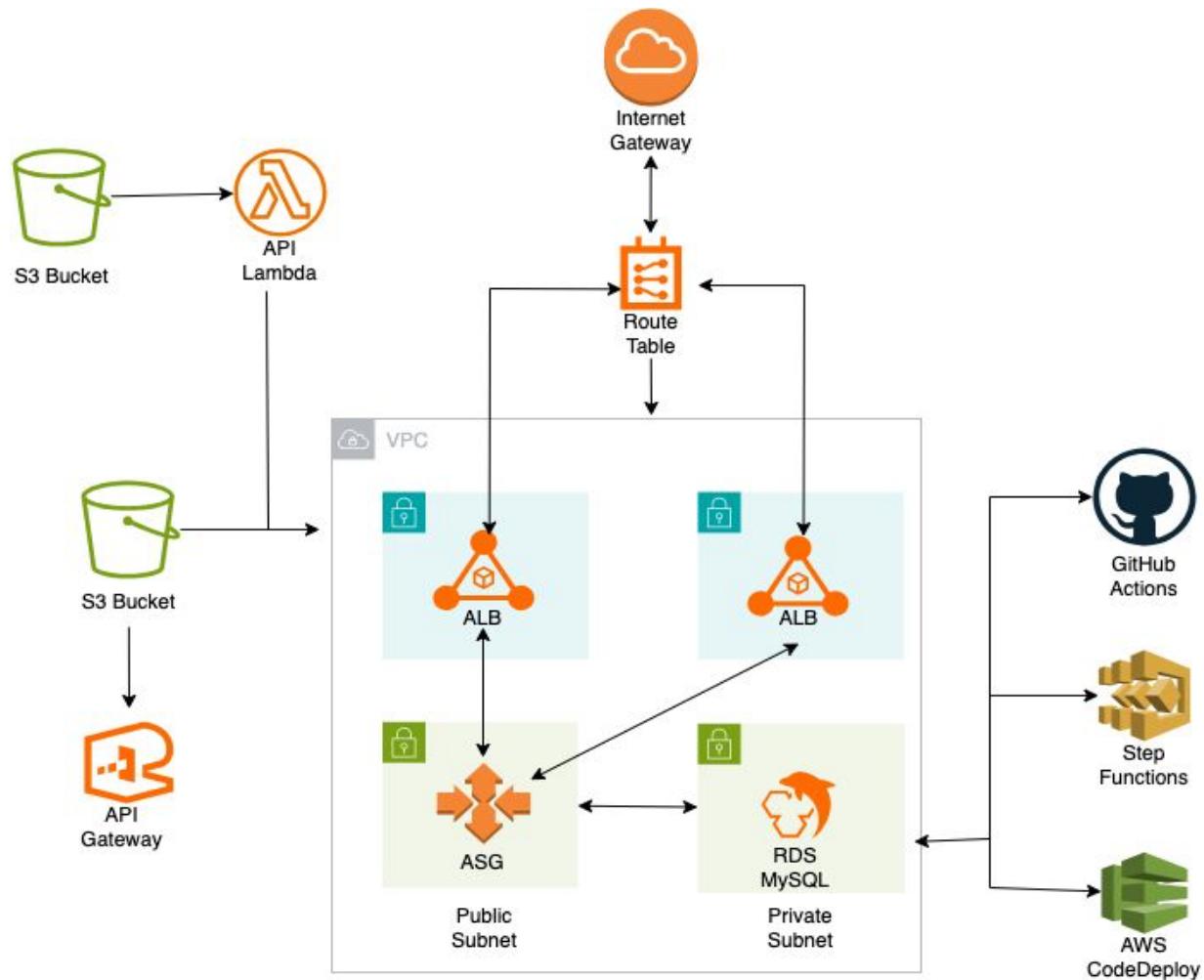


IS 698 Special Topics in Information Systems Project

Name- Akanksha Patel
Id- Ri79216

Deploying a Scalable AWS Architecture with Infrastructure as Code

1. Design an AWS Architecture

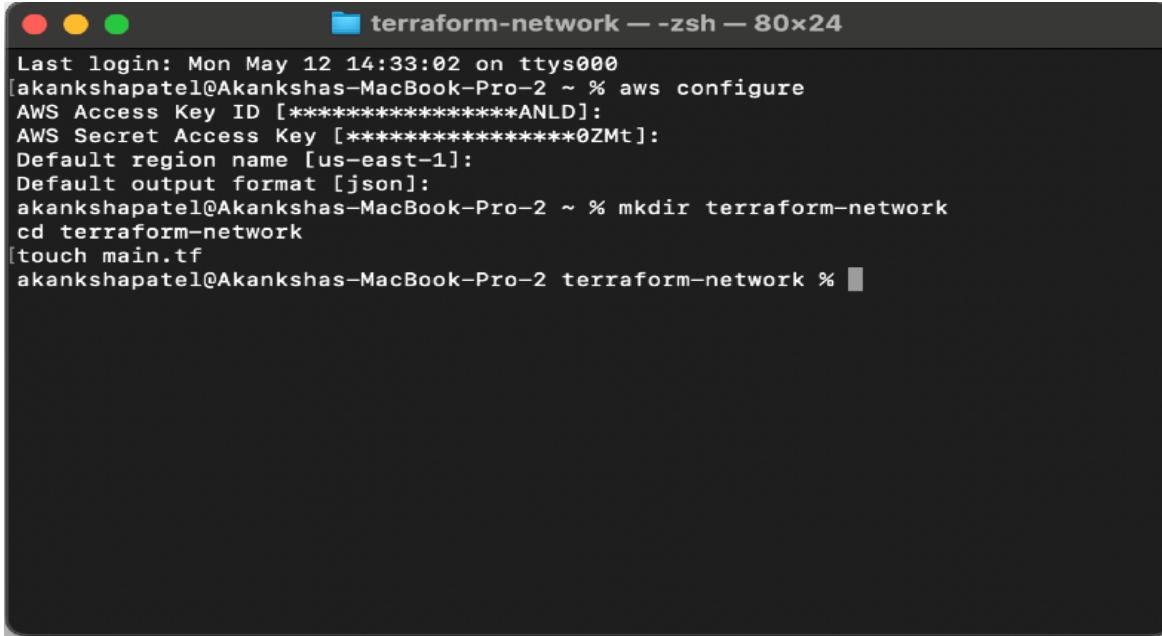


2. Implementation

A. Infrastructure Deployment

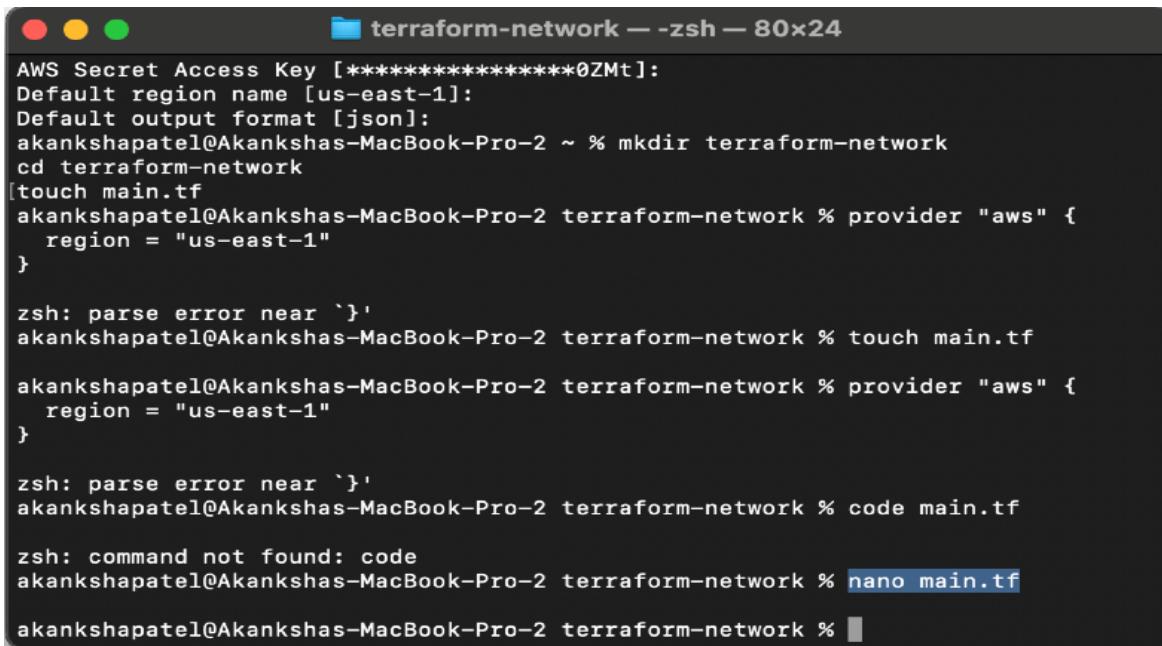
Create AWS Networking Using Terraform

STEP 1: Set Up the Project Folder



```
Last login: Mon May 12 14:33:02 on ttys000
[akankshapatel@Akankshas-MacBook-Pro-2 ~ % aws configure
AWS Access Key ID [*****ANLD]:
AWS Secret Access Key [*****0ZMt]:
Default region name [us-east-1]:
Default output format [json]:
akankshapatel@Akankshas-MacBook-Pro-2 ~ % mkdir terraform-network
cd terraform-network
[touch main.tf
akankshapatel@Akankshas-MacBook-Pro-2 terraform-network %
```

STEP 2: Configure the AWS Provider



```
AWS Secret Access Key [*****0ZMt]:
Default region name [us-east-1]:
Default output format [json]:
akankshapatel@Akankshas-MacBook-Pro-2 ~ % mkdir terraform-network
cd terraform-network
[touch main.tf
akankshapatel@Akankshas-MacBook-Pro-2 terraform-network % provider "aws" {
    region = "us-east-1"
}

zsh: parse error near `}'
akankshapatel@Akankshas-MacBook-Pro-2 terraform-network % touch main.tf

akankshapatel@Akankshas-MacBook-Pro-2 terraform-network % provider "aws" {
    region = "us-east-1"
}

zsh: parse error near `}'
akankshapatel@Akankshas-MacBook-Pro-2 terraform-network % code main.tf

zsh: command not found: code
akankshapatel@Akankshas-MacBook-Pro-2 terraform-network % nano main.tf

akankshapatel@Akankshas-MacBook-Pro-2 terraform-network %
```

STEP 3: Create a VPC

```
terraform-network — terraform apply — 80x24
akankshapatel@Akankshas-MacBook-Pro-2 terraform-network % terraform init
terraform plan
terraform apply

Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.97.0...
- Installed hashicorp/aws v5.97.0 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

```
terraform-network — zsh — 80x24
+ main_route_table_id = (known after apply)
+ owner_id = (known after apply)
+ tags =
  + "Name" = "MainVPC"
}
+ tags_all =
  + "Name" = "MainVPC"
}
}

Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: yes

aws_vpc.main_vpc: Creating...
aws_vpc.main_vpc: Still creating... [10s elapsed]
aws_vpc.main_vpc: Creation complete after 12s [id=vpc-0ed54be816b294d12]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
akankshapatel@Akankshas-MacBook-Pro-2 terraform-network %
```

The screenshot shows the AWS VPC Dashboard with the following details:

- Your VPCs (1/2) Info**: A table showing one VPC named "MainVPC" with ID "vpc-0ed54be816b294d12". It is in the "Available" state, has no Block Public Access policy, and its IP range is "10.0.0.0/16".
- Main route table**: Shows two routes: one via "aws-optimized-eni" to "0.0.0.0/0" and another via "aws-optimized-eni" to "10.0.0.0/16".
- Details** tab for "vpc-0ed54be816b294d12 / MainVPC":
 - VPC ID**: vpc-0ed54be816b294d12
 - State**: Available
 - Block Public Access**: Off
 - DNS resolution**: Enabled
 - Default VPC**: No
 - IPv4 CIDR**: 10.0.0.0/16
 - Network Address Usage metrics**: Disabled
 - Route 53 Resolver DNS Firewall rule groups**: None
 - Owner ID**: 664418549521

STEP 4: Create Public and Private Subnets

Public Subnet

```
terraform-network -- zsh -- 80x24

Initializing the backend...
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v5.97.0

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
aws_vpc.main_vpc: Refreshing state... [id=vpc-0ed54be816b294d12]

Terraform used the selected providers to generate the following execution plan.
Resource actions are indicated with the following symbols:
+ create

Terraform will perform the following actions:

# aws_subnet.public_subnet will be created
```

```
terraform-network -- zsh -- 80x24

+ arn                                     = (known after apply)
+ assign_ipv6_address_on_creation          = false
+ availability_zone                       = "us-east-1a"
+ availability_zone_id                   = (known after apply)
+ cidr_block                             = "10.0.1.0/24"
+ enable_dns64                           = false
+ enable_resource_name_dns_a_record_on_launch = false
+ enable_resource_name_dns_aaaa_record_on_launch = false
+ id                                     = (known after apply)
+ ipv6_cidr_block_association_id          = (known after apply)
+ ipv6_native                            = false
+ map_public_ip_on_launch                 = true
+ owner_id                               = (known after apply)
+ private_dns_hostname_type_on_launch     = (known after apply)
+ tags
  + "Name" = "PublicSubnet"
}
+ tags_all                                = {
  + "Name" = "PublicSubnet"
}
+ vpc_id                                  = "vpc-0ed54be816b294d12"

}

Plan: 1 to add, 0 to change, 0 to destroy.
```

```

 terraform-network -- zsh -- 80x24

+ tags
  + "Name" = "PublicSubnet"
}
+ tags_all
  + "Name" = "PublicSubnet"
}
+ vpc_id
}

Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: yes

aws_subnet.public_subnet: Creating...
aws_subnet.public_subnet: Still creating... [10s elapsed]
aws_subnet.public_subnet: Creation complete after 11s [id=subnet-012390fac2a3b8314]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
akankshapatel@Akankshas-MacBook-Pro-2:~/terraform-network % 

```

Subnets (1/3) Info

Name	Subnet ID	State	VPC	Block Public...	IPv4 CIDR	IPv6 CIDR	Actions
-	subnet-0b752fd30ddactb2	Available	vpc-0b31d3b51b9d5a2e0 pro...	Off	10.0.1.0/24	-	Create subnet
PublicSubnet	subnet-012390fac2a3b8314	Available	vpc-0ed54be816b294d12 Mai...	Off	10.0.1.0/24	-	
-	subnet-046c726c39e1da97d	Available	vpc-0b31d3b51b9d5a2e0 pro...	Off	10.0.2.0/24	-	

subnet-012390fac2a3b8314 / PublicSubnet

Details | Flow logs | Route table | Network ACL | CIDR reservations | Sharing | Tags

Details	Subnet ID : subnet-012390fac2a3b8314 IPv4 CIDR : 10.0.1.0/24 Availability Zone : us-east-1a Route table : rtb-0e12866204c71361 Auto-assign IPv6 address : No	Subnet ARN : arn:aws:ec2:us-east-1:664418949321:subnet-012390fac2a3b8314 Available IPv4 addresses : 251 Availability Zone ID : us-east-1a65 Network ACL : acl-06c20595c3e0a759c Auto-assign customer-owned IPv4 address : No	State : Available IPv6 CIDR : - Network border group : us-east-1 Default subnet : No Customer-owned IPv4 pool : -	Block Public Access : Off IPv6 CIDR association ID : - VPC : vpc-0ed54be816b294d12 MainVPC Auto-assign public IPv4 address : Yes Outpost ID : -
----------------	---	---	--	--

Private Subnet

```
terraform-network --zsh -- 80x24

A managed resource "aws_vpc" "main_vpc" has not been declared in the root
module.

