Lab 1: Java, JSON, AWS S3

Luca Telloli
luca.telloli@upf.edu

Pablo Martín Castagnaro
pablo.castagnaro@upf.edu

Summary

- Part 1: More on Java & Maven
- Part 2: The JSON format & the GSON library
- Part 3: AWS & S3
    - Motivation
    - Access
        - The web console
        - The CLI
        - The Java SDK
    - Services: S3
- Part 4: AWS Academy

Objectives of Lab 1

- Create a fully functional portable program in Java + Maven
- Learn and use the JSON format with an off-the-shelf library
- Get acquainted with AWS, AWS authentication, S3
- Benchmark the performance of a sequential application
- Functional programming: use Java 8 Optional type

More on Java & Maven

Introduction to Maven

- Is a build automation tool primarily used in Java projects
- Maven addresses two aspects of building software:
  - How the software is built
  - Its dependencies
- Configuration done in an XML file: pom.xml
- The project is hosted by the Apache foundation

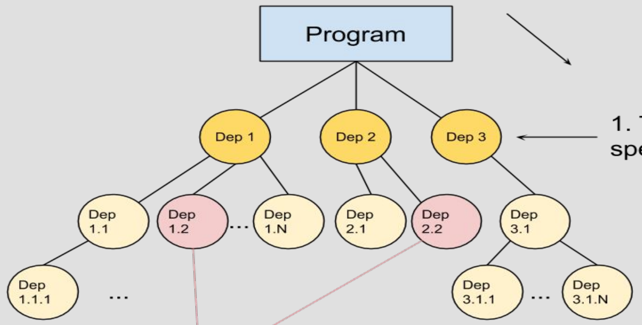
Maven

Manage lifecycle of an artifact

- Handle dependencies and possible conflicts
- Compile and generate portable artifacts

```
<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.11</version>
        <scope>test</scope>
    </dependency>
    <dependency>
    ...
    </dependency>
</dependencies>
```

Dependency tree



1. These are the only ones you specify in Maven's `pom.xml`

All others are dependencies of dependencies that are automagically resolved by Maven

Some dependencies might be in conflict (i.e. different versions of same library). Maven will try to automatically solve the conflicts

Running Java Applications

2 options when running a Java program:

- Retrieve and add all dependencies into the classpath

```
java -cp dep1:dep2:dep3 my.application.executable
```

- Build a jar containing all dependencies (a.k.a Fat Jars or Shadow Jars)

```
java -cp fat.jar my.application.executable
```

maven-shade-plugin

maven plugin to build “fat” jars

`mvn package`: the build will run the shade “goal” on this plugin

Shade will take all dependencies, extract all files, put them in a single jar, together with project classes

```xml
<build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-shade-plugin</artifactId>
        <version>3.2.1</version>
        <configuration>
          <!-- put your configurations here if need be -->
        </configuration>
        <executions>
          <execution>
            <phase>package</phase>
            <goals>
              <goal>shade</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
```

The Optional type

Bits of Functional Programming

Functional programming dislikes modifying variables:

- Can lead to unexpected application state
- Can lead to complicated branching

Functional programming dislikes the null value

- Can't call methods on it
- Can lead to complicated branching

Functional programming promotes function as first-class citizens, like variables

Instead of changing a variable to a new value, functional programming prefers to transform a variable of type T into a variable of type U using a function f: T -> U (T can be equal to U)

Examples:

- f: Int -> Int = x -> x + 1
  - Same as: f(x) = x + 1
- f: String -> Int = s -> Integer.parseInt(s)
  - Same as: f(s) = Integer.parseInt(s)

The Optional type

`Optional<T>` (Java 1.8+) is a generic container for type T, with two possible values:

- `Optional.empty()`, if the value is null
- `Optional` with value of type T, if not null

Equivalent to a “box”, either empty or with something in it: can continue the computation on if it’s not empty, otherwise it stays empty

Optional provides, among others, the following methods:

- `ofNullable(T val)`: constructor. if val is null, return Empty, else Optional(val)
- `isPresent()`: true if “box” contains a value
- `get()`: retrieve val or except
- `orElse(T other)`: retrieve val if present, otherwise return other
- `map(f: T->U): Optional<U>`: apply f on val if present, transforming type T to type U

The Optional type: a simple example

Return the integer value of a String representing an Integer or a default value (-1)

Without Optional

```
String intStr = null;
int val1 = -1;

if(intStr != null) {
      val1 = Integer.parse(intStr)
}
return val1;
```

With Optional

```
String intStr = null;

Integer val1 = Optional
  .ofNullable(intStr) //Optional<String>
  .map(s -> Integer.parse(s))
//Optional<Integer>
  .orElse(-1); //Integer
```

The Optional type: a more complex example

Accessing a nested field within a hierarchy of classes

Without Optional

```
if (userId != null) {
      User u = phoneBook.getUser(userId);
      if (u != null) {
            Address a = u.getAddress(u);
            if (a != null) {
                  City c = a.getCity()
                  return city;
            }
      }
}
if (city != null) {
      // do something
}
```

With Optional

```
Optional<City> = Optional
      .ofNullable(userId) // Optional<UserId>
      .flatMap(id -> Optional
            .ofNullable(phoneBook.getUser(id))
            //Optional<User>
      .flatMap(u -> Optional
            .ofNullable(u.getAddress(u))
      .map(a -> a.getCity())
      ;

if(city.isPresent()) {
      // do something
}
```

What’s JSON?

What’s JSON?

