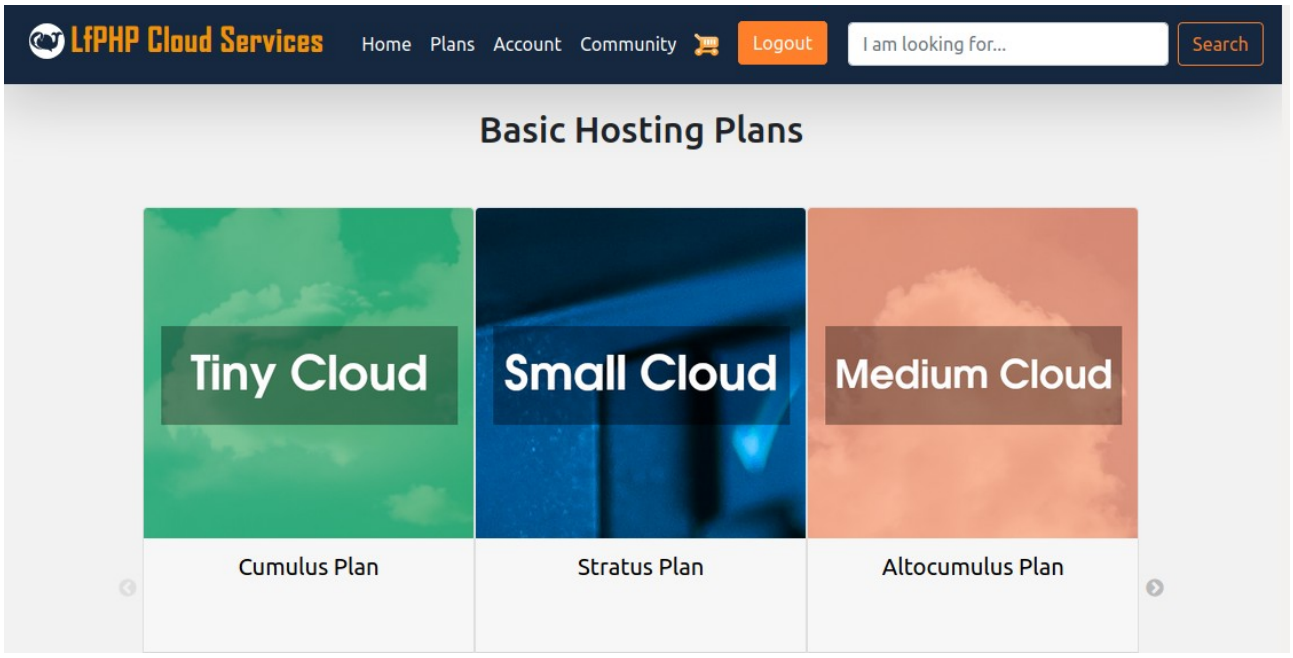
How to Create an Interactive HTML Website with LfPHP Lambda Cloud Services

Small business owners face a daunting task moving their businesses online. One of the barriers to entry into the online world is summed up in this simple question: *where to begin*? For those brave enough to take the plunge, one good place to start is to choose a *CMS* (Content Management System) such as Concrete5, WordPress, Drupal or Joomla, and start from there. However, the learning curve for many of these systems is formidable. In this article we explore how to build an HTML based website with PHP interactivity using Linux for PHP Lambda Cloud Services.

So Where Do I Start?

To start off, create an account on Linux for PHP Cloud Services. Go to this page: https://linuxforphp.com/signup, fill in the blanks, and click *Sign Up*. You will need to respond to the email in order to activate your account.

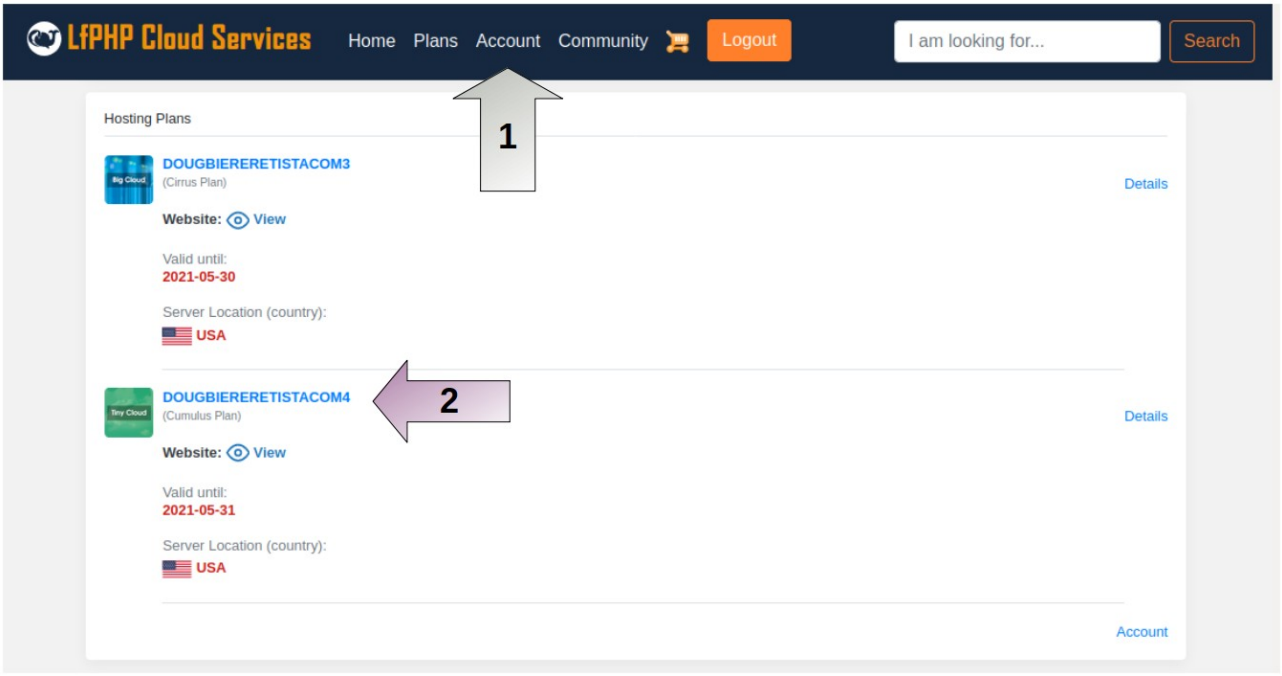
The next step is to login and choose *Plans*. If you hover your mouse over the plan a description pops up. After clicking *Buy Now*, a more detailed description appears with a server location choice. Although from a connectivity perspective it doesn't matter where the server is located, from a legal perspective, if sensitive customer data will be stored, the physical location of the server may be important.