[akankshapatel@Akankshas-MacBook-Pro-2 terraform-network % nano main.tf
akankshapatel@Akankshas-MacBook-Pro-2 terraform-network % terraform init
terraform plan

Initializing the backend...
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v5.97.0

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
aws_vpc.main_vpc: Refreshing state... [id=vpc-0ed54be816b294d12]
aws_subnet.public_subnet: Refreshing state... [id=subnet-012390fac2a3b8314]
```

```
terraform-network --zsh -- 80x24

+ enable_resource_name_dns_a_record_on_launch = false
+ enable_resource_name_dns_aaaa_record_on_launch = false
+ id = (known after apply)
+ ipv6_cidr_block_association_id = (known after apply)
+ ipv6_native = false
+ map_public_ip_on_launch = false
+ owner_id = (known after apply)
+ private_dns_hostname_type_on_launch = (known after apply)
+ tags =
  + "Name" = "PrivateSubnet"
}
+ tags_all =
  + "Name" = "PrivateSubnet"
}
+ vpc_id = "vpc-0ed54be816b294d12"

}

Plan: 1 to add, 0 to change, 0 to destroy.

Note: You didn't use the -out option to save this plan, so Terraform can't
guarantee to take exactly these actions if you run "terraform apply" now.
akankshapatel@Akankshas-MacBook-Pro-2 terraform-network %
```

```

 terraform-network -- zsh -- 84x24
+ owner_id = (known after apply)
+ private_dns_hostname_type_on_launch = (known after apply)
+ tags = {
  + "Name" = "PrivateSubnet"
}
+ tags_all = {
  + "Name" = "PrivateSubnet"
}
+ vpc_id = "vpc-0ed54be816b294d12"
}

Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: yes

aws_subnet.private_subnet: Creating...
aws_subnet.private_subnet: Creation complete after 1s [id=subnet-012ef858c48c2cec0]

akankshapatel@Akankshas-MacBook-Pro-2:~/terraform-network % 

```

Screenshot of the AWS VPC dashboard showing the Subnets (1/4) page. The table lists four subnets:

Name	Subnet ID	State	VPC	Block Public Access	IPv4 CIDR	IPv6 CIDR
PublicSubnet	subnet-012ef858c48c2cec0	Available	vpc-0ed54be816b294d12 MainVPC	Off	10.0.1.0/24	-
PrivateSubnet	subnet-012ef858c48c2cec0	Available	vpc-0ed54be816b294d12 MainVPC	Off	10.0.2.0/24	-
	subnet-0465726d59e1da97d	Available	vpc-0ed54be816b294d12 MainVPC	Off	10.0.2.0/24	-

The PrivateSubnet row is selected, showing its detailed configuration:

subnet-012ef858c48c2cec0 / PrivateSubnet

- Details** tab is active.
- Subnet ID:** subnet-012ef858c48c2cec0
- IPv4 CIDR:** 10.0.2.0/24
- Availability Zone:** us-east-1b
- Route Table:** rtb-0e512866204c71361
- Auto-assign IPv6 address:** No
- State:** Available
- IPv6 CIDR:** -
- Network border group:** us-east-1
- Default subnet:** No
- Customer-owned IPv4 pool:** -
- Block Public Access:** Off
- IPv6 CIDR association ID:** -
- VPC:** vpc-0ed54be816b294d12 | MainVPC
- Auto-assign public IPv4 address:** No
- Outpost ID:** -

STEP 5: Create an Internet Gateway

```
 terraform-network --zsh-- 84x24
+ id      = (known after apply)
+ owner_id = (known after apply)
+ tags    = {
+   "Name" = "MainIGW"
}
+ tags_all = {
+   "Name" = "MainIGW"
}
+ vpc_id  = "vpc-0ed54be816b294d12"
}

Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: yes

aws_internet_gateway.main_igw: Creating...
aws_internet_gateway.main_igw: Creation complete after 0s [id=igw-060d3acd0bc9d232f]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
aankshapatel@Aankshas-MacBook-Pro-2 terraform-network %
```

The screenshot shows the AWS VPC dashboard with the 'Internet gateways' section selected. A single Internet gateway named 'MainIGW' is listed, with its ID 'igw-060d3acd0bc9d232f'. The gateway is attached to the 'MainVPC' with the VPC ID 'vpc-0ed54be816b294d12'. The status is 'Attached'.

Name	Internet gateway ID	State	VPC ID	Owner
MainIGW	igw-060d3acd0bc9d232f	Attached	vpc-0ed54be816b294d12 MainVPC	664418949921

STEP 6: Create a Route Table

```

aws>s
 terraform-network --zsh --84x24

# aws_route_table_association.public_rt_assoc will be created
+ resource "aws_route_table_association" "public_rt_assoc" {
+   id          = (known after apply)
+   route_table_id = (known after apply)
+   subnet_id    = "subnet-012390fac2a3b8314"
}

Plan: 2 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: yes

aws_route_table.public_rt: Creating...
aws_route_table.public_rt: Creation complete after 1s [id=rtb-002f389b52a05b0b5]
aws_route_table_association.public_rt_assoc: Creating...
aws_route_table_association.public_rt_assoc: Creation complete after 0s [id=rtbassoc-072f637d2d29c8ffb]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
akankshapatel@Akankshas-MacBook-Pro-2 terraform-network %

```

Name	Route table ID	Explicit subnet associations	Main	VPC	Owner ID
PublicRT	rtb-002f389b52a05b0b5	subnet-012390fac2a3b8314	No	vpc-0ed54be816b294d12 MainVPC	664418949921
-	rtb-0e1234567890abc1234567890	-	Yes	vpc-0ed54be816b294d12 MainVPC	664418949921
-	rtb-0e1234567890abc1234567890	-	Yes	vpc-0b31d3b31b0d3a2e0 pro...	664418949921

rtb-002f389b52a05b0b5 / PublicRT

Details

Route table ID: rtb-002f389b52a05b0b5
Main: No
Owner ID: 664418949921

Explicit subnet associations: subnet-012390fac2a3b8314 / PublicSubnet

Edge associations: -

STEP 7: Create a Security Group

```
 terraform-network --zsh -- 84x24

+ owner_id          = (known after apply)
+ revoke_rules_on_delete = false
+ tags              =
  + "Name" = "WebSecurityGroup"
}
+ tags_all          =
  + "Name" = "WebSecurityGroup"
}
+ vpc_id            = "vpc-0ed54be816b294d12"
}

Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: yes

aws_security_group.web_sg: Creating...
aws_security_group.web_sg: Creation complete after 4s [id=sg-06efde22485d87dde]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
akankshapatel@Akankshas-MacBook-Pro-2 terraform-network %
```

The screenshot shows the AWS VPC Security Groups console. On the left, there's a navigation sidebar with sections like EC2 Global View, Virtual private cloud, Security, PrivateLink and Lattice, DNS firewall, and Network Firewall. The main area displays a table of security groups:

Name	Security group ID	Security group name	VPC ID	Description	Owner
-	sg-07a8a7bea79ef411	default	VPC-031d310963a2e0	default VPC security group	664418949921
-	sg-0a8a478c16c930630	default	VPC-0ed54be816b294d12	default VPC security group	664418949921
<input checked="" type="checkbox"/> WebSecurityGroup	sg-06efde22485d87dde	web-sg	VPC-0ed54be816b294d12	Allow HTTP and SSH	664418949921
-	sg-0bd2311954e078a54	launch-wizard-1	VPC-031d310963a2e0	launch-wizard-1 created 2025-05-07T2...	664418949921

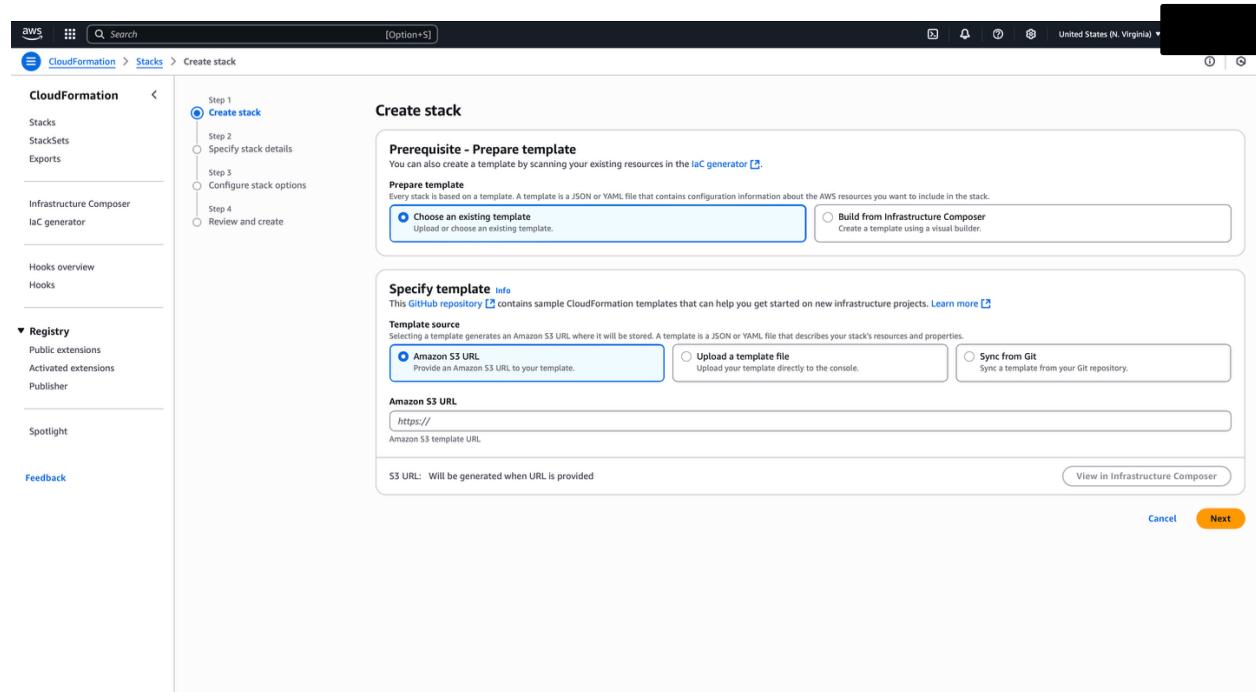
Below the table, a detailed view of the 'WebSecurityGroup' is shown. The 'Details' tab is selected, displaying the following information:

- Security group name: web-sg
- Owner: 664418949921
- Description: Allow HTTP and SSH
- VPC ID: VPC-0ed54be816b294d12

Use CloudFormation to deploy EC2 instances, RDS, and Lambda functions

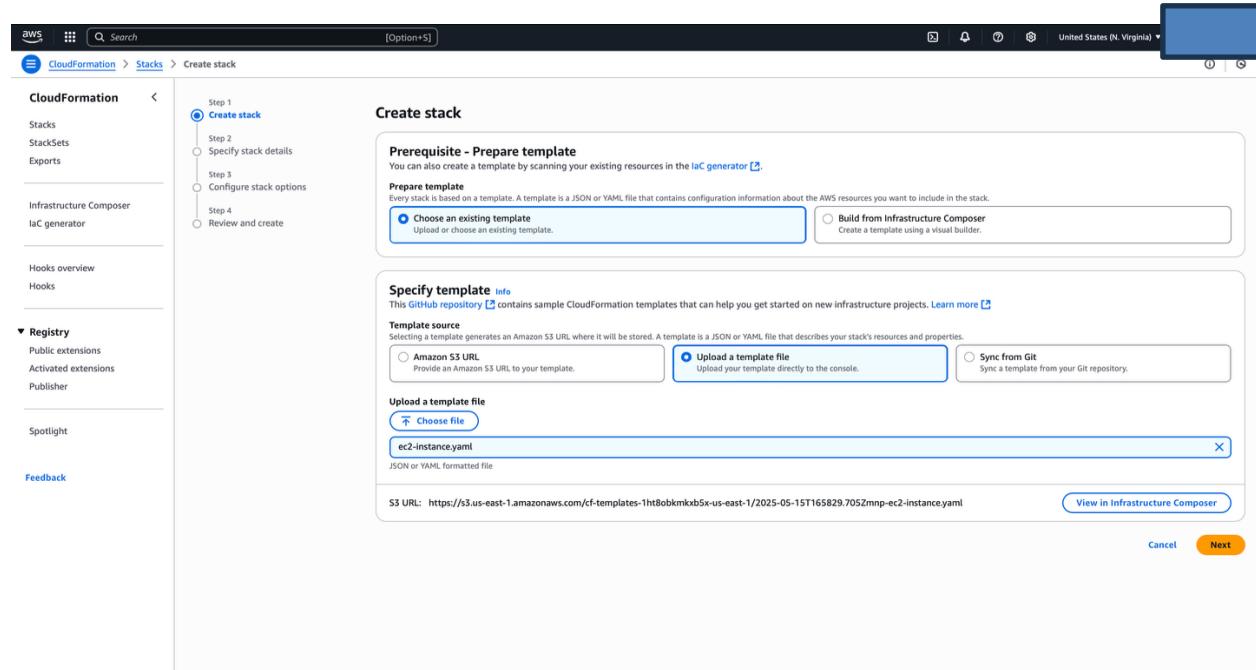
EC2 Instance:

CloudFormation



The screenshot shows the AWS CloudFormation 'Create stack' wizard. On the left, a sidebar lists 'CloudFormation' sections: Stacks, StackSets, Exports; Infrastructure Composer, IaC generator; Hooks overview, Hooks; Registry (Public extensions, Activated extensions, Publisher); and Spotlight, Feedback. The main area is titled 'Create stack' and shows 'Step 1 Create stack'. It has four tabs: Step 1 (selected), Step 2, Step 3, and Step 4. A box titled 'Prerequisite - Prepare template' contains instructions and two options: 'Choose an existing template' (selected) and 'Build from Infrastructure Composer'. Below this is a 'Specify template' section with 'Template source' options: 'Amazon S3 URL' (selected), 'Upload a template file', and 'Sync from Git'. An 'Amazon S3 URL' input field contains 'https://'. At the bottom right are 'Cancel' and 'Next' buttons.

Upload the Template



This screenshot continues the 'Create stack' wizard. The sidebar and Step 1 tabs are identical to the previous screenshot. In the 'Specify template' section, the 'Upload a template file' option is selected. A file input field shows 'ec2-instance.yaml' with a delete 'X' button. Below it, a note says 'JSON or YAML formatted file'. At the bottom right are 'Cancel' and 'Next' buttons.

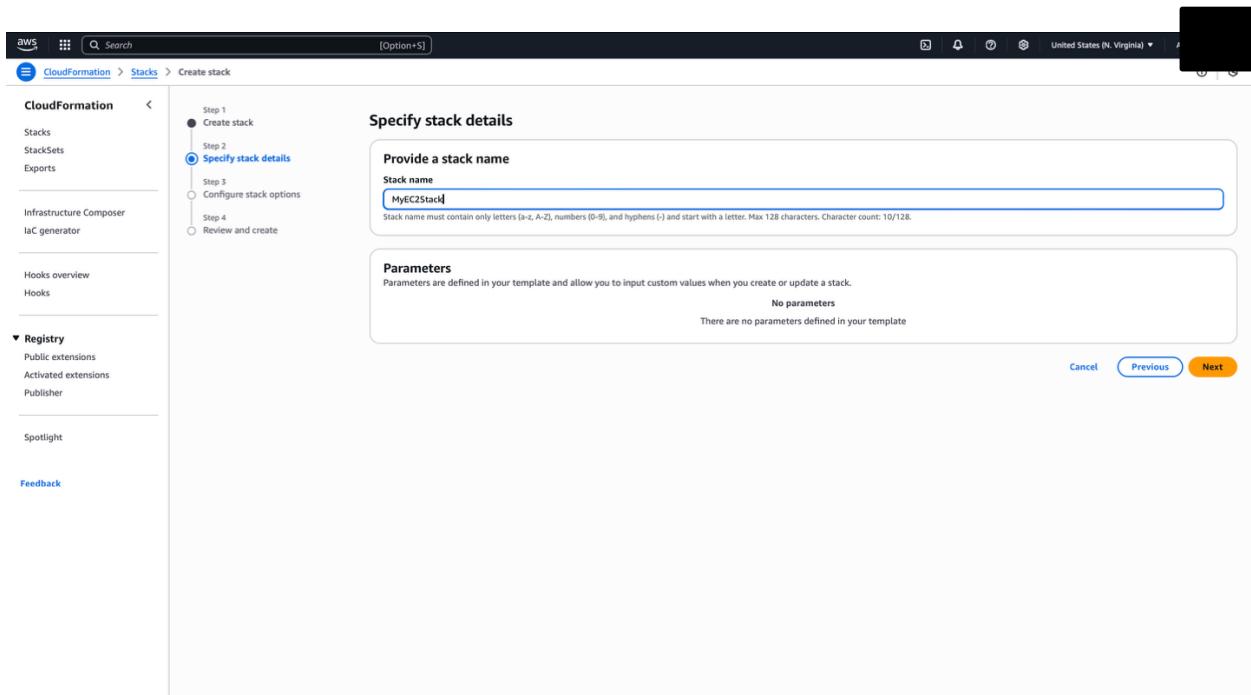
Create Stack

The screenshot shows the 'Specify stack details' step of the CloudFormation 'Create stack' wizard. The left sidebar lists navigation options like Stacks, StackSets, Infrastructure Composer, Registry, and Feedback. The main panel shows the 'Specify stack details' step selected. It includes fields for 'Stack name' (containing 'MyEC2Stack') and 'Parameters' (empty). Navigation buttons 'Cancel', 'Previous', and 'Next' are at the bottom.

Monitor Progress

This screenshot is identical to the one above, showing the 'Specify stack details' step of the CloudFormation 'Create stack' wizard. The left sidebar and main panel are the same, with the 'Specify stack details' step selected, a 'Stack name' field containing 'MyEC2Stack', and an empty 'Parameters' section. Navigation buttons 'Cancel', 'Previous', and 'Next' are at the bottom.

Verify EC2 Instance



Connect via SSH

