

Cloud Storage

John Wesley Hayhurst and David Hamblin

Introduction

- Files can be stored on a cloud for others to access
- Several notable services exist for this purpose
 - Amazon, Google, Microsoft
- Push from one device, pull from another

Amazon

- Cloud storage through Amazon Web Services (AWS)
- Many different services
 - Elastic Compute Cloud (EC2)
 - NoSQL databases (DynamoDB)
 - Simple Storage Service (S3)
- For our purposes - completely free!

What You Need

- Amazon AWS account
 - Different from Amazon account where you buy everything
- FTP/SSH Client
 - SSH in terminal or command prompt for Windows, OSX, Linux
 - For Windows, non-terminal clients
 - WinSCP
 - PuTTY
- Android Studio

Setting up an Amazon Server

- Create an Amazon AWS Account
- Go to EC2
- Launch Instance
- Select Ubuntu Server and free tier
- Create a security group
 - Ports 21, 22, and 80 opened (SSH, FTP, and HTTP)
 - Access to anywhere (or own IP)

Setting up an Amazon Server

- We create an EC2 instance, utilizing Ubuntu Linux

Create Instance

To start using Amazon EC2 you will want to launch a virtual server, known as an Amazon EC2 instance.

Launch Instance

Note: Your instances will launch in the US East (N. Virginia) region



Ubuntu Server 16.04 LTS (HVM), SSD Volume Type - ami-f4cc1de2

Ubuntu Server 16.04 LTS (HVM), EBS General Purpose (SSD) Volume Type. Support available from Canonical (<http://www.ubuntu.com/cloud/services>).

Free tier eligible

Root device type: ebs Virtualization type: hvm

Setting up an Amazon Server

- Next choose the instance type **t2.micro** for the free tier eligible server

	Family	Type
<input type="checkbox"/>	General purpose	t2.nano
<input checked="" type="checkbox"/>	General purpose	t2.micro Free tier eligible

- Followed by Review and Launch

[Cancel](#)[Previous](#)[Review and Launch](#)[Next: Configure Instance Details](#)

Setting up Amazon Server

- Configure Security Groups

▼ Security Groups			
Security group name		launch-wizard-2	
Description		launch-wizard-2 created 2017-04-15T11:06:21.093-04:00	
Type ⓘ	Protocol ⓘ	Port Range ⓘ	Source ⓘ
SSH	TCP	22	0.0.0.0/0

- Default allows SSH, add HTTP and Custom TCP Rule

Type ⓘ	Protocol ⓘ	Port Range ⓘ	Source ⓘ
HTTP	TCP	80	0.0.0.0/0
HTTP	TCP	80	::/0
SSH	TCP	22	0.0.0.0/0
SSH	TCP	22	::/0
Custom TCP Rule	TCP	21	0.0.0.0/0
Custom TCP Rule	TCP	21	::/0

Setting up an Amazon Server

- Ready to Launch!
- But wait...
- Key pair required for authentication
- Give name, and download
- Now we can Launch Instance, and go to View Instances

Select an existing key pair or create a new key pair

A key pair consists of a **public key** that AWS stores, and a **private key file** that you store. Together, they allow you to connect to your instance securely. For Windows AMIs, the private key file is required to obtain the password used to log into your instance. For Linux AMIs, the private key file allows you to securely SSH into your instance.

Note: The selected key pair will be added to the set of keys authorized for this instance. Learn more about [removing existing key pairs from a public AMI](#).

Create a new key pair

Key pair name

server_key







Download Key Pair

You have to download the **private key file** (*.pem file) before you can continue. **Store it in a secure and accessible location.** You will not be able to download the file again after it's created.

