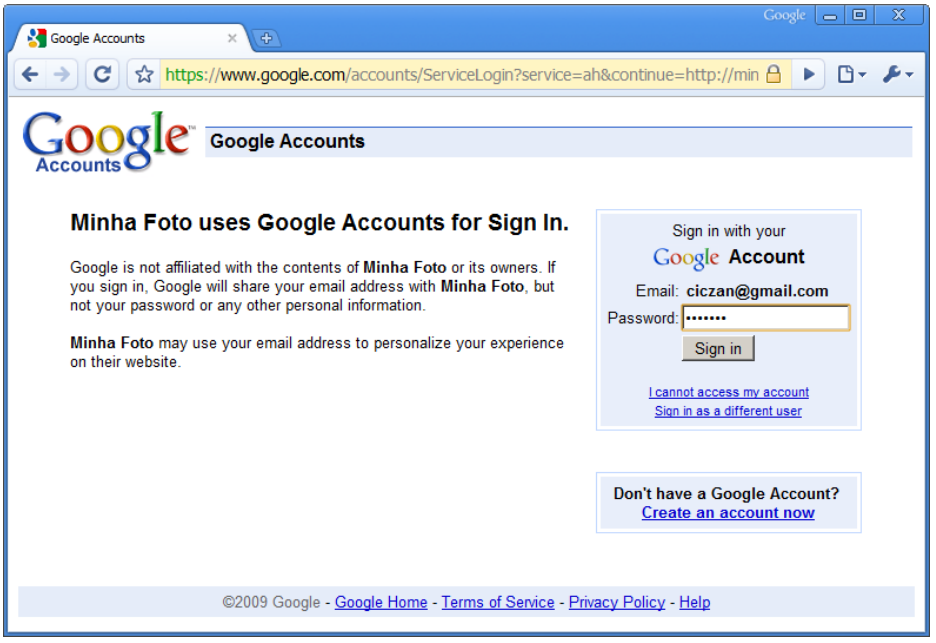

Figura 1. Autenticação via Google Accounts

Persistência

Uma das características mais peculiares do GAE é o mecanismo de persistência de dados. O serviço oferecido é chamado de Datastore e é baseado na solução de armazenamento usada internamente pelo Google, o Bigtable.
O Bigtable é um sistema de armazenagem distribuído, criado para armazenar grandes volumes de dados (petabytes) espalhados em um grande número de servidores. Para viabilizar esta

escalabilidade, alguns recursos dos bancos de dados relacionais tiveram que ser "sacrificados". Por exemplo, a integridade relacional não é garantida, e consultas com join não são suportadas. Uma apresentação completa do Bigtable está fora do escopo desde artigo, consulte a seção Links para mais detalhes.
O acesso ao Datastore pode ser feito através de três diferentes APIs: uma API própria de baixo nível, com JDO (Java Data Access) e JPA (Java Persistence API). Utilizar a API de baixo nível não é produtivo, e muito menos portável, na verdade ela voltada principalmente para os que desejem implementar outras APIs de persistência sobre ela. Entre JDO e JPA, muito provavelmente o leitor está mais familiarizado com o JPA, que é atualmente o "padrão de fato" para persistência em Java. Mesmo assim, a documentação oficial e os exemplos do SDK enfatizam o JDO, por ser uma API de persistência genérica, podendo ser implementada tanto para Bancos de Dados relacionais quanto para Bancos de Dados OOs, arquivos XML e outros tipos de back-end.

No exemplo desse artigo será apresentado o uso do JDO, que conceitualmente é parecido com o JPA. Ou seja, é baseado em annotations para a definição das entidades persistentes e o gerenciamento das entidades é feito a partir de uma classe gerenciadora, chamada PersistenceManager.

Imagens

Para o tratamento de imagens o Google App Engine oferece uma biblioteca própria, que suporta operações básicas de edição: redimensionar, rotacionar, cortar e inverter na horizontal e na vertical. Fora o básico, a API possui a função "Im Feeling Lucky" (Estou com Sorte), marca registrada do mecanismo de busca do Google, que neste caso faz um ajuste automático no contraste, brilho e gama da imagem. A utilização da API é muito simples, como será demonstrado no nosso exemplo.
A API de imagens é uma tentativa de suprir a ausência da API Java2D, que foi podada já que depende do AWT. Mas apesar de bem mais simples de utilizar, e de ser suficiente em muitos casos, não chega nem perto do Java2D em funcionalidades. Para aplicações web, essa falta será sentida na incompatibilidade com bibliotecas de Capcha e Gráficos (ex.: JFreeChart).