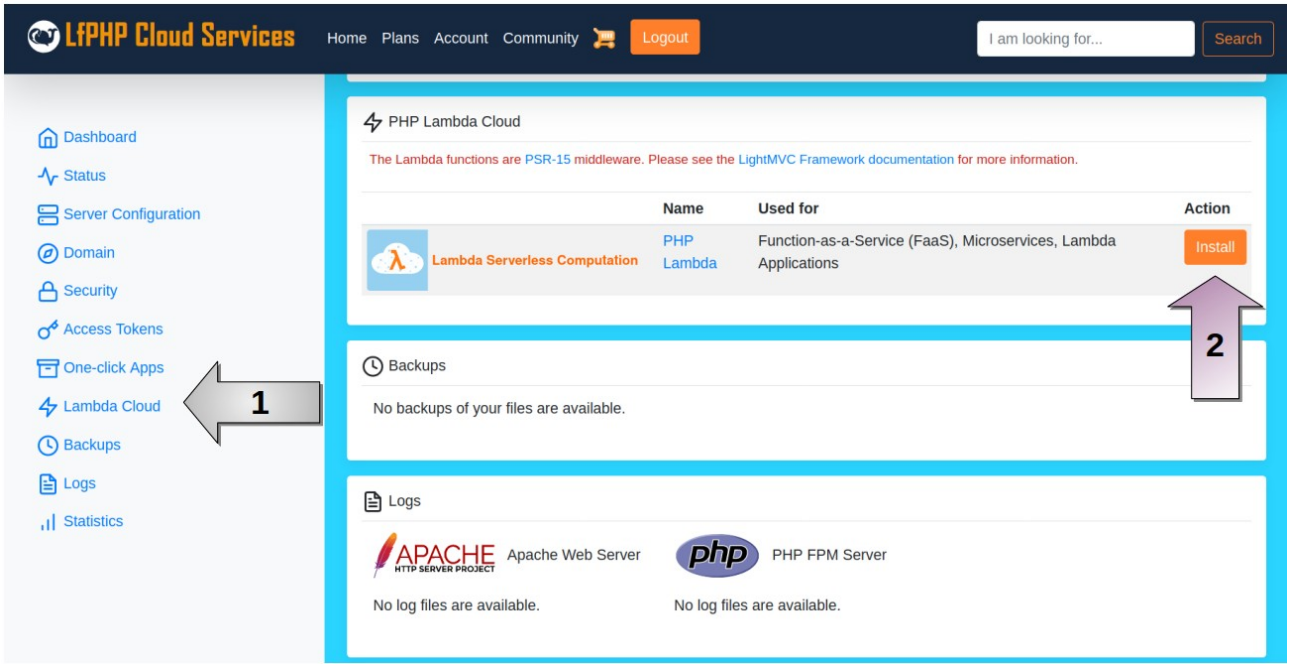
Prices are extremely reasonable and there are free trial offers available. Promotional codes can be applied when you go to checkout.

OK, Got the Plan ... Now What?

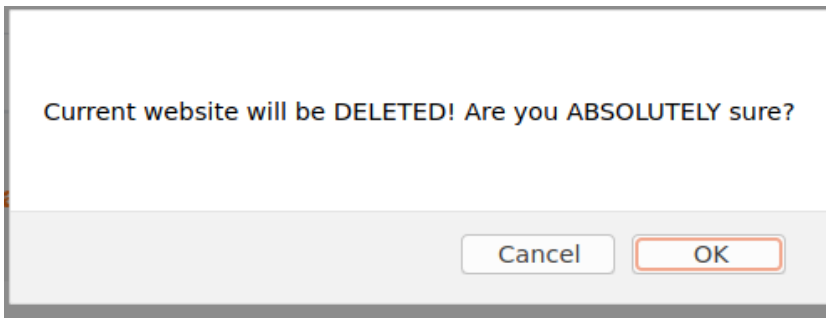
Once you have the plan, from your account page ([1] on the screenshot below), select the link to your plan ([2] on the screenshot). Note that website is not yet ready for view, so if you click on the *Website: View* link, only an invalid response code appears in your browser.



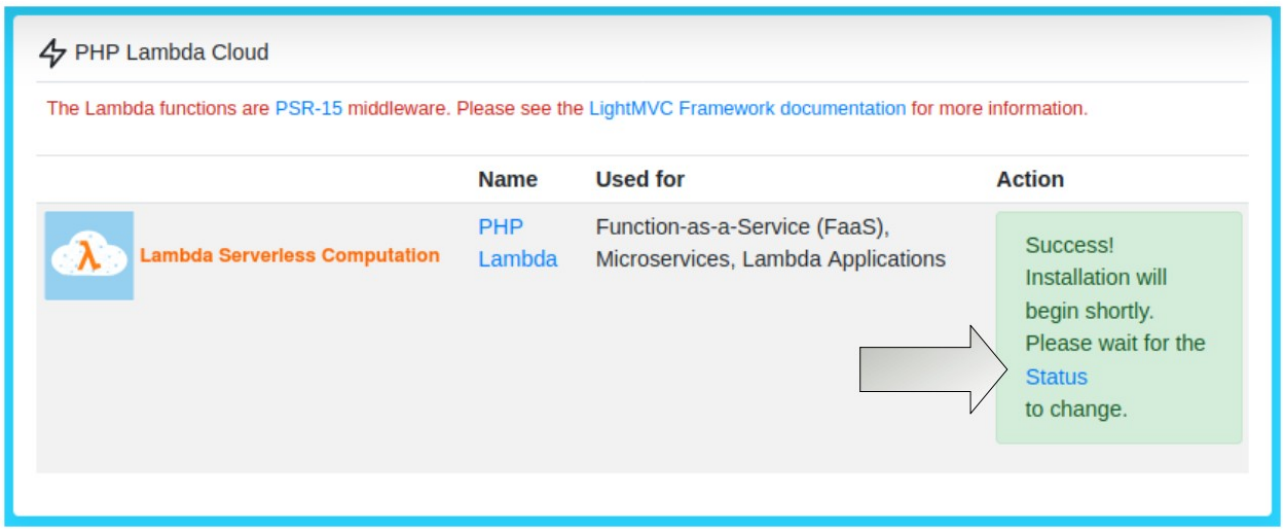
From the initial *Dashboard* screen that appears, select *Lambda Cloud* [1] and then *Install* [2].



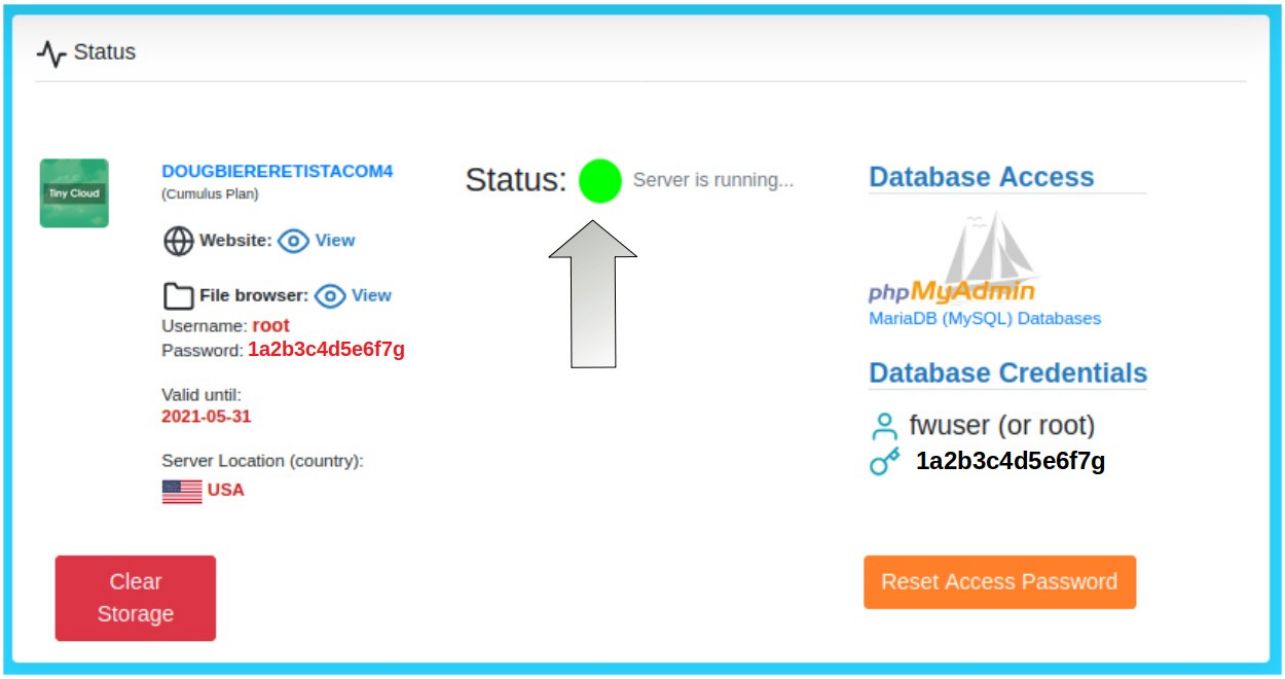
Click *OK* on the dialog box warning you that the current website will be deleted. This popup appears whenever you elect to install a new type of service (e.g. Lambda Cloud, Drupal, WordPress, etc.). In this case, of course, there *is* no website, so not to worry!



