S3 for Application Storage

**SPL-BE-200-DVSFAS-1 - Version 1.0.4**

© 2024 Amazon Web Services, Inc. or its affiliates. All rights reserved. This work may not be reproduced or redistributed, in whole or in part, without prior written permission from Amazon Web Services, Inc. Commercial copying, lending, or selling is prohibited. All trademarks are the property of their owners.

Note: Do not include any personal, identifying, or confidential information into the lab environment. Information entered may be visible to others.

Corrections, feedback, or other questions? Contact us at [*AWS Training and Certification*](https://support.aws.amazon.com/#/contacts/aws-training).

**Lab overview**

This lab focuses on updating the application code developed for the grid-maker application. After testing the application to create a new grid image locally, you update the application code to use images stored in an Amazon Simple Storage Service (Amazon S3) bucket instead. You verify the functionality of the application by accessing images stored in an S3 bucket, creating a new grid image, and generating an S3 presigned URL to view the grid image that gets stored in a separate S3 bucket.

This lab is designed to challenge developers who want to test their abilities. You are given high-level steps to perform and you have the opportunity to identify the solution on your own. This lab provides hints to help if you get stuck along the way.

If you are a new developer, you can refer to the step-by-step instructions in the *Solutions* expand-collapse section.

OBJECTIVES

By the end of this lab, you should be able to:

* Update Python code to list objects in an S3 bucket.
* Update Python code to get objects in an S3 bucket.
* Update Python code to put objects in an S3 bucket.
* Update Python code to create an S3 presigned URL for an image to be viewed in a browser.

TECHNICAL KNOWLEDGE PREREQUISITES

To successfully complete this lab:

* You should be familiar with basic navigation of the AWS Management Console.
* You should be comfortable editing and running scripts by using an AWS Cloud9 code editor and terminal.
* You should have a basic understanding of and familiarity with Amazon S3.

DURATION

This lab requires *60* minutes to complete.

ICON KEY

Various icons are used throughout this lab to call attention to different types of instructions and notes. The following list explains the purpose for each icon:

* **Command:** A command that you must run.
* **Expected output:** A sample output that you can use to verify the output of a command or edited file.
* **Note:** A hint, tip, or important guidance.
* **Additional information:** Where to find more information.
* **Consider:** A moment to pause to consider how you might apply a concept in your own environment or to initiate a conversation about the topic at hand.

**Start lab**

1. To launch the lab, at the top of the page, choose **Start lab**.

 You must wait for the provisioned AWS services to be ready before you can continue.

1. To open the lab, choose **Open Console**.

You are automatically signed in to the AWS Management Console in a new web browser tab.

**Do not change the Region unless instructed.**

COMMON SIGN-IN ERRORS

**Error: You must first sign out**



If you see the message, **You must first log out before logging into a different AWS account:**

* Choose the **click here** link.
* Close your **Amazon Web Services Sign In** web browser tab and return to your initial lab page.
* Choose **Open Console** again.

**Error: Choosing Start Lab has no effect**

In some cases, certain pop-up or script blocker web browser extensions might prevent the **Start Lab** button from working as intended. If you experience an issue starting the lab:

* Add the lab domain name to your pop-up or script blocker’s allow list or turn it off.
* Refresh the page and try again.

**Task 1: Run the application to store the image locally**

In this task, you connect to the AWS Cloud9 environment and start configuring the application.

TASK 1.1: CONNECT TO AWS CLOUD9

AWS Cloud9 is a cloud-based integrated development environment (IDE) that you can use to write, run, and debug your code with only a browser. It includes a code editor, debugger, and terminal. AWS Cloud9 comes prepackaged with essential tools for popular programming languages, including JavaScript, Python, PHP, and more. You don’t need to install files or configure your development machine to start new projects.

In this task, you connect to the AWS Cloud9 environment that’s provisioned as part of this lab.

1. From the **Lab Information** section to the left of these instructions, copy the **Cloud9Environment** URL link and in a new browser tab, paste the link.

The browser takes you to the AWS Cloud9 environment that you use during this lab.

You don’t need the **Cloud9 Welcome screen** or any of the other default tabs that appear when you first launch **AWS Cloud9**.

1. Close each tab by choosing the **X**.

This section of the IDE is where you update various file throughout this lab.

**Consider:** Take a moment to familiarize yourself with the **AWS Cloud9** IDE interface.

* In the middle of the screen, a single terminal session is open in the editor. You can open multiple tabs in this window to edit files and run terminal commands.
* The file navigator appears on the left side of the screen. As you build out your AWS Cloud Development Kit (AWS CDK) environment and application, additional directories and files appear here.
* A gear icon is on the right side of the screen. Choosing this icon opens the AWS Cloud9 **Settings** panel.