Cron

O serviço de Cron permite o agendamento de tarefas em horários ou intervalos determinados, de forma semelhante ao comando cron dos sistemas operacionais Unix. Os agendamentos são descritos no arquivo cron.xml, que deve ser colocado no diretório WEB-INF da estrutura do projeto.
A sintaxe do agendamento tem um estilo descritivo, "escrito como se fala", em inglês. Por exemplo, um agendamento que dispara o serviço a cada 6 horas seria: "every 6 hours", que em português é literalmente "a cada 6 horas".

Diferente das outras APIs de agendamento Java, onde se configura uma classe para ser chamada, o Cron do GAE requer um path, definido no web.xml, e na hora que o serviço é disparado este path é invocado via HTTP. Veja um cron.xml de exemplo na Listagem 1.
Por enquanto o Cron não é suportado no ambiente de desenvolvimento, só é possível testá-lo on-line.

Listagem 1. Exemplos de agendamento com o cron.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<cronentries>
  <cron>
```

```
    <url>/estatisticas</url>
    <description>Atualiza estatísticas
    dos usuários a cada meia hora
    </description>
    <schedule>every 30 minutes</schedule>
    <timezone>America/Sao_Paulo</timezone>
  </cron>
  <cron>
    <url>/relatorio</url>
    <description>Gera envia um relatorio
    todo o primeiro domingo do mês</description>
    <schedule>every sunday of
    month 09:00</schedule>
    <timezone>America/Sao_Paulo</timezone>
  </cron>
</cronentries>
```

Compatibilidade

Excluindo-se as APIs de Imagens e Cron, o Google optou por adotar os padrões de mercado para os demais serviços:

- JavaMail para envio de e-mails;
- JCache como interface do uso do Cache;
- java.net para acesso a URLs externas (URL Fetch);
- Log4J e java.util.logging para logging;
- JPA e JDO para persistência (já mencionados).

Desta forma o desenvolvedor não precisa aprender novas APIs, e o mais importante, torna as aplicações mais fáceis de serem migradas, tanto para dentro como para fora do GAE.

O SDK

Para o desenvolvimento de aplicativos para o App Engine, o Google disponibiliza um SDK que inclui um runtime para simular o ambiente do servidor localmente, um utilitário para deploy, um template para novos projetos, targets ANT para as funções principais e alguns exemplos (focando principalmente o suporte a JDO).

Ainda na primeira versão, o ambiente de desenvolvimento é limitado, até se comparado com o equivalente para Python, mas muitas melhorias foram prometidas para as versões futuras.

O Plugin do Eclipse

Juntamente com o lançamento do suporte Java, foi disponibilizado um plugin para o Eclipse. Verifique na seção Links o endereço que contém as instruções de download e instalação (que não foge do padrão Eclipse). O Plugin oferece funções para criação de projetos, execução local e deploy automatizado. Também é fornecido suporte ao Google Web Toolkit (ou GWT), o framework web do Google (apresentado na edição 64 da Java Magazine).

É possível inclusive criar um projeto GAE + GWT. Mais informações sobre o uso do plugin serão apresentadas no exemplo a seguir.