```
Downloads — ec2-user@ip-10-0-1-94:~ — ssh -i project_key.pem ec2-us...
zsh: parse error near `\'n'
akankshapatel@Akankshas-MacBook-Pro-2 Downloads % cd ~/Downloads
akankshapatel@Akankshas-MacBook-Pro-2 Downloads % chmod 400 project_key.pem
akankshapatel@Akankshas-MacBook-Pro-2 Downloads % ssh -i project_key.pem ec2-use
r@107.21.193.206

The authenticity of host '107.21.193.206 (107.21.193.206)' can't be established.
ED25519 key fingerprint is SHA256:OBjRjYy4mK/t10HVX/i/0/qhg0ZhIc4XpqeAZHGqU88.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '107.21.193.206' (ED25519) to the list of known hosts
.

      _|_ _|_
     _|_| /   Amazon Linux 2 AMI
    ___|_\___|___|_

https://aws.amazon.com/amazon-linux-2/
56 package(s) needed for security, out of 99 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-10-0-1-94 ~]$
```

RDS Instances

CloudFormation

The screenshot shows the AWS CloudFormation Stacks page. On the left is a navigation sidebar with links for CloudFormation, Stacks, StackSets, Exports, Infrastructure Composer, Hooks, Registry, Spotlight, and Feedback. The main area displays a table titled "Stacks (3)". The table has columns for Stack name, Status, Created time, and Description. The status column includes icons for CREATE_COMPLETE (green), CREATE_FAILED (red), and DELETE_FAILED (red). The created time column shows dates from May 15, 2025, to April 21, 2025. The description column provides brief details for each stack.

Stack name	Status	Created time	Description
MyEC2Stack	CREATE_COMPLETE	2025-05-15 13:00:43 UTC-0400	Deploy an EC2 instance using specified AMI and key pair
LambdaStack	CREATE_COMPLETE	2025-05-13 13:24:32 UTC-0400	Lambda Function and IAM Role to log messages to CloudWatch
MyStack	DELETE_FAILED	2025-04-21 16:13:01 UTC-0400	Full setup with VPC, EC2, S3, Lambda

Upload the YAML File

The screenshot shows the "Create stack" wizard, Step 1: Prerequisite - Prepare template. The sidebar on the left lists steps: Step 1 (Create stack, selected), Step 2 (Specify stack details), Step 3 (Configure stack options), and Step 4 (Review and create). The main area contains a "Prerequisite - Prepare template" section with instructions to create a template by scanning existing resources or using Infrastructure Composer. It shows two options: "Choose an existing template" (selected) and "Build from Infrastructure Composer". Below this is a "Specify template" section with a link to a GitHub repository containing sample templates. It shows three options: "Amazon S3 URL" (selected), "Upload a template file" (selected), and "Sync from Git". The "Upload a template file" section shows a file named "rds_instance.yaml" selected. At the bottom, there is a "S3 URL" field with the value "https://s3.us-east-1.amazonaws.com/cf-templates-1ht8obkmkx5x-us-east-1/2025-05-15T173340.42921fl-rds_instance.yaml", a "View in Infrastructure Composer" button, and "Cancel" and "Next" buttons.

Configure Stack Details

The screenshot shows the 'Specify stack details' step in the CloudFormation wizard. On the left, a sidebar lists navigation options like Stacks, StackSets, Infrastructure Composer, and Registry. The main area has a title 'Specify stack details' and two sections: 'Provide a stack name' and 'Parameters'. In 'Provide a stack name', the stack name 'MyRDSStack' is entered. Below it, a note says 'Stack name must contain only letters (a-z, A-Z), numbers (0-9), and hyphens (-) and start with a letter. Max 128 characters. Character count: 10/128.' In 'Parameters', it says 'Parameters are defined in your template and allow you to input custom values when you create or update a stack.' and 'No parameters'.

Review & Deploy

The screenshot shows the 'Stacks' page in the CloudFormation console. The sidebar includes options like Stacks, Drifts, Infrastructure Composer, and Registry. The main table displays four stacks:

Stack name	Status	Created time	Description
MyRDSStack	CREATE_COMPLETE	2025-05-15 13:36:15 UTC-0400	Deploy a private MySQL RDS Instance with VPC and subnets
MyEC2Stack	CREATE_COMPLETE	2025-05-15 13:00:43 UTC-0400	Deploy an EC2 instance using specified AMI and key pair
LambdaStack	CREATE_COMPLETE	2025-05-13 13:24:32 UTC-0400	Lambda Function and IAM Role to log messages to CloudWatch
MyStack	DELETE_FAILED	2025-04-21 16:13:01 UTC-0400	Full setup with VPC, EC2, S3, Lambda

Verify RDS Instance

```
Downloads — ec2-user@ip-10-0-1-94:~ — ssh -i project_key.pem ec2-us...
[akankshapatel@Akankshas-MacBook-Pro-2 ~ % mysql -h mydb.copagycoa7l4.us-east-1.rds.amazonaws.com -u admin -p

|Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 517
Server version: 8.0.41 Source distribution

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| myappdb |
| mysql |
| performance_schema |
+-----+
```

Lambda functions

CloudFormation and uploading the Template

The screenshot shows the AWS CloudFormation 'Create stack' wizard. On the left, there's a sidebar with navigation links like 'CloudFormation', 'Stacks', 'StackSets', 'Exports', 'Infrastructure Composer', 'IaC generator', 'Hooks overview', 'Hooks', 'Registry', 'Public extensions', 'Activated extensions', 'Publisher', 'Spotlight', and 'Feedback'. The main area is titled 'Create stack' and shows 'Step 1 Create stack' selected. It has four steps: Step 1 (selected), Step 2 (Specify stack details), Step 3 (Configure stack options), and Step 4 (Review and create). A box labeled 'Prerequisite - Prepare template' contains instructions: 'You can also create a template by scanning your existing resources in the IaC generator.' Below it, 'Prepare template' says 'Every stack is based on a template. A template is a JSON or YAML file that contains configuration information about the AWS resources you want to include in the stack.' There are three options: 'Choose an existing template' (selected), 'Build from Infrastructure Composer' (disabled), and 'Sync from Git' (disabled). The 'Specify template' section includes a note about a GitHub repository and a 'Template source' section with three options: 'Amazon S3 URL' (disabled), 'Upload a template file' (selected, with a file input field containing 'lambda-function.yaml'), and 'Sync from Git' (disabled). At the bottom, there's an 'S3 URL' field with the value 'https://s3.us-east-1.amazonaws.com/cf-templates-1ht8obkmxb5x-us-east-1/2025-05-15T175009.570Ziq2-lambda-function.yaml' and two buttons: 'View in Infrastructure Composer' and 'Next'.

Create Stack

The screenshot shows the 'Specify stack details' step of the CloudFormation wizard. On the left, a sidebar lists navigation options like Stacks, StackSets, Exports, Infrastructure Composer, and Registry. The main area shows a progress bar with four steps: Step 1 (Create stack), Step 2 (Specify stack details, which is selected and highlighted in blue), Step 3 (Configure stack options), and Step 4 (Review and create). The 'Provide a stack name' section contains a 'Stack name' input field with 'LambdaStack' typed in. Below it, a note states: 'Stack name must contain only letters (a-z, A-Z), numbers (0-9), and hyphens (-) and start with a letter. Max 128 characters. Character count: 11/128.' The 'Parameters' section indicates 'No parameters' and notes that there are no parameters defined in the template. At the bottom right are 'Cancel', 'Previous', and 'Next' buttons.

Configure Stack

The screenshot shows the 'Stacks' page in the CloudFormation console. The sidebar on the left includes options for Stacks, Drifts, StackSets, Exports, Infrastructure Composer, Registry, and Feedback. The main content area displays a list of stacks under the heading 'Stacks (4)'. One stack, 'LambdaStack', is selected and shown in detail. The 'Stack info' tab is active, showing the following details:

- Stack ID:** arn:aws:cloudformation:us-east-1:664418949921:stack/LambdaStack/7ec867f0-31b5-11e0-8c4c-12d2e14b57f3
- Status:** CREATE_COMPLETE
- Status reason:** -
- Parent stack:** -
- Deleted time:** -
- Last drift check time:** -
- IAM role:** -

The 'Description' field indicates: 'Deploy a basic AWS Lambda function with IAM Role'. The 'Tags' section at the bottom shows a note: 'Stack-level tags will apply to all supported resources in your stack. You can add up to 50 unique tags for each stack.' There is a search bar for tags and a table for key-value pairs, both currently empty.

Verify Lambda Function Created

The screenshot shows the AWS Lambda console interface. At the top, the URL is [aws.amazon.com/lambda/functions>HelloWorldFunction](#). The main area displays the function code in `index.py`:

```
def lambda_handler(event, context):
    print("Event received:", event)
    return {
        'statusCode': 200,
        'body': "Hello from Lambda!"
    }
```

Below the code, there's a test event named "mytest" with the following response:

```
{"statusCode": 200, "body": "Hello from Lambda!"}
```

The "TEST EVENTS" section shows "mytest" is selected. On the right, a "Create new test event" dialog is open, showing the same event name "mytest". The "Event JSON" field contains:

```
{ "key1": "value1" }
```

The "Code properties" tab is visible at the bottom.

This screenshot is identical to the one above, showing the AWS Lambda console with the Hello World function. The code, test event, and overall interface are the same.

Deploy a web application on EC2 (can be a simple HTML page).

Launched EC2 instance: MyWebServer

The screenshot shows the AWS EC2 Instances page. On the left, there's a navigation sidebar with sections like Dashboard, EC2 Global View, Events, Instances (with sub-options Instances, Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations), Images (AMIs, AMI Catalog), Elastic Block Store (Volumes, Snapshots, Lifecycle Manager), Network & Security (Security Groups, Elastic IPs, Placement Groups, Key Pairs, Network Interfaces), Load Balancing (Load Balancers, Target Groups, Trust Stores), and Auto Scaling (Auto Scaling Groups). The main content area has a header 'Instances (1/3) Info' with a search bar and filters. It lists three instances: 'i-0060827b4f724b9aef' (status Running, type t2.micro, zone us-east-1a, public IP 54.88.1.84), 'i-0a01f717d1812a54' (status Running, type t2.micro, zone us-east-1a, public IP ec2-107-21-193-206.co..., IP 107.21.193.206), and 'MyWebServer' (status Running, type t2.micro, zone us-east-1a, public IP ec2-54-88-1-84.compute-1.amazonaws.com, IP 54.88.1.84). Below this is a detailed view for 'MyWebServer' with tabs for Details, Status and alarms, Monitoring, Security, Networking, Storage, and Tags. The Details tab shows instance summary information including Public IPv4 address (54.88.1.84), Instance state (Running), Private IP DNS name (ip-10-0-1-138.ec2.internal), Instance type (t2.micro), VPC ID (vpc-0ed54be816b294d12), and Outbound ID.

Connect to the EC2 Instance (via SSH):

The screenshot shows the 'Connect to instance' dialog box. At the top, it says 'Connect to instance' and provides instructions: 'Connect to your instance i-066e93fb4a5989e1d (MyWebServer) using any of these options'. Below this are three tabs: EC2 Instance Connect, Session Manager, and SSH client (which is selected). The SSH client tab contains a section for 'Instance ID' (i-066e93fb4a5989e1d [MyWebServer]). It lists four steps for connecting via SSH: 1. Open an SSH client, 2. Locate your private key file. The key used to launch this instance is project_key.pem, 3. Run this command, if necessary, to ensure your key is not publicly viewable: chmod 400 "project_key.pem", and 4. Connect to your instance using its Public DNS: ec2-54-88-1-84.compute-1.amazonaws.com. An example command is shown: ssh -i "project_key.pem" ec2-user@ec2-54-88-1-84.compute-1.amazonaws.com. A note at the bottom states: 'Note: In most cases, the guessed username is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username.' At the bottom right are 'Cancel' and 'Connect to instance' buttons.