Seconds later, one the Lambda Cloud service has been installed, you can click on *Status* which takes you back to the top.



Don't take any further action until the status indicator goes green and you see *Server is running*, as shown here:



What About the HTML?

A great starting point would be to browse the wide world of free HTML templates. Many design companies offer HTML templates for WordPress, Drupal and Joomla, for example. These are generally referred to as *themes*. Although potentially useful as a starting point for an HTML website, it's better to locate a pure HTML template rather than a WordPress theme.

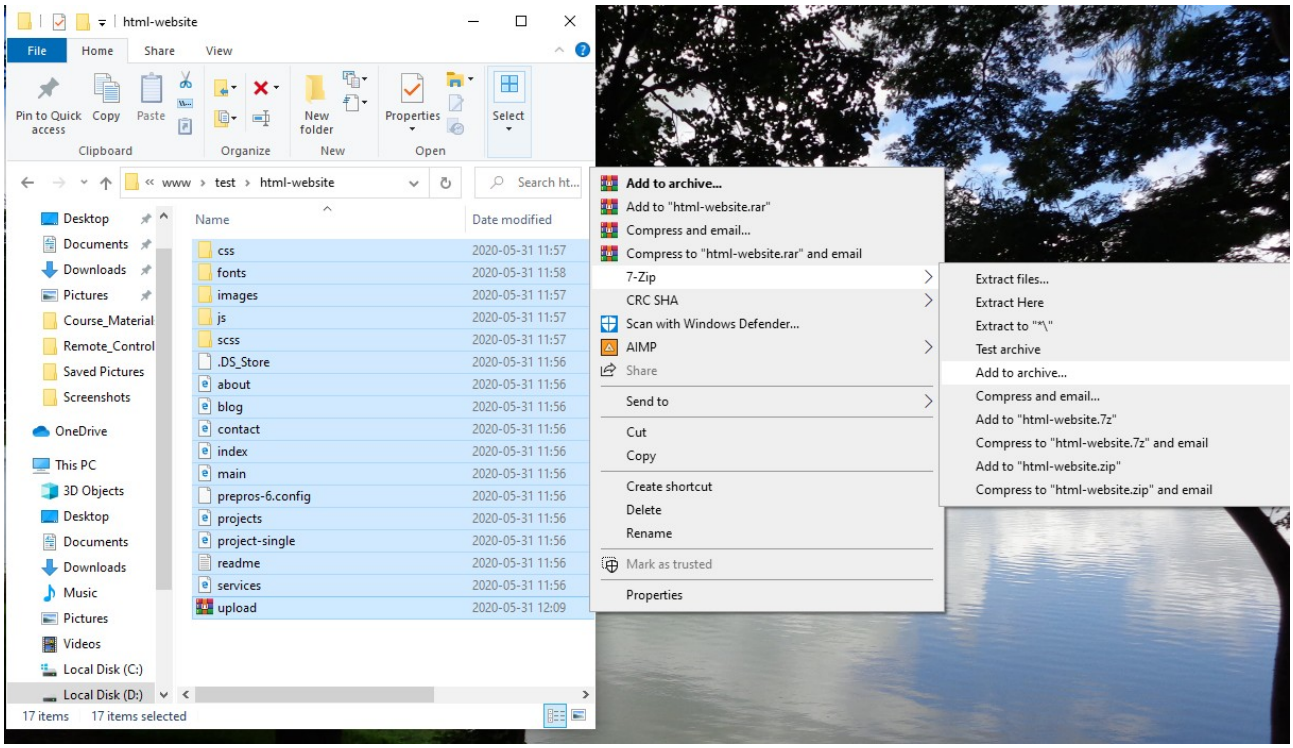
As an example, take *Colorlib*: if you go to this page: https://colorlib.com/wp/templates/, you will find several dozen free HTML templates. The templates are generally downloaded as a *zip* file as they contain not only HTML, but *CSS* (Cascading Style Sheets) as well.

Be sure to carefully read any license agreements that might apply to free template usage. As a general rule, the template designers want to get credit. In practical terms this means displaying a mandatory footer with a link back to the designers' website. This is a small price to pay for a great looking starting template that saves you days of work using tools you may not even have.

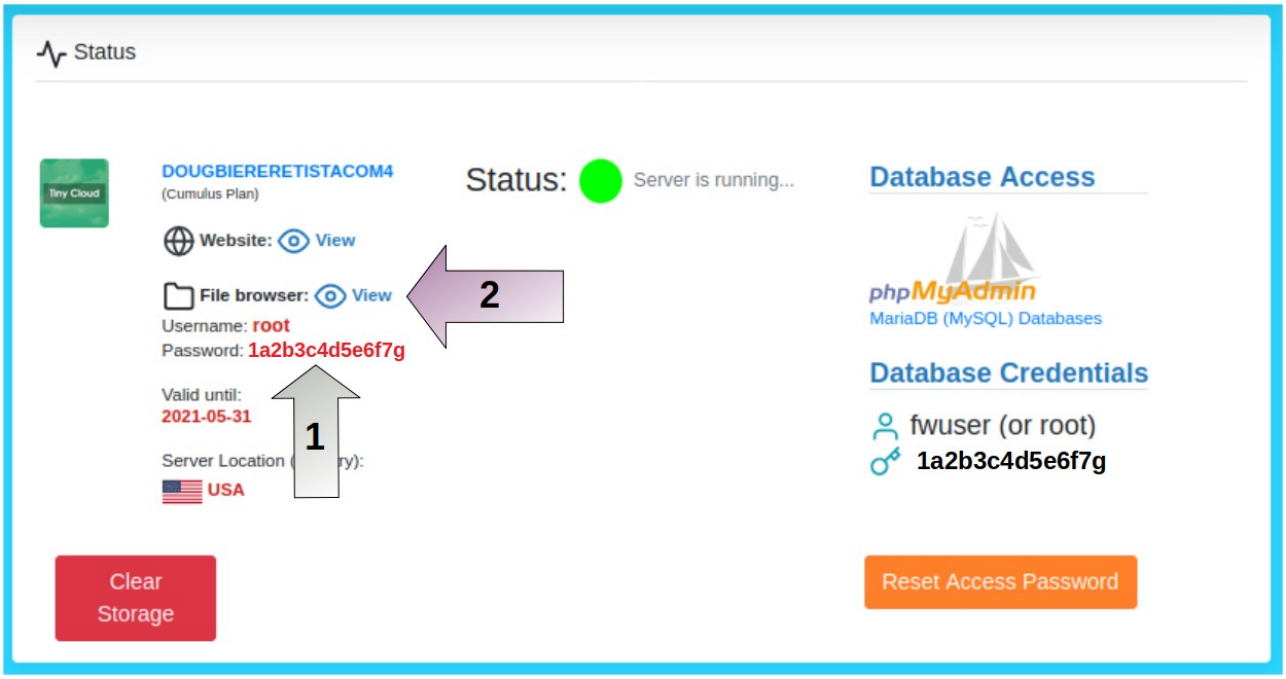
Once you've chosen a template, create a directory on your local computer, download and unzip the folder containing *index.html* into the new directory. You can then open up the *.html files and edit them, adding the proper wording and images appropriate to your company.

HTML is Done ... Now What?