Para demonstrar o uso do SDK, do plugin do Eclipse, das APIs e do processo de deploy, será apresentada uma aplicação de exemplo. A aplicação é um serviço de armazenamento e disponibilização de imagens. Basicamente, um usuário poderá fazer o upload de um arquivo de imagem e ele será disponibilizado publicamente, em uma URL simples.

Graças à oferta sem custo inicial do GAE, além de poder fazer o download do código fonte da aplicação no site da Java Magazine, o leitor poderá acessar e utilizar a aplicação online, no endereço *http://minha-foto.appspot.com.*

Criação do Projeto

A forma mais simples de se criar um projeto para o App Engine é com o plugin do Eclipse. Se o plugin foi instalado corretamente, selecione a opção F*ile>New>Web Application Project*, e uma janela semelhante à Figura 2 será mostrada.

Entre com o nome e o pacote inicial, como é feito em um projeto normal. Para este exemplo não será utilizado o GWT, por isso desmarque a opção Use Google Web Toolkit e clique em Finish para concluir.

Caso não deseje utilizar o Eclipse, a forma mais prática de iniciar um projeto é copiar a pasta new_project_template, localizada na pasta demos da instalação do SDK, e utilizá-la como ponto de partida.

Esta pasta já possui um arquivo build.xml do Ant pronto, faltando apenas configurar a variável appengine.sdk para apontar para a instalação do SDK. Outras IDEs não serão abordadas neste artigo, mas os usuários de NetBeans não terão dificuldades de importar um projeto que utilize o Ant.

[abrir imagem em janela]

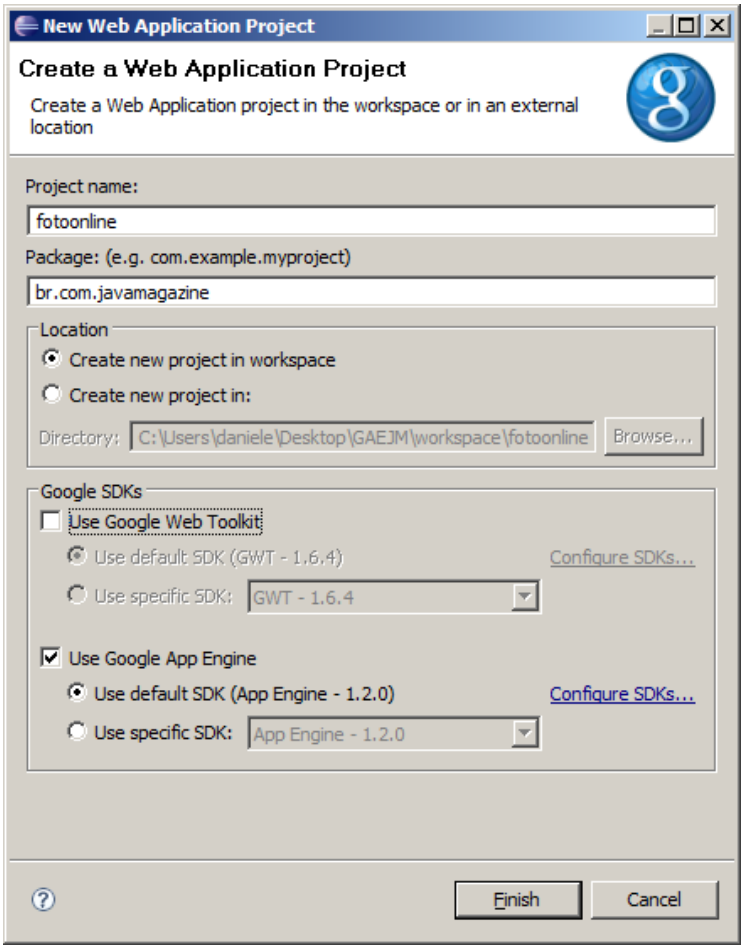

Figura 2. Criando projeto com o plugin do Eclipse

Estrutura do Projeto

A organização dos arquivos em um projeto não é rigidamente imposta, a única necessidade é

que exista uma pasta contendo os arquivos que serão copiados para o servidor. A organização desta pasta deve seguir a estrutura de um arquivo WAR, isto é, possuir uma pasta *WEB-INF* com o arquivo web.xml e as subpastas classes e lib. O papel das pastas e a visibilidade delas também funciona da mesma forma. Mas na verdade, em nenhum momento um arquivo WAR é gerado; como veremos mais adiante, o deploy no Google App Engine é um pouco diferente.

Em projetos criados com o plugin do Eclipse, esta “pasta de deploy” tem o nome de war. Caso você esteja utilizando o *new_project_template*, no script ANT o nome padrão é *www*, que pode ser alterado. O conteúdo desta pasta não é apagado durante um build, por isso deve-se colocar nela todos os arquivos que se deseja copiar para o servidor.

O arquivo *web.xml* é o mesmo que o utilizado pelas outras aplicações Java web. As únicas tags não suportadas são as relacionadas a JNDI e EJB. Na Listagem 2 é mostrado o conteúdo do web.xml da nossa aplicação exemplo. Nele estão definidos os quatro Servlets que criaremos:

- o servlet de entrada (*welcome*), que faz o login;
- o que gera a página para upload e visualização da foto (*verfoto*);
- o servlet que recebe os dados para serem persistidos (*upload*);
- e o que fornece a imagem (*image*).

Listagem 2. web.xml. O cabeçalho foi omitido