- JSON (from JavaScript Object Notation) is a widely used lightweight format used for data representation and exchange on the Internet
- Main advantages are:
  - it’s human readable
  - JSON serialisation produces text
  - aims to be self-describing (like XML)
  - it’s compact (unlike XML)
  - it’s easy to read, write and manipulate

What’s JSON?

- JSON has 3 type of structures:
  - Objects (contained within curly brackets {})
  - Arrays (contained within square brackets [])
  - Simple values
- Simple values can be of 4 types: strings, numerical values, booleans, null
- In an object, each value has a name, in an array it doesn’t
- Sequence of values are separated by comma

Examples of valid JSON

`{}`	Empty JSON object
`[]`	Empty JSON array
`false`	The boolean value `false`
`[1, 2, 3]`	Json array of numbers
`{“age”: 12}`	Object with a single numerical field

`{`
 `“name”: “tom”,`
 `“friends”: [“jerry”, “jack”]`
`}`
	Object with 2 fields, one is a string, the other is an array of strings

`[`
  `{“name”: “ted”, “IBAN”: 123},`
  `{“name”: “bob”, “IBAN”: 456}`
`]`
	Array of objects: each object contains one numerical field (“IBAN”) and one string field (“name”), that is: the two objects have the same structure.

Examples of invalid JSON

`{1}`	The numerical value is missing a name!
`[“age”: 11]`	Values don’t have a name in arrays!
`false, true`	An array of boolean would be missing the square brackets
`[1, 2, 3, “four”]`	This is actually **valid**, but you should think twice before writing it!
`{“age”: 12`	Missing closing bracket

`{“name”: “tom”`
` “friends”: [“jerry”, “jack”]}`
	The values are not separated by comma

`{“name”: “jerry”,`
`  true}`
	The boolean value doesn’t have a name

Parsing JSON with GSON

Parsing JSON

- There are plenty of libraries used to parse JSON for any language
- In Java, very common choices are:
  - Jackson
  - GSON
  - JSON simple
- In this Lab, we'll use GSON
- Gson provides two way to parse JSON:
  - Directly into a java class (POJO)
  - Abstractly into a JSON object

Parse JSON...	...using `Gson()`	...using `JsonParser()`
When?	Parsing structured objects directly into Java classes (all objects have same structure)	Unknown Objects or object that do not map directly to Java classes
Sample JSON	`var jsonStr = “{\”name\”: \”tom\”}”`	
Deserialization Example	`new Gson() .fromJson(jsonStr, User.class)`	`JsonParser. parseString(jsonStr)`
Returns	An instance of the target class	An instance of `JsonElement`
Expectations	The target class exists. JSON string and target class have same structure (including nested objects)	None, tries to parse the string into JSON directly
Type casting	Automatic conversion to target class member type	“manual” with JsonElement class methods to expected types
Retrieve fields	Use the target class accessors (getters)	each field has to be checked for existence and retrieved manually

Parsing with GSON using JsonParser

<table>
<tr>
<td>Java</td>
<td>

```java
class User {
  String name;
  int age;
  String phoneNumber;
}
```

</td>
<td rowspan="2">

```java
JsonElement je = JsonParser.parseString(jsonStr);
JsonObject jo = je.getAsJsonObject();
String user = null;
if (jo.has("user")) {
  JsonObject userObj = jo.get("user")
     .getAsJsonObject(); // cast method
  if (userObj.has("name")) {
    user = userObj.get("name")
     .getAsString(); // cast method
  }
}
… // parse for age and phoneNumber
return new User(user, age, phoneNumber);
```

</td>
</tr>
<tr>
<td>JSON</td>
<td>

```json
{
  "user": {
    "name": "john",
    "age": 22
  },
  "Contacts": {
    "Email": "john@gmail.com",
     "Mobile": "777-654-321"
  }
}
```

</td>
</tr>
</table>

Intro to AWS

What is AWS?

From AWS documentation:

> *Amazon Web Services (AWS) provides on-demand computing resources and services in the cloud, with pay-as-you-go pricing.*

Also known as:

> *IAAS: Infrastructure As A Service*

A bit of history

Project born in 2002, officially started in 2006 with only 3 services: Storage (S3), Computing (EC2), Queues (SQS)

Today it comprises now more than 90 different services spanning multiple areas of computing

In 2015 AWS was reported as profitable!

Notable customers: NASA, Netflix

Instagram migrated to Facebook infrastructure shortly after acquisition

What is AWS?

Integrated constellation of services that span many categories and purposes

- Computing resources (EC2)
- Storage (S3)
- Database storage (RDS, Redshift, …)
- NoSQL Storage (DynamoDB, …)
- Analytics & Data processing (EMR, ElasticSearch, Athena, …)
- Security and Policy (IAM, …)
- Networking (VPC, Route53, CloudFront, …)
- Machine Learning (Lex, Forecast, …)
- More categories: Robotics, Blockchain, Satellite, Media Services, …
- More goodies added every week!