```
Downloads — ec2-user@ip-10-0-1-138:~ — ssh -i project_key.pem ec2-u...
Public                                         terraform.tfstate.backup
PyCharmMiscProject                         test-file.txt
PyCharmMiscProject.iml                      testfile.txt
Sunflower.sql                               upload-test.txt
delete-markers.json                        upload_file.py
downloaded-file.txt                         versions.jsonaws
ec2-instance.yaml                          vpc-subnets.yaml
function.zip                                wekafiles
[akankshapatel@Akankshas-MacBook-Pro-2 ~ % cd Downloads
[akankshapatel@Akankshas-MacBook-Pro-2 Downloads % ssh -i "project_key.pem" ec2-u...
ser@54.88.1.84

      #
~\_\_ #####_          Amazon Linux 2023
~~ \_\#####\
~~   \###|
~~     \#/ ___ https://aws.amazon.com/linux/amazon-linux-2023
~~       V~' '-->
~~~      /
~~-.-
~/--/
~/m/'

Last login: Tue May 13 20:12:49 2025 from 130.85.59.148
[ec2-user@ip-10-0-1-138 ~]$
```

Install Apache Web Server on EC2

```
Downloads — ec2-user@ip-10-0-1-138:/var/www/html — ssh -i project_k...
      /
~/--/
~/m'

Last login: Tue May 13 20:28:11 2025 from 130.85.59.148
[ec2-user@ip-10-0-1-138 ~]$ sudo yum update -y
Last metadata expiration check: 4:09:15 ago on Tue May 13 20:15:59 2025.
=====
WARNING:
  A newer release of "Amazon Linux" is available.

  Available Versions:

  Version 2023.7.20250512:
    Run the following command to upgrade to 2023.7.20250512:

      dnf upgrade --releasever=2023.7.20250512

  Release notes:
    https://docs.aws.amazon.com/linux/al2023/release-notes/relnotes-2023.7.2025
0512.html

=====
Dependencies resolved.
Nothing to do.
```

```
=====
Dependencies resolved.
Nothing to do.
Complete!
[ec2-user@ip-10-0-1-138 ~]$ sudo yum install -y httpd
Last metadata expiration check: 4:09:32 ago on Tue May 13 20:15:59 2025.
Dependencies resolved.
=====
Package          Arch    Version        Repository      Size
=====
Installing:
  httpd           x86_64  2.4.62-1.amzn2023   amazonlinux   48 k
Installing dependencies:
  apr             x86_64  1.7.5-1.amzn2023.0.4   amazonlinux   129 k
  apr-util         x86_64  1.6.3-1.amzn2023.0.1   amazonlinux   98 k
  generic-logos-httpd  noarch  18.0.0-12.amzn2023.0.3   amazonlinux   19 k
  httpd-core       x86_64  2.4.62-1.amzn2023   amazonlinux   1.4 M
  httpd-filesystem  noarch  2.4.62-1.amzn2023   amazonlinux   14 k
  httpd-tools       x86_64  2.4.62-1.amzn2023   amazonlinux   81 k
  libbrotli         x86_64  1.0.9-4.amzn2023.0.2   amazonlinux   315 k
  mailcap          noarch  2.1.49-3.amzn2023.0.3   amazonlinux   33 k
Installing weak dependencies:
  apr-util-openssl x86_64  1.6.3-1.amzn2023.0.1   amazonlinux   17 k
```

Deploy a Simple HTML Page to Apache

```
=====
Installed:
  apr-1.7.5-1.amzn2023.0.4.x86_64
  apr-util-1.6.3-1.amzn2023.0.1.x86_64
  apr-util-openssl-1.6.3-1.amzn2023.0.1.x86_64
  generic-logos-httpd-18.0.0-12.amzn2023.0.3.noarch
  httpd-2.4.62-1.amzn2023.x86_64
  httpd-core-2.4.62-1.amzn2023.x86_64
  httpd-filesystem-2.4.62-1.amzn2023.noarch
  httpd-tools-2.4.62-1.amzn2023.x86_64
  libbrotli-1.0.9-4.amzn2023.0.2.x86_64
  mailcap-2.1.49-3.amzn2023.0.3.noarch
  mod_http2-2.0.27-1.amzn2023.0.3.x86_64
  mod_lua-2.4.62-1.amzn2023.x86_64

Complete!
[ec2-user@ip-10-0-1-138 ~]$ sudo systemctl start httpd
[ec2-user@ip-10-0-1-138 ~]$ sudo systemctl enable httpd
Created symlink /etc/systemd/system/multi-user.target.wants/httpd.service → /usr/lib/systemd/system/httpd.service.
[ec2-user@ip-10-0-1-138 ~]$ cd /var/www/html
[ec2-user@ip-10-0-1-138 html]$ sudo nano index.html
[ec2-user@ip-10-0-1-138 html]$
```

Result



Hello from EC2 Web Server!

Configure the database for the web application.

Step 1: SSH into EC2

```
p1: SSH into EC2
Downloads — ec2-user@ip-10-0-1-138:~ — ssh -i ~/Downloads/project_k...
Last login: Tue May 13 20:23:08 on ttys000
[akankshapatel@Akankshas-MacBook-Pro-2 ~ % aws configure
AWS Access Key ID [*****ANLD]:
AWS Secret Access Key [*****0ZMt]:
Default region name [us-east-1]:
Default output format [json]:
[akankshapatel@Akankshas-MacBook-Pro-2 ~ % cd Downloads
[akankshapatel@Akankshas-MacBook-Pro-2 Downloads % ssh -i ~/Downloads/project_key
.pem ec2-user@54.88.1.84

,
#_
~\_ #####_      Amazon Linux 2023
~~ \_#####\
~~   \###|
~~     \#/ ___  https://aws.amazon.com/linux/amazon-linux-2023
~~       V~' '-->
~~~   /
~~.~. /_
~/ _/
~/m/'

Last login: Wed May 14 00:24:59 2025 from 73.212.60.215
[ec2-user@ip-10-0-1-138 ~]$
```

Step 2: Install MySQL client on EC2

```
[mysql]> help

For information about MySQL products and services, visit:
  http://www.mysql.com/
For developer information, including the MySQL Reference Manual, visit:
  http://dev.mysql.com/
To buy MySQL Enterprise support, training, or other products, visit:
  https://shop.mysql.com/

List of all MySQL commands:
Note that all text commands must be first on line and end with ';'
?          (\?) Synonym for `help'.
clear      (\c) Clear the current input statement.
connect    (\r) Reconnect to the server. Optional arguments are db and host.
delimiter  (\d) Set statement delimiter.
edit      (\e) Edit command with $EDITOR.
ego       (\G) Send command to mysql server, display result vertically.
exit      (\q) Exit mysql. Same as quit.
go        (\g) Send command to mysql server.
help      (\h) Display this help.
nopager   (\n) Disable pager, print to stdout.
notee     (\t) Don't write into outfile.
pager     (\P) Set PAGER [to_pager]. Print the query results via PAGER.
```

Create the Database and Table

```
[mysql]> CREATE DATABASE myappdb;
Query OK, 1 row affected (0.06 sec)

mysql> USE myappdb;
Database changed
mysql>
mysql> CREATE TABLE users (
    ->     id INT AUTO_INCREMENT PRIMARY KEY,
    ->     name VARCHAR(100),
    ->     email VARCHAR(100)
    -> );
Query OK, 0 rows affected (0.08 sec)

mysql>
```

Insert Sample Data

```
akankshapatel — mysql -h mydb.copagycoa7l4.us-east-1.rds.amazonaws.com
ssl_session_data_print Serializes the current SSL session data to stdout or file
For server side help, type 'help contents'

mysql> CREATE DATABASE myappdb;
Query OK, 1 row affected (0.06 sec)

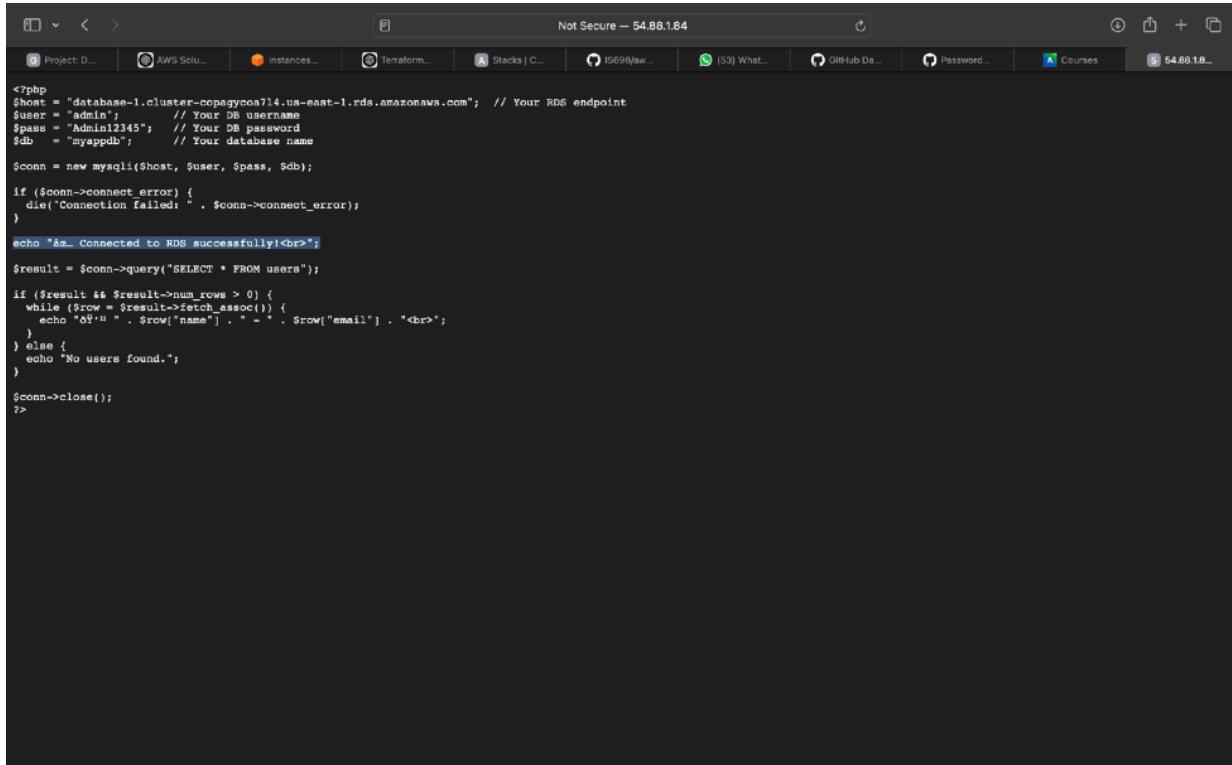
mysql> USE myappdb;
Database changed
mysql>
mysql> CREATE TABLE users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL DEFAULT "admin",
    email VARCHAR(100) NOT NULL DEFAULT "Admin12345",
    password VARCHAR(100) NOT NULL DEFAULT "Admin12345";
);
$host = "mydb.copagycoa7l4.us-east-1.rds.amazonaws.com";
$user = "admin";
$pass = "Admin12345";
$db = "myappdb";
Query OK, 0 rows affected (0.08 sec)

// Connect to MySQL using PHP
mysql> INSERT INTO users (name, email) VALUES ('Alice', 'alice@example.com');
$conn = new mysqli($host, $user, $pass, $db);
Query OK, 1 row affected (0.03 sec)

if ($conn->connect_error){
mysql> INSERT INTO users (name, email) VALUES ('Bob', 'bob@example.com');
Query OK, 1 row affected (0.03 sec)

mysql>
```

Configure Your EC2 App



```
</php>
$host = "database-1.cluster-copagycoa7l4.us-east-1.rds.amazonaws.com"; // Your RDS endpoint
$user = "admin"; // Your DB username
$pass = "Admin12345"; // Your DB password
$db = "myappdb"; // Your database name

$conn = new mysqli($host, $user, $pass, $db);

if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}

echo "Connected to RDS successfully!  
";

$result = $conn->query("SELECT * FROM users");

if ($result && $result->num_rows > 0) {
    while ($row = $result->fetch_assoc()) {
        echo "$row["name"] - $row["email"]  
";
    }
} else {
    echo "No users found.";
}

