Google App Engine

Google App Engine

- Introduction
- Cloud Computing
- Google App Engine
- Getting Start with GAE
- Using JSP in GAE
- Using Static Files

Google App Engine

- Storing Data
- Blobstore API
- Image API
- Struts
- Exercise: Shop Around
- Memcache API

Google App Engine

- Mail API
- Users Service
- URL Fetch
- Channel API
- Task Queue
- Apps Market

CLOUD COMPUTING

What is Cloud Computing ?

National Institute Of Standard and Technology Definition of Cloud Computing:

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

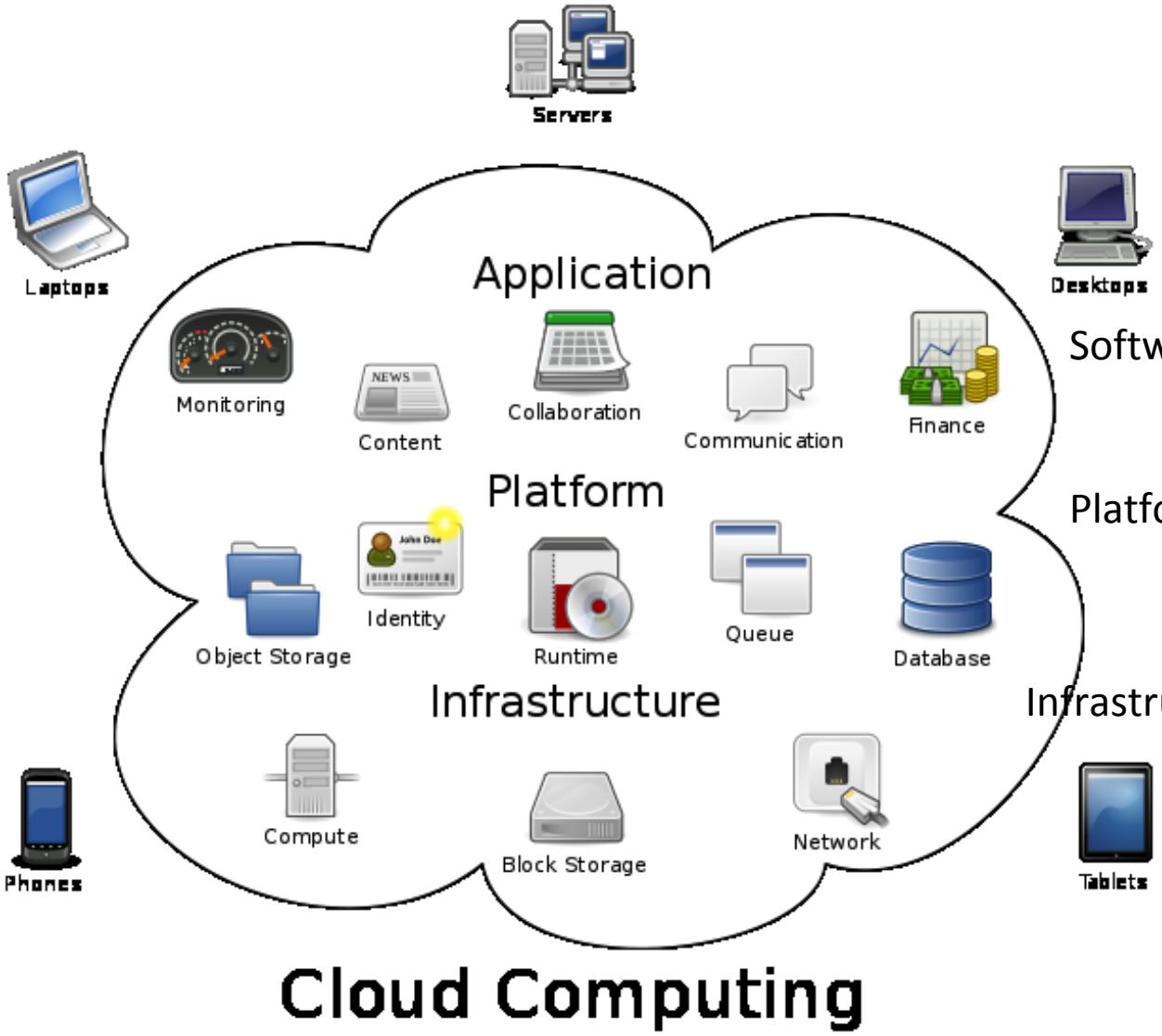
http://csrc.**nist**.gov/publications/**nist**pubs/800-145/SP800-145.pdf

Characteristics

- On-demand self-service
- Broad network access
- Resource pooling
- Rapid elasticity
- Measured service

What is Cloud Computing ?

“The delivery of computing requirements as a service”



Software as a Service (SaaS)

Platform as a Service (PaaS)

Infrastructure as a Service (IaaS)

From: http://en.wikipedia.org/wiki/File:Cloud_computing.svg

Cloud Computing

Shared resources

Cost reduction

Flexibility

Usage based cost model

Easy scaling

GOOGLE APP ENGINE

Google App Engine

Platform as a service (PaaS)

Google infrastructure

Automatic scalability

Free up to 1GB of storage serving ~5M page views/month

Google App Engine

- Programming languages
  - Java
  - Python
  - Go
- Storing data
  - App Engine Datastore – NoSQL datastore
  - Google Cloud SQL – relational SQL database
  - Google Cloud Storage – objects and files up to terabytes
- Integrating with Google Accounts

Google App Engine

Services

- Blobstore
- Image
- Mail
- Memcache
- Users
- URL Fetch
- Channel
- Task Queue

GETTING START WITH GAE

Getting Start – Installing

Java SDK

- GAE supports Java 5 and Java 6

Eclipse

Google Plugin for Eclipse

- http://dl.google.com/eclipse/plugin/3.3 (Europa)
- http://dl.google.com/eclipse/plugin/3.4 (Ganymede)
- http://dl.google.com/eclipse/plugin/3.5 (Galileo)
- http://dl.google.com/eclipse/plugin/3.6 (Helios)
- http://dl.google.com/eclipse/plugin/3.7 (Indigo)

Getting Start – Creating a Project

Creating a Project in Eclipse

New Project > Google > Web Application Project

Java Servlet Standard using WAR standard layout for Java web applications

HelloworldServlet

```
package helloworld;
import java.io.IOException;
import javax.servlet.http.*;

@SuppressWarnings("serial")
public class HelloworldServlet extends HttpServlet {
   public void doGet(HttpServletRequest req,
   HttpServletResponse resp)throws IOException {
     resp.setContentType("text/plain");
     resp.getWriter().println("Hello, world");
   }
}
```

Hello World

- Run > Run As > Web Application
- Open http://localhost:8888/helloworld

web.xml

```
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-
   instance" ...>
<servlet>
<servlet-name>Helloworld</servlet-name>
<servlet-class>helloworld.HelloworldServlet</servlet-
   class>
</servlet>
<servlet-mapping>
<servlet-name>Helloworld</servlet-name>
<url-pattern>/helloworld</url-pattern>
</servlet-mapping>
<welcome-file-list>
<welcome-file>index.html</welcome-file>
</welcome-file-list>
</web-app>
```

Uploading the application

Registering the application

https://appengine.google.com/

Uploading from Eclipse

Click the App Engine deploy button on the toolbar

Accessing the application

http://your_app_id.appspot.com/

appengine-web.xml

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

Quota and Limit

- The number of times the application has been uploaded by a developer. The current quota is 1,000 per day.
- 10,000 uploaded files per version.
- Each file is limited to a maximum size of 32 megabytes.
- The total size of all files can not exceed 150 megabytes

USING JSP

Using JSP

- JSP files under war folder is automatically mapped to a URL path

http://localhost:8888/hello.jsp → war/hello.jsp

hello.jsp

```
<%@ page contentType="text/html;charset=UTF-8"
  language="java" %>
<html>
<head>
</head>
<body>
<%
String hello = "Hello World!";
%>

<%=hello%>
</body>
</html>
```

