Amazon EC2 + EWS Lambda

Slide 1: Where Can You Store Data?

Slide Content (as text):

- **Instance Store**: Temporary, high-speed storage tied to the instance lifecycle
- **EFS (Elastic File System)** Support: Shared file storage
- **EBS (Elastic Block Storage)**: Persistent block-level storage
- **S3**: Large data set storage or EC2 backups

Expanded Explanation:

1. **Instance Store**
   - This is storage physically attached to the host machine running your EC2 instance.
   - It is extremely fast and ideal for temporary or ephemeral data.
   - **Important Note**: If you stop or terminate the EC2 instance, data on the instance store is lost.
2. **EFS (Elastic File System)**
   - A fully managed, shared file system that can be mounted on multiple EC2 instances simultaneously.
   - Ideal for scenarios where you need a common file system for multiple servers or services.
3. **EBS (Elastic Block Storage)**
   - Persistent storage volumes that can be attached to an EC2 instance.
   - Data remains available even if you stop or reboot the instance (but not if you terminate and delete the volume, unless you configure otherwise).
   - Great for typical server use-cases like a root volume or attached volumes for databases.
4. **S3 (Simple Storage Service)**
   - Object-based storage for large amounts of data (files, images, backups, static web content, etc.).
   - Highly durable and cost-effective.
   - Typically used for backups, hosting static sites, or data lakes.

Slide 2 (labeled as “6” in the screenshot): Amazon EC2 & Lambda

Slide Content (as text):

- **DS 4300**
- **Amazon EC2 & Lambda**
- **Mark Fontenot, PhD**
- **Northeastern University**

Expanded Explanation:

- This slide introduces the topic of Amazon EC2 (Elastic Compute Cloud) and AWS Lambda.
- **EC2** is Amazon's Infrastructure as a Service (IaaS) offering, allowing you to rent virtual machines ("instances") in the cloud.
- **AWS Lambda** is a serverless computing service that lets you run code without provisioning or managing servers.

Slide 3 (labeled "7" in the screenshot): Let's Spin Up an EC2 Instance

Slide Content (as text):

- "Let's Spin Up an EC2 Instance" (title)

Expanded Explanation:

- This slide indicates the beginning of a tutorial or demonstration on how to create ("spin up") a new EC2 instance.
- Spinning up an instance involves choosing an AMI (Amazon Machine Image), selecting instance type, configuring network settings, etc.
- The upcoming slides likely detail the step-by-step process.

Slide 4 (labeled "8"): Let's Spin Up an EC2 Instance

Slide Content (as text, partially visible):

- Screenshots from the AWS Console showing steps to create a new EC2 instance.

Expanded Explanation:

- **Step 1: Choose an AMI** – For example, "Ubuntu Server 20.04 LTS".
- **Step 2: Choose an Instance Type** – e.g., t2.micro for a small, free-tier-eligible instance.
- **Step 3: Configure Instance Details** – VPC settings, subnets, etc.
- **Step 4: Add Storage** – EBS volume size, etc.
- **Step 5: Add Tags** – Optional metadata for better organization.
- **Step 6: Configure Security Group** – Control inbound/outbound traffic with firewall rules.

- **Step 7: Review & Launch** – Launch the instance and create or select a key pair for SSH access.

Slide 5 (labeled "9"): Let's Spin Up an EC2 Instance

Slide Content (as text):

- Another screenshot continuing the setup steps.

Expanded Explanation:

- Likely continues detailing the creation process, focusing on security group configuration or final review.
- **Security Group** best practice:
    - Open only the ports you need.
    - Typically, port 22 for SSH (for Linux) or RDP (for Windows) is required.
    - If you plan to run a web server, open port 80 (HTTP) or 443 (HTTPS).

Slide 6 (labeled "10"): Ubuntu VM Commands

Slide Content (as text):

- **Initial user is ubuntu**
- Access super user commands with sudo
- Make sure apt-get is up to date
- Update the packages installed
- sudo apt update; sudo apt upgrade

Expanded Explanation:

- Once you SSH into the instance, you'll be logged in as the ubuntu user by default (on an Ubuntu AMI).
- To perform administrative tasks (install packages, change configurations), you'll use sudo.
- **Key commands**:
    - sudo apt update – Refreshes the package lists for upgrades/new packages.
    - sudo apt upgrade – Installs available updates.
- This ensures your instance is secure and has the latest security patches.