$conn->close();
?>
```

Implement autoscaling to manage web server load.

STEP 1: Create an AMI from Existing EC2

The screenshot shows the AWS EC2 console with the 'Amazon Machine Images (AMIs)' page. A new AMI, 'MyWebServerAMI', has been created from an existing EC2 instance. The AMI details are displayed, including its ID (ami-03176c9c58a043491), image type (machine), platform details (Linux/UNIX), and root device type (EBS). The AMI is currently available.

AMI ID	Image type	Platform details	Root device type
ami-03176c9c58a043491	machine	Linux/UNIX	EBS
AMI name	Owner account ID	Architecture	Usage operation
MyWebServerAMI	664418949921	x86_64	RunInstances
Root device name	Status	Source	Virtualization type
/dev/xvda	Available	664418949921/MyWebServerAMI	hvm
Boot mode	State reason	Creation date	Kernel ID
uefi-preferred	-	2025-05-14T18:44:33.000Z	-
Description	Product codes	RAM disk ID	Deprecation time
-	-	-	-
Last launched time	Block devices	Deregistration protection	Allowed image
-	/dev/xvda=snap-0a03153d881be80e@truegp3	Disabled	-

STEP 2: Create a Launch Template

The screenshot shows the AWS EC2 console with the 'Launch Templates' page. A new launch template, 'WebServerTemplate', has been created. The launch template details are displayed, including its ID (lt-07cd6415b63829969), name (WebServerTemplate), and default version (1).

Launch template ID	Launch Template Name	Default Version	Latest Version	Create Time	Created By	Managed	Operator
lt-08f0e27ee8851e3b7	AutoScaling-LT	1	1	2025-03-13T19:27:03.000Z	arn:aws:iam::664418949921:us...	false	-
lt-07cd6415b63829969	WebServerTemplate	1	1	2025-05-14T19:05:19.000Z	arn:aws:iam::664418949921:root	false	-

STEP 3: Create an Auto Scaling Group

The screenshot shows the AWS EC2 Auto Scaling groups page. On the left, there's a navigation sidebar with various EC2 services like Dashboard, Global View, Events, Instances, Images, Elastic Block Store, Network & Security, Load Balancing, Auto Scaling, and CloudWatch Metrics. The 'Auto Scaling Groups' section is selected. The main content area shows a table for 'Auto Scaling groups (1/1)'. The table has columns for Name, Launch template/configuration, Instances, Status, Desired capacity, Min, Max, and Availability Zones. One row is highlighted for 'WebASG' with a 'WebServerTemplate | Version Default' launch configuration, 1 instance, and 1 desired capacity across 3 availability zones. Below this, a detailed view for 'Auto Scaling group: WebASG' is shown with tabs for Details, Integrations - new, Automatic scaling, Instance management, Instance refresh, Activity, and Auto Scaling groups. The 'Automatic scaling' tab is selected, showing 'WebASG Capacity overview' with a desired capacity of 1, scaling limits from 1 to 3, and a status of 'Status -'. At the bottom, there are links for CloudWatch Metrics and Feedback.

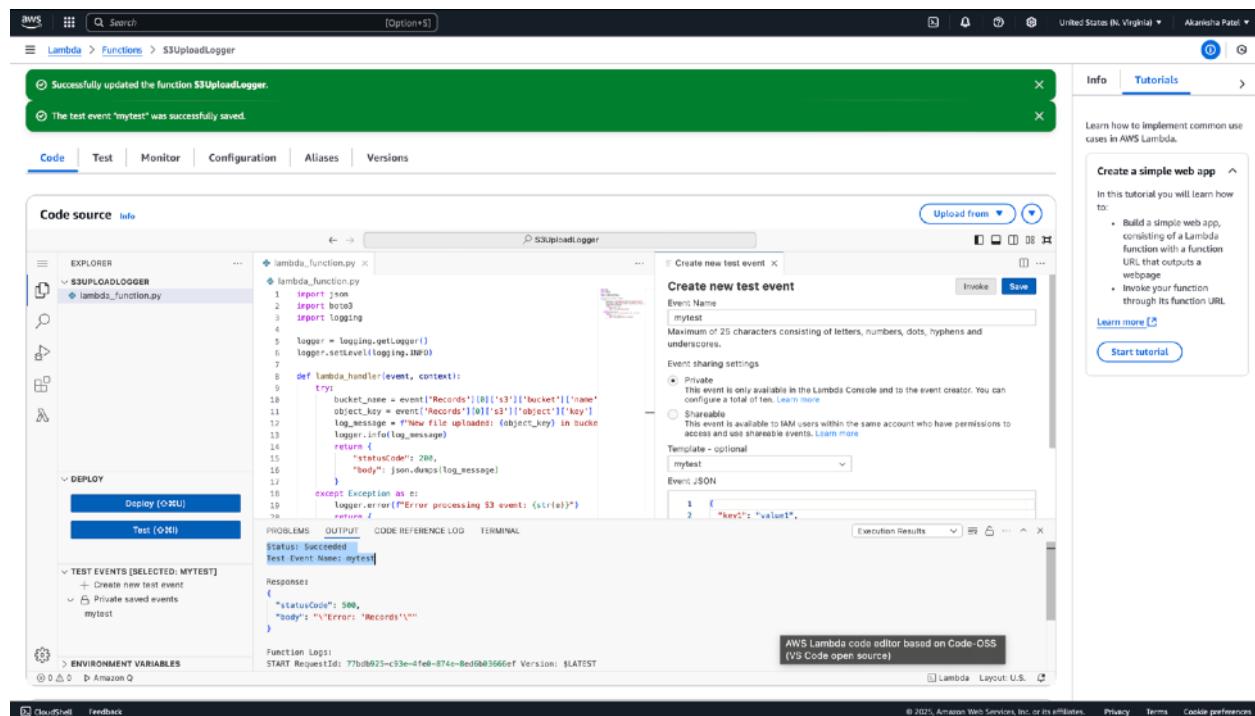
Test

The screenshot shows the AWS EC2 Instances page. The left sidebar includes sections for Instances, Images, Elastic Block Store, Network & Security, Load Balancing, and Auto Scaling. The 'Instances' section is selected. The main area displays a table for 'Instances (2/4)'. It lists two instances: 'myinstance' (t2.micro, running) and 'MyWebServer' (t2.micro, running). A third instance, 'CF-EC2-Instance', is listed but appears to be inactive. Below the table, it says '2 instances selected'. Under the 'Monitoring' tab, there are several line charts showing CPU utilization (%), Network in (bytes), Network out (bytes), Network packets in (count), Network packets out (count), Metadata no token (count), CPU credit usage (count), and CPU credit balance (count). The charts cover a time range from 18:45 to 19:30. At the bottom, there are links for CloudWatch Metrics and Feedback.

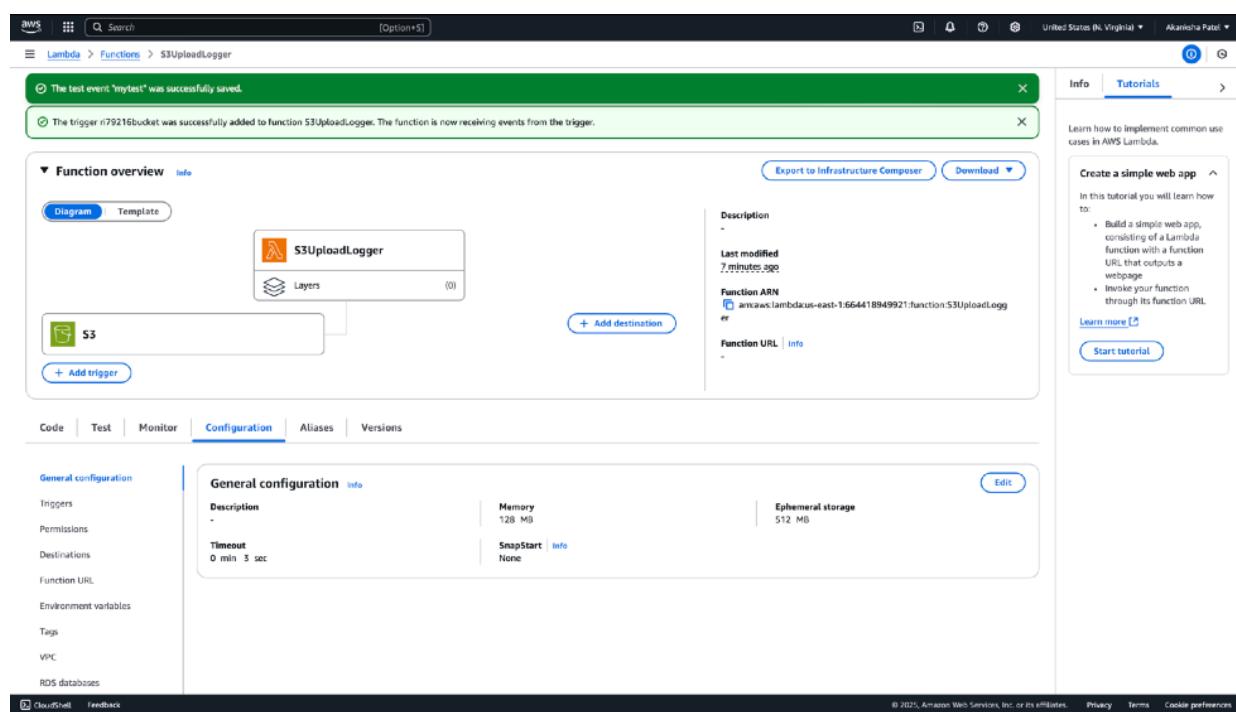
B. AWS Lambda for Logging S3 Uploads

Create an AWS Lambda function that logs S3 file uploads to CloudWatch Logs.

Step 1: Create the Lambda Function



Add S3 as a Trigger



Upload a File to S3

The screenshot shows the AWS S3 'Upload: status' page. At the top, a green banner displays a success message: 'Upload succeeded. For more information, see the Files and folders table.' Below this, a summary table shows one file uploaded successfully ('Succeeded') and zero files failed ('Failed'). The 'Files and folders' tab is selected, displaying a table with one row: 'file.txt.rtf' (Type: text/rtf, Size: 366.0 B, Status: Succeeded). A note at the top of the table area states: 'After you navigate away from this page, the following information is no longer available.'

View the Logs in CloudWatch

The screenshot shows the AWS CloudWatch 'Log groups' page. On the left, a navigation sidebar includes sections for CloudWatch (Favorites and recent, Dashboards, AI Operations, Alarms, Logs, Metrics, X-Ray traces, Events, Application Signals, Network Monitoring, Insights), Settings (Telemetry config, Getting Started, What's new), and CloudWatch shell (Feedback). The main content area shows a table of log groups. One group, '/aws/lambda/\$3UploadLogger', is selected and highlighted with a blue border. The table columns include Log group, Log class, Anomaly detection, Data protection, Sensitive data controls, Retention, Metric filters, and Contributor insights. The retention for the selected group is set to 'Never expire'. Other groups listed are '/aws/lambda/lambda-logger.lambda' and 'RDOSMetrics'.

The screenshot shows the AWS CloudWatch Log Events interface. The left sidebar includes links for CloudWatch, Favorites and recent, Dashboards, AI Operations, Alarms, Log groups, Log Anomalies, Live Tail, Logs Insights, Metrics, X-Ray traces, Events, Application Signals, Network Monitoring, Insights, Settings, Telemetry config, Getting Started, and What's new. The main content area displays log events for the 'S3UpLoadLogger' log group. The log entries are as follows:

- 2025-05-14T19:56:49.678Z INIT.START Runtime Version: python:3.9.v91 Runtime Version ARN: arn:aws:lambda:sous-east-1::runtime:c4127df9acf600313f596a2245bc413de98744c008c2b8a38ab334f663cd6
- 2025-05-14T19:56:49.945Z START RequestId: 77bd925-c93e-4fe8-8e0d03666ef Version: \$LATEST
- 2025-05-14T19:56:49.946Z [ERROR] 2025-05-14T19:56:49.946Z 77bd925-c93e-4fe8-8e0d03666ef Error processing S3 event: 'Records'
- 2025-05-14T19:56:49.967Z END RequestId: 77bd925-c93e-4fe8-8e0d03666ef Duration: 19.30 ms Billed Duration: 20 ms Memory Size: 128 MB Max Memory Used: 56 MB Init Duration: 206.72 ms
- 2025-05-14T19:56:49.967Z REPORT RequestId: 77bd925-c93e-4fe8-8e0d03666ef Duration: 19.30 ms Billed Duration: 20 ms Memory Size: 128 MB Max Memory Used: 56 MB Init Duration: 206.72 ms

No newer events at this moment. Auto retry paused. [Resume](#)

At the bottom right, there are links for CloudShell, Feedback, Privacy, Terms, and Cookie preferences.

C. Interaction with AWS

Use AWS Console to verify infrastructure deployment.

STEP 1: Verify EC2 Web Server

The screenshot shows the AWS EC2 Instances interface. The left sidebar includes links for EC2, Dashboard, EC2 Global View, Events, Instances, Instances Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations, Images, AMI Catalog, Elastic Block Store, Volumes, Snapshots, Lifecycle Manager, Network & Security, Security Groups, Elastic IPs, Placement Groups, Key Pairs, Network Interfaces, Load Balancing, Load Balancers, Target Groups, Trust Stores, and Auto Scaling, Auto Scaling Groups. The main content area displays a table of instances:

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 IP	Elastic IP
myinstance	I-0060827b4f74bfaef	Running	t2.micro	2/2 checks passed	View alarms	us-east-1a	-	-	-
CF-EC2-Instance	I-0a01f71f7d1812a54	Running	t2.micro	2/2 checks passed	View alarms	us-east-1a	ec2-107-21-193-206.co...	107.21.193.206	-
MyWebServer	I-066e93fb4a5989e1d	Running	t2.micro	2/2 checks passed	View alarms	us-east-1a	ec2-54-88-1-84.comput...	54.88.1.84	-
	I-0f6daf11694fd75bc	Running	t2.micro	2/2 checks passed	View alarms	us-east-1a	ec2-54-158-41-63.com...	54.158.41.63	-

Below the table, a modal window titled "Select an instance" is open, showing a list of instances with a "instances" button at the bottom.

At the bottom right, there are links for CloudShell, Feedback, Privacy, Terms, and Cookie preferences.

STEP 2: Verify S3 Bucket and File Uploads

The screenshot shows the AWS S3 console interface. On the left, there's a sidebar with navigation links like 'Amazon S3', 'General purpose buckets', 'Storage Lens', and 'AWS Marketplace for S3'. The main area is titled 'ri79216bucket' and shows a table of objects. There is one object listed: 'file.txt.rtf' (rtf type, 366.0 B size, last modified May 14, 2025, 16:05:12 UTC-04:00). At the top, there are several action buttons: 'Copy S3 URI', 'Copy URL', 'Download', 'Open', 'Delete', 'Actions', 'Create folder', and 'Upload'.

STEP 3: Verify Lambda Function and Logs

The screenshot shows the AWS Lambda console. On the left, there's a sidebar with 'Lambda' selected, followed by 'Dashboard', 'Applications', 'Functions', 'Additional resources', and 'Related AWS resources'. The main area is titled 'Functions (1/2)' and lists two functions: 'S3UploadLogger' (selected) and 'LogHelloLambda'. Both functions are in 'Zip' package type, Python 3.9 runtime, and were last modified 2 hours ago. To the right, there's a 'Tutorials' section with a 'Create a simple web app' card. The card describes the tutorial, lists steps (building a simple web app, invoking the function), and includes 'Learn more' and 'Start tutorial' buttons. The bottom of the page has standard AWS footer links: CloudShell, Feedback, Privacy, Terms, and Cookie preferences.

STEP 4: Verify Logs in CloudWatch

The screenshot shows the AWS CloudWatch Log groups interface. The left sidebar navigation includes 'CloudWatch' (selected), 'Dashboards', 'AI Operations', 'Alarms', 'Logs' (selected), 'Metrics', 'X-Ray traces', 'Events', 'Application Signals', 'Network Monitoring', and 'Insights'. Under 'Logs', there are links for 'Log groups' (New), 'Log Anomalies', 'Live Tail', 'Logs Insights' (New), and 'Contributor insights'. The main content area displays 'Log groups (1/3)' with a note: 'By default, we only load up to 10000 log groups.' A search bar says 'Filter log groups or try prefix search' with an 'Exact match' checkbox. The table lists three log groups:

Log group	Log class	Anomaly d...	Data protection	Sensitive data co...	Retention	Metric filters	Contributor insig...
/aws/lambda/1.logHelloLambda	Standard	Configure	-	-	Never expire	-	-
/aws/lambda/S3UploadLogger	Standard	Configure	-	-	Never expire	-	-
/DPSMetrics	Standard	Configure	-	-	1 month	-	-

At the bottom, there are links for 'CloudShell', 'Feedback', and 'What's new'. The top right corner shows 'United States (N. Virginia)', 'Akashika Patel', and other standard UI elements.

Use AWS CLI to interact with EC2, S3, and Lambda.

Step 1: EC2: List Running Instances

```
akankshapatel — ec2-user@ip-10-0-1-138:/var/www/html — -zsh — 80x22
...t-1.rds.amazonaws.com -u admin -p
...-0-1-138:/var/www/html — -zsh
~~-.- _/
_/_/
/_m/'_
Last login: Wed May 14 00:52:01 2025 from 73.212.60.215
[ec2-user@ip-10-0-1-138 ~]$ cd /var/www/html
[ec2-user@ip-10-0-1-138 html]$ sudo nano index.php
[ec2-user@ip-10-0-1-138 html]$ client_loop: send disconnect: Broken pipe
akankshapatel@Akankshas-MacBook-Pro-2 ~ % aws ec2 describe-instances \
--filters Name=instance-state-name,Values=running \
--query "Reservations[*].Instances[*].[InstanceId,InstanceType,PublicIpAddress
,State.Name]" \
--output table

-----
|           DescribeInstances           |
+-----+-----+-----+-----+
| i-0060827b4f24b9aef | t2.micro | None   | running |
| i-0a01f71f7d1812a54 | t2.micro | 107.21.193.206 | running |
| i-066e93fb4a5989e1d | t2.micro | 54.88.1.84   | running |
| i-0f6daf11694fd75bc | t2.micro | 54.158.41.63 | running |
+-----+-----+-----+-----+
akankshapatel@Akankshas-MacBook-Pro-2 ~ %
```

Step2: S3: Upload a File to a Bucket

```
Downloads — ec2-user@ip-10-0-1-138:/var/www/html — -zsh — 80x22
...t-1.rds.amazonaws.com -u admin -p ...0-1-138:/var/www/html — -zsh +
,State.Name]" \
--output table