JSPs on development server

- The development server converts the JSP into Java source code, then compiles the Java source into Java bytecode.
- The development server regenerates and compiles JSPs automatically if the original JSP files change

JSPs on GAE server

- SDK compiles all JSPs to bytecode, and only uploads the bytecode to GAE server
- App Engine server uses the compiled JSP classes

USING STATIC FILES

Using Static Files

- Static files: Images, CSS, JavaScript, Movies, Flash, etc.
- By default, App Engine makes all files in the WAR available as static files except JSPs and files in WEB-INF/
- Configure which files App Engine treats as static files using theappengine-web.xml file.
- App Engine serves static files from separate servers than those that invoke servlets

hellocss.jsp

```
<%@ page contentType="text/html;charset=UTF-8"
   language="java" %>
<html>
<head>
<link type="text/css" rel="stylesheet" href="/css/
   hello.css">
</head>
<body>
<%
String hello = "Hello World!";
%>

<%=hello%>
</body>
</html>
```

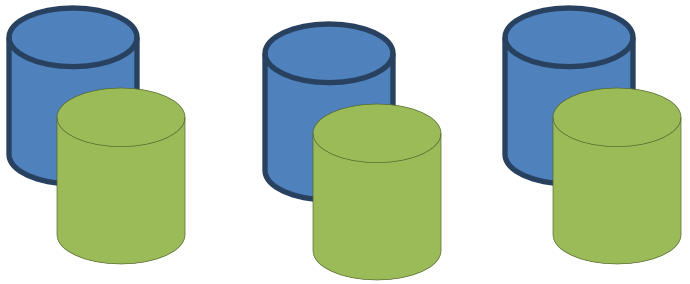
hello.css

```
body {
  color: white;
  background-color: blue;
}
```

USING DATASTORE

Using Datastore

- Storing data in a scalable web application
  - Replication
  - Distribution
  - Load balancing



App Engine Datastore

- App Engine's datastore is built on top of Bigtable
- Distributed architecture to automatically manage scaling
- High Replication Datastore (HRD), uses the Paxos algorithm to replicate data across multiple data center
- Eventual consistency

App Engine Datastore

- Datastore writes scale by automatically distributing data as necessary
- Datastore only supported queries that performance scales with the size of the result set (as opposed to the data set)
- A query whose result set contains 100 entities performs the same whether it searches over a hundred entities or a million

App Engine Datastore

- The following are not supported:
  - Join operations
  - Inequality filtering on multiple properties
  - Filtering of data based on results of a subquery

App Engine Datastore

App Engine Datastore

- Entities of the same kind can have different properties
- Different entities can have properties with the same name but different value types

Data Types

- Integers
- Floating-point numbers
- Strings
- Dates
- Binary data

https://developers.google.com/appengine/docs/java/datastore/entities#Properties_and_Value_Types

Using Datastore

- Low-level API
- Java Data Objects(JDO)
- Java Persistence API (JPA)
- Other frameworks:
  - Objectify
  - Twig
  - Slim3

HelloDatastoreServlet.java

```
public class HelloDatastoreServlet extends
   HttpServlet {
    public void doGet(HttpServletRequest req,
      HttpServletResponse resp) throws IOException {
    Entity data = new Entity("HelloDatastore");
    data.setProperty("firstname", "John");
    data.setProperty("lastname", "Smith");
    DatastoreService datastore =
      DatastoreServiceFactory.getDatastoreService();
    datastore.put(data);
    }
}
```

Kind

Properties

Entity

ReadServlet.java

```
public class ReadServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse
      resp) throws IOException {
        DatastoreService datastore =
          DatastoreServiceFactory.getDatastoreService();
        Query query = new Query("HelloDatastore");
        List<Entity> dataList =
          datastore.prepare(query).asList(FetchOptions.Builder.wit
          hDefaults());
        for (Entity data : dataList) {
            resp.getWriter().println(data.getProperty("firstname")
              );
            resp.getWriter().println(data.getProperty("lastname")
              );
        }
    }
}
```

Transactions

- A transaction is an operation or set of operations that is atomic
- An operation may fail when:
  - Too many users try to modify an entity group simultaneously.
  - The application reaches a resource limit.
  - The Datastore encounters an internal error.
- When two or more transactions simultaneously attempt to modify entities in one or more **common entity groups**, only the first transaction to commit its changes can succeed; all the others will fail on commit

HelloDatastoreServlet – with TXN

```
public class HelloDatastoreServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse resp)
       throws IOException {
        DatastoreService datastore =
          DatastoreServiceFactory.getDatastoreService();
        Transaction txn = datastore.beginTransaction();
        try {
            Entity data = new Entity("HelloDatastore");
            data.setProperty("firstname", "John");
            data.setProperty("lastname", "Smith");
            datastore.put(data);
            txn.commit();
        }
        finally {
        if (txn.isActive()) {
                txn.rollback();
            }
        }
    }
}
```

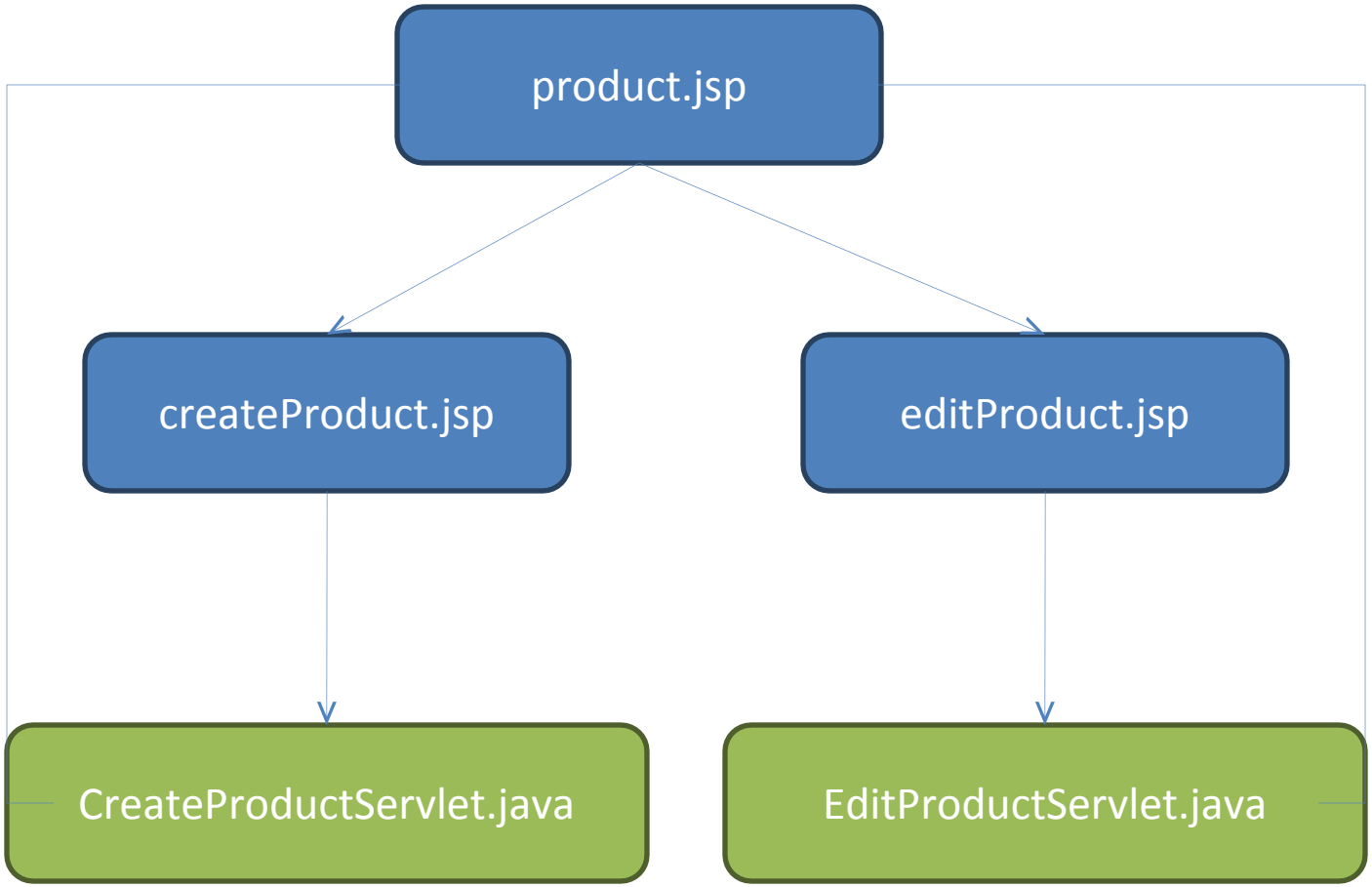
Transactional Task Enqueuing

- Enqueue up to 5 tasks when the transaction is committed successfully
- Such as sending an email to confirm a purchase

```
Queue queue = QueueFactory.getDefaultQueue();

queue.add(TaskOptions.Builder.url("/path/to/queue"));
```