```
<web-app xmlns="http://java.sun.com/xml/
   ns/javaee" version="2.5">
    <servlet>
        <servlet-name>welcome</servlet-name>
        <servlet-class>br.com.javamagazine.
          WelcomeServlet</servlet-class>
    </servlet>
<servlet>
        <servlet-name>verfoto</servlet-name>
        <servlet-class>br.com.javamagazine.
          VerFotoServlet</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>upload</servlet-name>
        <servlet-class>br.com.javamagazine.
          UpdateServlet</servlet-class>
    </servlet>
    <servlet>
        <servlet-name>image</servlet-name>
        <servlet-class>br.com.javamagazine.
          FotoImageServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>image</servlet-name>
        <url-pattern>/foto/*</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>upload</servlet-name>
        <url-pattern>/update</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>verfoto</servlet-name>
```

```
        <url-pattern>/user/*</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>welcome</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
</web-app>
```

O arquivo com as configurações específicas para o App Engine é o *appengine-web.xml*, nele são definidos o nome e a versão da aplicação (Listagem 3). Falaremos mais sobre ele na seção de Deploy.
O outro arquivo que precisa ser comentado, mesmo que brevemente, é o *datastore-indexes.xml*. O JDO possui uma linguagem de consulta chamada JDOQL. Para todas as consultas na Datastore é necessário existir um índice previamente definido. Em uma aplicação do GAE estes índices são definidos no arquivo *datastore-indexes.xml*. Na maioria dos casos o desenvolvedor não precisa se preocupar, pois o conteúdo deste arquivo é gerado automaticamente, na pasta *WEB-INF/appengine-generated*, o que vai ser o caso na nossa aplicação de exemplo.

Para os casos mais complexos o desenvolvedor deve criar outro arquivo com o mesmo nome, só que na pasta *WEB-INF*, para que não seja sobrescrito. Ambos os arquivos serão consultados durante a execução, sendo que o definido pelo usuário tem precedência.

Modelo de Dados

Para o nosso exemplo será necessário criar apenas um bean (ou entity, ou POJO se preferir), que chamaremos de UserFoto, com os atributos: ID, nome do usuário, arquivo da imagem, texto de comentário e o horário da última atualização. Pelos motivos citados anteriormente foi optado por utilizar JDO ao invés de JPA. O código fonte da classe é apresentado na Listagem 4.

Listagem 3. appengine-web.xml

```
<appengine-web-app xmlns="http://appengine.google.com/ns/1.0">
  <application>minha-foto</application>
  <version>1</version>
</appengine-web-app
```

Listagem 4. Bean UserFoto com as anotações do JDO. Imports e métodos de acesso omitidos.

```
@PersistenceCapable(identityType = IdentityType.APPLICATION, detachable = "true")
public class UserFoto implements Serializable {

  //O Memcache exige que a classe seja serializavel e o GAE exige que um UID seja
definido
  private static final long serialVersionUID = 54321L;

  public UserFoto(String userName, String texto, Blob imageData) {
    this.userName = userName;
    this.texto = texto;
    this.imageData = imageData;
    this.updateTime = new Date();
  }

  @PrimaryKey
```

```
  @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
  private Long id;

  @Persistent
  private String userName;

  @Persistent
  private com.google.appengine.api.datastore.Blob imageData;

  @Persistent
  private Date updateTime;

    //Métodos de acesso omitidos

}
```

A anotação que define a classe como persistente é @PersistentCapable. O atributo identityType define em que escopo a classe será persistida, neste caso para a aplicação. O segundo atributo, detachable, indica que os atributos da classe podem ser manipulados fora do contexto de uma transação.
Os atributos a serem persistidos devem possuir a anotação @Persistent, e devem ser serializáveis. Os métodos de acesso (getters e setters) foram omitidos na listagem, mas na verdade, para o JDO, eles não precisariam ser definidos, já que o acesso é feito por introspecção.
Este exemplo é uma introdução bem superficial à JDO, que é uma API bem completa. Infelizmente uma apresentação mais profunda das suas funcionalidades não é possível no espaço deste artigo.
O leitor mais atento deve ter percebido que o atributo do arquivo da imagem é do tipo com.google.appengine.api.datastore.Blob, uma dependência da API proprietária, o que não é desejável por razões de portabilidade. Isso foi necessário porque, devido a um bug no GAE, não é possível persistir arrays de bytes. Mas de acordo com os desenvolvedores, é um problema que será resolvido já na próxima versão.

O código que utiliza JDO foi encapsulado em uma classe DAO, mostrado na Listagem 5. No DAO é possível ver que o uso da API do JDO é muito semelhante ao da JPA. As principais operações de persistência são feitas através de uma Interface de gerenciamento, o PersistenceManager, que é obtido por uma PersistenceManagerFactory. Como a criação desta factory é considerada custosa, o Google recomenda que só exista uma instância por aplicação. No nosso exemplo, para manter as coisas simples, foi criada uma classe Singleton para controlar a construção, apresentada na Listagem 6.

Listagem 5. DAO para o bean UserFoto com uso do JCache

```
public class UserFotoDAO {

  private static final Logger log =
   Logger.getLogger(UpdateServlet.class
      .getName());

  private static Cache cache;

  public static UserFoto getUserFoto(String userName) {
    //Procura primeiro no cache
```

```
  UserFoto uf = (UserFoto) getCache().get(userName);

  if (uf != null) {
    log.info("Utilizando UserFoto do cache");
    return uf;
  }

  PersistenceManager pm = null;
  try {
    pm = PMF.get().getPersistenceManager();
    Query query = pm.newQuery("select from " +
     UserFoto.class.getName());
    query.setFilter("userName == paramUser");
    query.declareParameters("String paramUser");
    query.setUnique(true);
    uf = (UserFoto) query.execute(userName);

    getCache().put(userName, uf);
    return uf;
  }
  finally {
    if (pm != null) pm.close();
  }

