morph AppSpace™

Complete Hosting and Fully Managed Service for Web Applications

Java Developers' Cookbook

December 2008

About this Cookbook

This cookbook is intended for Java developers and provides technical information on how to deploy a Java application to Morph AppSpace. For more information about Morph Labs and its services, visit www.mor.ph.

Table of Contents

Deploying a Java Application on the Morph AppSpace

Deploying any Java application on the Morph AppSpace is easy and fast. Follow these simple steps:

Preparing a Java Application for Deployment

> You can deploy applications that run on Jetty with either a PostgreSQL or MySQL database. This would be a Web application written against the standard 2.5 Servlet specification, with the addition of PostgreSQL or MySQL database wrapped as a standard XA data source available via JNDI.

To prepare your Java application and reduce the likelihood of running into any hitches during deployment, read the following items:

- JDK 1.5 or lower must be installed on your system.
- If your application needs a database, make sure that you have a way to populate this. You can perform any of the following ways:
  - use an Object-Relational Mapping technique such as Hibernate.
  - write a filter or servlet that creates tables and inserts data.
- Your application must be compatible with the PostgreSQL or MySQL database servers.
- You can use `jndiNames` to retrieve a DataSource for the database connection. To do this, specify a name on the application's `web.xml` file. The jetty-runner detects this action and creates the resource for it. The following is a sample code that you can add to your `web.xml` file:

```
<resource-ref>
    <description>Morphlabs Datasource</description>
   <!-- any name will do for the res-ref-name -->
    <res-ref-name>jdbc/morph-ds</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
  </resource-ref>
```

> **Note:** You can get the database information from the Morph AppSpace you have created. See Deploying your Java application on how to create a Morph AppSpace.

- Compile your application's source code into a war file. Make sure that all needed libraries are included in the war file.
- If your application needs to send emails, you can use the mail session. You can get this using the `jndiname "mail/session"`. The following displays the code on how to do this:

```
InitialContext initialContext = new InitialContext();
Context context = (Context)initialContext.lookup("java:/comp/env");
Session session = (Session)context.lookup("mail/Session");
```

> **Warning:** Do not add the `mail.jar` and `activation.jar` in the war file.

- Your application only has write permission on its current working directory. This is the "./" path. You can also retrieve the path using the command `System.getProperty("user.dir")`.

- To connect to a database using JNDI Datasource, you need these classes:

```
 javax.naming.Context
javax.naming.InitialContext
javax.sql.DataSource
```

The following displays the code:

```
InitialContext initialContext = new InitialContext();
Context context = (Context)initialContext.lookup("java:/comp/
env");
datasource = (DataSource)context.lookup("jdbc/morph-ds");
Connection dbConnection = datasource.getConnection(); //get a
database connection instance
```

Note: Adding the `postgresql` and `mysql connector jar` in the war file is not required.

Preparing a Grails Application for Deployment

Read the following items to prepare your Grails application and reduce the likelihood of running into any hitches during deployment.

- JDK 1.5 or lower must be installed on your system.
- Grails and Apache Ant must be installed on your system.
- Your application must be compatible with the PostgreSQL or MySQL database servers.
- If your Grails application does not support PostgreSQL, perform these steps to configure `DataSource` in your application:

  1. Open `DataSource.groovy` in the `{app.dir}/grails-app/conf` folder.
  2. Set `driverClassName` to `org.postgresql.Driver`, the dialect to `org.hibernate.dialect.PostgreSQLDialect` and `pooled` to `true`. Setting pooled to true will improve the speed of you application since database connections are being pooled. The following displays a sample code :

     ```
     pooled = true
     driverClassName = "org.postgresql.Driver"
     dialect = "org.hibernate.dialect.PostgreSQLDialect"
     ```

  3. Set the url to `jdbc:postgresql://<database_host>/<database_name>`. You should also set the username and password of the database.

Note: You can get the database information from the Morph AppSpace you have created. See ***Deploying your Java or Grails application*** to learn how to create your Morph AppSpace.