Example: Product Pages



product.jsp

```
<%@ page contentType="text/html;charset=UTF-8"
  language="java" %>
<%@ page import="java.util.List" %>
<%@ page
  import="com.google.appengine.api.datastore.*" %>

<html>
<head>
<link type="text/css" href="/bootstrap/css/
  bootstrap.css" rel="stylesheet" />
</head>
<body>
</body>
</html>
```

product.jsp - body

```
<%
DatastoreService datastore = DatastoreServiceFactory.getDatastoreService();
Query query = new Query("Product");
List<Entity> products =
    datastore.prepare(query).asList(FetchOptions.Builder.withDefault());
if (products.isEmpty()) {
%>
        <p>Empty store.</p>
<%
    } else {
    for (Entity product : products) {
%>
<a href="editProduct.jsp?key=<%=product.getKey().getId()%>">edit</a> Name:
    <%=product.getProperty("name")%> - <%=product.getProperty("price")%><br>
<%
}
}
%>
<br>
<a href="createProduct.jsp">Create new product</a>
```

createProduct.jsp

```
<%@ page contentType="text/html;charset=UTF-8"
   language="java" %>
<html>
<head>
</head>
<body>
<form method="post" action="/createProduct">
Product Name: <input type="text"
   name="productName"><br>
Price: <input type="text" name="productPrice"><br>
<input type="submit">
</form>
</body>
</html>
```

CreateProductServlet.java

```
public class CreateProductServlet extends HttpServlet {
   public void doPost(HttpServletRequest req, HttpServletResponse
   resp)
        throws IOException {
        String name = req.getParameter("productName");
        String price = req.getParameter("productPrice");


        Entity product = new Entity("Product");
        product.setProperty("name", name);
        product.setProperty("price", Float.parseFloat(price));
        DatastoreService datastore =
   DatastoreServiceFactory.getDatastoreService();
        datastore.put(product);
        resp.sendRedirect("/datastore/product.jsp");
   }
}
```

editProduct.jsp

```
<%@ page contentType="text/html;charset=UTF-8"
  language="java" %>
<%@ page import="java.util.List" %>
<%@ page import="java.lang.Long" %>
<%@ page
  import="com.google.appengine.api.datastore.*"
  %>
<html>
<head>
</head>
<body>

</body>
</html>
```

editProduct.jsp – body

```
<%
Key key = KeyFactory.createKey("Product",
   Long.parseLong(request.getParameter("key")));
Entity product = null;
try {
    product = datastore.get(key);
} catch (EntityNotFoundException e) {
}
%>
```

editProduct.jsp – body continue

```
<%if(product == null) { %>
   Entity not found!!!
<%}else{%>
    <form method="post" action="/editProduct">
    <input type="hidden" name="key"
      value="<%=request.getParameter("key")%>">
    Product Name: <input type="text" name="productName"
      value="<%=product.getProperty("name")%>"><br>
    Price: <input type="text" name="productPrice"
      value="<%=product.getProperty("price")%>"><br>
    <input type="submit">
    </form>
<%}%>
```

EditProductServlet.java

```
public class EditProductServlet extends HttpServlet {
     public void doPost(HttpServletRequest req, HttpServletResponse resp)
     throws IOException {
          String name = req.getParameter("productName");
          String price = req.getParameter("productPrice");
          Key key = KeyFactory.createKey("Product",
             Long.parseLong(req.getParameter("key")));
          DatastoreService datastore = DatastoreServiceFactory.getDatastoreService();
          Entity product;
          try {
               product = datastore.get(key);
               product.setProperty("name", name);
               product.setProperty("price", Float.parseFloat(price));
               datastore.put(product);
          } catch (EntityNotFoundException e) {
               resp.getWriter().println("Entity not found!!");
               return;
          }
          resp.sendRedirect("/datastore/product.jsp");
     }
}
```

Queries

- Query.addFilter
- Query.FilterOperator
- Query.addSort

```
Query q = new Query(“Product");
q.addFilter("name", Query.FilterOperator.EQUAL, name);
q.addFilter NOT_EQUAL Query.FilterOperator.LESS_THAN, maxPrice);
q.addSort("price", SortDirection.ASCENDING);
```

FilterOperator

Operator	Meaning
EQUAL	Equal to
LESS_THAN	Less than
LESS_THAN_OR_EQUAL	Less than or equal to
GREATER_THAN	Greater than
GREATER_THAN_OR_EQUAL	Greater than or equal to
NOT_EQUAL	Not equal to
IN	Member of (equal to any of the values in a specified list)

NOT_EQUAL

- NOT_EQUAL filter is replaced with a LESS_THAN filter and a GREATER_THAN filter
- A query can have no more than one NOT_EQUAL filter

product.jsp - queries

```
Query query = new Query("Product");
if(request.getParameter("name") != null) {
   query.addFilter("name", Query.FilterOperator.EQUAL,
   request.getParameter("name"));
}
if(request.getParameter("maxPrice") != null) {
   query.addFilter("price", Query.FilterOperator.LESS_THAN,
   Float.parseFloat(request.getParameter("maxPrice")));
}
query.addSort("price", Query.SortDirection.ASCENDING);
```

Limit and Offset

- FetchOptions.Builder.withLimit(5).offset(5))

```
FetchOptions fetchOptions = FetchOptions.Builder.withLimit(5);
if(request.getParameter("offset") != null) {
   fetchOptions.offset(Integer.parseInt(request.getParameter("offset")));
}
List<Entity> products = datastore.prepare(query).asList(fetchOptions);
```

Indexes

- Index configuration file:

  war/WEB-INF/appengine-generated/datastore-indexes.xml
- The development web server automatically adds suggestions to this file as it encounters queries that do not yet have indexes configured

Data Consistency

- High Replication Datastore (HRD) provides high availability for your reads and writes by storing data synchronously in multiple data centers
- **Strongly consistent** queries guarantee the freshest results, but may take longer to complete.
- **Eventually consistent** queries generally run faster, but may occasionally return stale results.