-----+-----+-----+-----+
| i-0060827b4f24b9aef | t2.micro | None | running |
| i-0a01f71f7d1812a54 | t2.micro | 107.21.193.206 | running |
| i-066e93fb4a5989e1d | t2.micro | 54.88.1.84 | running |
| i-0f6daf11694fd75bc | t2.micro | 54.158.41.63 | running |
-----+
[akankshapatel@Akankshas-MacBook-Pro-2 ~ % aws s3 cp file.txt.rtf s3://ri79216bucket/
The user-provided path file.txt.rtf does not exist.
[akankshapatel@Akankshas-MacBook-Pro-2 ~ % cd Downloads
akankshapatel@Akankshas-MacBook-Pro-2 Downloads % aws s3 cp file.txt.rtf s3://ri79216bucket/
upload: ./file.txt.rtf to s3://ri79216bucket/file.txt.rtf
akankshapatel@Akankshas-MacBook-Pro-2 Downloads % ]
```

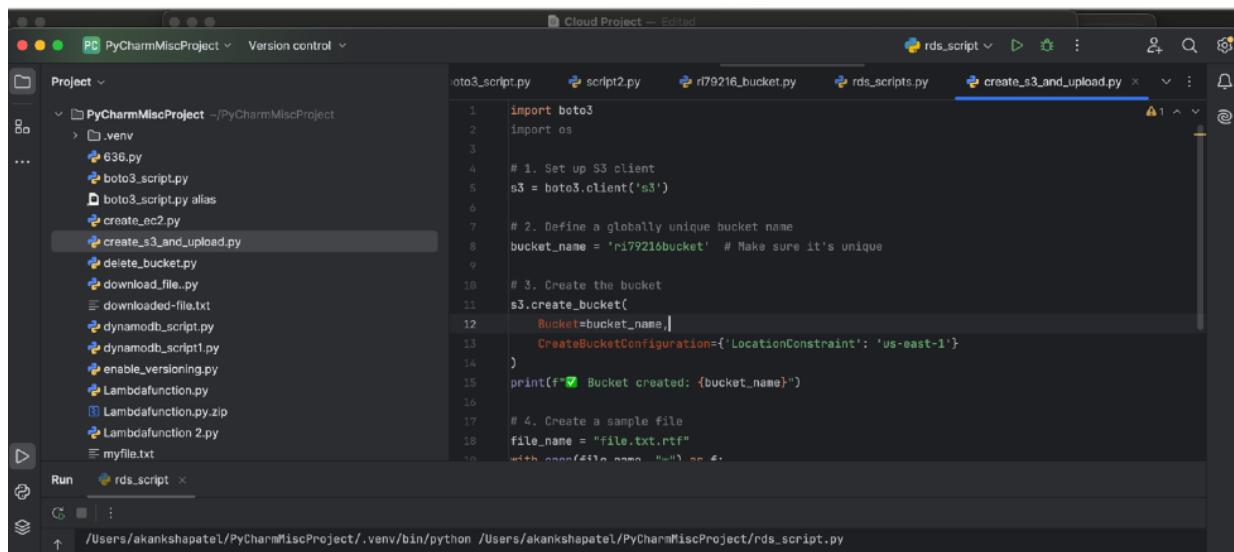
```
Downloads — ec2-user@ip-10-0-1-138:/var/www/html — -zsh — 88x22
east-1.rds.amazonaws.com -u admin -p ...ip-10-0-1-138:/var/www/html — -zsh +
-----+-----+-----+-----+
| 0060827b4f24b9aef | t2.micro | None | running |
| 01f71f7d1812a54 | t2.micro | 107.21.193.206 | running |
| 066e93fb4a5989e1d | t2.micro | 54.88.1.84 | running |
| 0fdaf11694fd75bc | t2.micro | 54.158.41.63 | running |
-----+
[apatel@Akankshas-MacBook-Pro-2 ~ % aws s3 cp file.txt.rtf s3://ri79216bucket/
The user-provided path file.txt.rtf does not exist.
[apatel@Akankshas-MacBook-Pro-2 ~ % cd Downloads
apatel@Akankshas-MacBook-Pro-2 Downloads % aws s3 cp file.txt.rtf s3://ri79216bucket/
upload: ./file.txt.rtf to s3://ri79216bucket/file.txt.rtf
apatel@Akankshas-MacBook-Pro-2 Downloads % aws s3 ls s3://ri79216bucket/
-14 17:37:15      366 file.txt.rtf
apatel@Akankshas-MacBook-Pro-2 Downloads % akaakaakanakanakaakakaakakaaaaaaa
apatel@Akankshas-MacBook-Pro-2 Downloads % ]
```

Step 3: Lambda

```
Downloads — ec2-user@ip-10-0-1-138:~ — ssh -i ~/Downloads/project_key.pem ec2-user@54.8...
[ec2-user@ip-10-0-1-138 ~]$ aws lambda list-functions
{
  "Functions": [
    {
      "FunctionName": "S3UploadLogger",
      "FunctionArn": "arn:aws:lambda:us-east-1:664418949921:function:S3UploadLogger",
      "Runtime": "python3.9",
      "Role": "arn:aws:iam::664418949921:role/service-role/S3UploadLogger-role-xims4k68",
      "Handler": "lambda_function.lambda_handler",
      "CodeSize": 454,
      "Description": "",
      "Timeout": 3,
      "MemorySize": 128,
      "LastModified": "2025-05-14T19:52:06.000+0000",
      "CodeSha256": "QUJ5FXLQcda8Pynjx3WTxj+YPOzT/XJvmp9YE9xRuZY=",
      "Version": "$LATEST",
      "TracingConfig": {
        "Mode": "PassThrough"
      },
      "RevisionId": "8474f9fd-b537-47a0-bb65-4661cf43436d",
      "PackageType": "Zip",
      "Architectures": [
        "x86_64"
      ],
    }
  ]
}
```

Write Python scripts using Boto3 to:

1. Create an S3 bucket and upload a file.



The screenshot shows the PyCharm IDE interface with a 'Cloud Project' selected. The project structure on the left includes files like .venv, 636.py, boto3.script.py, boto3.script.py alias, create_ec2.py, create_s3_and_upload.py, delete_bucket.py, download_file.py, downloaded-file.txt, dynamodb_script.py, dynamodb_script1.py, enable_versioning.py, Lambdafunction.py, Lambdafunction.py.zip, Lambdafunction 2.py, and myfile.txt. The 'create_s3_and_upload.py' script is open in the main editor window. The code uses Boto3 to create an S3 bucket and upload a file:

```
import boto3
import os

# 1. Set up S3 client
s3 = boto3.client('s3')

# 2. Define a globally unique bucket name
bucket_name = 'ri79216bucket' # Make sure it's unique

# 3. Create the bucket
s3.create_bucket(
    Bucket=bucket_name,
    CreateBucketConfiguration={'LocationConstraint': 'us-east-1'}
)
print(f"Bucket created: {bucket_name}")

# 4. Create a sample file
file_name = "file.txt.rtf"
with open(file_name, "w") as f:
    f.write("Hello, World!")

# 5. Upload the file to the bucket
s3.upload_file(file_name, bucket_name, file_name)
```

The 'Run' tab at the bottom shows the command: /Users/akankshapatel/PyCharmMiscProject/.venv/bin/python /Users/akankshapatel/PyCharmMiscProject/rds_script.py

```

Downloads — zsh — 93x24
ID: 6, Name: Charlie, Email: charlie@example.com
ID: 7, Name: Diana, Email: diana@example.com
ID: 8, Name: Eve, Email: eve@example.com
akankshapatel@Akankshas-MacBook-Pro-2 ~ % python3 create_s3_and_upload.py
[ /Library/Frameworks/Python.framework/Versions/3.13/Resources/Python.app/Contents/MacOS/Python
: can't open file '/Users/akankshapatel/create_s3_and_upload.py': [Errno 2] No such file or directory
akankshapatel@Akankshas-MacBook-Pro-2 ~ % cd ~/Downloads

akankshapatel@Akankshas-MacBook-Pro-2 Downloads % ls create_s3_and_upload.py

ls: create_s3_and_upload.py: No such file or directory
akankshapatel@Akankshas-MacBook-Pro-2 Downloads % ls *.py

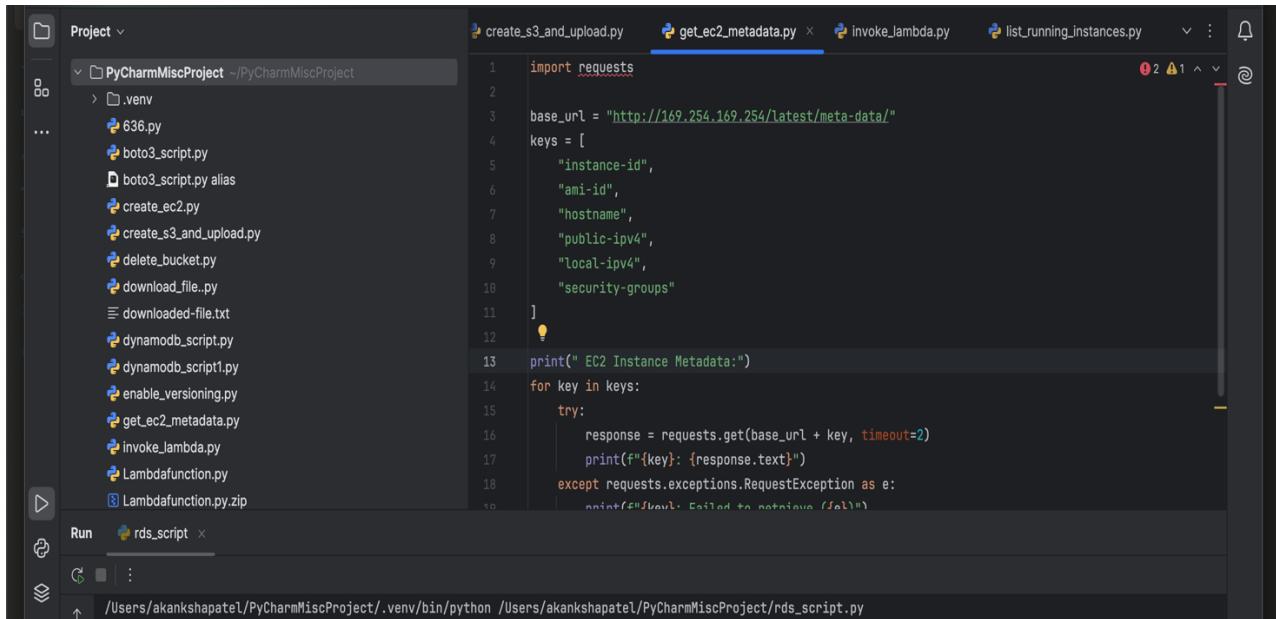
cli_lambda.py      prime_factorization.py  user_insert.py
dynamo_script.py   user_details.py
akankshapatel@Akankshas-MacBook-Pro-2 Downloads % nano create_s3_and_upload.py

akankshapatel@Akankshas-MacBook-Pro-2 Downloads % python3 create_s3_and_upload.py

✓ File 'file.txt.rtf' created or found locally.
✓ File 'file.txt.rtf' uploaded to bucket 'ri79216bucket'
akankshapatel@Akankshas-MacBook-Pro-2 Downloads %

```

2. Retrieve EC2 instance metadata.



```

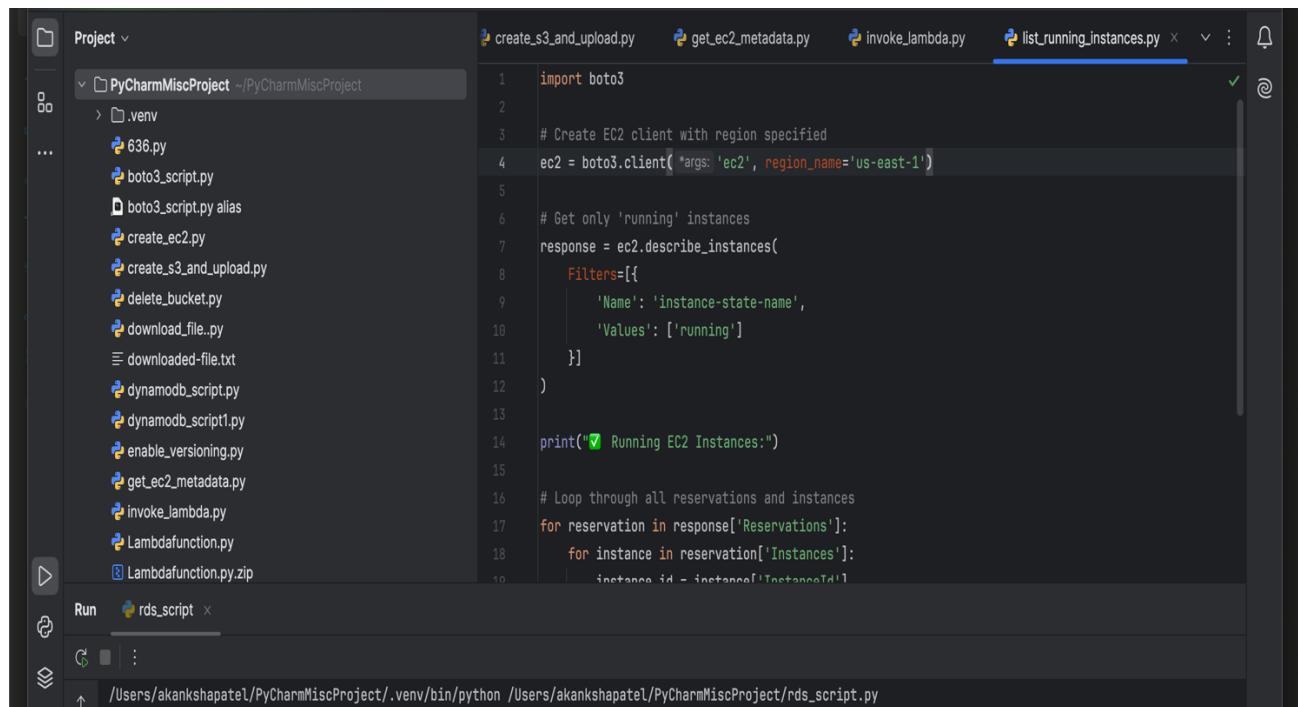
Project
PyCharmMiscProject ~/PyCharmMiscProject
  .venv
    636.py
    boto3_script.py
    boto3_script.py alias
    create_ec2.py
    create_s3_and_upload.py
    delete_bucket.py
    download_file.py
    downloaded-file.txt
    dynamodb_script.py
    dynamodb_script1.py
    enable_versioning.py
    get_ec2_metadata.py
    invoke_lambda.py
    Lambdafunction.py
    Lambdafunction.py.zip
Run rds_script x
Run : 
  /Users/akankshapatel/PyCharmMiscProject/.venv/bin/python /Users/akankshapatel/PyCharmMiscProject/rds_script.py

1 import requests
2
3 base_url = "http://169.254.169.254/latest/meta-data/"
4 keys = [
5     "instance-id",
6     "ami-id",
7     "hostname",
8     "public-ipv4",
9     "local-ipv4",
10    "security-groups"
11 ]
12
13 print(" EC2 Instance Metadata:")
14 for key in keys:
15     try:
16         response = requests.get(base_url + key, timeout=2)
17         print(f"{key}: {response.text}")
18     except requests.exceptions.RequestException as e:
19         print(f"Error: Failed to retrieve {key}:")


```