Here is a complete sample of the `DataSource.groovy` :

```
dataSource {
      pooled = true
}
hibernate {
    cache.use_second_level_cache=true
    cache.use_query_cache=true
    cache.provider_class='org.hibernate.cache.EhCacheProvider'
}
// environment specific settings
environments {
      development {
            dataSource {
                  dbCreate = "create-drop"
                  url = "jdbc:postgresql://<host>/<database name>"
                  driverClassName = "org.postgresql.Driver"
                  dialect =
"org.hibernate.dialect.PostgreSQLDialect"
                  username = "postgres"
                  password = "postgres"
            }
```

```
        }
        test {
            dataSource {
                dbCreate = "update"
                url = "jdbc:postgresql://<host>/<database name>"
                driverClassName = "org.postgresql.Driver"
                dialect =
"org.hibernate.dialect.PostgreSQLDialect"
                username = "postgres"
                password = "postgres"
            }
        }

        production {
            dataSource {
                dbCreate = "update"
```

Note: Set `dbCreate` property to update. This is important for your production environment. Otherwise, data would be deleted from your database each time you would restart your application. You could also use a JNDI name to connect to a database. See below.

```
                url = "jdbc:postgresql://<host>/<database name>"
                driverClassName = "org.postgresql.Driver"
                dialect =
"org.hibernate.dialect.PostgreSQLDialect"
                username = "user_425"
                password = "a6pzvu6x"
            }
        }
    }
```

Note: If you wish to use MySQL, replace the URL with `"jdbc:mysql://<host>/<database name>"`.

- To connect to MySQL using Grails, edit `DataSource.groovy`.

  `set DriverClassName to com.mysql.jdbc.Driver`
  `set dialect to org.hibernate.dialect.MySQLDialect` or
  `org.hibernate.dialect.MySQL5Dialect`

Note: Adding the `postgresql` and `mysql connector jar` in the war file is not required.

- To connect to the database using a JNDI name, perform the following steps:

  1. Set `jndiName` to "`java:comp/env/`" and the JNDI name. If your JNDI name is "`jdbc/morph-ds`", then your `dataSource` would look like this:

     ```
     dataSource {
         dbCreate = "update"
         jndiName = "java:comp/env/jdbc/morph-ds"
     }
     ```

2. Specify the JNDI datasource resource on your application's web.xml file.
   - Run the command grails install-templates. This will create the web.xml file.
   - Modify the web.xml template, which is located in src/templates/war/web.xml, adding the resource-ref element that references the database at the end of the template web.xml:

```
<resource-ref>
   <res-ref-name>jdbc/morph-ds</res-ref-name>
   <res-type>javax.sql.DataSource</res-type>
   <res-auth>Container</res-auth>
</resource-ref>
```

Note: The res-ref-name morph-ds must be the same of what is configured in the Grails DataSource file (see above).

- Compile your application's source code into a war file. In Grails, you can type this command grails <env> war; where env is for test environment, dev is for development environment, and prod is for production environment.

  By default, Grails adds all dependencies in the WEB-INF/lib directory of the war. These dependencies include jdbc2_0-stdext.jar. This jar contains the JDBC 2.0 extensions, but these classes have been included in JDK 1.4 long time ago, and are only needed when running with JDK 1.3. By default, this obsolete jar is still included. This jdbc2_0-stdext.jar must be removed from the WEB-INF/lib directory of the war. Otherwise, this error occurs:

```
org.springframework.jndi.TypeMismatchNamingException: Object of
type [class com.atomikos.jdbc.nonxa.NonXADataSourceBean] available
at JNDI location [java:comp/env/jdbc/glogds] is not assignable to
[javax.sql.DataSource]
```

  The following lists two ways to remove the jdbc2_0-stdext.jar:

  - Modify the Grails installation, by removing the jdbc2_0-stdext.jar from the DEFAULT_DEPS variable defined in $GRAILS_HOME/scripts/War.groovy. This will modify permanently the Grails installation.
  - Specify the application dependencies as described in the Grails User Manual here: http://grails.org/doc/1.0.x/guide/single.html#17.%20Deployment

  The war created would be the one to be deployed to the Morph AppSpace.

- Your application only has write permission on its current working directory. This is the "./" path. You can also retrieve the path using the command System.getProperty("user.dir").
- To use JNDI for sending mails, perform the following steps:
  1. Run command grails install-templates to create the web.xml on src/templates/war (if you have not yet run this command).