Data Consistency

- To obtain **strongly consistent** query results, you need to use an ancestor query limiting the results to a single **entity group**
- Entity groups are a unit of consistency as well as transactionality

```
Key storeKey = KeyFactory.createKey("Store", "store1");
Entity product = new Entity("Product", storeKey);
```

Using JPA

- Standard interface for storing objects containing data into a relational database
- An application that uses the JPA interface can work with different databases without using any vendor-specific database code
- Simplifies porting an application between different database vendors

Google App Engine Datastore

USING JPA

Using JPA

- Create persistence.xml
- Create Data Classes
- Create EntityManagerFactory

META-INF/persistence.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
    http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd" version="1.0">
  <persistence-unit name="transactions-optional">
  <provider>org.datanucleus.store.appengine.jpa.DatastorePersistenceProvider</provider>
    <properties>
      <property name="datanucleus.NontransactionalRead" value="true"/>
      <property name="datanucleus.NontransactionalWrite" value="true"/>
      <property name="datanucleus.ConnectionURL" value="appengine"/>
    </properties>
  </persistence-unit>
</persistence>
```

META-INF/persistence.xml

- datastoreReadConsistency (STRONG/ EVENTUAL) Default is STRONG

```
<property name="datanucleus.appengine.datastoreReadConsistency"
   value="EVENTUAL" />
```

- query.timeout

```
<property name="javax.persistence.query.timeout" value="5000" />
```

- datastoreWriteTimeout

```
<property name="datanucleus.datastoreWriteTimeout" value="10000" />
```

- multiple <persistence-unit> elements

Data Class - Product.java

```
package jpa;

import com.google.appengine.api.datastore.Key;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
public class Product {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Key key;

    private String name;
    private float price;
    // getters and setters
}
```

EMF.java

```
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
public final class EMF {
   private static final EntityManagerFactory emfInstance =
   Persistence.createEntityManagerFactory("transactions-
   optional");

   private EMF() {}

   public static EntityManagerFactory get() {
       return emfInstance;
   }
}
```

product.jsp - query

```
<%@ page import="java.util.List" %>
<%@ page import="jpa.*" a%>
<%@ page import="javax.persistence.*" %>

<%
EntityManager em = EMF.get().createEntityManager();
List<Product> products = null;
try {
  Query q = em.createQuery("select p from Product p");
  products = q.getResultList();
%>
// getting properties
<%
}
finally {
   em.close();
}
%>
```

product.jsp – getting properties

```
<%if (products.isEmpty()) { %>
    <p>Empty store.</p>
<% } else { %>
    <% for (Product product : products) { %>
    <a href="editProduct.jsp?
      key=<%=KeyFactory.keyToString(product.getKey())%>">
    edit</a>
    Name: <%=product.getName()%> - <%=product.getPrice()%><br>
    <% } %>
<% } %>
```

CreateProductServlet.java

```
Product p = new Product();
p.setName(name);
p.setPrice(Float.parseFloat(price));
EntityManager em = EMF.get().createEntityManager();
try {
   em.persist(p);
}
finally {
   em.close();
}
```

editProduct.jsp

```
<%@ page import="java.util.List" %>
<%@ page import="jpa.*" %>
<%@ page import="javax.persistence.*" %>
<%@ page import="com.google.appengine.api.datastore.*" %>
<%
EntityManager em = EMF.get().createEntityManager();
List<Product> products = null;
try {
    Query q = em.createQuery("select p from Product p where p.key = :key");
    q.setParameter("key", request.getParameter("key"));
    products = q.getResultList();
    Product product = products.get(0);
%>
// getting properties
<%
}
finally {
   em.close();
}
%>
```

editProduct.jsp – getting properties

```
<%if(product == null) { %>
   Entity not found!!!
<%}else{%>
    <form method="post" action="/jpaEditProduct">
    <input type="hidden" name="key"
      value="<%=request.getParameter("key")%>">
    Product Name: <input type="text" name="productName"
      value="<%=product.getName()%>"><br>
    Price: <input type="text" name="productPrice"
      value="<%=product.getPrice()%>"><br>
    <input type="submit">
    </form>
<%}%>
```

EditProductServlet.java

```
EntityManager em = EMF.get().createEntityManager();
List<Product> products = null;
try {
    Query q = em.createQuery("select p from Product p where
      p.key = :key");
    q.setParameter("key", req.getParameter("key"));
    products = q.getResultList();
    Product product = products.get(0);
    product.setName(name);
    product.setPrice(Float.parseFloat(price));
    em.persist(product);
}
finally {
   em.close();
}
```

Google App Engine Datastore

USING JDO

Using JDO

- Java Data Objects (JDO) is a standard interface for storing objects containing data into a database
- An application that uses the JDO interface can work with different kinds of databases without using any database-specific code

META-INF/jdoconfig.xml

```
<?xml version="1.0" encoding="utf-8"?>
<jdoconfig xmlns="http://java.sun.com/xml/ns/jdo/jdoconfig"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="http://java.sun.com/xml/ns/jdo/jdoconfig">

   <persistence-manager-factory name="transactions-optional">
       <property name="javax.jdo.PersistenceManagerFactoryClass"
    value="org.datanucleus.store.appengine.jdo.DatastoreJDOPersistenceManagerFacto
    ry"/>
       <property name="javax.jdo.option.ConnectionURL" value="appengine"/>
       <property name="javax.jdo.option.NontransactionalRead" value="true"/>
       <property name="javax.jdo.option.NontransactionalWrite" value="true"/>
       <property name="javax.jdo.option.RetainValues" value="true"/>
    <property name="datanucleus.appengine.autoCreateDatastoreTxns" value="true"/>
   </persistence-manager-factory>
</jdoconfig>
```

Data Class – product.java

```
package jdo;
import javax.jdo.annotations.PersistenceCapable;
import javax.jdo.annotations.IdGeneratorStrategy;
import javax.jdo.annotations.Persistent;
import javax.jdo.annotations.PrimaryKey;
import com.google.appengine.api.datastore.Key;

@PersistenceCapable
public class Product {
    @PrimaryKey
     @Persistent(valueStrategy = IdGeneratorStrategy.IDENTITY)
     private Key key;
    @Persistent
    private String name;
    @Persistent
    private float price;
    public Key getKey() {
          return key;
    }

    // getters and setters
}
```

PMF.java

```
package jdo;
import javax.jdo.JDOHelper;
import javax.jdo.PersistenceManagerFactory;

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

    private PMF() {}

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

product.jsp

```
<%@ page import="java.util.List" %>
<%@ page import="jdo.*" %>
<%@ page import="javax.jdo.*" %>

<%
PersistenceManager pm = PMF.get().getPersistenceManager();
List<Product> products = null;
Query query = pm.newQuery(Product.class);

products = (List<Product>) query.execute();
%>
```

CreateProductServlet.java

```
public class CreateProductServlet extends HttpServlet {
    public void doPost(HttpServletRequest req, HttpServletResponse resp)
    throws IOException {
        String name = req.getParameter("productName");
        String price = req.getParameter("productPrice");
        Product p = new Product();
        p.setName(name);
        p.setPrice(Float.parseFloat(price));

        PersistenceManager pm = PMF.get().getPersistenceManager();
        try {
            pm.makePersistent(p);
        } finally {
            pm.close();
        }
        resp.sendRedirect("/jdo/product.jsp");
    }
}
```

editProduct.jsp

```
<%@ page import="java.util.List" %>
<%@ page import="jdo.*" %>
<%@ page import="javax.jdo.*" %>
<%@ page import="com.google.appengine.api.datastore.*" %>


PersistenceManager pm = PMF.get().getPersistenceManager();
try {
Key k = KeyFactory.createKey(Product.class.getSimpleName(),
   Long.parseLong(request.getParameter("key")));
    Product product = pm.getObjectById(Product.class, k);
```

EditProductServlet.java

```
PersistenceManager pm = PMF.get().getPersistenceManager();
try {
   Product product = pm.getObjectById(Product.class,
    Long.parseLong(req.getParameter("key")));
  product.setName(name);
  product.setPrice(Float.parseFloat(price));
  pm.makePersistent(product);
}
finally {
  pm.close();
}
```

Quota

Resource	Free Default Limit	Billing Enabled Default Limit
Stored Data	1 GB	1 GB free; no maximum
Number of Indexes	200	200

Limit

Limit	Amount
Maximum entity size	1 megabyte
Maximum transaction size	10 megabytes
Maximum number of values in all indexes for an entity	5000

GAE – Day 2

- Blobstore API
- Image API
- Struts
- Exercise: Shop Around
- Memcache API
- Mail API
- Users Service
- URL Fetch
- Channel API
- Task Queue