Every *AWS Cloud9* workspace is automatically assigned *AWS Identity and Access Management (IAM)* credentials. These credentials provide the workspace with limited access (based on your federated role) to some AWS services in your account. We call these AWS managed temporary credentials.

TASK 1.2: REVIEW AND RUN THE APPLICATION

In this task, you review the application and learn about the main sections and what they do.

The **terminal pane** is at the bottom of the IDE. You can expand the pane up halfway to have more visibility when you run commands. You can also close it and open a new terminal session from the top menu. (To open a new terminal session, choose the  icon and choose **New Terminal**.)

1. From the file tree, expand the **local-script** folder.
2. To review the code, open **main.py**.

The following list describes the main sections of this application:

* *Import modules*: The first three lines import some modules that are needed for the script. *glob* is used to find files that match a pattern; *math* is used for mathematical functions; and *PIL* (or Python Imaging Library) is used for image processing.
* *Define variables*: The next four lines define variables that are used throughout the script. *tile\_size* is the size of each image tile in *pixels*. *source\_images* is a list of file names that match the pattern *source/\*.jpg*. *image\_count* is the number of source *images*. Finally, *tiles\_width* and *tiles\_height* are the number of tiles in each row and column of the grid, respectively.
* *Print information*: The next line prints information about what the script is doing, such as converting source images and creating a grid with certain dimensions.
* *Create destination image*: The next line creates a new empty image object with the *mode RGB* (or red-green-blue) and a size equal to the total width and height of the grid in pixels.
* *Loop through source images*: The next two lines start two nested loops that iterate over each row (y) and column (x) of the grid. These loops contain an *if* statement that checks if there are any source images left in the list. If yes, then:
* *Pop filename*: The next line pops (removes and returns) one file name from the list and assigns it to a variable called filename.
* *Open image*: The next two lines open the file with *binary mode (‘rb’)* by using a context manager (*with*) that automatically closes it after use. Then, they create an image object from it by using *Image.open()*.
* *Crop image*: The next six lines crop (cut out) a square region from the image. The cropped region has the same length as the shorter side of the image. This is done by calculating the coordinates of the center and the corners of the square by using *img.size*, *min()*, and *// (integer division)*. Then, they use *img.crop()* to get a new image object with only that region.
* *Resize image*: The next line resizes (changes the dimensions) of the cropped image to match the tile size by using *img.resize()*.
* *Paste image*: The next line pastes (copies) the resized image onto the destination image at a position that depends on the row and column index (x and y) multiplied by the tile size by using *destination\_image.paste()*.
* *Save destination image*: The last line saves (writes) the destination image to a file called *destination/grid.jpg* by using *destination\_image.save()*.

TASK 1.3: RUNNING THE APPLICATION TO CREATE THE GRID.JPG IMAGE

In this task, you install the application dependencies and then run the application.

1. **Command:** Change directories into the **local-script** folder by running the following command:

cd local-script

**Expected output:**

*None, unless an error occurs.*

1. **Command:** Install the Python packages required for this application (which are based on the **requirements.txt** file) by running the following command:

pip install -r requirements.txt

**Expected output:** Versions might differ from the versions in the following output.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Collecting pillow

Downloading Pillow-9.4.0-cp39-cp39-manylinux\_2\_17\_x86\_64.manylinux2014\_x86\_64.whl (3.3 MB)

━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.3/3.3 MB 51.5 MB/s eta 0:00:00

Installing collected packages: pillow

Successfully installed pillow-9.4.0

WARNING: You are using pip version 22.0.4; however, version 23.0.1 is available.

You should consider upgrading via the '/home/ec2-user/.pyenv/ver

The **local-script/source** folder contains 16 images. After running the **main.py** application, you create the **grid.jpg** image that is a grid of all 16 images.

1. **Command:** To create the initial **grid.jpg** image that includes the 16 image files, run the following command:

python main.py

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Converting: 16 source images. Creating: 4 x 4 grid.

1. View the image by opening the **local-script > destination** folder and then opening the **grid.jpg** file.

The grid should look similar to the following image.



**Congratulations!** You successfully ran the application to create an image made up of 16 images in a grid format.

**Task 2: Update the main.py code**

In this task, you update the code in the *main.py* file to re-create the same *grid.jpg* file by using images stored in Amazon S3. You update the script so you can *list S3 objects*, *get S3 objects*, *put S3 objects*, and generate a *presigned URL* to view the image in a browser tab.