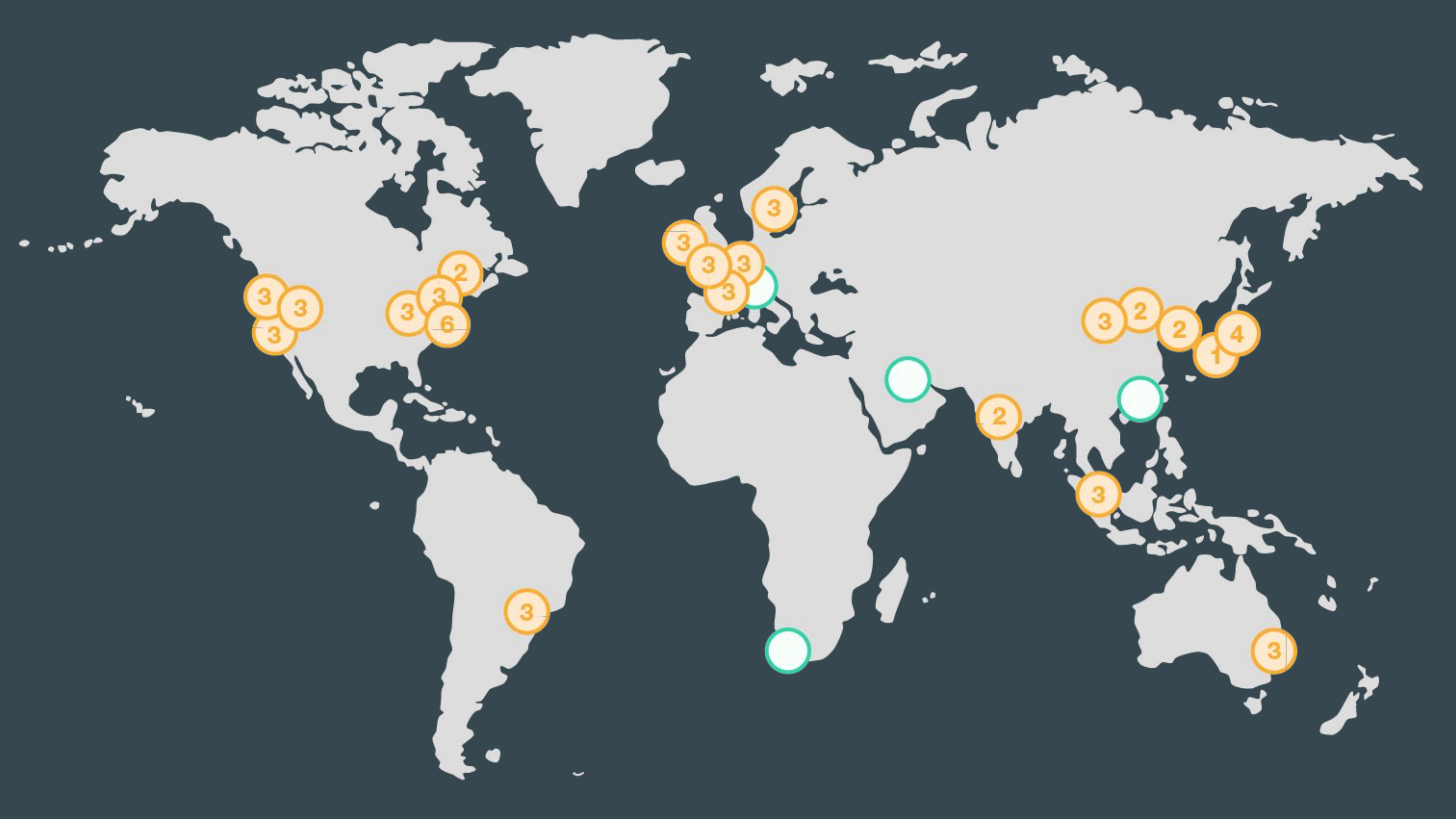
Regions and Availability Zones

Amazon services are hosted in multiple locations world-wide.

These locations are composed of Regions and Availability Zones. Each Region is a separate geographic area. Each Region has multiple, isolated locations known as Availability Zones.

Amazon EC2 provides you the ability to place resources, such as instances, and data in multiple locations.

Resources aren't replicated across regions (unless you do so specifically).


3
3
3
2
3
3
6
3
3
3
3
3
3
3
2
2
4
1
2
3
3

Benefits

- **On demand**: Pay-as-you-go (each service billed differently)
- **Scaling**: easily go up and down horizontally or vertically from 0 to massive
- **Faster time-to-market**: little to no set-up needed, especially for common use cases

Elastic

can handle “any” load

scales up-and-down

Access

To access AWS services you need to authenticate

Username and password are called *aws_access_key_id* and *aws_secret_access_key*

For AWS Educate: you'll need also a *aws_session_token*

Access

The **AWS console** is a web interface

The **AWS CLI** is a command line interface to most AWS services

The **AWS SDKs** provide embeddable AWS parts for most services in popular languages