2. Append the following lines to the `web.xml` file:

```
<resource-ref>
    <description>Morphlabs Mail Session</description>
    <res-ref-name>mail/Session</res-ref-name>
    <res-type>javax.mail.Session</res-type>
    <res-auth>Container</res-auth>
  </resource-ref>
```

3. Edit the `resources.groovy` file found at grails-app/conf/spring/. Add the following beans:

```
 mailSession(org.springframework.jndi.JndiObjectFactoryBean)
        {
          jndiName = 'java:/comp/env/mail/Session'
        }
mailSender(org.springframework.mail.javamail.JavaMailSenderImpl)
{
          session = mailSession
        }
```

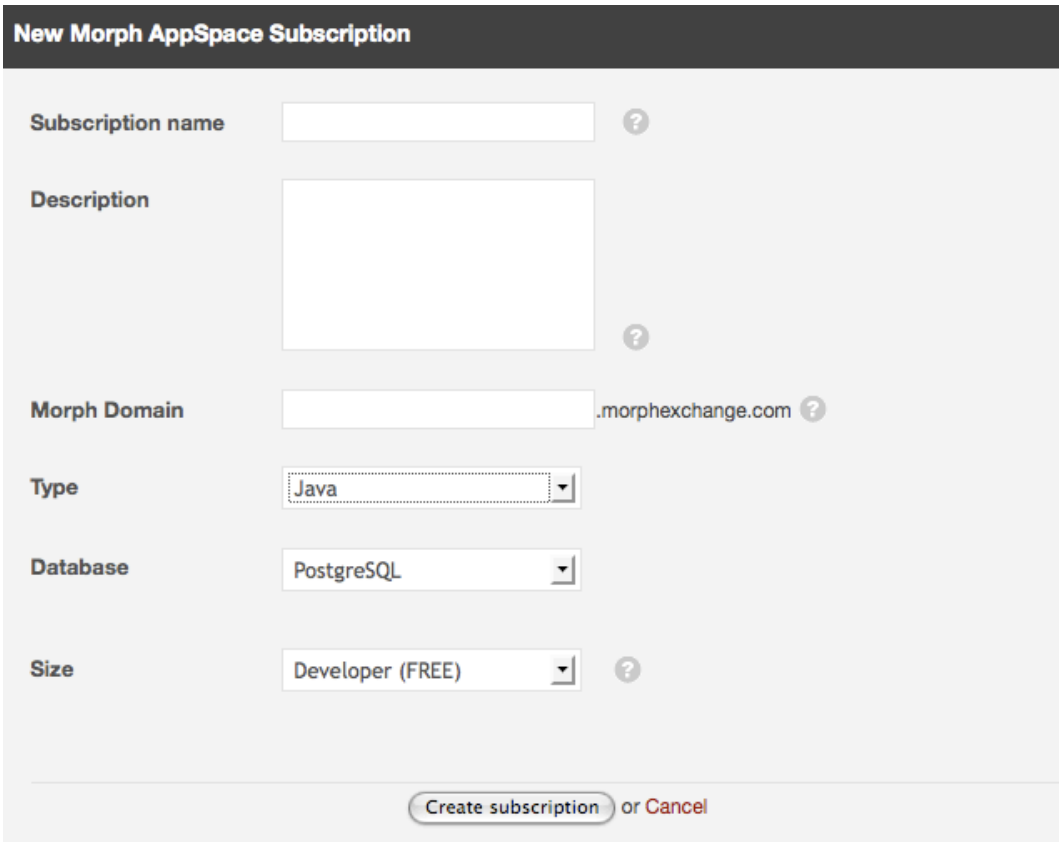
This would configure the `mailSender` to use the provided `JNDI mail session`.

Note: Do not add the `mail.jar` and `activation.jar` to your war file.

Deploying your Java or Grails application

Follow these easy steps to deploy and run your Java or Grails application on the Morph AppSpace:

1. Log in to the Morph Control Panel.
2. From the Morph Control Panel **Subscriptions** *page*, create a new subscription by clicking **New Subscription.** This displays:



3. Fill in the required fields. If you have created your Morph AppCloud, you may choose to deploy your application on it.
4. Click **Create subscription** to create your new Morph AppSpace.
5. Select your newly added Morph AppSpace widget. If your application needs a database, click **Create Database**.

6. Download the deployment properties file from the Morph AppSpace widget. Save this file to your application's root directory.
7. You can deploy your Java application in two different methods. Perform any of the following:

Method	Steps
Use the `morph_deployer.jar`.	1. Download the deployment jar from the widget. Save this to the location of your war file. 2. Place the deployment properties file to the location of the war file. 3. Upload your application on the Morph AppSpace. In the command console, type `java -jar morph-deploy.jar --config morph_deploy.properties your_war_file.war`. **Note:** You can also add your morph username and password. Type this command: `java -jar morph_deploy.jar --user your_username --password your_password --config morph_deploy.properties your_warfile.war`. You can also add a `--quiet` option to disable log messages.