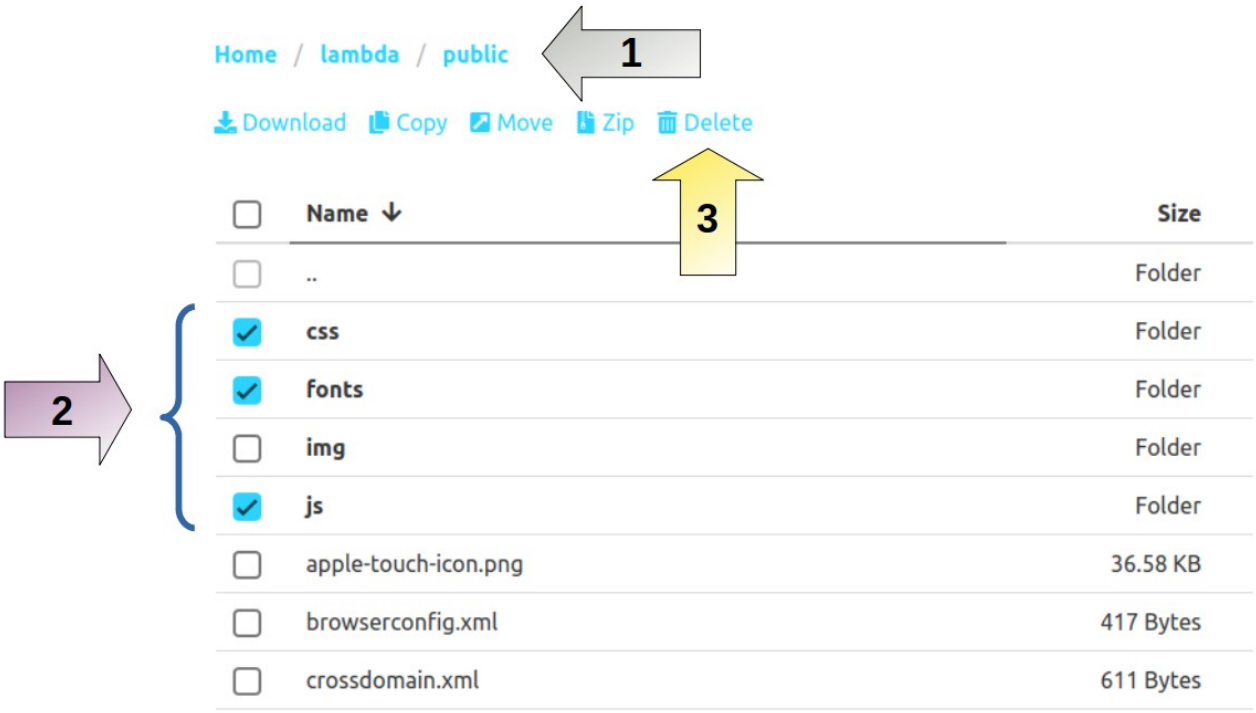
Once the HTML editing is complete, it's time to upload it to your new LfPHP website. First, be sure to zip all files and directories, starting with the directory containing *index.html* into a zip file, using the tools available on your operating system. The screenshot below shows using the Windows 7Zip utility to compress files to be uploaded.



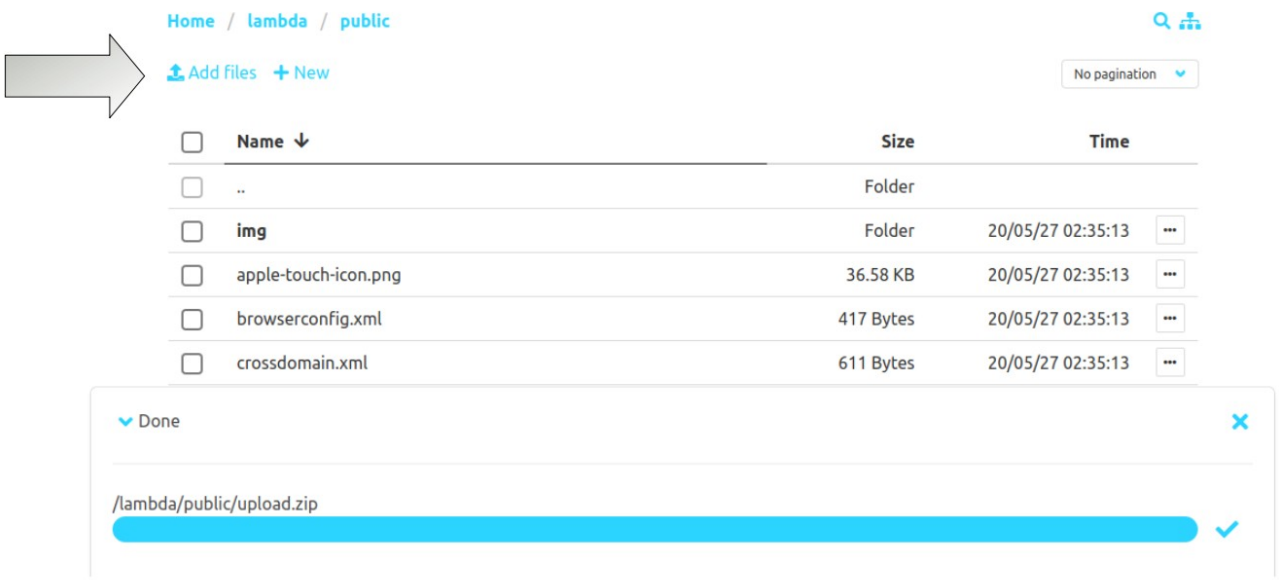
Returning to the Linux for PHP Cloud Services *Status* screen for your plan, locate *File* browser and first write down the username and password [1]. Secondly, click on *View* [2]:



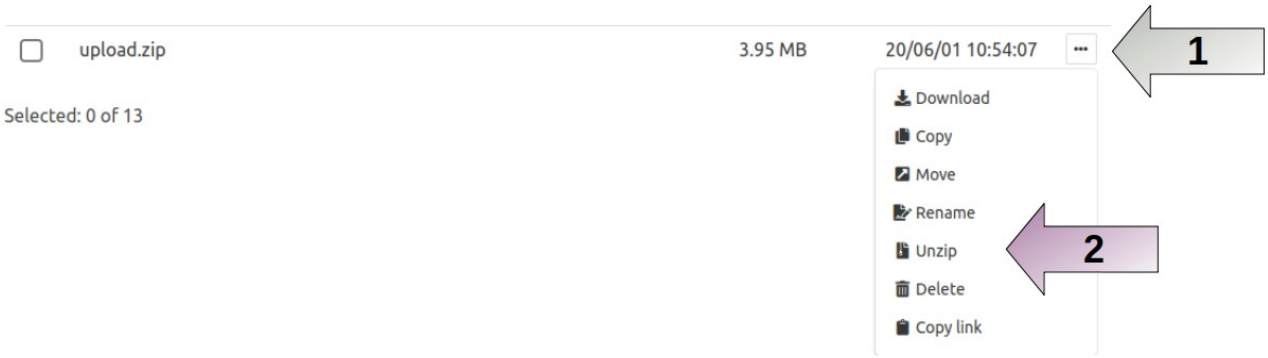
When prompted, enter the username and password and click *Log in*. From the *File browser* dialog window (which should appear as a new tab in your browser), click on *Lambda*. Scroll down a bit and click on *Public* [1]. You will then need to select these three folders: **css**, **fonts** and **js**. Once selected [2], click on *Delete* [3]. The reason why these need to be deleted is that when the zip file is unzipped following upload, it will create duplicate directories giving you more work. By deleting these folders in advance, you can unzip without any future problems.