}

public static void saveOrUpdateUserFoto(String userName,
 String texto, Blob imageData)
{
  PersistenceManager pm = null;
  try {
    pm = PMF.get().getPersistenceManager();
    UserFoto uf = getUserFoto(userName);

    if (uf == null) {
      log.info(userName + " registrando foto.");
      uf = new UserFoto(userName, texto, imageData);
      pm.makePersistent(uf);
    }
    else {
      log.info(userName + " atualizando os dados. ");
      uf.setUpdateTime(new Date());
      uf.setTexto(texto);
      uf.setImageData(imageData);
      getCache().put(userName, uf);
    }
  }
  finally {
    if (pm != null) pm.close();
  }
}

private static Cache getCache() {
  if (cache == null) {
    try {
      CacheFactory factory = CacheManager.
```

```
        getInstance().getCacheFactory();
        cache = factory.
        createCache(Collections.emptyMap());
      }
      catch (CacheException cex) {
        log.severe("Problemas com o cache: " +
        cex.getMessage());
      }
    }
    return cache;
  }
}
```

Listagem 6. Classe para acesso ao PersistenceManagerFactory

```
public final class PMF {
    private static final PersistenceManagerFactory pmfInstance =
        JDOHelper.
        getPersistenceManagerFactory("transactions-optional");

    private PMF() {}

    public static PersistenceManagerFactory get() {
        return pmfInstance;
    }
}
```

Ainda no DAO da Listagem 5, no método getUserFoto(), é mostrado como pode ser feita uma consulta usando JDOQL, a linguagem de consulta do JDO. A API permite que a consulta seja feita de várias formas. Na listagem é mostrado o estilo por métodos, mas poderíamos ter usado uma única String, desta forma:

```
Query query = pm.newQuery(
  "select from UserFoto where userName == paramUser " +
  "parameters String paramUser");
```

A persistência deve ser feita dentro de uma transação, que é gerenciada usando o PersistenceManger, como mostrado no método saveOrUpdateUserFoto().

Autenticação

Como este artigo é sobre o GAE, vamos utilizar o serviço de autenticação padrão da plataforma. A Interface com os principais métodos é a UserService. Para obter uma instância de UserService deve-se utilizar a factory UserServiceFactory.

A utilização da API pode ser vista no Servlet de entrada da aplicação, exibido na Listagem 7. Primeiro é verificado se o usuário está logado, caso não esteja, é feito um redirecionamento para a página de login. Caso esteja logado, é verificado se já possui uma foto adicionada. Caso possua, os atributos do bean são adicionados ao objeto do request para ser acessado pela página de edição.

Listagem 7. Método doGet() do Servlet de entrada da aplicação, WelcomeServlet.java

```
    public void doGet(HttpServletRequest req,
```

```
                                HttpServletResponse resp)
      throws IOException, ServletException
{
  UserService userService = UserServiceFactory.
   getUserService();
  User user = userService.getCurrentUser();

  if (user != null) {
    req.setAttribute("user", user);
    req.setAttribute("logouturl",
    userService.createLogoutURL("/"));