Use the `morph-deployer.jar` through a proxy.	Add the following parameters:
	`-DproxyHost=<proxy host>`
	`-DproxyPort=<proxy port (default is 80 if not specified)>`
	`-DproxySet=true`
	Example:
	`java -DproxySet=true -DproxyHost=127.0.0.1 -DproxyPort=8080 -jar morph-deploy.jar testApp-1.0-SNAPSHOT.war`

Method	Steps

Use the `ant-task` plugin.	1. Download the `ant-task-plugin.jar` from the widget. Save this to the location of your war file.
	• If you are using Apache Ant, you can save it at the `lib/ folder.`
	• If you are using Eclipse, you can add the ant task by opening the **Preferences** dialog. Go to **Window** > **Preferences**. Select **Ant** > **Runtime** node in the tree. Select the **Classpath** tab and the **Global Entries** node. Click the **Add External JARS** button and add `morph-ant-task.jar`. Select the **Task** tab and click the **Add Task** button. Select the `morph-ant-task.jar` from the **Location** combo box. Select the `class com.morph.ant.task.MorphDeployer`. Name the task as `morph-deploy`.
	2. Append the following to your `build.xml` file:
	• If you are using Apache Ant, you need to define the task by adding the following line:
	`<taskdef name="morph-deploy" classname="com.morphexchange.ant.task.MorphDeployer"/>`
	Note : If you did not add the `morph-ant-task.jar` at your lib folder, you must define its location through the `classpath` parameter.
	• Append the following line to deploy your war file:
	`<morph-deploy configFile="morph_deploy.properties" warFile="your_war_file.war"/>`
	Note : The ant task takes the following parameters:
	• `username` - the morph username.
	• `password` - the morph password.
	• `configFile` - the deployment properties file.
	• `warFile` - the war file to be deployed.
	• `quiet` - set to true to disable log messages; false by default.

Method	Steps

Use the `ant-task` plugin through a proxy.	Add the following parameters:
	`proxyHost - the proxy host`
	`proxyPort - the proxy port, default is 80`
	Example:
	`<project name="testApp" default="deploy">`
	`<taskdef name="morph-deploy" classname="com.morphexchange.ant.task.MorphDeployer"/>`
	`<target name="deploy">`
	`<morph-deploy configFile="morph_deploy.properties" warFile="testApp-1.0-SNAPSHOT.war" proxyHost="127.0.0.1" proxyPort="8080"/>`
	`</target>`
	`</project>`
	Note: Leave the username and password blank If the proxy does not require any authentication.

Note: To learn more about using the deployer jar, type `java -jar morph-deploy.jar` without any parameters to display the usage. Your username and password are not required. If you have not specified your username and password, it will prompt you to authenticate.

8. Once the command has finished, wait for a few minutes while your application is being initialized and then in a browser go to `<your_app>.morphexchange.com.`