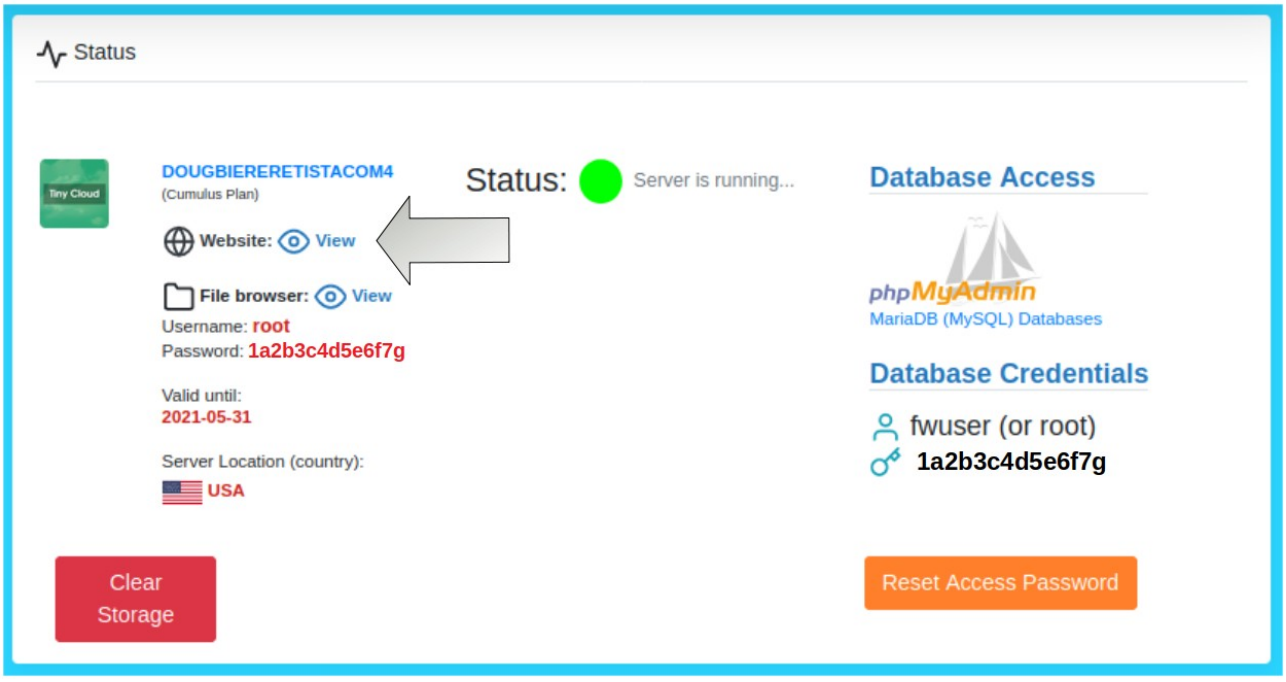
Of course, you need to answer *Delete* when the *Are you sure ...* dialog box appears. Next, select *Add Files* , locate the zip file just created on your local computer, and proceed with the upload.



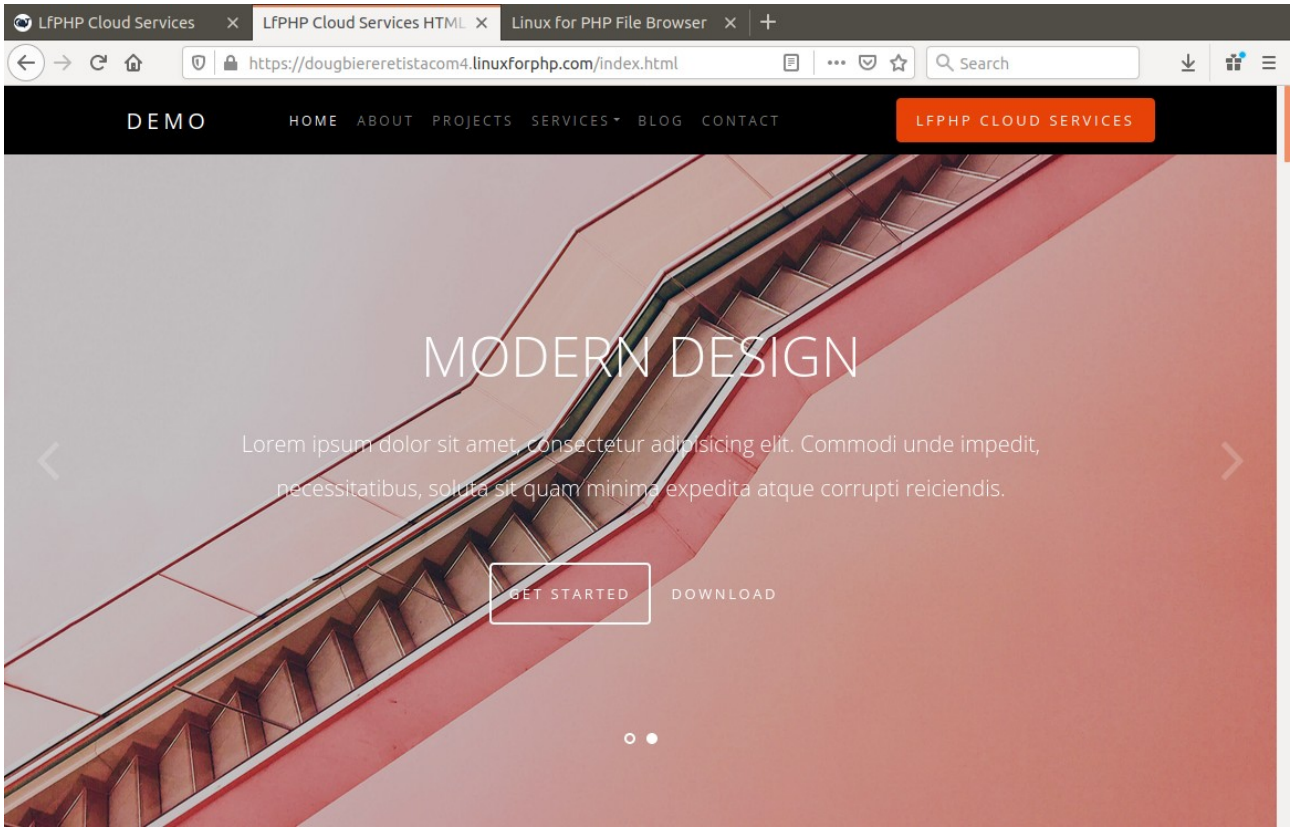
Locate the uploaded zip file, and click on the 3 dots icon to the right side of the file listing. In the dropdown menu select *Unzip*. Click on *Unzip* when you see the *Are you sure ...* dialog box appear.



At this point, if you return to your plan *Dashboard* page, you can select *Website: view*.



If you then append *index.html* to the URL you are given when purchasing the plan, you can now see your HTML website:

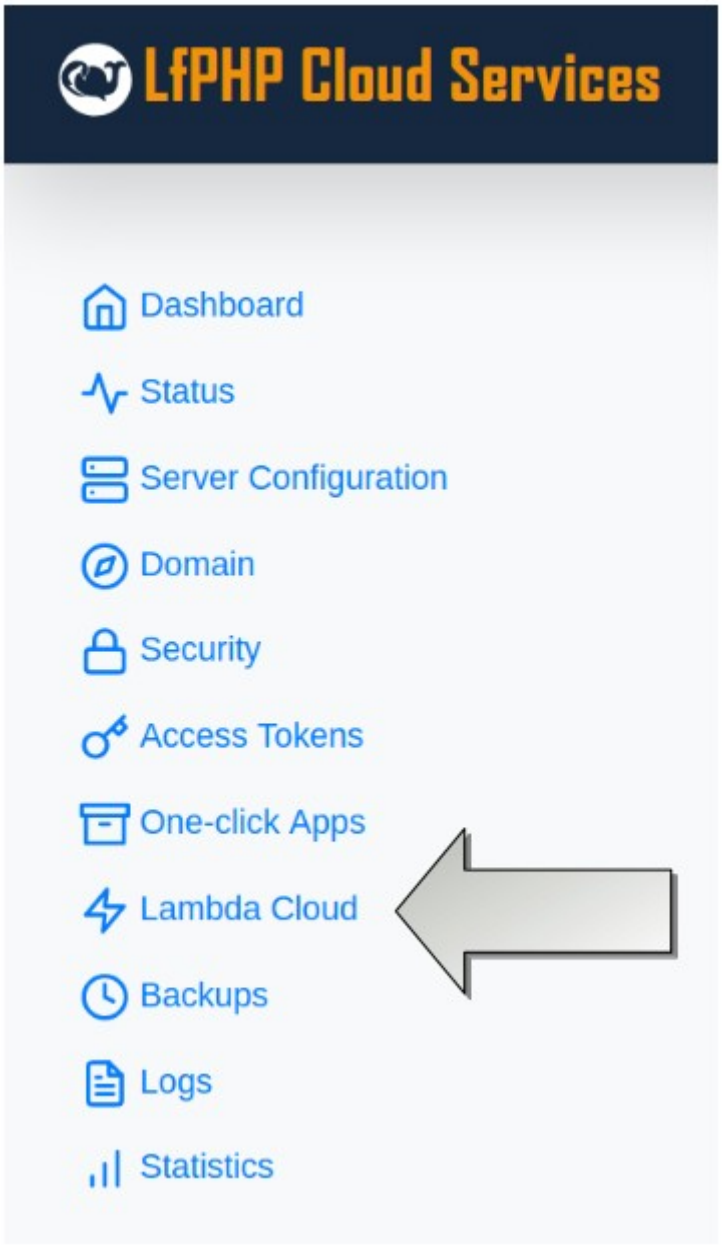


Great! So Are We Done?

We're not quite out of the woods, however! Although the HTML now appears as expected, there are still a couple of details to arrange. One issue is that if you remove *index.html* from the URL you enter in your browser, you'll notice we're back to the unsightly default home page. The other issue is that part of our objective was to add some PHP interactivity, in which we'll implement a *Contact* form whereby customers can sign up for an email newsletter. Let's have a look at reprogramming the default home page first.

Refactoring the Home Page

In order to have our new *index.html* page replace the default home page, we need to return to PHP *Lambda Cloud* from our plan dashboard page:



In the black box, enter the following PHP code [1]:

```
$index = '/srv/tempo/lambda/public/index.html';
$body = file_get_contents($index);
$response = new Laminas\Diactoros\Response();
$response = $response->withStatus(200);
$response->getBody()->write($body);
return $response;
```

The first two lines read the new *index.html* file. The last 4 lines render the HTML using the PHP Lambda Cloud mechanism. When you've finished enter the PHP code, click on *Deploy* [2].

You can now view the website without having to add *index.html* to the URL.

Adding a Signup Form

At last, we're ready to tackle the *Signup* form. To start we need to return to the HTML page that contains the signup form. Write down the values for the *name* attributes for all the HTML *input* elements to be submitted. Here is an example form that accepts a name and email address:

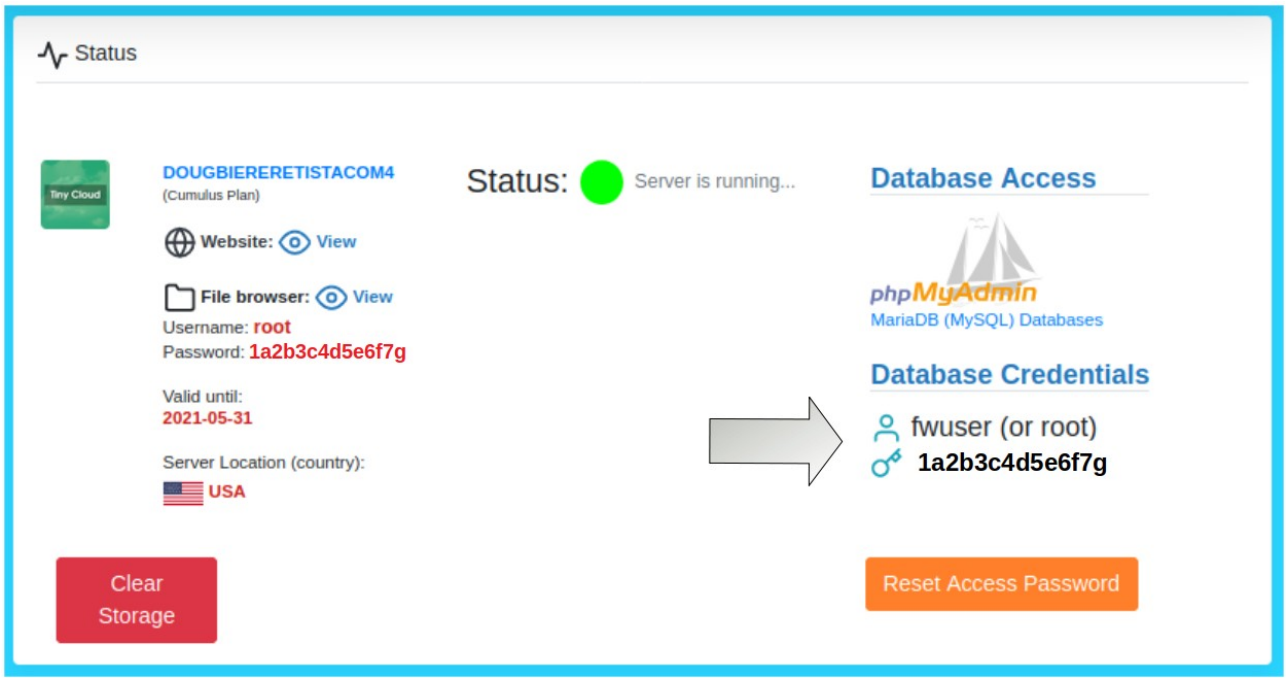
```
<form action="/signup" method="post">
  <div class="row">
    <div class="col-md-6 form-group">
      <label for="name">Name</label>
      <input type="text" id="name" name="name" class="form-control ">
    </div>
    <div class="col-md-6 form-group">
      <label for="email">Email</label>
      <input type="email" id="email" name="email" class="form-control ">
    </div>
  </div>
  <div class="row">
    <div class="col-md-6 form-group">
      <input type="submit" value="Send Message" class="btn">
    </div>
  </div>
</form>
```

Aside from the actual HTML *input* tags, what is of extreme importance is to configure the form *action* to a Lamba function route. In the example above, that route is **/signup**.

> NOTE: unfortunately, using the HTML page approach, if you need to change the HTML, you will need to first delete the HTML file using the *File browser*, and the select *Add files* to re-upload the changed file.

Now that the HTML signup form has been created and uploaded to the cloud, we need to define a new *Lambda Function*, with a route to match */signup*. Returning to the PHP Lambda Cloud section on the Linux for PHP Cloud Services plan page, click on the *Add* button to the right.