    UserFoto userFoto = (UserFoto) UserFotoDAO
        .getUserFoto(user.getNickname());
    if (userFoto != null) {
      req.setAttribute("texto", userFoto.getTexto());
      req.setAttribute("updateTime",
      SimpleDateFormat.getDateTimeInstance()
          .format(userFoto.getUpdateTime()));
    }
    req.getRequestDispatcher("/fotouser.jsp").
     forward(req, resp);
  }
  else {
    req.setAttribute("loginurl",
     userService.createLoginURL("/"));
    req.getRequestDispatcher("/welcome.jsp").
     forward(req, resp);
  }
}
```

Para simular o login offline o ambiente de desenvolvimento possui uma implementação de teste onde é possível simular o login de um usuário (Figura 3). Neste momento qualquer nome de usuário é aceito.

[abrir imagem em janela]

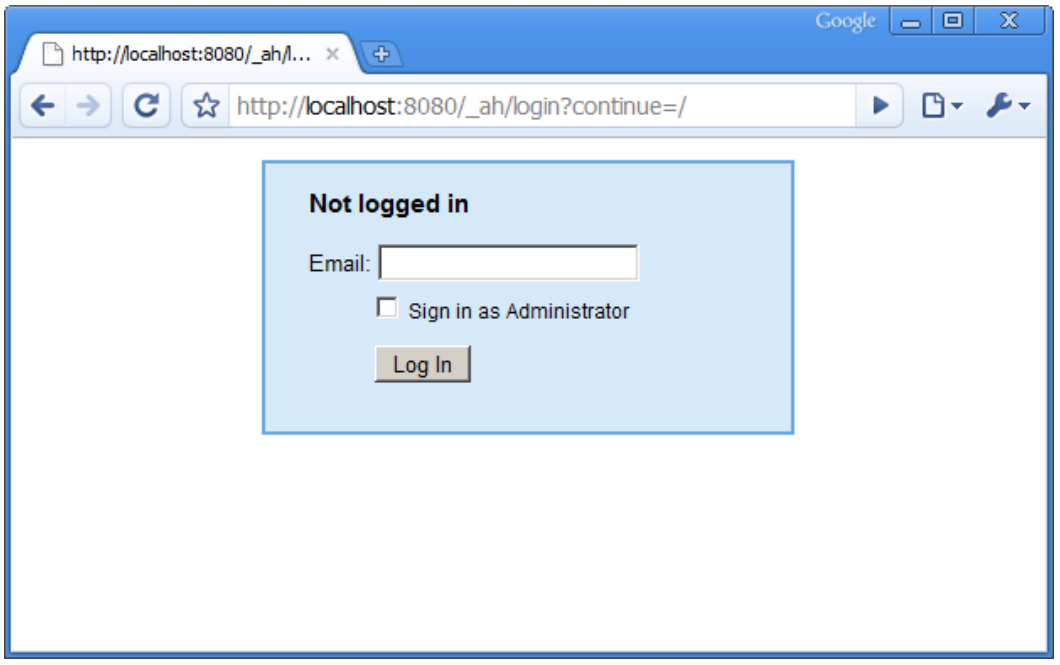

Figura 3. Simulação de autenticação

Cache

Para uma aplicação que disponibiliza imagens, o uso do Cache é fundamental. Conforme mencionado, o serviço de Cache do GAE implementa a API JCache (JSR107). Esta API é simples de utilizar, a principal interface é javax.cache.Cache, que extende java.util.Map, muito familiar aos programadores Java.

No nosso exemplo, todo acesso ao Cache foi encapsulado no DAO. O método privado getCache(), na Listagem 5, cuida da inicialização do Cache, que é consultado antes do acesso à base de dados e atualizado quando o objeto UserFoto é modificado.

Para que um objeto possa ser armazenado no cache é necessário que ele seja serializável, isto é, implemente a interface java.io.Serializable. O GAE também exige que classes serializáveis definam o atributo serialVersionUID, que é um identificador usado para diferenciar versões de classes. O programador pode definir qualquer número neste campo, inclusive o mesmo para classes diferentes, mas é uma boa prática definir um número diferente para cada classe.

Nota

É o processo de acessar atributos e métodos de um objeto de forma dinâmica, durante a execução. Utilizando a API de reflection do Java é possível acessar até mesmo atributos privados.

Upload e Tratamento da Imagem

Para upload do arquivo da foto foi utilizada a biblioteca Apache File Upload. Sua utilização é relativamente simples, e por não ser diretamente relacionada com o App Engine, o código não será mostrado no artigo.

Para demonstrar o uso da API de imagens vamos incluir no Servlet de upload um código para redimensionar as imagens para que tenham no máximo 500 pixels na sua dimensão maior, sem que haja mudança nas proporções (Listagem 8). Para isso, primeiro é necessário obter um objeto Image a partir da classe ImagesServiceFactory. A transformação a ser aplicada na imagem é encapsulada em um objeto do tipo Transform. Dessa forma é possível agrupar várias transformações para que sejam aplicadas em uma única operação.

Listagem 8. Redimensionamento de imagem