Cancel

Launch Instances

Server Information

Instance ID ▴	Instance Type ▾	Availability Zone ▾	Instance State ▾	Status Checks ▾	Alarm Status	Public DNS (IPv4) ▾	IPv4 Public IP ▾	IPv6 IPs ▾	Key Name ▾
i-0cf1f12be3...	t2.micro	us-east-1d	 running	 Initializing	None	 ec2-54-147-181-10.com...	54.147.181.10	-	server_key
i-0db80b825...	t2.micro	us-east-1a	 running	 2/2 checks ...	None	 ec2-34-201-19-233.com...	34.201.19.233	-	graduate

- Shows state, type, ID, associated key, etc.
- Public DNS (IPv4) for connections
 - Long name, but can acquire Elastic IP for short IPv4 address at a price
 - All we need from here for AWS

Connecting to an Amazon Server

- Requires an SSH client, e.g. SSH or PuTTY
- With a username, password, and key-pair when launching the server
- Default user: ubuntu, no password
- Yes to authenticity pop-up
- Requires additional setup on server

SSH Connection

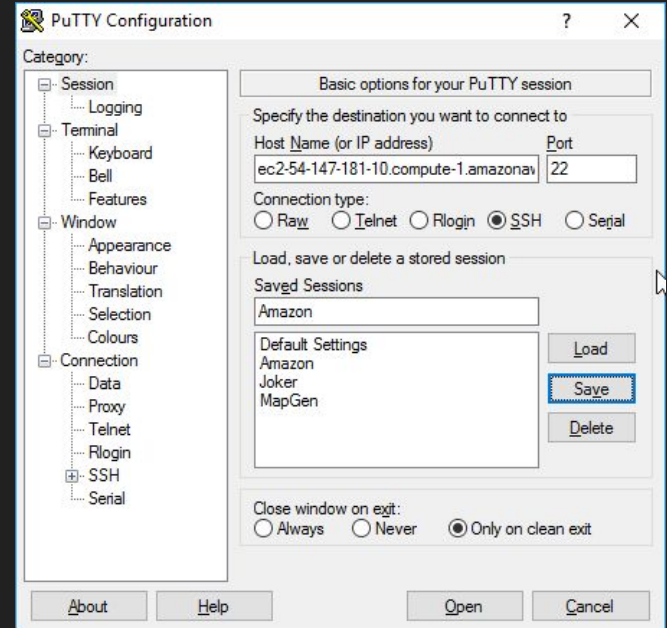
- In command prompt or terminal, with SSH

```
F:\Downloads>ssh -i server_key.pem ubuntu@ec2-54-147-181-10.compute-1.amazonaws.com
```

- Requires the key .pem file to authenticate server, and user “ubuntu” accompanied with the public DNS from your instance

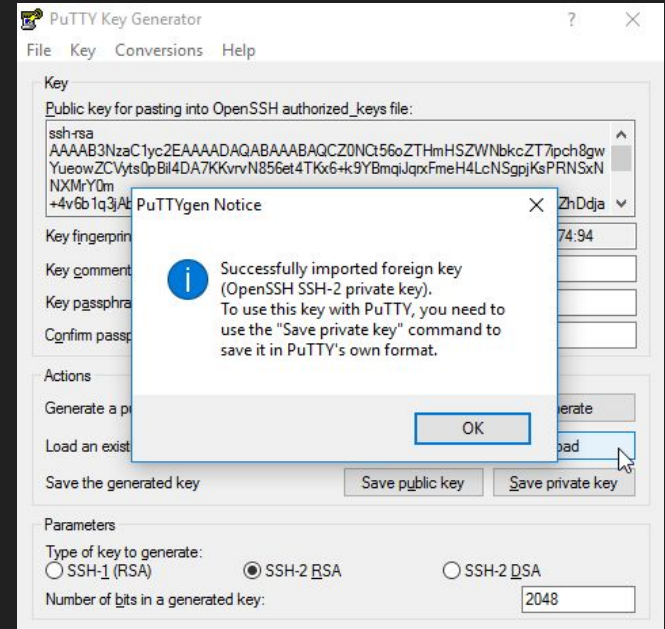
PuTTY (Optional)

- SSH client with many options
- DNS goes in “Host Name” with port 22, type as SSH
- Requires private key from PuTTYgen