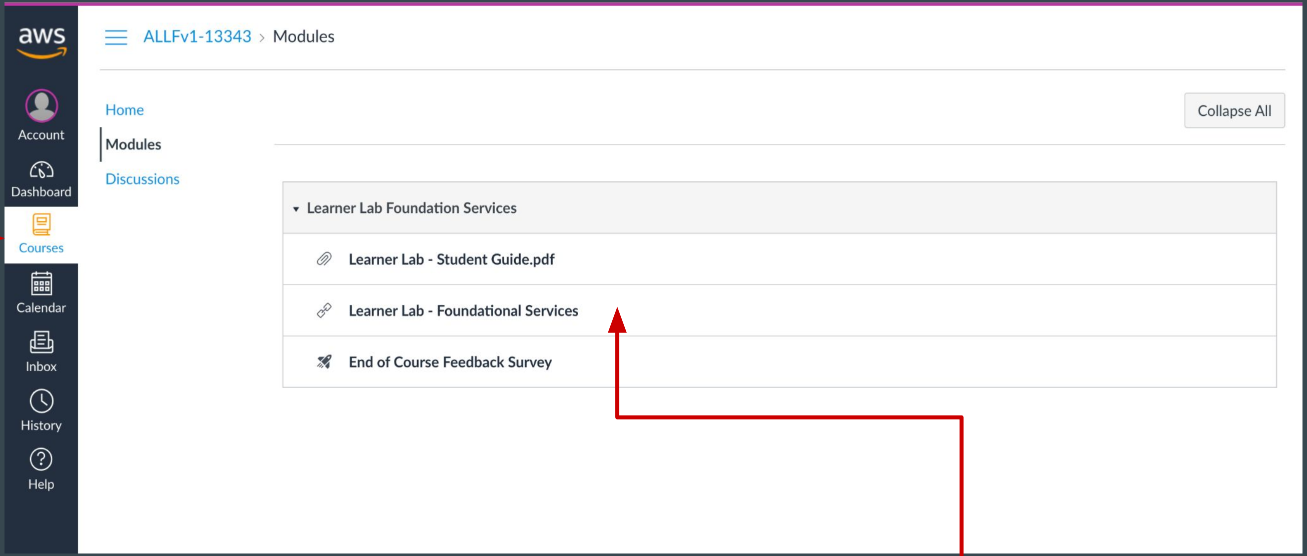
aws training and certification

AWS Academy

Empowering higher education institutions to prepare students for industry-recognized certifications and careers in the cloud

Access to AWS Academy and its resources

0. Check your email and register in AWS Academy with the provided link



ALLFv1-13343 › Modules

Home
Modules
Discussions

▾ Learner Lab Foundation Services
- Learner Lab - Student Guide.pdf
- Learner Lab - Foundational Services
- End of Course Feedback Survey

1. Click on Courses to access the AWS resources

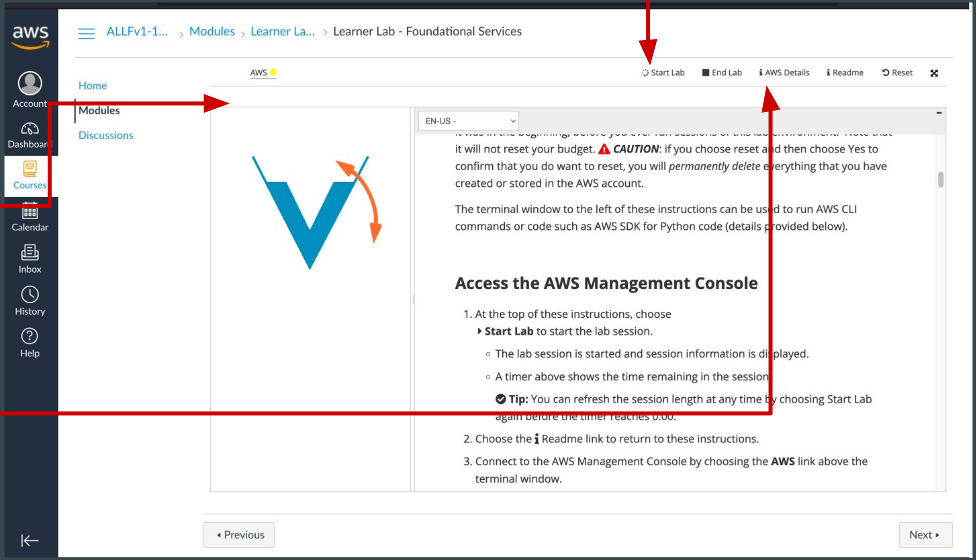
2. Click on Academy Learner Lab and access terms and conditions

Access to AWS Academy and its resources

3. Click on Start Lab to access the AWS resources

4. Wait until your lab is ready

5. Click AWS Details to access your account credentials



it will not reset your budget. ⚠ *CAUTION*: if you choose reset and then choose Yes to confirm that you do want to reset, you will *permanently delete* everything that you have created or stored in the AWS account.

The terminal window to the left of these instructions can be used to run AWS CLI commands or code such as AWS SDK for Python code (details provided below).

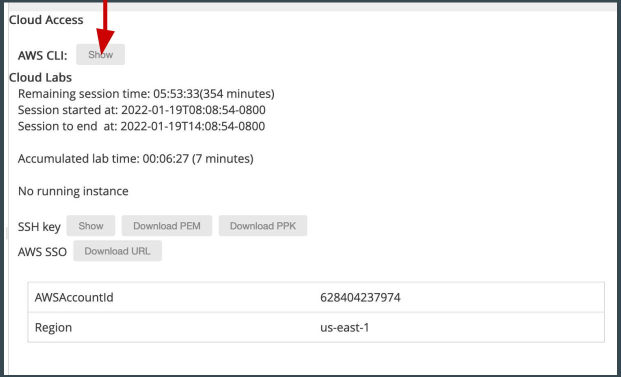
Access the AWS Management Console

1. At the top of these instructions, choose
   ▸ **Start Lab** to start the lab session.
   - The lab session is started and session information is displayed.
   - A timer above shows the time remaining in the session.
     ✔ **Tip:** You can refresh the session length at any time by choosing Start Lab again before the timer reaches 0:00.
2. Choose the ℹ Readme link to return to these instructions.
3. Connect to the AWS Management Console by choosing the **AWS** link above the terminal window.

What does AWS Academy offer?