```
//O objeto fotoBytes é um array de bytes
Image foto = ImagesServiceFactory.
makeImage(fotoBytes);
 if (foto.getHeight() > 500 || foto.getWidth() > 500) {
   Transform resize =
   ImagesServiceFactory.makeResize(500, 500);
   foto = ImagesServiceFactory.getImagesService().
   applyTransform(resize, foto);
 }
   fotoBytes = foto.getImageData();
```

Cadastro da Aplicação

Depois de feita a aplicação, o próximo passo é colocá-la no ar. Para isso é necessário criar

uma conta no Google App Engine, o que é bem simples, principalmente para quem já possui uma conta no Google. Basta acessar *appengine.google.com*, logar-se e seguir os passos indicados. Durante o cadastro o usuário precisa entrar com o número do seu telefone celular. Feito o cadastro você receberá um SMS com um código que deve ser usado para terminar o cadastro. Por padrão apenas o suporte Python é habilitado. Para ativar o Java clique no banner na parte superior da página de criação de aplicações.
Depois de efetuado o cadastro, o próximo passo é registrar a aplicação. Os únicos valores necessários são o nome e o título (Figura 4). O nome é utilizado na URL de acesso: *http://.appspot.com*, e uma vez criado não pode ser alterado. O título aparece na tela de autenticação, quando o visitante do site tentar se logar usando Google Accounts, e pode ser alterado a qualquer momento. Atenção para outra limitação, cada usuário pode criar no máximo 10 aplicações, mas como veremos ainda neste artigo, esta limitação pode ser contornada com o recurso de versionamento, que possibilita que mais de uma aplicação rode no mesmo domínio.

O leitor provavelmente vai perceber que é difícil encontrar um bom nome de aplicação disponível. Isso acontece porque não é possível cadastrar um nome que já esteja sendo usado por um usuário do Gmail. Por exemplo, se já existe um ideiax@gmail.com, você não conseguirá cadastrar o domínio ideiax.appspot.com, a não ser que este seja o seu e-mail.

[abrir imagem em janela]

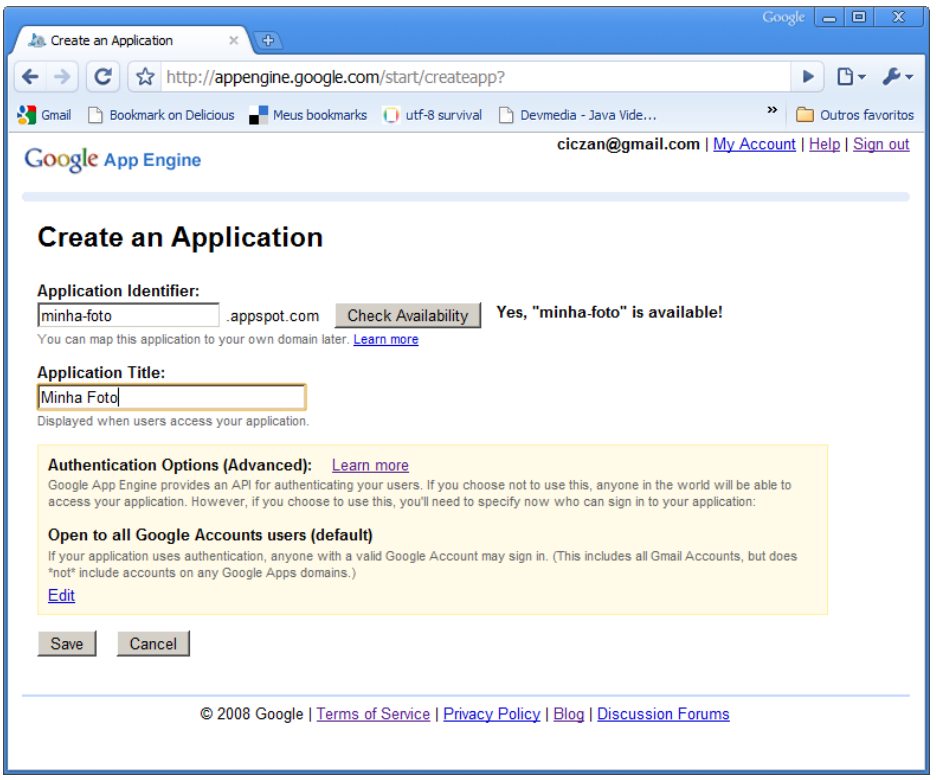

Figura 4. Criando uma Aplicação no Servidor

Deploy

Para que seja realizado o deploy (upload e instalação) da sua aplicação é preciso criar uma conta e uma aplicação no servidor, o que foi visto no passo anterior. Feito isso, atualize o nome definido na tag do arquivo *appengine-web.xml* para o mesmo nome definido no servidor. O passo final é o deploy propriamente dito, que pode ser feito com o utilitário *appcfg* ou com o plugin do Eclipse.

O executável do appcfg fica na pasta bin da instalação do SDK. No Windows o arquivo se chama *appcfg.cmd* e no Linux *appcfg.sh*, mas em ambos os casos a utilização é a mesma: *appcfg update [nome-aplicação]/war*. Antes do upload todo o código é compilado, inclusive arquivos JSF.

Nota

Quando o SDK é instalado junto com o plugin do Eclipse, a pasta do SDK é [Instalação do Eclipse]/plugins/ com.google.appengine.sdkbundle_[versão do build]

O deploy com o plugin do Eclipse é o mais simples possível, basta apertar o botão com o logo do GAE (uma turbina de avião) na barra de ferramentas e entrar com o login e senha do seu usuário. Não se esqueça de atualizar o nome da aplicação no *appengine-web.xml* antes. A Figura 5 apresenta uma página da aplicação em execução já no ambiente on-line.