Slide 7 (labeled "11"): MiniConda on EC2

Slide Content (as text):

- Make sure you're logged in to your EC2 instance.
- Let's install MiniConda:
  - Example command: wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
  - Then run bash Miniconda3-latest-Linux-x86_64.sh.

Expanded Explanation:

- **MiniConda** is a minimal installer for Conda, a package manager that helps you install Python and data-science-related libraries easily.
- After installation, you typically need to source your ~/.bashrc or restart your shell so that the conda command is recognized.
- You can then create or manage virtual environments on your EC2 instance.

Slide 8 (labeled "12"): Installing & Using Streamlit

Slide Content (as text):

- Log out of your EC2 instance and log back in.
- streamlit is now available!
- In your environment, install Streamlit.
- Make a test Streamlit file.
- Basic usage: streamlit run my_app.py
- cd web

Expanded Explanation:

- After installing Streamlit with pip or conda install streamlit, you can run it by calling streamlit run <filename.py>.
- You might log out and back in to refresh your PATH or environment variables.
- Creating a quick test file (e.g., my_app.py) helps confirm that Streamlit is installed correctly.

Slide 9 (labeled "13"): Basic Streamlit App

Slide Content (as text):

```
import streamlit as st

st.title("My Streamlit App")
st.write("Hello, world!")
```

- Save as test.py
- nano test.py (to edit)
- streamlit run test.py

Expanded Explanation:

- **Streamlit** is a Python library for building simple web apps quickly, especially for data visualization and data science demos.
- A minimal example includes a title and a simple “Hello, world!” text.
- st.title() sets the main heading, while st.write() can display text, data frames, or other content.

Slide 10 (labeled “14”): Opening Up The Streamlit Port

Slide Content (as text):

- Screenshot of AWS security group settings.
- Possibly showing how to open the port used by Streamlit (default is 8501).

Expanded Explanation:

- By default, Streamlit runs on localhost:8501.
- To make it accessible externally, you can either:
    1. Use an SSH tunnel (not covered here), or
    2. Open port 8501 in your EC2 security group inbound rules so you can access http://<public-ip>:8501.
- **Security Group**:
    - Add a rule:
        - Type: Custom TCP
        - Port Range: 8501
        - Source: Your IP (or 0.0.0.0/0 if you want it accessible to everyone—less secure)

Slide 11 (labeled “15”): In a Browser

Slide Content (as text):

- Screenshot of the Streamlit app running in the browser, with a title "Welcome to my Streamlit App."

Expanded Explanation:

- After you've opened the port and started Streamlit on the server (streamlit run test.py), navigate to http://<EC2-Public-IP>:8501.
- You should see your basic Streamlit app.
- This confirms your setup is correct: instance is running, port is open, and the app is served.

Slide 12 (labeled "16"): AWS Lambda

(This slide seems to be a title or transition slide.)

Slide Content (as text):

- "AWS Lambda" (title)

Expanded Explanation:

- Now shifting the topic from EC2 to AWS Lambda.
- Lambda is a *serverless* compute service where you only pay for the time your code is running.
- You don't need to manage any infrastructure; AWS handles the underlying servers.

Slide 13 (labeled "17"): Lambdas

Slide Content (as text):

- Lambdas provide **serverless computing**.
- Automatically run code in response to events.
- Worry about the code, not the servers.
- You only pay for execution time, not idle compute time (different from EC2).

Expanded Explanation:

- **Serverless** means you don't provision or manage servers. Instead, you write functions that AWS executes on-demand.
- **Events** can be triggered by S3 uploads, API Gateway calls, DynamoDB updates, scheduled CloudWatch events, etc.
- **Pay-as-you-go** model:
    - Billed by the number of requests and the time your code runs.

- No charges when your code isn't running.

Slide 14 (labeled "18"): Lambda Features

Slide Content (as text):

- **Event-driven execution** – can be triggered by many different events in AWS.
- Supports a large number of runtimes: Python, Java, Node.js, etc.
- **Highly integrated** with other AWS services.
- **Extremely scalable** and scales automatically.