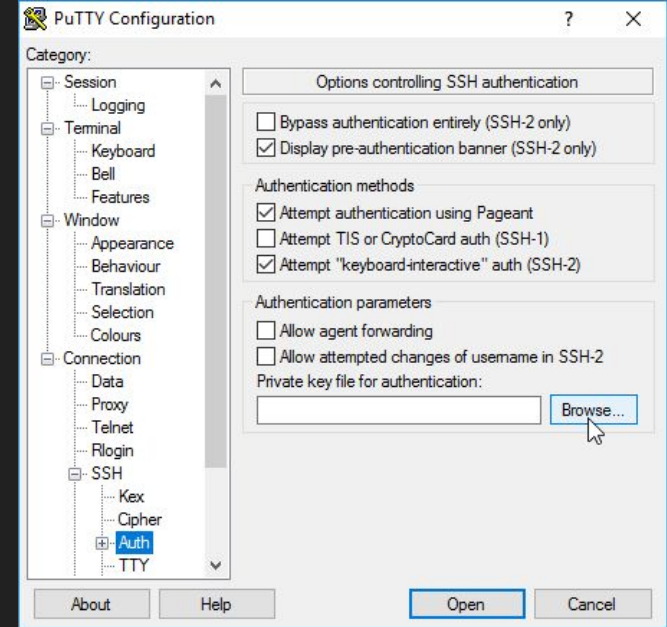
Setup Key-Pair in PuTTY (Optional)

- Launch PuTTYgen (included with PuTTY)
- Click load, switch to All Files, find .pem file
- Save private key as .ppk
- Switch back to PuTTY



Setup Key-Pair in PuTTY (Optional)

- With the Amazon server selected, expand Connection, SSH, and click Auth
- Browse for .ppk file
- Ready to Open



Connecting to an Amazon Server

- After SSH, despite method, the server will open as

```
Welcome to Ubuntu 16.04.2 LTS (GNU/Linux 4.4.0-64-generic x86_64)

* Documentation:  https://help.ubuntu.com
* Management:    https://landscape.canonical.com
* Support:        https://ubuntu.com/advantage

Get cloud support with Ubuntu Advantage Cloud Guest:
http://www.ubuntu.com/business/services/cloud

0 packages can be updated.
0 updates are security updates.

Last login: Sat Apr 15 15:26:48 2017 from 98.166.211.95
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-20-3:~$
```

- First, update all of the server's packages through the package manager

```
ubuntu@ip-172-31-20-3:~$ sudo apt-get update -y _
```


Allow Access Without Key-Pair

- To use the server without the key, we need password authentication
- Add a new user to the server through

```
ubuntu@ip-172-31-20-3:~$ sudo adduser student_
```

- Enter a password, and then enter through the remaining fields
- Next, enable password authentication for the server through

```
ubuntu@ip-172-31-20-3:~$ sudo vim /etc/ssh/sshd_config_
```

- Arrow key down to the following block, hit “i” for insert, change no to yes, hit Escape, type “:wq” and hit enter

```
# Change to no to disable tunnelled clear text passwords
PasswordAuthentication yes_
```

- Finally, restart the SSH service with

```
ubuntu@ip-172-31-20-3:~$ sudo /etc/init.d/ssh restart
[ ok ] Restarting ssh (via systemctl): ssh.service.
```

Setting up a Web Server

- Now, we need to install a web service to access the server through HTTP on port 80
 - Apache works for this purpose (Typically a LAMP stack may be required, with MySQL)

```
ubuntu@ip-172-31-28-3:~$ sudo apt-get install apache2 -y
```

- Once installed, user access is required to add/remove files from the server

```
ubuntu@ip-172-31-28-148:~$ sudo chmod 777 /var/www/html
```