- "Student" version of the AWS console
- Pre-loaded with some expendable credit ($100)
- The CLI credentials expire every 3 hours

6. Click on Show to get CLI Credentials



Cloud Access

AWS CLI: Show

Cloud Labs

Remaining session time: 05:53:33(354 minutes)
Session started at: 2022-01-19T08:08:54-0800
Session to end at: 2022-01-19T14:08:54-0800

Accumulated lab time: 00:06:27 (7 minutes)

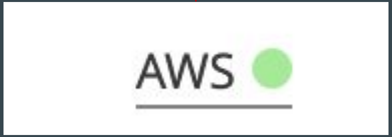
No running instance

SSH key Show Download PEM Download PPK

AWS SSO Download URL

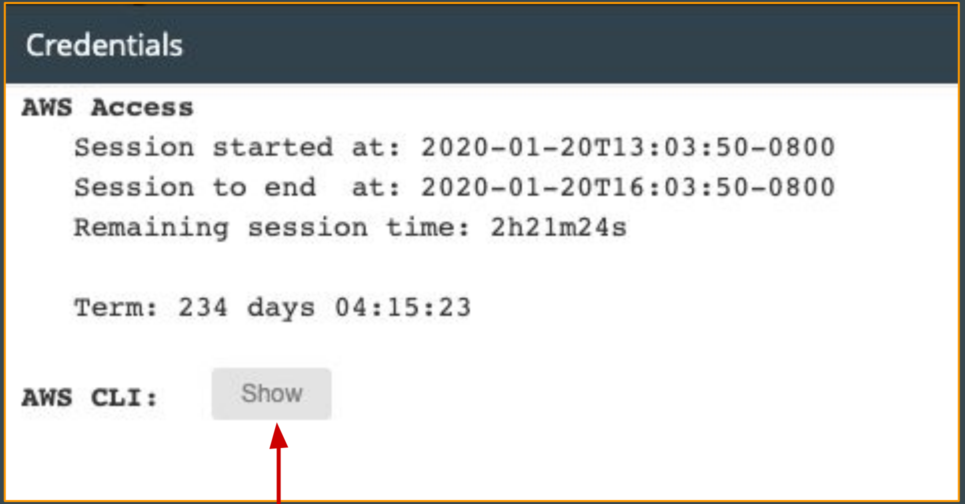
AWSAccountId	628404237974
Region	us-east-1

7. Click on AWS Console to open the Web Console



Account Details

- Provides the credentials and the token you'll need to copy into `~/.aws/credentials` on your workstation

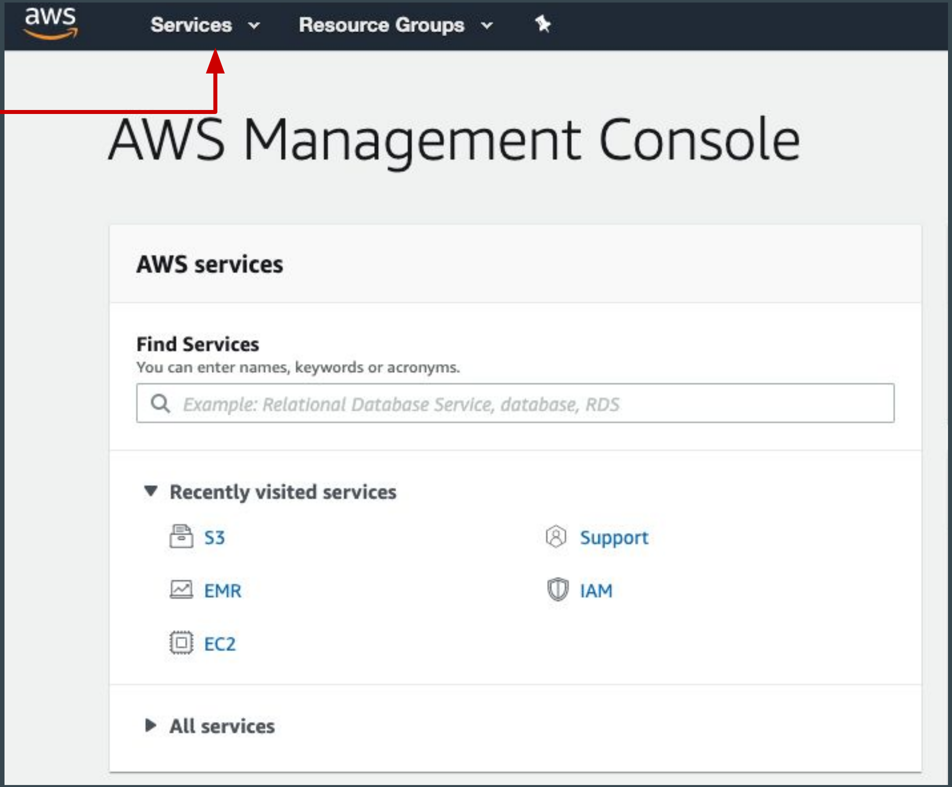


```
Credentials
AWS Access
    Session started at: 2020-01-20T13:03:50-0800
    Session to end  at: 2020-01-20T16:03:50-0800
    Remaining session time: 2h21m24s

    Term: 234 days 04:15:23

AWS CLI:
```

1. Click on Show to get your credentials
2. Copy them to the expected file
3. Try a CLI command from your shell, for instance: `aws s3 ls`

AWS Console

- Provides access to all available services (including S3)



AWS Management Console

AWS services

Find Services
You can enter names, keywords or acronyms.

Example: Relational Database Service, database, RDS

▼ **Recently visited services**

S3
Support
EMR
IAM
EC2

▶ **All services**

S3: Simple Storage Service

What is AWS S3?