You need to find the auto-generated database username and password for you when the PHP Lambda Cloud was installed! Those can be found at the top of the dashboard here:



In the *Route* box enter **/signup**. Inside the code box, enter PHP code to intercept and process form data. First we need to check to see if the request is an HTTP POST:

```
$message = 'SORRY!  Unable to sign you up.';
if ($request->getMethod() == 'POST') {
```

If so, we grab the data and sanitize:

```
    $data = $request->getParsedBody();
    $email = strip_tags(trim($data['email'] ?? ''));
    $name = strip_tags(trim($data['name'] ?? ''));
```

We then create a PDO connection to the database.

```
    if ($name && $email) {
        try {
            $dsn = 'mysql:host=localhost;dbname=fw';
            $opts = [\PDO::ATTR_ERRMODE => \PDO::ERRMODE_EXCEPTION];
            $pdo = new \PDO($dsn, 'fwuser', 'PASSWORD', $opts);
```

We now define two SQL statements: one to check for the existence of the signee, the other to insert data for new customers.

```
$sqlCheck = 'SELECT * FROM email_list WHERE email = ?';
$sqlInsert = 'INSERT INTO email_list (name, email) VALUES (?,?)';
```

If the customer is already signed up, the appropriate message is displayed, otherwise a new database entry is made:

```
$stmt = $pdo->prepare($sqlCheck);
$stmt->execute([$email]);
$result = $stmt->fetch(\PDO::FETCH_NUM);
if ($result) {
    $message = 'GREAT!  You are already signed up, thanks!';
} else {
    $stmt = $pdo->prepare($sqlInsert);
    $stmt->execute([$name, $email]);
    if ($stmt->rowCount()) {
        $message = 'Thanks for signing up!';
    }
}
```

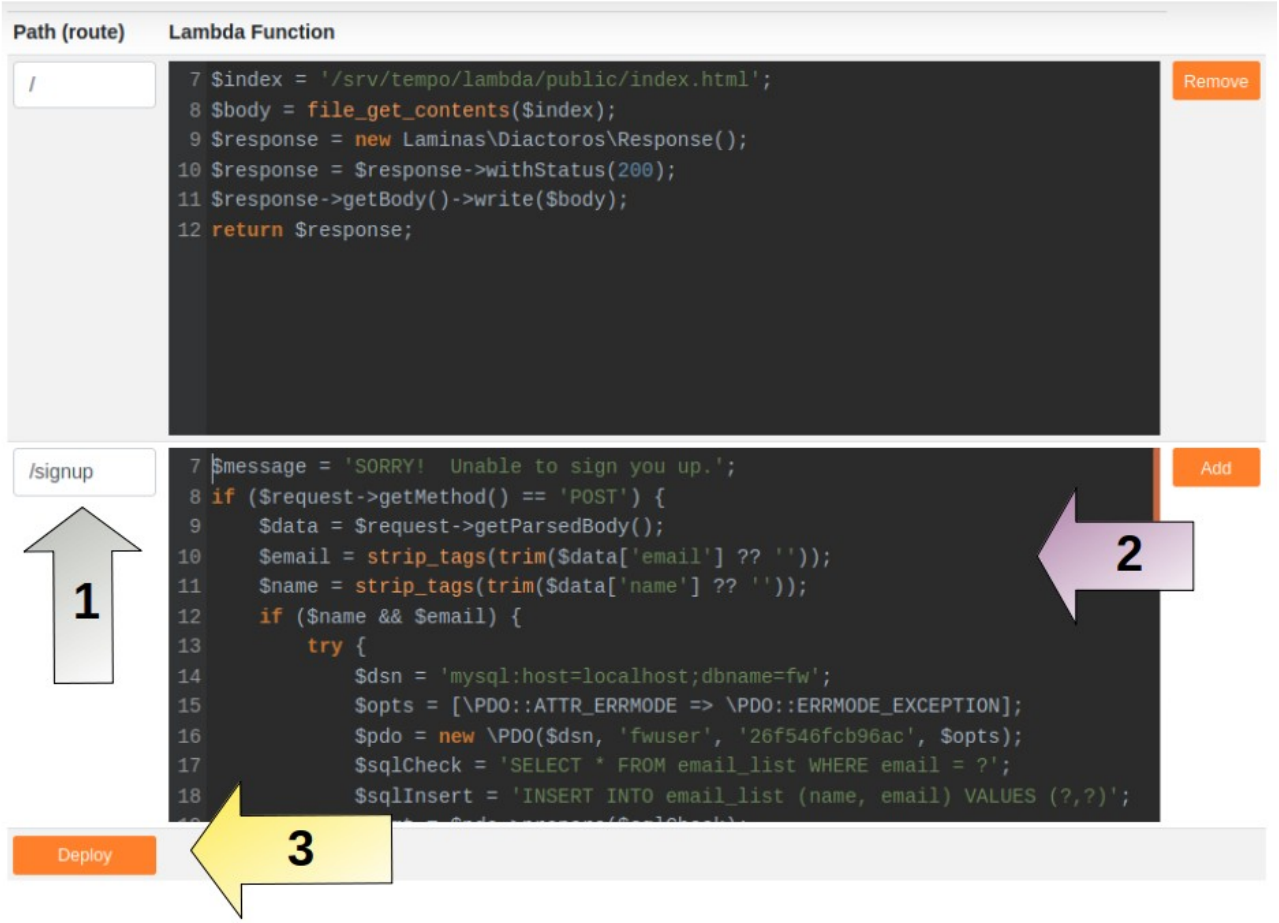
Finally, we need to return a response. In this case we simply read in the contents of our old friend *index.html*, and replace **<!-- MESSAGE -->** with the appropriate signup message:

```
$index = '/srv/tempo/lambda/public/index.html';
$body = file_get_contents($index);
$body = str_replace('<!-- MESSAGE -->', $message, $body);
$response = new Laminas\Diactoros\Response();
$response = $response->withStatus(200);
$response->getBody()->write($body);
return $response;
```

Here's the complete code block:

```
$message = 'SORRY!  Unable to sign you up.';
if ($request->getMethod() == 'POST') {
    $data = $request->getParsedBody();
    $email = strip_tags(trim($data['email'] ?? ''));
    $name = strip_tags(trim($data['name'] ?? ''));
    if ($name && $email) {
        try {
            $dsn = 'mysql:host=localhost;dbname=fw';
            $opts = [\PDO::ATTR_ERRMODE => \PDO::ERRMODE_EXCEPTION];
            $pdo = new \PDO($dsn, 'fwuser', '26f546fcb96ac', $opts);
            $sqlCheck = 'SELECT * FROM email_list WHERE email = ?';
            $sqlInsert = 'INSERT INTO email_list (name, email) VALUES (?,?)';
            $stmt = $pdo->prepare($sqlCheck);
            $stmt->execute([$email]);
            $result = $stmt->fetch(\PDO::FETCH_NUM);
            if ($result) {
                $message = 'GREAT!  You are already signed up, thanks!';
            } else {
                $stmt = $pdo->prepare($sqlInsert);
                $stmt->execute([$name, $email]);
                if ($stmt->rowCount()) {
                    $message = 'Thanks for signing up!';
                }
            }
        } catch (Throwable $t) {
            error_log(get_class($t) . ':' . $t->getMessage());
        }
    }
}
$index = '/srv/tempo/lambda/public/index.html';
$body = file_get_contents($index);
$body = str_replace('<!-- MESSAGE -->', $message, $body);
$response = new Laminas\Diactoros\Response();
$response = $response->withStatus(200);
$response->getBody()->write($body);
return $response;
```