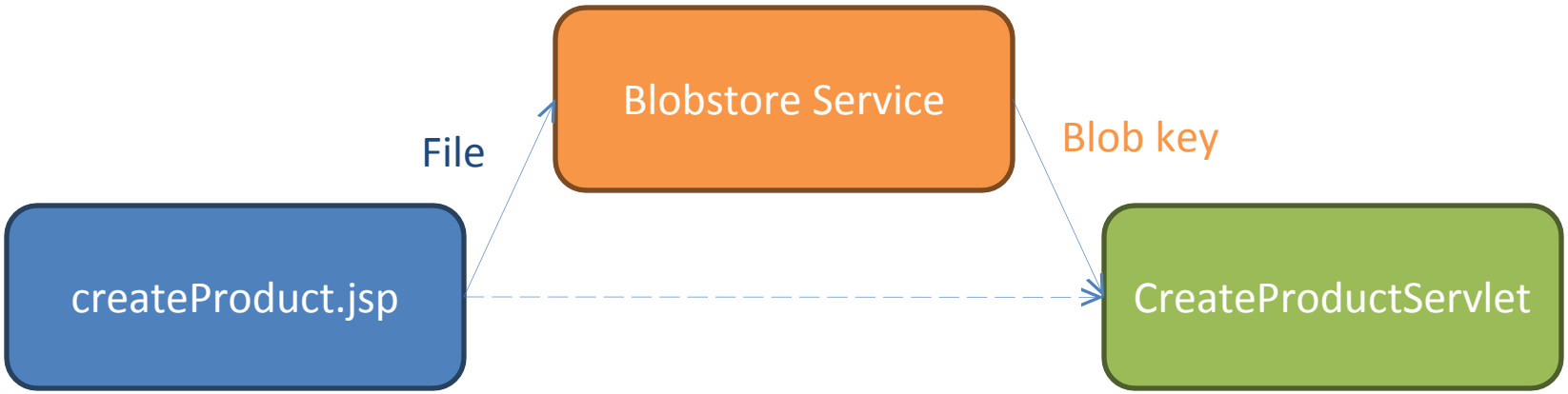
Google App Engine Service

BLOBSTORE API

Blobstore API

- Blob - binary large object
- Useful for serving large files, such as video or image files, and for allowing users to upload large data files
- Blobs are created by uploading a file through an HTTP request

Blobstore API



createProduct.jsp

```
<%
BlobstoreService blobstoreService =
   BlobstoreServiceFactory.getBlobstoreService();
%>
<form method="post" action="<%=
   blobstoreService.createUploadUrl("/blobCreateProduct") %>"
   enctype="multipart/form-data">
Product Name: <input type="text" name="productName"><br>
Price: <input type="text" name="productPrice"><br>
Photo: <input type="file" name="productPhoto"><br>
<input type="submit">
</form>
```

CreateProductServlet.java

```
BlobstoreService blobstoreService =
   BlobstoreServiceFactory.getBlobstoreService();
Map<String, List<BlobKey>> blobs =
   blobstoreService.getUploads(req);
List<BlobKey> blobKeys = blobs.get("productPhoto");

if (blobKeys != null && blobKeys.size() > 0) {
   product.setProperty("photo", blobKeys.get(0).getKeyString());
}
```

ProductPhotoServlet.java

```
package blobstore;

import java.io.IOException;

import javax.servlet.http.*;

import com.google.appengine.api.blobstore.BlobKey;
import com.google.appengine.api.blobstore.BlobstoreService;
import com.google.appengine.api.blobstore.BlobstoreServiceFactory;

public class ProductPhotoServlet extends HttpServlet {
   public void doGet(HttpServletRequest req, HttpServletResponse res)
    throws IOException {
        BlobKey blobKey = new BlobKey(req.getParameter("photo"));
   BlobstoreService blobstoreService =
   BlobstoreServiceFactory.getBlobstoreService();
        blobstoreService.serve(blobKey, res);
   }
}
```

product.jsp

```
<%if(product.getProperty("photo") != null) {%>
<img width="100px" src="/productPhoto?
   photo=<%=product.getProperty("photo")%>">
<br>
<%}%>
```

Google App Engine Service

IMAGE API

Image API

- The Image service Java API apply transformations to images, using a service instead of performing image processing on the application server

Using ImageServiceFactory

- Transforming Images from the Blobstore
- Image Transformations
  - Resize
  - Rotate
  - Flip
  - Crop

ProductPhotoServlet.java

```
BlobKey blobKey = new BlobKey(req.getParameter("photo"));
ImagesService imagesService =
   ImagesServiceFactory.getImagesService();

Image oldImage = ImagesServiceFactory.makeImageFromBlob(blobKey);
Transform resize = ImagesServiceFactory.makeResize(100, 300);

Image newImage = imagesService.applyTransform(resize, oldImage,
   OutputEncoding.JPEG);

byte[] newImageData = newImage.getImageData();

res.setContentType("image/jpeg");
res.getOutputStream().write(newImageData);
```

Using getServingUrl

- Dynamically resize and crop images
- returns a URL that serves the image, and transformations to the image are encoded in this URL
- http://imageurl**=sxx-c**
  where **xx** is an integer from 0–1600 representing the cropped image size in pixels, and **-c** tells the system to crop the image.

product.jsp

```
<img src="<%=ImagesServiceFactory.getImagesService().getServingUrl(new
   BlobKey((String)product.getProperty("photo")))%>=s100-c">
```

Limits

Limit	Amount
maximum data size of image sent to service	32 megabytes
maximum data size of image received from service	32 megabytes
maximum size of image sent or received from service	50 megapixels

Web Framework

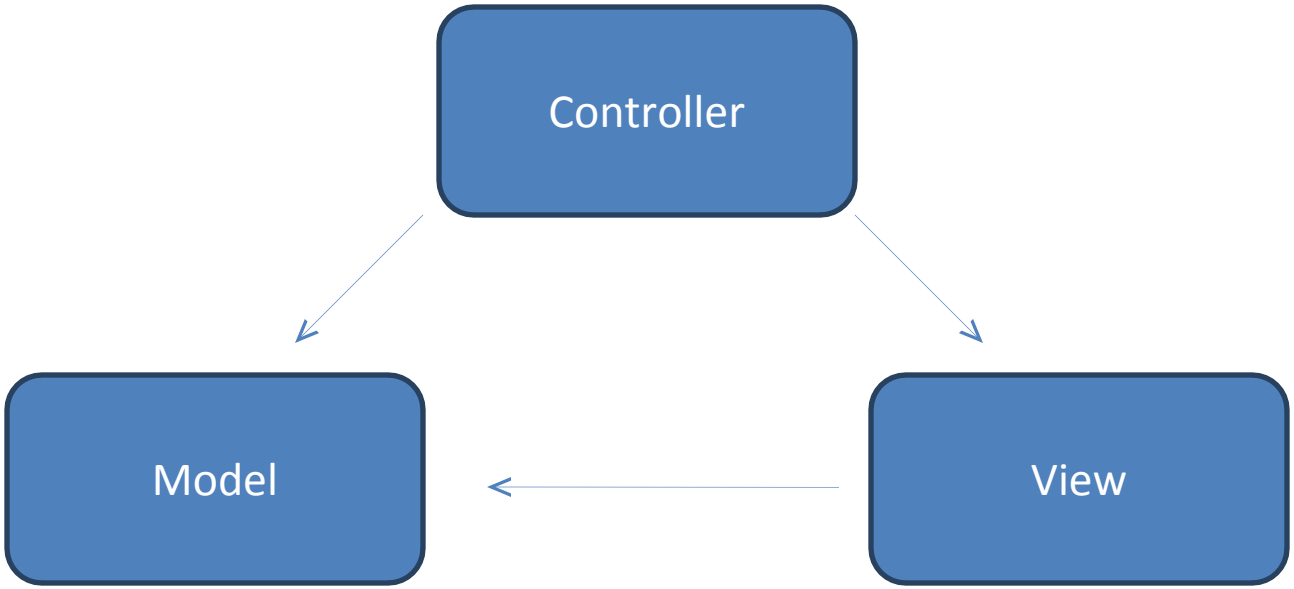
STRUTS

Struts

- open-source framework for creating Java web applications
- The framework provides three key components:
  - A "request" handler provided by the application developer that is mapped to a standard URI.
  - A "response" handler that transfers control to another resource which completes the response.
  - A tag library that helps developers create interactive form-based applications with server pages.