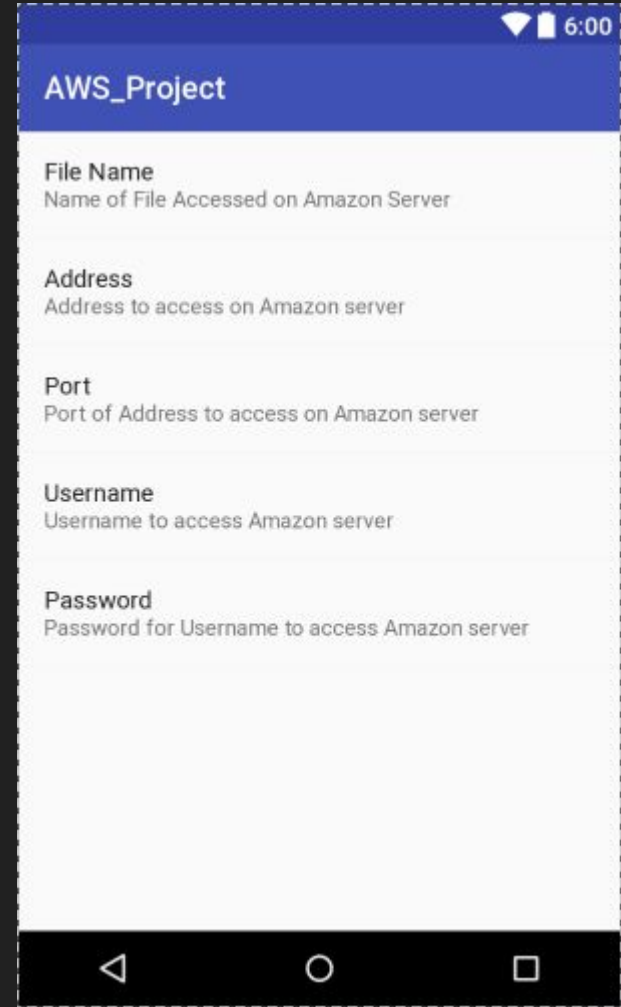
- Setup on the AWS server is finished, on to the Android project...

Basic Android Setup

- Set up a layout with an EditText and two Buttons
 - These buttons should have an onClick attribute to execute code
- Create a Settings activity and preferences XML
 - This settings preferences will allow for the file name, server, port, username, and password to be set by the user.
- Create Two AsyncTasks to download the file from the server and push the file from the server
- Within the main activity handle general setup and create methods
 - These methods will handle the settings, pushing, and pulling from the server.
- Update the Gradle file
 - Compile 'com.jcraft:jsch:0.1.53'

Settings

- Create EditTextPreferences for the File Name, Address, Port, Username, Password
- It is best practice to set a default value for the Address of the server
- Create a Preference Fragment to allow the application to launch the Settings activity if the user is missing a Setting



AsyncTask Setup - Download Json

- Create an AsyncTask that takes in a String and that will return a String
- The constructor should take in a reference to the main activity and the name of the file
- The doInBackground method should set up the connection between the application and the server and retrieve the file.
 - To establish the connection use HttpURLConnection to directly access the url of the file
 - Use a BufferedReader to then read in the data within the file
 - Return a string containing the file contents
- The onPostExecute should call a helper function from the main activity to process this string
- Process is exactly the same as DownloadTask used in the Bikes Project

AsyncTask Setup - Upload Json

- Create an AsyncTask that takes in a String and that will return a Boolean
- The constructor should take in a reference to the main activity, username, password, file name, port number, and JSON object containing from the text box
- The doInBackground should establish an SFTP connection and create a file with the file contents
 - Establish a connection using JSch and Session classes
 - Create a Channel to establish an SFTP connection using ChannelSftp
 - Use the SFTP connection to change directory where the file should be held
 - Create an Input Stream to extract contents of the JSON object to be written
 - Use the SFTP connection to put the contents into a file with the same file name

```
protected Boolean doInBackground(String... params) {
    String server = params[0];
    try{
        JSch jsch = new JSch();
        Session session = jsch.getSession(USERNAME, server, PORT);
        session.setConfig("StrictHostKeyChecking", "no");
        session.setPassword(PASSWORD);
        session.connect(TIMEOUT);
        Channel channel = session.openChannel("sftp");
        ChannelSftp sftp = (ChannelSftp) channel;
        sftp.connect(TIMEOUT);
        sftp.cd("/var/www/html/");

        InputStream stream = new ByteArrayInputStream(FILE_CONTENTS.toString().getBytes(StandardCharsets.UTF_8));
        sftp.put(stream, FILE_NAME+".json");

        stream.close();
        sftp.disconnect();
        sftp.exit();
    }
    catch (Exception e){
        e.printStackTrace();
        return false;
    }
    return true;
}
```