[abrir imagem em janela]

Figura 5. Página para upload da foto

O Console Administrativo

Para possibilitar o gerenciamento e monitoramento de aplicações, o GAE oferece uma interface do tipo painel de controle, chamado Console Administrativo.

Entre as funções de monitoramento é possível acompanhar o volume de dados transmitidos, uso de CPU, dados persistidos e todos os outros parâmetros que possuam uma cota de utilização. O usuário também conta com o auxílio de gráficos e um ranking dos recursos

(Servlets ou arquivos estáticos) mais acessados (Figura 6). Além das cotas, é possível acompanhar o serviço de Cron, os Logs da aplicação e consultar a base de dados por meio da GQL, um SQL voltado ao Datastore.

[abrir imagem em janela]

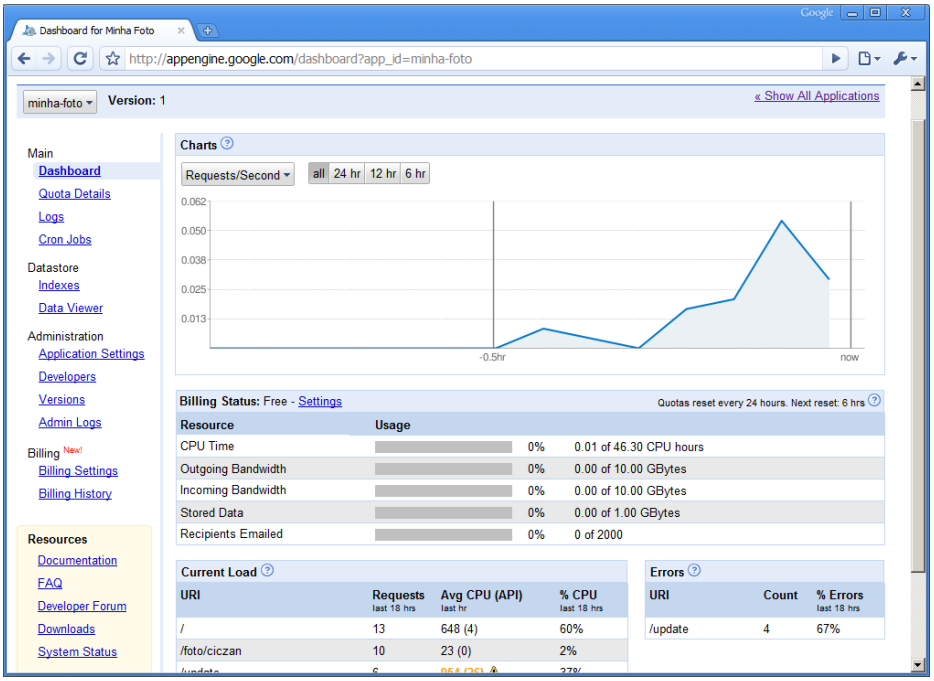

Figura 6. Console Administrativo

Na parte administrativa é possível configurar o acesso de outros desenvolvedores, editar a base de dados (também via GQL), habilitar a cobrança e controlar o versionamento da aplicação. A cobrança permite um ajuste fino na disponibilização dos recursos. No exemplo da Figura 7 configurei um orçamento máximo de U$0,50 ao dia, o que foi suficiente para 3 horas de processamento, 20GB de armazenamento e algumas centenas de MB de banda. Esta é a divisão “Standard” de dinheiro, é possível fazer o ajuste manual da distribuição.
Outro ponto importante na administração é o versionamento. O GAE permite a criação de apenas 10 aplicações simultâneas, e não é possível removê-las depois de criadas, o que é um problema quando se deseja testar modificações em um aplicativo que já tenha uma versão em produção. Com o versionamento é possível testar versões novas sem interferir na atual, basta alterar o nome da versão no arquivo *appengine-web.xml* que ele não sobrescreverá mais a versão anterior e estará acessível em http://.latest..appspot.com.

A qualquer momento pode-se alterar qual é a versão acessível na URL oficial no Console Administrativo. A versão não precisa ser um número, pode ser qualquer texto. O único problema é que a base de dados é compartilhada entre as versões, mas se bem planejado o risco de comprometimento da base é muito pequeno.

[abrir imagem em janela]

Figura 7. Configuração de cobrança

Conclusão

Ainda é cedo para dizer qual o impacto que o Google App Engine terá no mundo do desenvolvimento web. Neste artigo tentamos apresentar não só o que ele é e faz, mas também o que ele não é e não faz (ainda), de forma a dar ao leitor subsídios para decidir se a adoção da plataforma é vantajosa.

Observando as listas de discussões, é perceptível que a equipe do GAE está empenhada em melhorar a plataforma e as ferramentas. Com certeza as próximas versões trarão boas novidades, espera-se que principalmente na parte de gerenciamento do Datastore, que ainda é muito cru. Não se sabe se o Google planeja transformar o GAE em um Application Server completo, com EJB e JMS. Ao que parece o foco são pequenas e médias empresas, que não desejam montar ou manter uma infra-estrutura grande, e empresas do tipo “Start-up”, que estão começando com pouco capital. Mas estas são todas suposições, a única coisa certa até agora é que os desenvolvedores Java ganharam uma excelente ferramenta para colocarem suas idéias “no ar”.