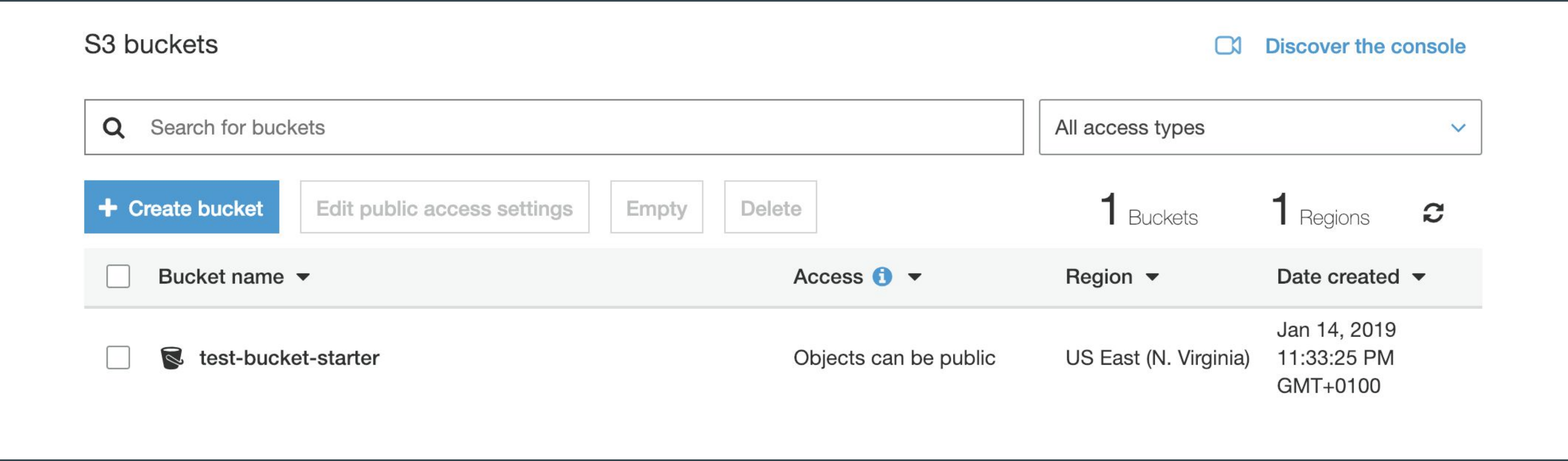
File System in the cloud, accessible with HTTP interface

HTTP Requests are authenticated

Bucket

equivalent to a disk in a “traditional” filesystem

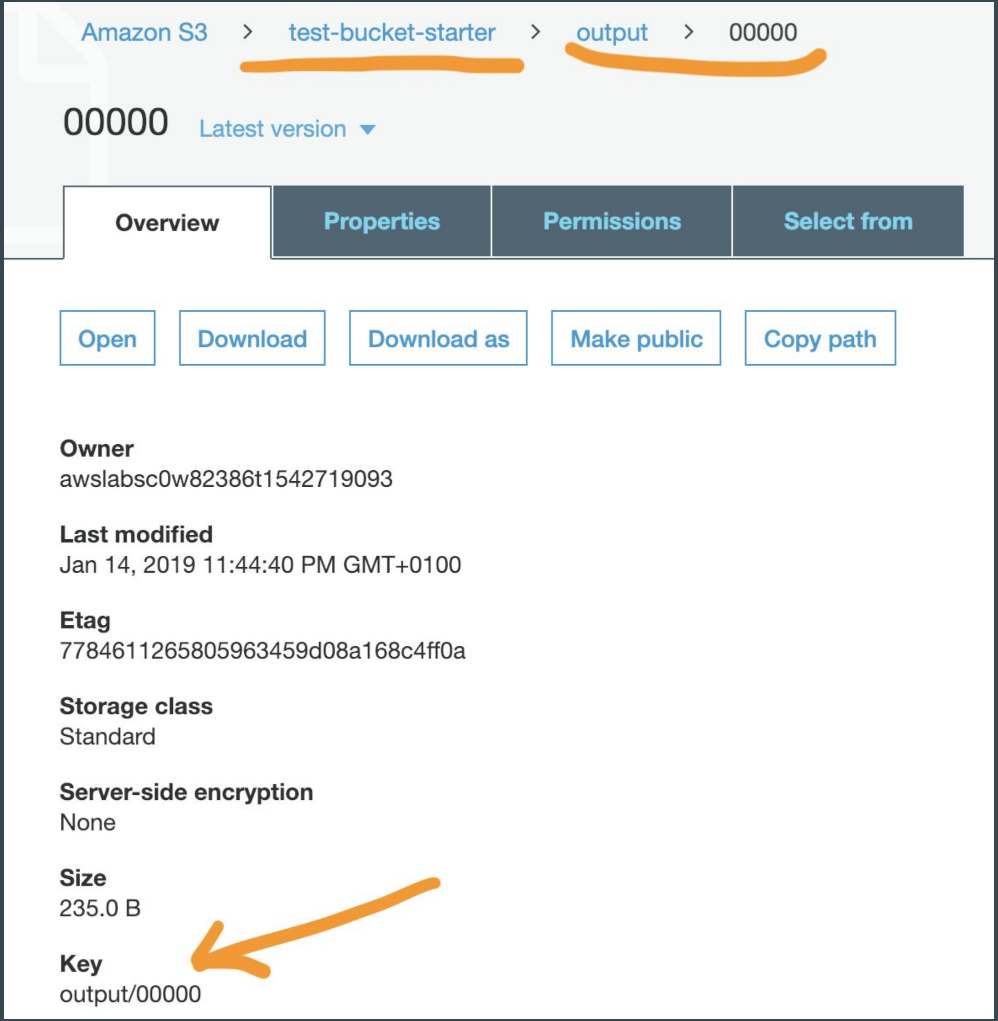
Amazon S3 has a global namespace. (i.e. No two S3 buckets can have the same name.)