Main Activity Helper Methods

- Helper methods include
 - Creating a JSONObject
 - Extracting a string from an JSONObject
 - Calls to execute AsyncTasks
 - Opening settings if settings are missing
 - Retrieving settings from the SharedPreferences

Creating a JSONObject

- Should grab string contents from the EditText box and convert that into a JSON object.

```
public JSONObject createJSONObject() {  
    JSONObject textToPush = new JSONObject();  
    String enteredText = editText.getText().toString();  
    try {  
        textToPush.put("text", enteredText);  
    } catch (JSONException e) {  
        e.printStackTrace();  
    }  
    return textToPush;  
}
```

Extracting a string from an JSONObject

- From a JSONObject it should grab the relevant contents and put them into the EditText box

```
public void extractStringFromJSON(String stringToExtract) {  
    try {  
        JSONObject extractedString = new JSONObject(stringToExtract);  
        String newText = extractedString.getString("text");  
        editView.setText(newText);  
    } catch (JSONException e) {  
        e.printStackTrace();  
        Toast.makeText(this, "Invalid JSON file", Toast.LENGTH_SHORT).show();  
        Log.e("extractStringFromJSON", "Invalid JSON file");  
    }  
}
```

Calls to execute AsyncTasks

- Method should handle button presses and call relevant AsyncTasks

```
public void pushPull(View view) {
    boolean connected = checkForNetworkConnectivity();
    final String fileTitle = retrieveTitle();
    final String address = retrieveAddress();
    final int port = retrievePort();
    if(fileTitle != null && address != null && port != 0 && connected) {
        switch (view.getId()) {
            case R.id.push_button:
                final JSONObject objToWrite = createJSONObject();
                saveFileToAmazon(objToWrite, fileTitle, address, port);
                break;
            case R.id.pull_button:
                readFileFromAmazon(fileTitle, address);
                break;
        }
    }
}
```

Opening settings if settings are missing

- If the settings are missing then it should alert the user and open the settings activity

```
private void openSettingsIfMissingInfo(String missing) {  
    Toast.makeText(this, missing + " not set in Settings", Toast.LENGTH_LONG).show();  
    Intent myIntent = new Intent(this, SettingsActivity.class);  
    startActivity(myIntent);  
}
```

Retrieving settings from the SharedPreferences

- These methods should grab relevant information from the SharedPreferences

```
private int retrievePort() {  
    int port = Integer.valueOf(myPreference.getString("port", "0"));  
    if(port == 0)  
        openSettingsIfMissingInfo("Port");  
    return port;  
}
```

```
private String retrieveTitle() {  
    String fileTitle = myPreference.getString("filename", "");  
    if(fileTitle.isEmpty()) {  
        openSettingsIfMissingInfo("File Name");  
        return null;  
    }  
    else  
        return fileTitle;  
}
```

```
private String retrieveUsername() {  
    String username = myPreference.getString("username", "");  
    if(username.isEmpty()) {  
        openSettingsIfMissingInfo("Username");  
        return null;  
    }  
    else  
        return username;  
}
```

Workflow

- User enters relevant information within the settings, if not they will be prompted to do so
- User will then enter text into the text box to be stored onto the server
- User presses the push button to save text to the server stored into a JSON file
- User goes to another device updates settings on that device
- User then presses the pull button to grab text stored in the JSON file

Demonstration

Questions?