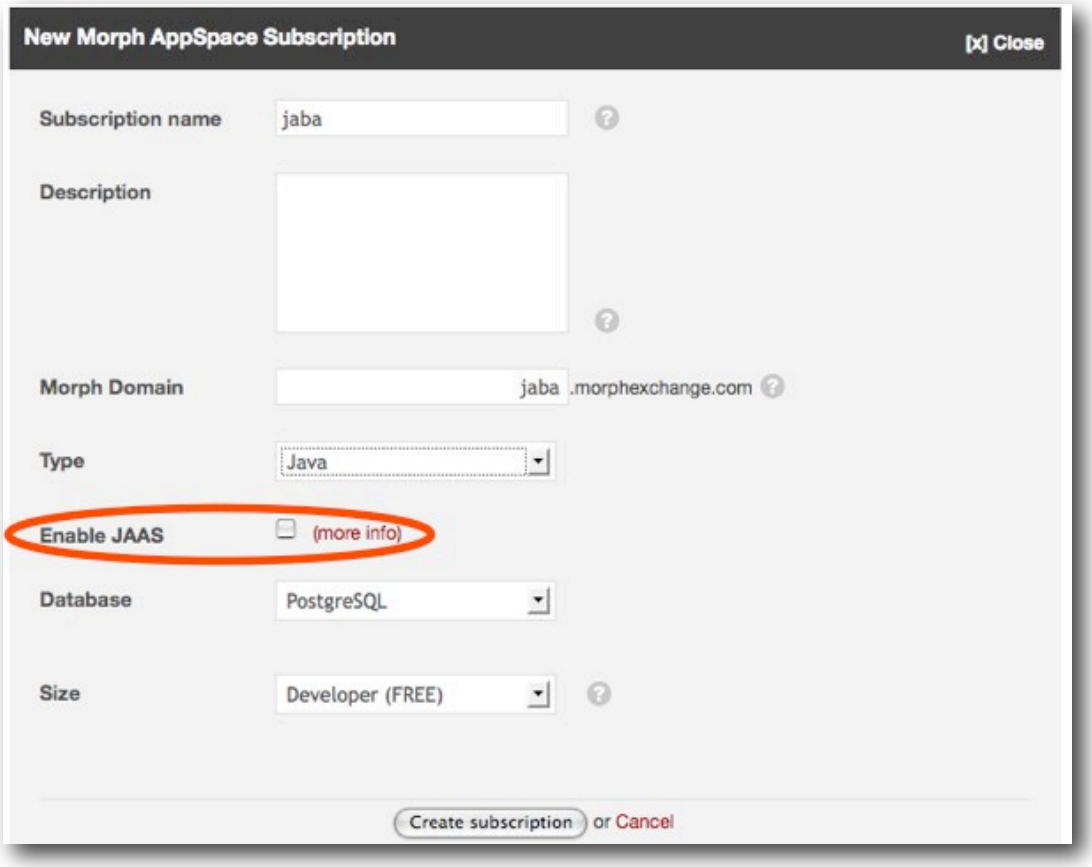
Enabling JAAS on your Morph AppSpace for Java

[1]Java™ Authentication and Authorization Service (JAAS) is most commonly used for the following purposes:

- authentication of users; JAAS provides a reliable and secure means to determine who is currently running or executing Java code.
- authorization of users; Using JAAS ensures users that they have the access control right or permissions required to do the actions performed.

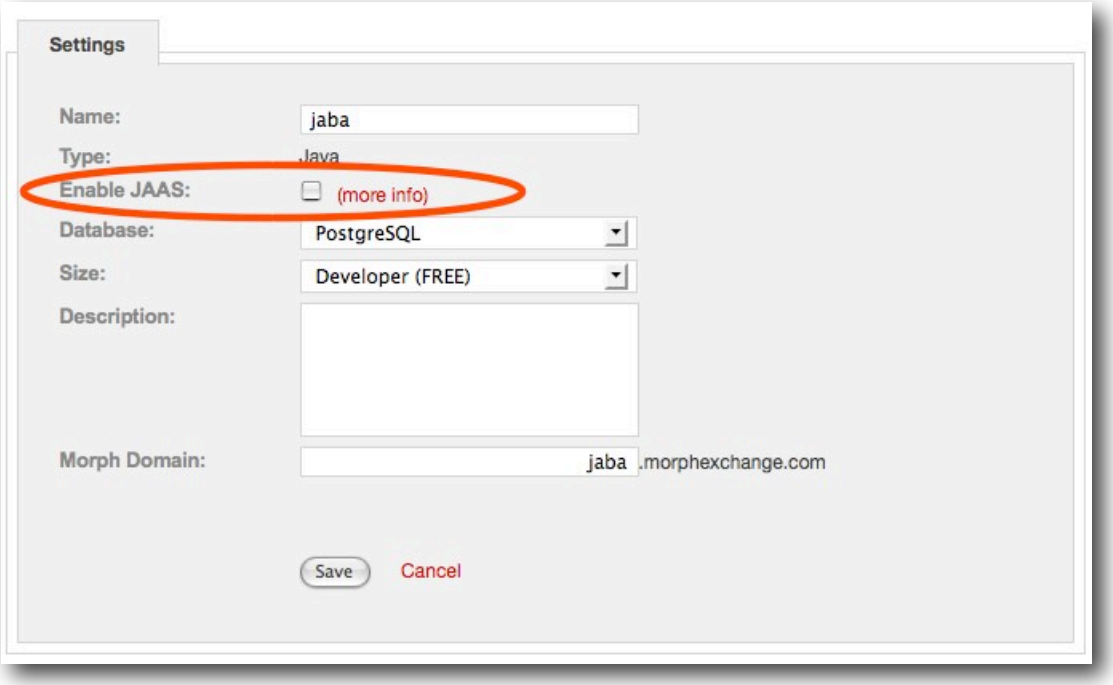
Follow these steps to enable JAAS on your Morph AppSpace:

1. Log in to the Morph Control Panel.
2. Create a new Morph AppSpace for Java or edit an existing one.
3. If you are creating a new Morph AppSpace for Java, simply click the **Enable JAAS** checkbox. Then, click **Create Subscription**.