	Bucket name	Access	Region	Date created
☐	test-bucket-starter	Objects can be public	US East (N. Virginia)	Jan 14, 2019 11:33:25 PM GMT+0100

Key

equivalent to a file path in a “traditional” file system



Owner
awslabsc0w82386t1542719093

Last modified
Jan 14, 2019 11:44:40 PM GMT+0100

Etag
7784611265805963459d08a168c4ff0a

Storage class
Standard

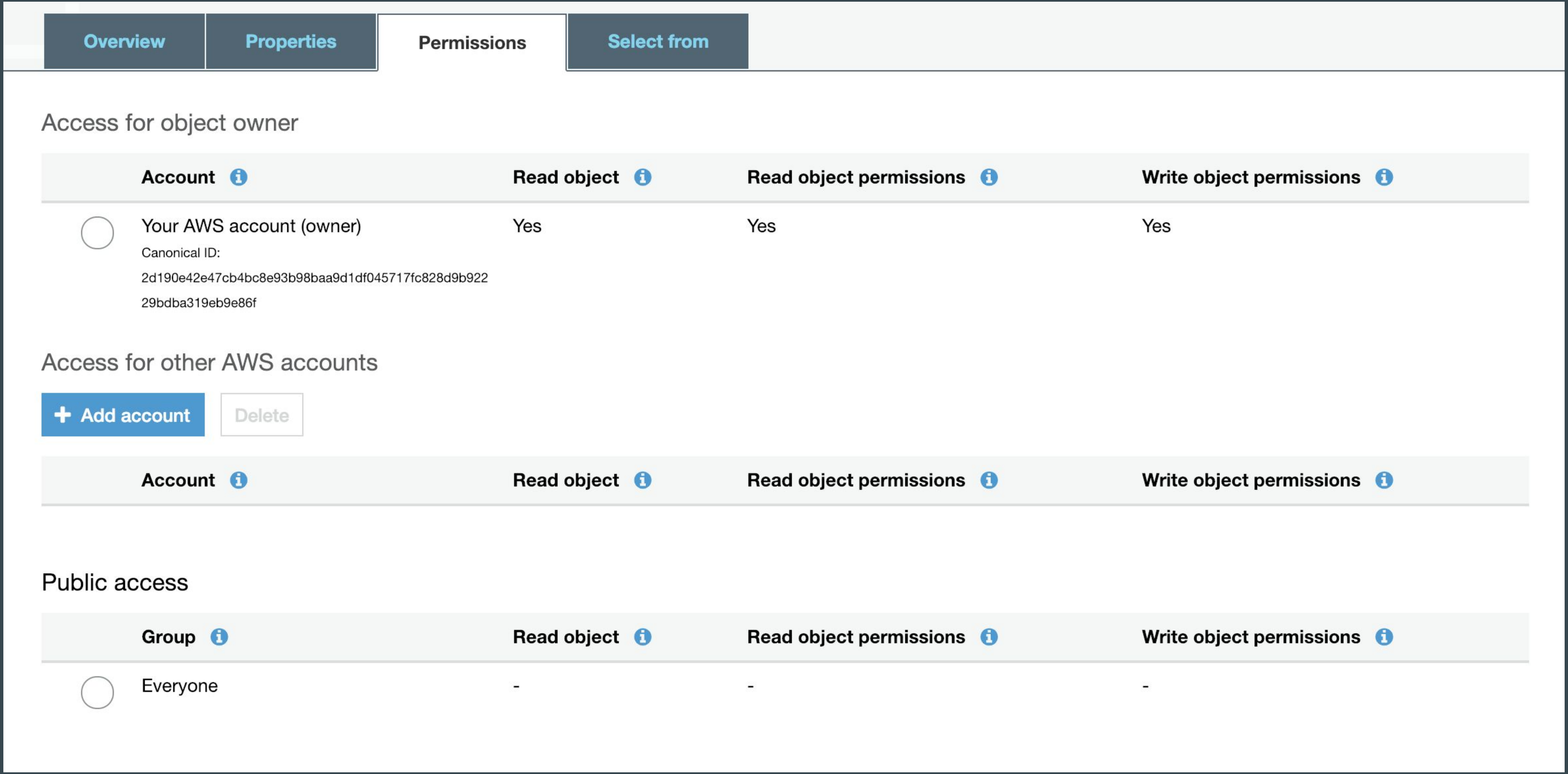
Server-side encryption
None

Size
235.0 B

Key
output/00000

Permissions

Access can be: private, for a specific account or list of accounts, or public



Access for object owner

	Account	Read object	Read object permissions	Write object permissions
○	Your AWS account (owner) Canonical ID: 2d190e42e47cb4bc8e93b98baa9d1df045717fc828d9b922 29bdba319eb9e86f	Yes	Yes	Yes

Access for other AWS accounts

	Account	Read object	Read object permissions	Write object permissions

Public access

	Group	Read object	Read object permissions	Write object permissions
○	Everyone	-	-	-

S3 using the CLI

S3 through CLI

```
aws s3 ls s3://bucket-name
```

List content of bucket `bucket-name`

```
aws s3 cp s3://bucket-name/dir1/file1 /tmp/
```