MVC

- model–view–controller (MVC) architectural pattern



Setting Up Struts on GAE

- Download Struts from http://struts.apache.org/
- Copy Jar files to war/WEB-INF/lib
  - commons-fileupload-X.X.X.jar
  - commons-io-X.X.X.jar
  - commons-lang3-X.X.jar
  - commons-logging-X.X.X.jar
  - commons-logging-api.X.X.jar
  - freemarker-X.X.X.jar
  - ognl-X.X.X.jar
  - struts2-core-X.X.X.X.jar
  - xwork-core-X.X.X.jar
  - javassist-X.X.X.jar

Model – HelloWorldModel.java

```
package hellostruts.model;

public class HelloWorldModel {
  private String message;

  public HelloWorldModel() {
    setMessage("Hello World");
  }

  public String getMessage() {
    return message;
  }

  public void setMessage(String message) {
    this.message = message;
  }


}
```

View – helloStruts.jsp

```
<%@ page language="java" contentType="text/html;"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<body>
  <h1>
    <s:property value="helloWorld.message" />
  </h1>
</body>
</html>
```

Controller – HelloWorldAction.java

```
package hellostruts.action;

import hellostruts.model.HelloWorldModel;

import com.opensymphony.xwork2.ActionSupport;

@SuppressWarnings("serial")
public class HelloWorldAction extends ActionSupport {
private HelloWorldModel helloWorld;
  public String execute() {
    helloWorld = new HelloWorldModel();
    return SUCCESS;
  }

  public HelloWorldModel getHelloWorld() {
    return helloWorld;
  }
}
```

web.xml

```
<filter>
<filter-name>struts2</filter-name>
<filter-class>
 org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter
</filter-class>
</filter>

<filter-mapping>
<filter-name>struts2</filter-name>
<url-pattern>/struts/*</url-pattern>
</filter-mapping>

<listener>
<listener-class>hellostruts.OgnlListener</listener-class>
</listener>
```

src/struts.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC
    "-//Apache Software Foundation//DTD Struts Configuration 2.0/
   /EN"
    "http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
<constant name="struts.devMode" value="true" />
<package name="basicstruts2" extends="struts-default">
   <action name="hello"
   class="hellostruts.action.HelloWorldAction" method="execute">
    <result name="success">/HelloStruts.jsp</result>
  </action>
</package>
</struts>
```

OgnlListener

```
package hellostruts;

import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;

import ognl.OgnlRuntime;

public class OgnlListener implements ServletContextListener {

    @Override
    public void contextDestroyed(ServletContextEvent arg0) {
          // TODO Auto-generated method stub

    }

    @Override
    public void contextInitialized(ServletContextEvent arg0) {
          OgnlRuntime.setSecurityManager(null);
    }

}
```

Hello Struts

- Run http://localhost:8888/hellostruts

Model – Product.java

```
public class Product {
    private String name;
    private double price;

    public Product() {}
    public Product(Entity e) {
      name = (String) e.getProperty("name");
      price = (Double) e.getProperty("price");
  }
}
```

View – product.jsp

```
<%@ page language="java" contentType="text/html;"%>
<%@ taglib prefix="s" uri="/struts-tags" %>
<html>
<body>
    <s:iterator value="productList">
    <s:property value="name"/> - <s:property value="price"/><br>
</s:iterator>
</body>
</html>
```

Controller - ProductAction.java

```
public class ProductAction extends ActionSupport {
  private List<Product> productList;
  public String execute() {
    DatastoreService datastore =
    DatastoreServiceFactory.getDatastoreService();
     Query query = new Query("Product");
    List<Entity> entityList =
    datastore.prepare(query).asList(FetchOptions.Builder.withDefaults());
     productList = new ArrayList<Product>();
     for (Entity entity : entityList) {
       productList.add(new Product(entity));
     }
     return SUCCESS;
}

  public List<Product> getProductList() {
    return productList;
  }
}
```

struts.xml

```
<action name="sProductList" class="hellostruts.action.ProductAction"
   method="execute">
  <result name="success">/hellostruts/product.jsp</result>
</action>
```

SHOP AROUND

Template

- http://chocotemplates.com/ecommerce/shop-around/
- Rename index.html to index.jsp and put in war/hellostruts/
- Add JSP and Struts tags
- Copy css and js to war/struts/

MEMCACHE API

Memcache

- In-memory data cache to speed up common datastore queries

ProductAction.java

```
MemcacheService cache = MemcacheServiceFactory.getMemcacheService();
List<Entity> entityList = (List<Entity>) cache.get("product"); // read from
    cache
if (entityList == null) {
    // get value from other source
    DatastoreService datastore =
    DatastoreServiceFactory.getDatastoreService();
  Query query = new Query("Product");
    entityList =
    datastore.prepare(query).asList(FetchOptions.Builder.withLimit(3));
  cache.put("product", entityList); // populate cache
}
productList = new ArrayList<Product>();
for (Entity entity : entityList) {
  productList.add(new Product(entity));
}
return SUCCESS;
```

Cached Data Expires

- By default, values stored in memcache are retained as long as possible
- The app can provide an expiration time when a value is stored, as either a number of seconds relative to when the value is added, or as an absolute Unix epoch time

Limit

Limit	Amount
maximum size of a cached value	1 megabyte
•A key can be any size. If the key is larger than 250 bytes, it is hashed to a 250-byte value before storing or retrieving. •The "multi" batch operations can have any number of elements. The total size of the call and the total size of the data fetched must not exceed 32 megabytes.	

Google App Engine Service

MAIL API

Mail API

- App Engine applications can send email messages on behalf of the app's administrators, and on behalf of users with Google Accounts
- Apps can receive incoming email at *string@appid*.appspotmail.com addresses

Sending Mail

- The Mail service Java API supports the JavaMail (javax.mail) interface for sending email messages
- An app cannot use the JavaMail interface to connect to other mail services for sending or receiving email messages
- Calls to the Mail service are asynchronous, and return immediately
- SMTP configuration added to the Transport or Session is ignored

Sender

- The sender address must be one of the following types:
  - The address of a registered administrator for the application
  - The address of the user for the current request signed in with a Google Account. The user's account must be a Gmail account, or be on a domain managed by Google Apps.
  - Any valid email receiving address for the app (such as xxx@APP-ID.appspotmail.com).

HelloMailServlet

```
package mail;

import javax.servlet.http.*;

import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.util.Properties;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.AddressException;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
```

HelloMailServlet

```
public class HelloMailServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws IOException {
        Properties props = new Properties();
        Session session = Session.getDefaultInstance(props, null);

        String msgBody = "Hello Mail sample works!!";

        try {
            Message msg = new MimeMessage(session);
            msg.setFrom(new InternetAddress("admin@[appid].appspotmail.com", "Admin"));
            msg.addRecipient(Message.RecipientType.TO,
                             new InternetAddress("[YOUR EAMIL]", "[YOUR NAME]"));
            msg.setSubject("Hello from GAE workshop");
            msg.setText(msgBody);
            Transport.send(msg);
            res.getWriter().println("Message Sent!");
        } catch (AddressException e) {
            e.printStackTrace(res.getWriter());
        } catch (MessagingException e) {
            e.printStackTrace(res.getWriter());
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace(res.getWriter());
        }
    }
}
```

Receiving Email

- Apps can receive incoming email at *string@appid*.appspotmail.com addresses
- Email messages sent to your app are implemented as HTTP POST requests to /_ah/mail/<address>
- Associate email addresses with servlets in app configuration
- Incoming email is disabled by default. Enable in appengine-web.xml

appengine-web.xml

```
<inbound-services>
   <service>mail</service>
</inbound-services>
```

web.xml

```
<servlet>
  <servlet-name>mailhandler</servlet-name>
  <servlet-class>mail.MailHandlerServlet</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>mailhandler</servlet-name>
  <url-pattern>/_ah/mail/*</url-pattern>
</servlet-mapping>
<security-constraint>
  <web-resource-collection>
    <url-pattern>/_ah/mail/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>admin</role-name>
  </auth-constraint>
</security-constraint>
```

Handlers for different email addresses