[1]Source: Java™ Authentication and Authorization Service Reference Guide, http://java.sun.com/j2se/1.4.2/docs/guide/security/jaas/JAASRefGuide.html.

4. If you are editing an existing Morph AppSpace for Java, access **Manage** > **Dashboard** > **Settings** > **Edit**. Then, click the **Enable JAAS** checkbox. Click **Save**.



Note: If you have edited an existing Morph AppSpace for Java, you need to restart your application for changes to be reflected.

Configuring the Web application to use the User Realm

Perform the following steps:

1. If you have enabled the JAAS feature on your Morph AppSpace, you may now use the Morph User Realm.

> **Note:** The users and roles are stored in the database that you have created for your Morph AppSpace. Log in to the **DB Admin** using your login credentials from **Dashboard** > **Database** > **Show Details**.

2. Edit the web.xml file found on the WEB-INF folder in your application.
3. Specify the URLs that have some security constraints:

```
<web-app>
   . . .
   <security-constraint>
     <web-resource-collection>
       <web-resource-name>A Protected Page</web-resource-name>
       <url-pattern>/*</url-pattern>
     </web-resource-collection>
     <auth-constraint>
       <role-name>admin</role-name>
       <role-name>user</role-name>
       <role-name>moderator</role-name>
     </auth-constraint>
   </security-constraint>
   . . .
 </web-app>
```

4. You may use any of the three types of authentication: Basic, Digest, and Form. The following displays sample configuration for each type:

   - Basic Authentication

```
<web-app>
   . . .
   <login-config>
      <auth-method>BASIC</auth-method>
      <realm-name>Morph User Realm</realm-name>
   </login-config>
   . . .
</web-app>
```

   - Digest Authentication

```
<web-app>
    . . .
    <login-config>
       <auth-method>DIGEST</auth-method>
       <realm-name>Morph User Realm</realm-name>
    </login-config>
    . . .
</web-app>
```

- Form Authentication

```
<web-app>
      . . .
      <login-config>
              <auth-method>FORM</auth-method>
              <realm-name>Morph User Realm</realm-name>
              <form-login-config>
                <form-login-page>/login/login</form-login-page>
                <form-error-page>/login/error</form-error-page>
              </form-login-config>
       </login-config>
       . . .
 </web-app>
```

Adding a User and Assigning a Role

The users and roles are stored in the database that you have created for your Morph AppSpace. The following lists the information of the tables and its fields:

Table name: __morph_users
Description: holds the username and password
Fields: id - integer primary key
username - varchar 100 unique
password - varchar 20

Table name: __morph_roles
Description: holds the defined roles
Fields: id - integer primary key
role - varchar 100

Table name: __morph_user_roles
Description: defines the roles of each user
Fields: user_id - integer
role_id - integer

1. To add a user, use the SQL command: `insert into __morph_users values (<the_user_id>, '<the_username>', '<the_password>');`
2. To add a role, use the SQL command: `insert into __morph_roles values (<the_role_id>, '<the_role>');`
3. To assign a user to a role, use the SQL command: `insert into __morph_user_roles values (<user_id>, <role_id>);`

Note: You can also use the DB Admin on the Morph Control Panel.

Contact Morph Labs Customer Support

For questions or comments, contact us at support@mor.ph. For more information, visit our online Forums at http://forums.mor.ph.

Morph Labs is the leading provider of Platform as a Service (PaaS) that virtualizes the application environment through the use of open source technologies to simplify the deployment, delivery, and management of Web based applications.

Morph Labs uses virtual infrastructures including Amazon Web Services to provide a truly elastic environment for Web applications that can be instantly provisioned and seamlessly scaled.

Morph Labs is a global company with headquarters in Cebu City, Philippines with additional in-country operations in Manila along with Los Angeles, California in the U.S.A.

www.mor.ph

World Headquarters
Unit 1A, Asiatown IT Park,
Lahug, Cebu City 6000
Philippines

Worldwide Inquiries
info@mor.ph
www.mor.ph