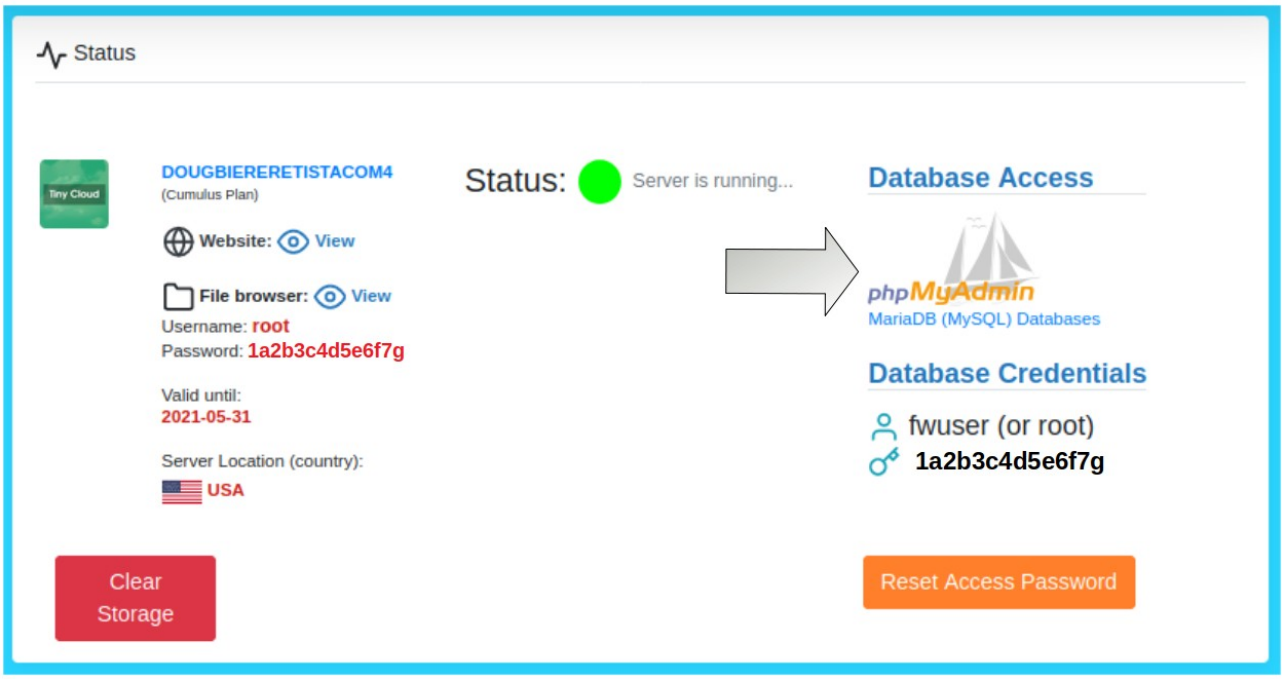
To deploy, be sure to enter the new route **/signup** [1], insert the form processing code [2] and click *Deploy* [3]:



What About the Database?

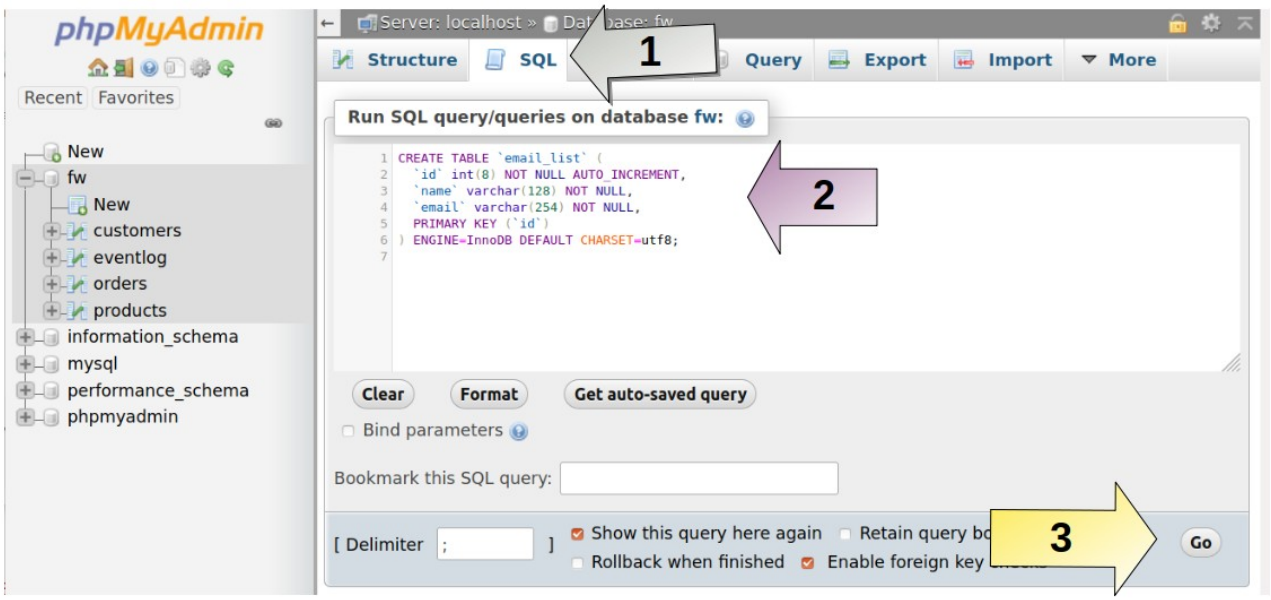
We need to create a database table to contain our new email list. Linux for PHP Cloud Services automatically creates a database for you called **fw**. You can create as many additional databases as you wish, and can also add tables to the existing database.

As we mentioned above, the auto-generated database username and password is shown in your account plan dashboard. Also, in the same area, is a link to access the database. Click on the *phpMyAdmin* icon, and, when prompted, enter the database username and password to manage the database.



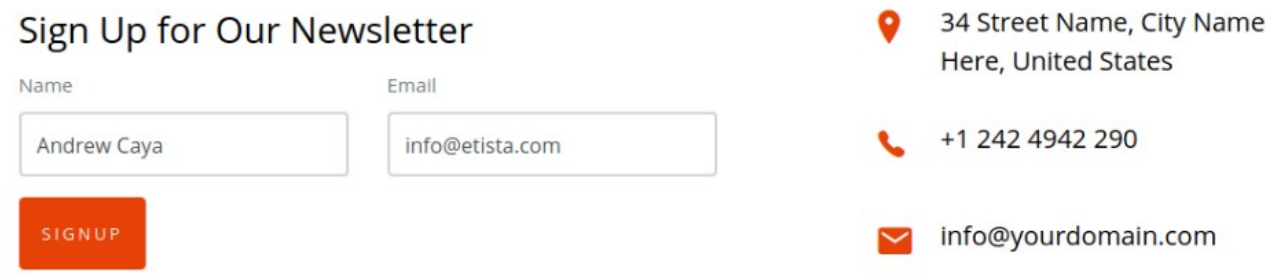
You can then select *SQL* [1], paste in this block of text [2] and click Go [3] to create the new table:

```
CREATE TABLE `email_list` (
  `id` int(8) NOT NULL AUTO_INCREMENT,
  `name` varchar(128) NOT NULL,
  `email` varchar(254) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

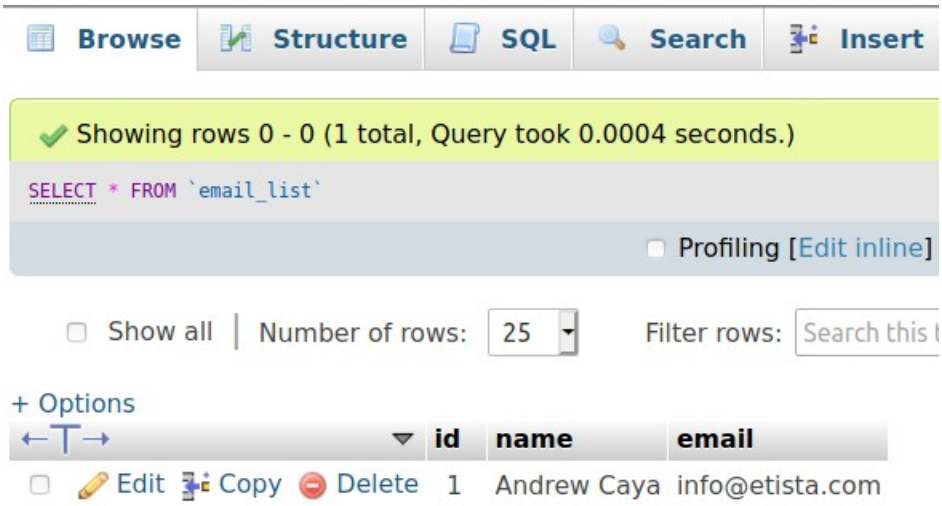


Ready, Set, Go!

All the pieces are now in place. To test, return to your browser and view your new website. Bring up the *Contact* page and enter your first customer:



If you return to *phpMyAdmin*, you will see that the new entry has been added:



And in the words of a famous character: *buddy, buddy, buddy ... that's all folks*. Happy coding on your new interactive HTML websites using the Linux for PHP Lambda Cloud!