```
<servlet>
  <servlet-name>handlesupport</servlet-name>
  <servlet-class>mail.SupportMailHandlerServlet
  </servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>handlesupport</servlet-name>
  <url-pattern>/_ah/mail/support*</url-pattern>
</servlet-mapping>
```

MailHandlerServlet

```
package mail;

import javax.servlet.http.*;
import java.io.IOException;
import java.util.Properties;

import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;


public class MailHandlerServlet extends HttpServlet {
  public void doPost(HttpServletRequest req, HttpServletResponse res)
   throws IOException {
  }
}
```

MailHandlerServlet - doPost

```
Properties props = new Properties();
Session session = Session.getDefaultInstance(props, null);
try {
  MimeMessage message = new MimeMessage(session, req.getInputStream());

  String msgBody = "Thank you for your message - " + message.getSubject();

  Message msg = new MimeMessage(session);
   msg.setFrom(new
   InternetAddress("admin@nitikorn.appspotmail.com", "Admin"));
  msg.addRecipient(Message.RecipientType.TO, message.getFrom()[0]);
  msg.setSubject("Thank you for your message");
  msg.setText(msgBody);
  Transport.send(msg);
} catch (MessagingException e) {
// TODO Auto-generated catch block
  e.printStackTrace();
}
```

Receiving Email

- https://appengine.google.com
- Administration > Application Settings > Configured Services - checking the Application Settings section to verify that incoming email is enabled
- Try sending email to **admin@appid.appspotmail.com**
- Main > Logs – check log when incoming email is processed

Quota

Resource	Free Default Limit		Billing Enabled Default Limit	
	Daily Limit	Maximum Rate	Daily Limit	Maximum Rate
Mail API Calls	100 calls	32 calls/minute	1,700,000 calls	4,900 calls/minute
Recipients Emailed	100 recipients	8 recipients/minute	100 recipients until first charge cleared; 100 recipients free and no maximum thereafter	5,100 recipients/ minute
Admins Emailed	5,000 mails	24 mails/minute	3,000,000 mails	9,700 mails/minute
Message Body Data Sent	60 MB	340 KB/minute	29 GB	84 MB/minute
Attachments Sent	2,000 attachments	8 attachments/minute	2,900,000 attachments	8,100 attachments/ minute
Attachment Data Sent	100 MB	10 MB/minute	100 GB	300 MB/minute

Limit

Limit	Amount
maximum size of outgoing mail messages, including attachments	10 megabytes
maximum size of incoming mail messages, including attachments	10 megabytes
maximum size of message when an administrator is a recipient	16 kilobytes

USERS SERVICE

Users API

- Integrate with Google user accounts
- Authenticate users using any one of 3 options:
  - A Google Account
  - An account on your Google Apps domain
  - An OpenID identifier

HelloUserServlet.java

```
public class HelloUserServlet extends HttpServlet {
  public void doGet(HttpServletRequest req, HttpServletResponse resp)
   throws IOException {
    UserService userService = UserServiceFactory.getUserService();
    User user = userService.getCurrentUser();

    if (user != null) {
      resp.setContentType("text/plain");
      resp.getWriter().println("Hello, " + user.getNickname());
    } else {
      resp.sendRedirect(
         userService.createLoginURL(req.getRequestURI()));
    }
  }
}
```

Required Login or Administrator Status

- A security constraint includes an authorization constraint that specifies which Google Accounts users can access the path

web.xml

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

URL FETCH

URL Fetch Service

- App Engine applications can communicate with other applications or access other resources on the web by fetching URLs
- Use the URL Fetch service to issue HTTP and HTTPS requests and receive responses
- The URL Fetch service uses Google's network infrastructure for efficiency and scaling purposes

UrlFetcherServlet.java

```
public void doGet(HttpServletRequest req, HttpServletResponse resp) {
  try {
    URL url = new URL("http://rss.cnn.com/rss/edition.rss");
   BufferedReader reader = new BufferedReader(new
     InputStreamReader(url.openStream()));
    String line;

    while ((line = reader.readLine()) != null) {
      resp.getWriter().println(line);
    }
    reader.close();