CHALLENGE A: INSTALL THE APPLICATION REQUIREMENTS

1. Using the *s3-scripts/requirements.txt* file, install the application requirements.

**Note:** If you want help with solving the challenge, expand the **Hint** or **Solution** sections.

**Hint**

**Solution**

CHALLENGE B: UPDATE MAIN.PY TO LIST AMAZON S3 OBJECTS

In this task, you update the code in the *~/s3-script/main.py* file. Update the *CHALLENGE-B BEGIN* section to *list* objects in the *source-images* S3 bucket.

**Hint**

**Solution**

CHALLENGE C: UPDATE MAIN.PY TO GET AMAZON S3 OBJECTS

In this task, you update the code in the *~/environment/s3-script/main.py* file. Update the *CHALLENGE-C* section to perform an Amazon S3 get\_objects operation, which you can use to get the contents of an image in the bucket.

**Hint**

**Solution**

CHALLENGE D: UPDATE MAIN.PY TO PUT AMAZON S3 OBJECTS

In this task, you update the code in the *main.py* file. Update the *CHALLENGE-D BEGIN* section with the code snippet that *puts* the grid image to a randomly named object in the *destination-images* Amazon S3 bucket.

**Hint**

**Solution**

CHALLENGE E: UPDATING MAIN.PY TO CREATE AN AMAZON S3 PRESIGNED URL

In this task, you update the code in the *main.py* file. Update the *CHALLENGE-E BEGIN* section to create an *Amazon S3 presigned URL*. The presigned URL is used to provide short-term access to a private object in your S3 bucket. In this case, it grants access to the *grid.jpg* image that is created in a later task.

**Hint**

**Solution**

**Task 3: Run the updated application**

In this task, you set two variables. One variable is for the *source bucket name*, which is prepopulated with the 16 images used to create the grid.jpg locally. The second variable is for *destination bucket name*, which is used to store the new grid.jpg image after it’s created. After you create those variables, you run the *main.py* application with those two variables as parameters. The application creates the grid.jpg image from images stored in Amazon S3, and then stores the *grid.jpg* image in the S3 *destination-images* bucket. Lastly, you view the *grid.jpg* image stored in the private *destination-images* Amazon S3 bucket in a browser by using an *Amazon S3 presigned URL*.

The commands you run list the buckets in the account. They search for strings that identify the *source-images* and *destination-images*, and return those values. Then, they print out the actual bucket names to verify that the values are set properly.

1. **Command:** To set the variables for **source\_bucket** and **destination\_bucket**, run the following commands:

source\_bucket=$(aws s3api list-buckets --output text --query 'Buckets[?contains(Name, `source-images`) == `true`] | [0].Name')

destination\_bucket=$(aws s3api list-buckets --output text --query 'Buckets[?contains(Name, `destination-images`) == `true`] | [0].Name') ;

printf "\nSource bucket: $source\_bucket\nDestination bucket: $destination\_bucket\n\n"

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

$ source\_bucket=$(aws s3api list-buckets --output text --query 'Buckets[?contains(Name, `source-images`) == `true`] | [0].Name')

$ destination\_bucket=$(aws s3api list-buckets --output text --query 'Buckets[?contains(Name, `destination-images`) == `true`] | [0].Name') ;

$

$ printf "\nSource bucket name: $source\_bucket\nDestination bucket name: $destination\_bucket\n\n"

Source bucket: source-images-us-west-2-2766912

Destination bucket: destination-images-us-west-2-2766912

1. **Command:** To initiate the **main.py** application, create the new **grid.jpg** image, and create an **Amazon S3 presigned-url** for the **grid.jpg** image, run the following command:

python main.py $source\_bucket $destination\_bucket

**Note:** If you see an error when you run this command, double-check all the challenges and make sure you have completed each challenge correctly. If you are unable to identify the issue, you can find the *solutions file* under *~/environment/s3-script/lab1-solution.py*.

**Expected output:**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\* This is OUTPUT ONLY. \*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Converting: 16 source images. Creating: 4 x 4 grid.

Creating temp file /tmp/tmpo6xu2ukh.jpg

Saved file to s3 bucket: destination-images-us-west-2-2766912, key: 60db3a46105959214468cba5f5786f16.jpg