```
A newer release of "Amazon Linux" is available.
Version 2023.7.20250512:
Run "/usr/bin/dnf check-release-update" for full release and version update info
'      #_
~\_\_ #####_      Amazon Linux 2023
~~ \_\#\#\#\`_
~~ \#\#\#|
~~ \#/--- https://aws.amazon.com/linux/amazon-linux-2023
~~ V~' '-->
~~ /_
~~ ._. /_
~/m/
Last login: Wed May 14 16:46:46 2025 from 73.212.60.215
[ec2-user@ip-10-0-1-138 ~]$ nano get_ec2_metadata.py
[ec2-user@ip-10-0-1-138 ~]$ python3 get_ec2_metadata.py
✓ EC2 Instance Metadata:
instance-id:
ami-id:
hostname:
public-ipv4:
local-ipv4:
security-groups:
[ec2-user@ip-10-0-1-138 ~]$
```

3. List running EC2 instance



The screenshot shows the PyCharm IDE interface with the following details:

- Project:** PyCharmMiscProject
- Files:** The project contains several Python files: create_s3_and_upload.py, get_ec2_metadata.py, invoke_lambda.py, and list_running_instances.py (which is currently open). There are also .venv, 636.py, boto3.script.py, and a Lambdafunction.py file.
- Code Editor:** The code for list_running_instances.py is displayed:

```
import boto3

# Create EC2 client with region specified
ec2 = boto3.client("ec2", region_name='us-east-1')

# Get only 'running' instances
response = ec2.describe_instances(
    Filters=[{
        'Name': 'instance-state-name',
        'Values': ['running']
    }]
)

print("✓ Running EC2 Instances:")

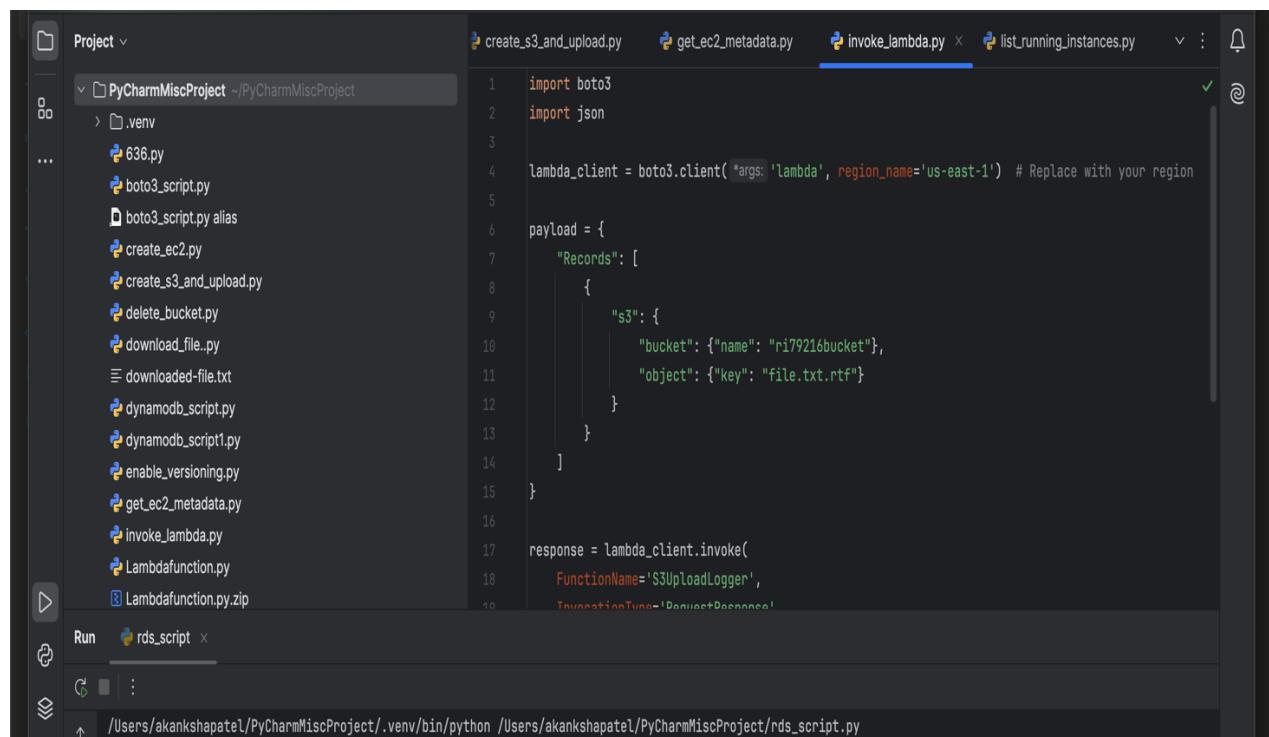
# Loop through all reservations and instances
for reservation in response['Reservations']:
    for instance in reservation['Instances']:
        instance_id = instance['InstanceId']
```
- Run Tab:** The 'rds_script' run configuration is selected.
- Status Bar:** The status bar at the bottom shows the command: /Users/akankshapate1/PyCharmMiscProject/.venv/bin/python /Users/akankshapate1/PyCharmMiscProject/rds_script.py

```

Downloads — ec2-user@ip-10-0-1-138:~ — ssh -i ~/Downloads/project_key.pem ec2-user@54.8...
dpoint_config
    return self._resolve_endpoint(**resolve_endpoint_kwargs)
  File "/home/ec2-user/.local/lib/python3.9/site-packages/botocore/args.py", line 537, in _resolve_en
dpoint
    return endpoint_bridge.resolve(
  File "/home/ec2-user/.local/lib/python3.9/site-packages/botocore/client.py", line 655, in resolve
    resolved = self.endpoint_resolver.construct_endpoint(
  File "/home/ec2-user/.local/lib/python3.9/site-packages/botocore/regions.py", line 233, in construc
t_endpoint
    result = self._endpoint_for_partition(
  File "/home/ec2-user/.local/lib/python3.9/site-packages/botocore/regions.py", line 281, in _endpoi
t_for_partition
    raise NoRegionError()
botocore.exceptions.NoRegionError: You must specify a region.
[ec2-user@ip-10-0-1-138 ~]$ nano list_running_instances.py
[ec2-user@ip-10-0-1-138 ~]$ python3 list_running_instances.py
Running EC2 Instances:
- ID: i-0060827b4f24b9aef, Type: t2.micro, Public IP: N/A
- ID: i-0a01f71f7d1812a54, Type: t2.micro, Public IP: 107.21.193.206
- ID: i-066e93fb4a5989e1d, Type: t2.micro, Public IP: 54.88.1.84
- ID: i-0f6daf11694fd75bc, Type: t2.micro, Public IP: 54.158.41.63
[ec2-user@ip-10-0-1-138 ~]$ nano invoke_lambda.py
[ec2-user@ip-10-0-1-138 ~]$ python3 invoke_lambda.py
Traceback (most recent call last):

```

4. Invoke the AWS Lambda function manually.



The screenshot shows the PyCharm interface with the 'invoke_lambda.py' file open in the editor. The code uses the boto3 library to invoke a Lambda function named 'S3UploadLogger'. The Lambda function is triggered by an S3 event, specifically for an object named 'file.txt.rtf' in a bucket named 'ri79216bucket'. The code also handles the response from the Lambda function.

```

import boto3
import json

lambda_client = boto3.client("lambda", region_name='us-east-1') # Replace with your region

payload = {
    "Records": [
        {
            "s3": {
                "bucket": {"name": "ri79216bucket"},
                "object": {"key": "file.txt.rtf"}
            }
        }
    ]
}

response = lambda_client.invoke(
    FunctionName='S3UploadLogger',
    InvocationType='RequestResponse'
)

```

```

Downloads — ec2-user@ip-10-0-1-138:~ — ssh -i ~/Downloads/project_key.pem ec2-user@54.8...
- ID: i-0060827b4f24b9aef, Type: t2.micro, Public IP: N/A
- ID: i-0a01f71f7d1812a54, Type: t2.micro, Public IP: 107.21.193.206
- ID: i-066e93fb4a5989e1d, Type: t2.micro, Public IP: 54.88.1.84
- ID: i-0f6daf11694fd75bc, Type: t2.micro, Public IP: 54.158.41.63
[ec2-user@ip-10-0-1-138 ~]$ nano invoke_lambda.py
[ec2-user@ip-10-0-1-138 ~]$ python3 invoke_lambda.py
Traceback (most recent call last):
  File "/home/ec2-user/invoke_lambda.py", line 17, in <module>
    response = lambda_client.invoke(
  File "/home/ec2-user/.local/lib/python3.9/site-packages/botocore/client.py", line 595, in _api_call
    return self._make_api_call(operation_name, kwargs)
  File "/home/ec2-user/.local/lib/python3.9/site-packages/botocore/context.py", line 123, in wrapper
    return func(*args, **kwargs)
  File "/home/ec2-user/.local/lib/python3.9/site-packages/botocore/client.py", line 1058, in _make_api_call
    raise error_class(parsed_response, operation_name)
botocore.exceptions.ClientError: An error occurred (AccessDeniedException) when calling the Invoke operation: User: arn:aws:sts::664418949921:assumed-role/myrole/i-066e93fb4a5989e1d is not authorized to perform: lambda:InvokeFunction on resource: arn:aws:lambda:us-east-1:664418949921:function:S3UploadLogger because no identity-based policy allows the lambda:InvokeFunction action
[ec2-user@ip-10-0-1-138 ~]$ python3 invoke_lambda.py
Lambda response:
{"statusCode": 200, "body": "\"New file uploaded: file.txt.rtf in bucket: ri79216bucket\""}
[ec2-user@ip-10-0-1-138 ~]$ █

```

Report and Documentation

1. Architecture Diagram explaining the setup.

1. VPC (Virtual Private Cloud)

- A custom isolated network where all AWS resources live.
- Divided into:
 - Public Subnets – Can access internet
 - Private Subnets – No direct internet access

All EC2s, ALBs, RDS, and NAT routing happen inside this VPC.

2. Internet Gateway + Route Table

- Internet Gateway (IGW):
 - Connects your VPC to the internet
 - Required for any public-facing resources (like ALB, EC2)
- Route Table:
 - Routes traffic:
 - 0.0.0.0/0 → IGW for public subnet (Internet access)
 - Private subnet route does not include IGW

3. Public Subnet

a. ALB (Application Load Balancer)

- Distributes incoming HTTP/HTTPS traffic to:
 - Auto Scaling Group (EC2 app servers)
 - Lambda (in ALB-Lambda integration)

b. ASG (Auto Scaling Group)

- Maintains minimum and maximum EC2 instances based on demand
- Launches EC2s in multiple AZs
- Registers EC2s behind the ALB

4. Private Subnet

a. RDS MySQL

- Fully managed relational database
- Deployed in a private subnet (no internet access)
- Only EC2s or Lambdas in VPC can connect

This ensures high security — RDS can't be attacked directly from the internet.

5. Amazon S3 Buckets

a. Frontend Hosting S3 Bucket

- Stores static frontend files
- Optionally connected to CloudFront + API Gateway for public delivery

b. Data Upload S3 Bucket

- File uploads
- Trigger Lambda function for processing

6. Lambda Functions

a. Triggered by S3

- When a new object is uploaded, S3 sends an event to Lambda
- Lambda:
 - Reads file

- Parses or processes it
- Stores result in RDS or another S3 bucket

b. API Lambda

- Also invoked through API Gateway (explained next)

7. API Gateway

- Fully managed API layer
- Exposes REST or HTTP endpoints to the public
- Routes traffic to:
 - Lambda functions
 - ALB/EC2 backend
 - S3 static websites

8. Automation & DevOps Integration

a. GitHub Actions

- CI/CD pipeline:
 - On git push, GitHub runs tests
 - Builds code, zips artifacts
 - Deploys via Code Deploy or updates Lambda

b. AWS Code Deploy

- Handles automated deployments:
 - EC2 code (via install scripts)
 - Lambda function versioning
 - Blue/green and canary deployments

c. AWS Step Functions

- Orchestrates Lambda and API tasks
- Example: A data pipeline triggered by S3 upload, processed in 3 steps:
 - Parse → Validate → Store
- Can retry failed steps and define workflows

2. Use Terraform to create networking components (VPC, subnets, security groups).

Step 1: Create Project Directory

```
mkdir aws-networking
```

```
cd aws-networking
```

Step 2: Create main.tf with the following Terraform code

```
provider "aws" {
    region = "us-east-1"
}

# VPC
resource "aws_vpc" "main" {
    cidr_block      = "10.0.0.0/16"
    enable_dns_support = true
    enable_dns_hostnames = true
    tags = {
        Name = "main-vpc"
    }
}

# Internet Gateway
resource "aws_internet_gateway" "igw" {
    vpc_id = aws_vpc.main.id
    tags = {
        Name = "main-igw"
    }
}

# Public Subnet
resource "aws_subnet" "public" {
    vpc_id          = aws_vpc.main.id
    cidr_block      = "10.0.1.0/24"
    map_public_ip_on_launch = true
    availability_zone = "us-east-1a"
    tags = {
        Name = "public-subnet"
    }
}

# Private Subnet
```

```

resource "aws_subnet" "private" {
  vpc_id      = aws_vpc.main.id
  cidr_block   = "10.0.2.0/24"
  availability_zone = "us-east-1a"
  tags = {
    Name = "private-subnet"
  }
}

# Public Route Table and Route
resource "aws_route_table" "public" {
  vpc_id = aws_vpc.main.id
  tags = {
    Name = "public-rt"
  }
}

resource "aws_route" "public_internet" {
  route_table_id      = aws_route_table.public.id
  destination_cidr_block = "0.0.0.0/0"
  gateway_id          = aws_internet_gateway.igw.id
}

resource "aws_route_table_association" "public_assoc" {
  subnet_id     = aws_subnet.public.id
  route_table_id = aws_route_table.public.id
}

# NAT Gateway
resource "aws_eip" "nat" {
  vpc = true
}

resource "aws_nat_gateway" "natgw" {
  allocation_id = aws_eip.nat.id
  subnet_id    = aws_subnet.public.id
  depends_on   = [aws_internet_gateway.igw]
  tags = {
    Name = "nat-gateway"
  }
}

```

```

# Private Route Table and Route
resource "aws_route_table" "private" {
  vpc_id = aws_vpc.main.id
  tags = {
    Name = "private-rt"
  }
}

resource "aws_route" "private_nat" {
  route_table_id      = aws_route_table.private.id
  destination_cidr_block = "0.0.0.0/0"
  nat_gateway_id      = aws_nat_gateway.natgw.id
}

resource "aws_route_table_association" "private_assoc" {
  subnet_id      = aws_subnet.private.id
  route_table_id = aws_route_table.private.id
}