Copy remote key `/dir1/file1` from bucket `bucket-name` into local directory `/tmp/`.
cp from local to s3 is also a valid operation; in that case you should provide the full key name

```
aws s3 cp --recursive s3://bucket/dir2/ /tmp/dir2.local
```

Copy all keys under `/dir2` from bucket `bucket-name` into local directory `/tmp/dir2.local`

```
aws s3api create-bucket --bucket test-bucket
```

Create bucket `test-bucket` in the default account and region

```
aws s3api list-buckets
```

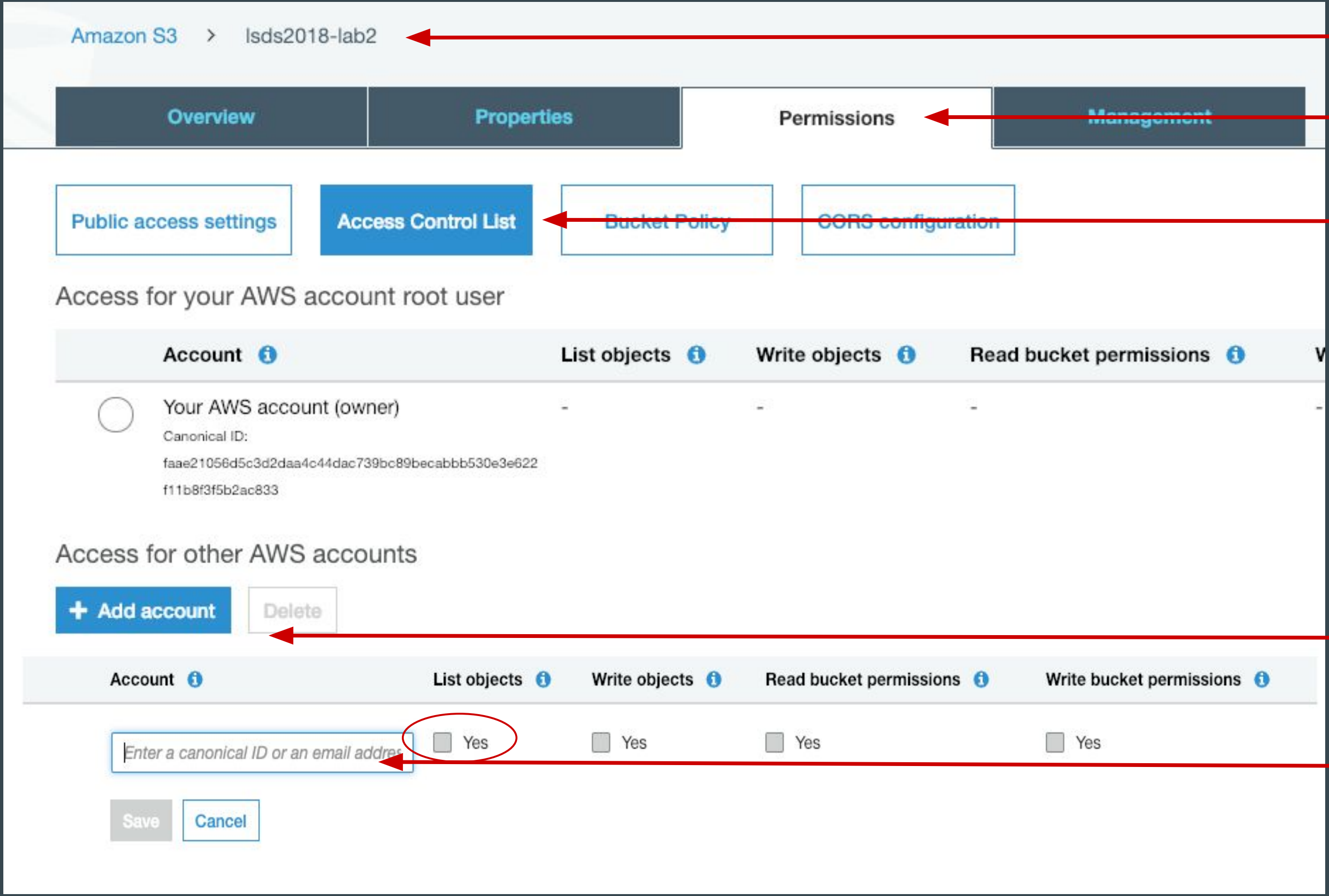
List all the created buckets in the default account. **Additionally, displays the canonical id.**

Giving custom access to a bucket

- To give access to another account for a bucket, you need to know the canonical ID of that account. It's a long alphanumeric string.
    - Hint! The `list-buckets` operations provides you with such information
- As usual, you can perform the operation through the CLI, or the web interface, or programmatically
- From the cli, the command is something like:

```
aws s3api put-bucket-acl --bucket <bucket-name> --grant-read id=<canonical-id>
```

- Warning! This overrides any existing policy! See: https://docs.aws.amazon.com/cli/latest/reference/s3api/put-object-acl.html



Access for your AWS account root user

Account	List objects	Write objects	Read bucket permissions
Your AWS account (owner) Canonical ID: faae21056d5c3d2daa4c44dac739bc89becabbb530e3e622f11b8f3f5b2ac833	-	-	-

Access for other AWS accounts

Account	List objects	Write objects	Read bucket permissions	Write bucket permissions
Enter a canonical ID or an email addres	Yes	Yes	Yes	Yes

1. Select bucket
2. Select *Permissions*
3. Select *ACL*
4. Select *Add Account*
5. Insert the Canonical ID and select *List Objects*