Presigned url: https://destination-images-us-west-2-2766912.s3.amazonaws.com/60db3a46105959214468cba5f5786f16.jpg?AWSAccessKeyId=ASIAQ4LBBKYBH6CU4ESR&Signature=2rV48mUXhgSzX1vMwdOs85FJO3o%3D&x-amz-security-token=IQoJb3JpZ2luX2VjEAQaCXVzLXdlc3QtMiJGMEQCIGzCykNA5NHtqsIm%2FTpFYdivCb3L72YrrxpDQUyiRiHWAiBSzXRkKsvGBuAnaTy3TUW%2B6kBsc41HdinYhOGS4ChzvCq5BAit%2F%2F%2F%2F%2F%2F%2F%2F%2F%2F8BEAEaDDA2MDg2OTkyNDM1NCIMNCbjUvkBdDH6hxftKo0EXAT8GIGzX8FoH4oKqMsNyBLZrVh3z9IeQyIAE8hDskn6BcmgzbZJQlcp%2FQAoBhYoY3E5iUpguNzsrOjsUUe6E5eEeWLc5t1ICjbnfKgIRo6k0QG1FHAh63ZbZCf3%2Ff6EZdpqrtTzjWhy28jm3%2B39rwOmsqFIxsglqOCcfFykO98AKSOw4lqeAeYRcHtgLsKstO4KKk8HU1PQ%2FfzAFpgMS0JxsNctMRTAHdw9gRzx1d1VR6BZLWEOdyM5lb7toyViRQ3RjuULb2cwOLF3jpFRgDgAeofd58OpI0sa6vZ0jOmpbRXacFCs3MVTy9exVtYfdnbe6c396b6S%2BgKK4A%2F98%2BMgOfx49MwPmiS2PTDpnbUOPXpj5pHMJhP8UqXPUXXmwD4ZbZS%2F1CuB%2BrPJYz14mkHJk2SdBzDZzekHbrTFbeSqudXZ6A4hZaLazcx4kNjpkU96dy%2BzMh2tyKtL4BiGgq3iyUi5wNzzAj6XBNQYon%2Fynp8JzaWbqQB30j8cXqTc9XnQkRz4iCunUDtW2awzIkRrOAF%2BNOgMYR7R9jDIHVl%2FFJFDR7u4y7Ao9ZUQpcfjhG0soP3C6HQA0sP036FPlMDJQdA4zdSNNBJ%2BIEavuAJWk5CCECk68Lv7B%2FMCM5jRw%2BDWVr%2BhX%2FKQF9Oqn0%2Fjyf%2FFhIJrcmEVVT%2FkLNqcaztHDMLhXW1Ezyz9QcHcMMzng6AGOqcCMFE7k27FyCgVTUt5NCaTJ1myVBGKfAAd42eT0wBUzZVqDnQ0GgHyrz%2BlPN1rHHMQBKiS9kf8XKbKu1HOkq3sTirZ9efNTGCqPsObf2ENXShHh%2FNQJNA4phy1GCJJNyQP5wK1BkWlQn6fae9upSyBm6LRJccHXf%2FIna5MF%2Fup0KleFexJ8FQe4673NGepD9KGCjIgbfnrLJnYR%2FV0jImhhVHpHxKKmeA9ccwjQC45F43WnAosigIBRqN%2B1JpeUgMckqZZ6UhPTQBeppSPXkBGiLO%2F9oJMOKk2Q%2BDglRxEW3pJjQDDgZS27fkYvTIfLeqDO1uLdiIknbBzJERLd88laxlp%2BppdoHWsxdZlPXMEpfGXeY8PyLof2J6u7GEi7lZzkmolWeUJVw%3D%3D&Expires=1677788046

1. To view the **grid.jpg** image stored in the **destination-images** S3 bucket, copy the **Amazon S3 presigned URL** and paste it in a new browser tab.

**Congratulations!** You successfully created a *grid.jpg* image made from 16 individual images that were merged into one image and stored locally. Then, you updated the *s3-script/main.py* Python application to create the same *grid.jpg* image based on the same 16 images that were stored in the *source-images* Amazon S3 bucket. The new *grid.jpg* image was created and stored in the *destination-images* Amazon S3 bucket. You then viewed the *grid.jpg* image in a browser by creating and using an *Amazon S3 presigned URL*.

**Conclusion**

 Congratulations! You now have successfully:

* Updated Python code to list objects in an S3 bucket.
* Updated Python code to get objects in an S3 bucket
* Updated python code to put objects in an S3 bucket.
* Updated Python code to create an S3 presigned URL for an image to be viewed in a browser.

**End lab**

Follow these steps to close the console and end your lab.

1. Return to the **AWS Management Console**.
2. At the upper-right corner of the page, choose **AWSLabsUser**, and then choose **Sign out**.
3. Choose **End lab** and then confirm that you want to end your lab.

For more information about AWS Training and Certification, see [*https://aws.amazon.com/training/*](https://aws.amazon.com/training/).

*Your feedback is welcome and appreciated.*  
If you would like to share any feedback, suggestions, or corrections, please provide the details in our [*AWS Training and Certification Contact Form*](https://support.aws.amazon.com/#/contacts/aws-training).