# Security Group for EC2 (allow SSH)
resource "aws_security_group" "ec2_sg" {
  name      = "ec2-sg"
  description = "Allow SSH"
  vpc_id    = aws_vpc.main.id

  ingress {
    from_port  = 22
    to_port    = 22
    protocol   = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    from_port  = 0
    to_port    = 0
    protocol   = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
    Name = "ec2-sg"
  }
}

```

}

Step 3: Initialize and Deploy

A. Initialize Terraform
terraform init

B. See the plan
terraform plan

C. Apply the configuration
terraform apply

Confirm with yes when prompted.

Step 4: Output and Verify

- Go to AWS Console → VPC
 - Verify:
 - VPC
 - Subnets (public/private)
 - Internet Gateway
 - NAT Gateway
 - Route Tables
 - Security Group

3. Use CloudFormation to deploy EC2 instances, RDS, and Lambda functions

PART 1: Deploy EC2 Instance

Step 1: Create a CloudFormation Template ec2-instance.yaml

AWSTemplateFormatVersion: '2010-09-09'

Description: Deploy an EC2 instance using specified AMI and key pair

Resources:

 MyVPC:

 Type: AWS::EC2::VPC

 Properties:

 CidrBlock: 10.0.0.0/16

 Tags:

 - Key: Name

 Value: MyVPC

MySubnet:

Type: AWS::EC2::Subnet

Properties:

VpcId: !Ref MyVPC

CidrBlock: 10.0.1.0/24

MapPublicIpOnLaunch: true

AvailabilityZone: !Select [0, !GetAZs "]

Tags:

- Key: Name

- Value: MySubnet

MyInternetGateway:

Type: AWS::EC2::InternetGateway

AttachGateway:

Type: AWS::EC2::VPCGatewayAttachment

Properties:

VpcId: !Ref MyVPC

InternetGatewayId: !Ref MyInternetGateway

MyRouteTable:

Type: AWS::EC2::RouteTable

Properties:

VpcId: !Ref MyVPC

DefaultRoute:

Type: AWS::EC2::Route

DependsOn: AttachGateway

Properties:

RouteTableId: !Ref MyRouteTable

DestinationCidrBlock: 0.0.0.0/0

GatewayId: !Ref MyInternetGateway

SubnetRouteTableAssociation:

Type: AWS::EC2::SubnetRouteTableAssociation

Properties:

SubnetId: !Ref MySubnet

RouteTableId: !Ref MyRouteTable

MySecurityGroup:

Type: AWS::EC2::SecurityGroup

Properties:

GroupDescription: Allow SSH access

VpcId: !Ref MyVPC

SecurityGroupIngress:

- IpProtocol: tcp

FromPort: 22

ToPort: 22

CidrIp: 0.0.0.0/0

MyEC2Instance:

Type: AWS::EC2::Instance

Properties:

InstanceType: t2.micro

ImageId: ami-0953476d60561c955

KeyName: project_key

NetworkInterfaces:

- AssociatePublicIpAddress: true

SubnetId: !Ref MySubnet

DeviceIndex: 0

GroupSet:

- !Ref MySecurityGroupStep 2: Deploy via Console

1. Go to AWS CloudFormation → **Create Stack**

2. Choose Upload a template file

3. Upload ec2-instance.yaml

4. Click **Next**, name the stack (e.g., EC2Stack)

5. Click Next → Next → Acknowledge → Create Stack

Output:

- A running EC2 instance in your public subnet
- Accessible via SSH:

ssh -i project_key.pem ec2-user@<public-ip>

- HTML page available at <http://<public-ip>>

PART 2: Deploy RDS MySQL Instance

Step 1: Create CloudFormation Template rds-mysql.yaml

AWSTemplateFormatVersion: '2010-09-09'

Description: RDS MySQL Instance

Resources:

MyDBSubnetGroup:

Type: AWS::RDS::DBSubnetGroup

Properties:

DBSubnetGroupDescription: Subnets for RDS

SubnetIds:

- subnet-private-1 # From Terraform

- subnet-private-2 # Optional second AZ

MyRDSInstance:

Type: AWS::RDS::DBInstance

Properties:

DBInstanceIdentifier: mymysql-db

DBInstanceClass: db.t3.micro

Engine: mysql

EngineVersion: 8.0.35

AllocatedStorage: 20

MasterUsername: admin

MasterUserPassword: Anchal#10

VPCSecurityGroups:

- sg-rds-id # Only accessible from EC2 SG

DBSubnetGroupName: !Ref MyDBSubnetGroup

PubliclyAccessible: false

Step 2: Deploy via Console

1. Go to CloudFormation → Create Stack
2. Upload rds-mysql.yaml
3. Click through wizard → Name stack RDSStack
4. Confirm → Create Stack

Output:

- RDS instance deployed in private subnet
- Accessible only from EC2 inside VPC:

```
mysql -h <endpoint> -u admin -p
```

PART 3: Deploy Lambda Function

Step 1: Create lambda-function.yaml

AWSTemplateFormatVersion: '2010-09-09'

Description: Lambda Hello World

Resources:

LambdaExecutionRole:

Type: AWS::IAM::Role

Properties:

RoleName: LambdaExecutionRole

AssumeRolePolicyDocument:

Version: '2012-10-17'

Statement:

- Effect: Allow

Principal:

Service: lambda.amazonaws.com

Action: sts:AssumeRole

ManagedPolicyArns:

- arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole

HelloWorldFunction:

Type: AWS::Lambda::Function

Properties:

FunctionName: HelloWorldFunction

Runtime: python3.9

Handler: index.lambda_handler

Timeout: 10

Role: !GetAtt LambdaExecutionRole.Arn

Code:

ZipFile: |

```
def lambda_handler(event, context):
```

```
    return {
```

```
        'statusCode': 200,
```

```
        'body': 'Hello from Lambda!'
```

```
}
```

Step 2: Deploy

1. Go to CloudFormation → Create Stack
2. Upload lambda-function.yaml
3. Name it LambdaStack

4. Click Create Stack

Step 3: Test Lambda

1. Go to **AWS Lambda**
2. Select HelloWorldFunction
3. Click **Test** → use default test event
4. Click Test again

Output:

```
{  
  "statusCode": 200,  
  "body": "Hello from Lambda!"  
}
```

4. Deploy a web application on EC2 (can be a simple HTML page).

Step 1: Prerequisites

- A VPC and public subnet were already created using Terraform
- A security group was configured to allow:
 - SSH access (TCP port 22)
 - HTTP access (TCP port 80)
- A Key Pair (project_key.pem) was available for SSH

Step 2: Create CloudFormation Template

A CloudFormation template ec2-webapp.yaml was created with:

- An EC2 instance resource using the Amazon Linux 2 AMI
- Apache Web Server installed and started via a User Data script
- An HTML file (index.html) created automatically in /var/www/html/

Key Code Snippet:

UserData:

```
Fn::Base64: |  
#!/bin/bash  
yum update -y  
yum install -y httpd  
systemctl start httpd
```

```
systemctl enable httpd  
echo "<h1>Hello from EC2 Web Server</h1>" > /var/www/html/index.html
```

Step 3: Deploy the Stack

1. Opened AWS Console → CloudFormation
2. Selected **Create Stack** → With new resources
3. Uploaded ec2-webapp.yaml
4. Provided parameter: project_key as Key Pair
5. Clicked **Create Stack** and monitored the stack status
6. Stack status changed to: CREATE_COMPLETE

Step 4: Verify the EC2 Instance

- Navigated to **EC2 > Instances**
- Located the instance deployed by CloudFormation
- Copied its Public IPv4 address

Step 5: Test the Web Application

- Opened a browser and visited:

<http://<public-ip>>

Output:

```
pgsql  
CopyEdit  
Hello from EC2 Web Server
```

5. Configure the database for the web application.

Step 1: Prerequisites

- A RDS MySQL instance was deployed using CloudFormation (rds-mysql.yaml)
- The RDS instance was:
 - Placed in a private subnet
 - Attached to a security group that allows access from the EC2 security group
- The EC2 web server is already running in the **public subnet** and is configured to allow outbound access to RDS

Step 2: Connect to RDS from EC2

1. SSH into EC2:

```
ssh -i project_key.pem ec2-user@<ec2-public-ip>
```

2. Install MySQL client on EC2:

```
sudo yum install mysql -y
```

3. Connect to RDS:

```
mysql -h <rds-endpoint> -u admin -p  
    o Password: *****
```

Connection successful confirms that networking, security groups, and routing are correctly configured.

Step 3: Create Database and Table

Once connected to MySQL shell:

```
CREATE DATABASE myappdb;  
USE myappdb;
```

```
CREATE TABLE users (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    name VARCHAR(100),  
    email VARCHAR(100),  
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);
```

The schema is now ready for use by the web app.

◆ Step 5: Validate Data Operations

- Insert data:

```
INSERT INTO users (name, email) VALUES ('Akanksha',  
'akanksha@example.com');
```

- Query data:

```
SELECT * FROM users;
```

6. Implement autoscaling to manage web server load.

Step 1: Prerequisites

- A VPC, public subnet, internet gateway, and security group were already provisioned using Terraform
- A key pair (project_key.pem) is available
- A simple web app (index.html) is already tested on a standalone EC2 instance

Step 2: Create a Launch Template

1. Go to EC2 Console → Launch Templates → Create launch template
2. Set values:
 - Name: WebServerTemplate
 - AMI: ami-0953476d60561c955
 - Instance type: t2.micro
 - Key pair: project_key
 - Security group: Allow HTTP (port 80) and SSH (port 22)
3. Under Advanced details → User Data, add:

```
#!/bin/bash
yum update -y
yum install -y httpd
systemctl start httpd
echo "<h1>Hello from Auto Scaling EC2</h1>" > /var/www/html/index.html
```

4. Click Create launch template

The launch template defines the EC2 configuration that will be used by the Auto Scaling Group.

Step 3: Create an Application Load Balancer (ALB)

1. Go to EC2 → Load Balancers → Create Load Balancer
2. Choose Application Load Balancer
3. Name: WebALB
4. Scheme: Internet-facing

5. Listener: HTTP on port 80
6. Availability Zones: select at least 2 public subnets
7. Security group: allow HTTP (port 80)
8. Target group:
 - Name: WebTargetGroup
 - Target type: instance
 - Protocol: HTTP
 - Port: 80
 - Health check path: /
9. Click Create

The ALB will distribute traffic across EC2 instances and handle health checks.

Step 4: Create an Auto Scaling Group (ASG)

1. Go to EC2 → Auto Scaling Groups → Create Auto Scaling Group
2. Name: WebASG
3. Launch template: WebServerTemplate
4. VPC: Select your VPC
5. Subnets: Select public subnets
6. Attach to existing ALB:
 - Target Group: WebTargetGroup
7. Set desired capacity:
 - Desired: 1
 - Min: 1
 - Max: 3
8. Scaling policy:
 - Choose Target tracking scaling policy
 - Metric: Average CPU utilization
 - Target value: 60%
9. Click Create Auto Scaling Group

ASG automatically launches new EC2 instances based on CPU utilization.

Step 5: Test Autoscaling

1. Get the DNS name of the ALB
2. Open in browser <http://<ALB-DNS-name>>
3. We should see: Hello from Auto Scaling.

B. AWS Lambda for Logging S3 Uploads

Step 1: Prerequisites

- An Amazon S3 bucket was already created
- We have AWS IAM permissions to create Lambda functions and attach event triggers
- CloudWatch Logs permissions are included in the Lambda role

Step 2: Write the Lambda Function (Python 3.9)

Function name: LogS3Uploads

Code:

```
import json
import boto3
import logging

# Set up logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    try:
        bucket_name = event['Records'][0]['s3']['bucket']['name']
        object_key = event['Records'][0]['s3']['object']['key']

        log_message = f"New file uploaded: {object_key} in bucket: {bucket_name}"
        logger.info(log_message)

        return {
            "statusCode": 200,
            "body": json.dumps(log_message)
        }
    except Exception as e:
        logger.error(f"Error processing S3 event: {str(e)}")
        return {
            "statusCode": 500,
            "body": json.dumps(f"Error: {str(e)}")
        }
```

Step 3: Create IAM Role for Lambda

1. Go to IAM > Roles → Create role
2. Choose Lambda as the trusted entity
3. Attach these policies:
 - AWSLambdaBasicExecutionRole (CloudWatch Logs)
4. Name it: LambdaS3LoggingRole

Step 4: Create Lambda Function

1. Go to AWS Lambda → Create function
2. Choose:
 - Author from scratch
 - Function name: LogS3Uploads
 - Runtime: Python 3.9
 - Execution Role: Use an existing role → LambdaS3LoggingRole
3. Paste the function code into the editor
4. Click Deploy

Step 5: Add S3 Trigger to Lambda

1. Under Lambda → Triggers, click Add trigger
2. Choose S3
3. Bucket: my-upload-logs-bucket (your existing bucket)
4. Event type: PUT (All object creates events)
5. Prefix/suffix: *(optional)*
6. Check the box to acknowledge permission changes
7. Click Add

Step 6: Test the Setup

1. Go to S3 → my-upload-logs-bucket
2. Click Upload → upload a test file (e.g., sample.txt)
3. Wait a few seconds

Step 7: Verify CloudWatch Logs

1. Go to CloudWatch → Logs → Log groups
2. Find: /aws/lambda/LogS3Uploads

3. Open the latest log stream
4. Output: INFO:root:New file uploaded: sample.txt in bucket: my-upload-logs-bucket

C. Interaction with AWS

Use AWS CLI for Basic Operations

1. List EC2 Instances

```
aws ec2 describe-instances --query "Reservations[*].Instances[*].InstanceId" --output text
```

Output: List of EC2 Instance IDs currently running

2. List S3 Buckets

```
aws s3 ls
```

Output: List of existing S3 buckets

3. Invoke Lambda Function Manually

```
aws lambda invoke \
--function-name LogS3Uploads \
--payload '{}' \
output.json
```

Output saved to output.json, viewable with:

C. Python Scripts Using Boto3

All scripts were written using boto3 to programmatically manage AWS resources.

1. Create S3 Bucket & Upload a File

```
import boto3
import os

# 1. Set up S3 client
s3 = boto3.client('s3')

# 2. Define a globally unique bucket name
bucket_name = 'ri79216bucket' # Make sure it's unique

# 3. Create the bucket
s3.create_bucket(
    Bucket=bucket_name,
    CreateBucketConfiguration={'LocationConstraint': 'us-east-1'})
```

```

)
print(f" Bucket created: {bucket_name}")

# 4. Create a sample file
file_name = "file.txt.rtf"
with open(file_name, "w") as f:
    f.write("Hello from Boto3!")

# 5. Upload file to S3
s3.upload_file(file_name, bucket_name, file_name)
print(f" File '{file_name}' uploaded to bucket '{bucket_name}'")

```

2. Retrieve EC2 Metadata from Instance (runs on EC2)

import requests

```

import requests

base_url = "http://169.254.169.254/latest/meta-data/"
keys = [
    "instance-id",
    "ami-id",
    "hostname",
    "public-ipv4",
    "local-ipv4",
    "security-groups"
]

print(" EC2 Instance Metadata:")
for key in keys:
    try:
        response = requests.get(base_url + key, timeout=2)
        print(f"{key}: {response.text}")
    except requests.exceptions.RequestException as e:
        print(f"{key}: Failed to retrieve ({e})")
5

```

3. List Running EC2 Instances

import boto3

```

import boto3

# Create EC2 client with region specified
ec2 = boto3.client('ec2', region_name='us-east-1')

# Get only 'running' instances
response = ec2.describe_instances(
    Filters=[{
        'Name': 'instance-state-name',
        'Values': ['running']
    }]
)

print(" Running EC2 Instances:")

# Loop through all reservations and instances
for reservation in response['Reservations']:

```

```

for instance in reservation['Instances']:
    instance_id = instance['InstanceId']
    instance_type = instance['InstanceType']
    public_ip = instance.get('PublicIpAddress', 'N/A')
    print(f" - ID: {instance_id}, Type: {instance_type}, Public IP: {public_ip}")

```

4. Manually Invoke Lambda Function

```

import boto3
import json

lambda_client = boto3.client('lambda', region_name='us-east-1') # Replace with your region

payload = {
    "Records": [
        {
            "s3": {
                "bucket": {"name": "ri79216bucket"},
                "object": {"key": "file.txt.rtf"}
            }
        }
    ]
}

response = lambda_client.invoke(
    FunctionName='S3UploadLogger',
    InvocationType='RequestResponse',
    Payload=json.dumps(payload),
)

response_payload = response['Payload'].read().decode()
print("Lambda response:")
print(response_payload)

```

. No Challenges Faced

. Github link - <https://github.com/Akanksha576/cloud-project>