    } catch (MalformedURLException e) {
            // ...
    } catch (IOException e) {
            // ...
    }
}
```

Quota

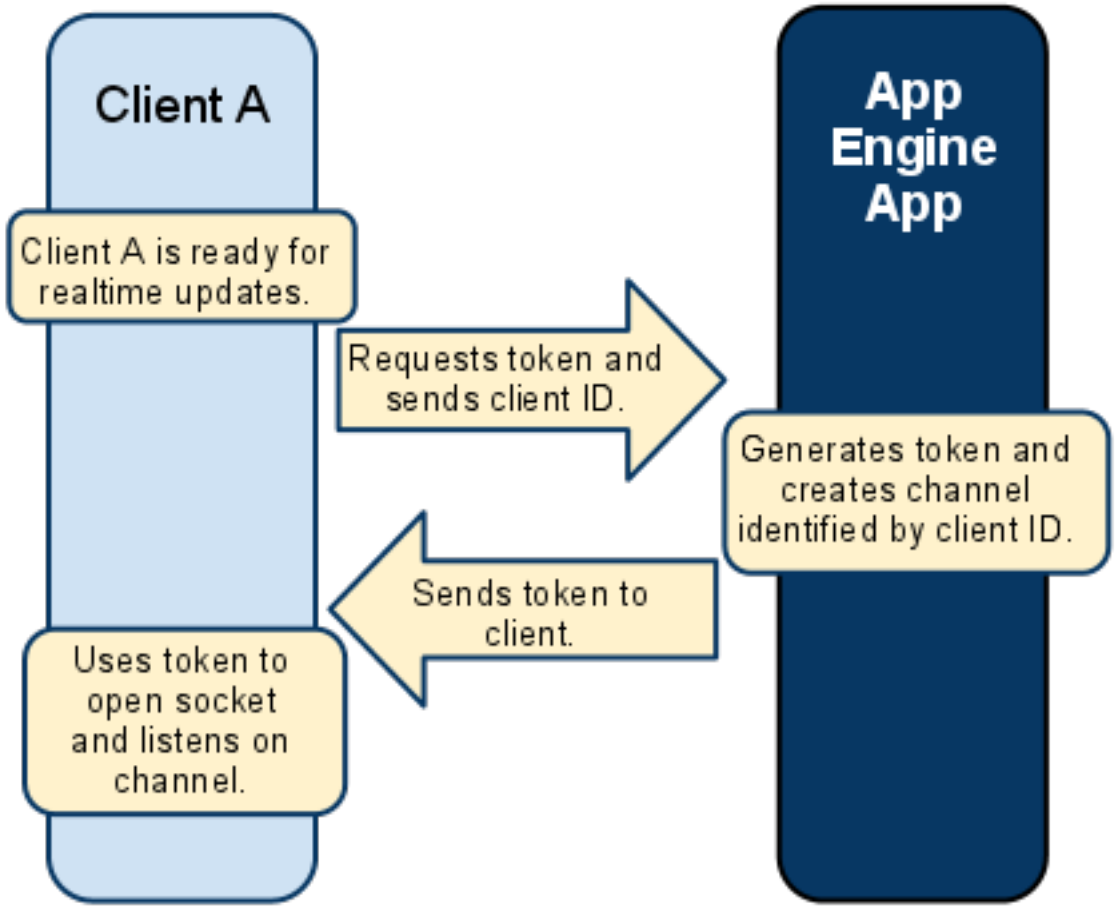
Resource	Free Default Limit		Billing Enabled Default Limit	
	Daily Limit	Maximum Rate	Daily Limit	Maximum Rate
UrlFetch API Calls	657,000 calls	3,000 calls/minute	46,000,000 calls	32,000 calls/minute
UrlFetch Data Sent	up to the Outgoing Bandwidth quota	22 MB/minute	up to the Outgoing Bandwidth quota	740 MB/minute
UrlFetch Data Received	up to the Incoming Bandwidth quota	22 MB/minute	up to the Incoming Bandwidth quota	740 MB/minute

CHANNEL API

Channel API

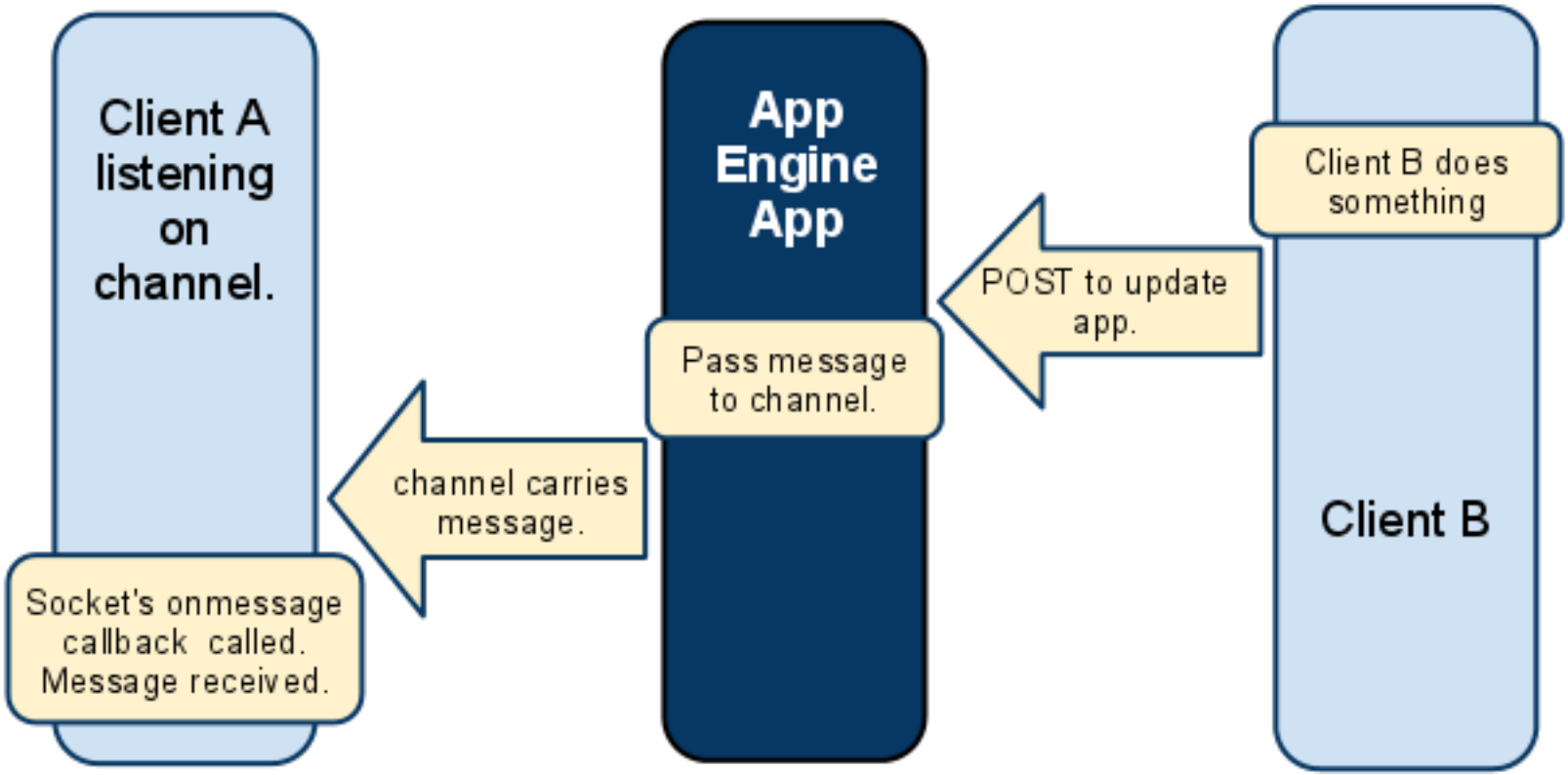
- Creates a persistent connection between your application and Google servers allowing your application to send messages to JavaScript clients in real time

Channel API



https://developers.google.com/appengine/docs/java/channel/overview

Channel API



https://developers.google.com/appengine/docs/java/channel/overview

ChatServlet.java

```
public class ChatServlet extends HttpServlet {
public void doGet(HttpServletRequest req, HttpServletResponse resp) throws
    IOException {
    ChannelService channelService =
    ChannelServiceFactory.getChannelService();
     String token = channelService.createChannel("chat");
     resp.setContentType("text/plain");
     resp.getWriter().write(token);
}

public void doPost(HttpServletRequest req, HttpServletResponse resp) throws
    IOException {
ChannelService channelService = ChannelServiceFactory.getChannelService();
channelService.sendMessage(new ChannelMessage("chat",
    req.getParameter("message")));
}
}
```

chat.html

```
<html>
<head>
...
</head>
<body>
<div id="messageList"></div>
<input type="text" id="message"/><button id="sendBtn">Send</button>
</body>
</html>
```

chat.html - JavaScript

```
<script type="text/javascript" src="/_ah/channel/jsapi"></script>
<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/
    jquery.min.js"></script>
<script type="text/javascript">
var channel = null;
$(function() {
$.get('/chatServer', function(token) {
  channel = new goog.appengine.Channel(token);
  var socket = channel.open();
  socket.onopen = function() {};
  socket.onmessage = function(message) {
    $('#messageList').append($('<div>').html(message.data));
  };
  socket.onerror = function() {};
  socket.onclose = function() {};
});

$('#sendBtn').click(function() {
  $.post("/chatServer", {message: $('#message').val()}, function() {
    $('#message').val('');
  });
});
});
</script>
```

Quota

Resource	Free Default Limit		Billing Enabled Default Limit	
	Daily Limit	Maximum Rate	Daily Limit	Maximum Rate
Channel API Calls	657,000 calls	3,000 calls/ minute	91,995,495 calls	32,000 calls/ minute
Channels Created	100 channels	6 creations/ minute	Based on your budget	60 creations/ minute
Channels Hours Requested	200 hours	12 hours requested/ minute	Based on your budget	120 hours requested/ minute
Channel Data Sent	Up to the Outgoing Bandwidth quota	22 MB/minute	2 GB	740 MB/minute

TASK QUEUE

Task Queue API

- Perform background work outside of a user request
- Simply configure a queue and add tasks to it. App Engine handles the rest
- Manage task queues for an application using the Task Queue tab of the Administration Console

TaskServlet.java – add task

```
public class TaskServlet extends HttpServlet {
  public void doGet(HttpServletRequest req, HttpServletResponse res) {
    Queue queue = QueueFactory.getDefaultQueue();
     queue.add(
       TaskOptions.Builder
        .withUrl("/task")
        .method(TaskOptions.Method.POST));
  }
}
```

TaskServlet.java - task

```
public void doPost(HttpServletRequest req, HttpServletResponse res) {
  DatastoreService datastore =
  DatastoreServiceFactory.getDatastoreService();
   Query query = new Query("Product");
  List<Entity>entityList =
  datastore.prepare(query).asList(FetchOptions.Builder.withDefaults());
   for (Entity entity : entityList) {
     entity.setProperty("price", 200);
     entity.setProperty("updated", new Date());
     datastore.put(entity);
   }
 }
```

Quota

Resource	Free Default Limit		Billing Enabled Default Limit	
	Daily Limit	Maximum Rate	Daily Limit	Maximum Rate
Task Queue API Calls	100,000	*n/a*	1,000,000,000	*n/a*

Apps Market

- Google Apps Marketplace
  https://www.google.com/enterprise/marketplace/
- Chrome web store
  https://chrome.google.com/webstore
- Apple Web Apps
  http://www.apple.com/webapps/
- Facebook Apps
  https://developers.facebook.com/apps