Expanded Explanation:

- Because Lambda is event-driven, you don't have to keep an instance running.
- Official runtimes include Node.js, Python, Ruby, Java, Go, .NET, etc. You can also use custom runtimes.
- Integrations with S3, SNS, SQS, DynamoDB, API Gateway, and more let you build sophisticated serverless architectures.
- Lambda automatically handles concurrency. If multiple events occur at once, Lambda will spin up additional executions.

Slide 15 (labeled "19"): How it Works

Slide Content (as text):

1. Add/Upload your code through AWS Management Console (or via CLI, or CI/CD).
2. Configure event source(s).
3. Watch your Lambda run when one of the event sources triggers it.

Expanded Explanation:

- **Step 1**: You can write your function code in the console's built-in editor, upload a ZIP, or reference an S3 bucket.
- **Step 2**: An event source (e.g., S3 bucket event, CloudWatch event) is configured to call your function.
- **Step 3**: Once the event occurs, AWS Lambda automatically executes your function.

Slide 16 (labeled "20"): Let's Make One

(Likely an introduction to a hands-on Lambda creation demo.)

Slide Content (as text):

- "Let's Make One" (title)

Expanded Explanation:

- You will walk through creating a new Lambda function in the AWS Console.
- The following slides presumably show each step in the process.

Slide 17 (labeled "21"): Making a Lambda

Slide Content (as text):

- Screenshot from the AWS Lambda console, showing the "Create function" button or the initial steps in creation.

Expanded Explanation:

- On the Lambda main page, click **Create function**.
- You'll be prompted to choose between:
    1. **Author from scratch**
    2. **Use a blueprint**
    3. **Use container image**
    4. **Browse serverless app repository**
- Typically, you'll "Author from scratch" to get started with a basic function.

Slide 18 (labeled "22"): Creating a Function

Slide Content (as text):

- Possibly shows form fields:
    - Function name
    - Runtime (Python, Node.js, etc.)
    - Permissions/Role

Expanded Explanation:

- **Function name**: Choose something descriptive, e.g., my-first-lambda.
- **Runtime**: Select the language environment (e.g., Python 3.9).
- **Permissions**: You can create or select an IAM role that gives Lambda the permissions it needs (for example, to log to CloudWatch).

Slide 19 (labeled "23"): Sample Code

Slide Content (as text):

```
def lambda_handler(event, context):
   # Your code here
   return {
      'statusCode': 200,
      'body': 'Hello from Lambda!'
   }
```

Expanded Explanation:

- This is a simple Python Lambda handler.
- The lambda_handler function is the entry point for the Lambda.
- **Parameters**:
  - event: data passed in by the triggering service or event.
  - context: runtime information about the function execution (e.g., request ID, memory limits, etc.).
- The returned dictionary typically includes a statusCode and body (especially if used behind an API Gateway).

Slide 20 (labeled "24"): Edit the code, Deploy the code!

Slide Content (as text):

- A screenshot showing the Lambda code editor in the console, with a "Deploy" button.

Expanded Explanation:

- After writing or modifying your code in the console, you must **Deploy** it to save the changes and make them live.
- You can test the code by configuring a test event or by hooking up an actual event source.

Slide 21 (labeled "25"): Test It

Slide Content (as text):

- A screenshot showing the "Test" button and a sample event configuration.

Expanded Explanation:

- You can create a **test event** in the console that mimics the payload from a real event source (e.g., S3, API Gateway).
- Once you click **Test**, Lambda will run your function with the provided event.
- The console will display the result, including logs and the return value.

Final Notes and Summary

By following these slides and the expanded explanations, you have a roadmap for:

1. **Creating and managing an EC2 instance**:
   - Choosing an AMI (like Ubuntu).
   - Updating and installing necessary packages (including MiniConda).
   - Deploying a basic web application (e.g., Streamlit).
   - Opening ports in the security group for external access.
2. **Introduction to AWS Lambda**:
   - Understanding serverless concepts.
   - Creating a basic Lambda function in Python.
   - Event-driven execution and the pay-per-execution model.
   - Testing your Lambda function with custom events.

   These are foundational AWS concepts—EC2 for on-demand virtual machine instances and Lambda for serverless functions. With these, you can build a wide range of applications in the AWS ecosystem.

End of Expanded Lecture Notes