# Part 1: Environment Setup

Question: Why might IAM not require a region parameter?

- IAM is a global (account-level) service, so its resources and control plane are not scoped to a single AWS region; therefore `boto3.client('iam')` does not need a `region_name`.
- IAM roles, users, groups, and instance profiles exist at the account level rather than inside a specific region, so API calls operate against the global IAM control plane. Some SDKs or endpoints may internally route requests to a specific region (often `us-east-1`) for the control plane, but you normally don't supply a region when creating an IAM client.

# Part 2: Setting Up Logging and Configuration

Instance Details :

region: us-east-1
bucket_name: faizan-ds5220-us-east-1-001
instance:
    ami_id: ami-0b6c6ebed2801a5cb
    instance_type: t3.micro
    key_name: ds5220
    security_group_id: sg-05f0b7e635c036b91
    role_name: EC2-S3-Limited-Access
    instance_profile_name: EC2-S3-Limited-Access


**A Jupyter Server is running.**

## Part 3: Create an S3 Bucket

S3 bucket - s3://faizan-ds5220-us-east-1-001

## Part 4: Creating IAM Role and Instance Profile

**Result:** `aws s3 ls s3://faizan-ds5220-us-east-1-001` returned `test.txt`, so the instance role **EC2-S3-Limited-Access** is correctly attached and the S3 permissions are working.

- **Role attached:** instance profile **EC2-S3-Limited-Access** is associated with `i-085ee7d879ec1a277`.
- **Assumed identity:** `aws sts get-caller-identity` returned `arn:aws:sts::441700731647:assumed-role/EC2-S3-Limited-Access/i-085ee7d879ec1a277`.
- **S3 access tested:** `aws s3 ls` showed `test.txt`, and you removed it with `aws s3 rm` — the role's S3 permissions are working.


```
{
  "AssociationId": "iip-assoc-036c187b10ca705f4",
  "InstanceId": "i-085ee7d879ec1a277",
  "IamInstanceProfile": {
    "Arn": "arn:aws:iam::441700731647:instance-profile/EC2-S3-Limited-Access",
    "Id": "AIPAWNV3KJL74AHI3GW6A"
  },
  "State": "associated"
}
```

# Part 5: Allocating and Associating an Elastic IP

### Elastic IP allocation and association

- **Action performed:** Allocated an Elastic IP (EIP) and associated it to EC2 instance **i-085ee7d879ec1a277**.
- **AllocationId: eipalloc-0dd4fe4c2c9b4028a**
- **Public IP assigned: 34.193.151.4**
- **AssociationId: eipassoc-0f0ea8fea370bab06**
- **Instance network interface: eni-0100f92b04b2b2e01**
- **Private IP used for association: 172.31.19.150**

# Part 6: Launching an EC2 Instance

**Instance launched:** `i-02e3ac20e17df5655` **Public IP:** `54.82.81.119` **Private IP:** `172.31.28.231` **AZ:** `us-east-1c` **Security group used:** `sg-0ebc2214e0241dd08` (default in your VPC)

# Part 7: Main Orchestration Function

### Provisioning summary

- **Instance ID:** `i-0ce1c23ced1917f2e`
- **Public IP:** `18.205.137.154`
- **S3 bucket:** `iac-python-bucket-79daea90`
- **EIP allocation:** `eipalloc-0ebf5833cd1b02b3e`
- **EIP association:** `eipassoc-0a039604f6114eec5`

## Part 8: Testing and Validation

JupyterLab is installed and reachable at `http://18.205.137.154:8888` with token `my-token`. Apache is serving `http://18.205.137.154/`. The EC2 instance is using the assumed role `EC2-S3-Limited-Access` and that role now has an inline policy allowing `s3:PutObject`, `s3:GetObject`, and `s3:ListBucket`. A test file was successfully uploaded to `s3://iac-python-bucket-79daea90/`.

### Evidence

- `curl` returned JupyterLab HTML for `:8888` with the configured token.
- `aws sts get-caller-identity` on the instance shows `arn:aws:sts::441700731647:assumed-role/EC2-S3-Limited-Access/i-0ce1c23ced1917f2e`.
- `aws s3 ls` shows the uploaded file `s3_test_from_manual.txt`.

# Reflection Questions

Consider these questions and be sure you understand the inner workings of this script:

1. What are the advantages of using boto3 for infrastructure provisioning compared to the AWS Console or CLI?

- **Programmability and automation** — boto3 lets you embed AWS API calls inside scripts and programs, so provisioning becomes repeatable, testable, and part of CI/CD pipelines rather than a manual sequence of clicks or one-off shell commands.

- **Fine-grained control and error handling** — you get structured responses, exceptions you can catch, and the ability to implement retries, backoff, and conditional logic (e.g., “create only if missing”), which is harder to do reliably with manual Console work.

- **Idempotence and stateful workflows** — code can check resource state and make minimal changes; combined with tagging and naming conventions this reduces drift compared with ad-hoc Console changes.

2. How does IAM role propagation affect automation workflows? How might you handle this in production?

- **Propagation delay** — when you create or attach an IAM role or inline policy, there can be a short delay before the new permissions are usable by an already-running principal (EC2 instance, Lambda, etc.). Automation that immediately relies on those permissions can fail with AccessDenied.

- **How to handle it in production**

    - **Design for eventual consistency**: add retries with exponential backoff and check for specific error codes (AccessDenied) before failing the workflow.

    - **Validate identity on the resource**: call `sts:GetCallerIdentity` from the instance to confirm the assumed role, then attempt a harmless API call (e.g., `s3:ListBucket`) in a loop until it succeeds or a timeout is reached.

- **Provision role and instance profile early**: create role and instance profile, attach role to profile, and only launch instances after IAM resources are fully created; or bootstrap with a small wait and verification step in user-data.

3. What security considerations should you keep in mind when writing infrastructure code?

- **Least privilege** — grant the minimum actions and resources required (narrow S3 ARNs, specific KMS keys, limited EC2 actions). Avoid broad * permissions.

- **Protect secrets** — never hard-code credentials, tokens, or private keys in code or VCS; use IAM roles, Secrets Manager, or parameter stores and ensure secrets are rotated.

- **Network exposure** — restrict security group CIDRs to known addresses (your script used caller IP /32 — good practice), avoid opening management ports to 0.0.0.0/0.

- **Auditability and logging** — enable CloudTrail, S3 access logging, and resource tags so changes are traceable; log provisioning actions and failures.

- **Safe defaults and review** — require code review for infra changes, use automated policy checks (e.g., IAM policy linter, Terraform Sentinel, or AWS Config rules) before applying.

4. How would you extend this code to be reusable across multiple environments (dev, staging, prod)?

- **Parameterize everything** — region, AMI, instance size, bucket names, CIDR ranges, and role names should be inputs (CLI args, environment variables, or a config file).

- **Use environment overlays** — keep a base configuration and apply small environment-specific overlays (YAML/JSON) or use a templating layer so the same code can produce different outputs.

- **Separate state and naming** — include environment prefixes/suffixes in resource names and isolate state (different S3 buckets, different tags) so resources don't

collide.

- **Promote via CI/CD** — run provisioning in pipelines with gated approvals for higher environments; run automated tests (smoke tests, policy checks) before promoting to staging/prod.

5. Why must Elastic IP association happen so late in the logical flow?

- **EIP association is a scarce, billable resource** — allocating and associating an EIP before you actually have a running instance risks wasting an address (and incurring charges) if the instance launch fails. Allocate only when you have a concrete instance to attach.

- **Association requires a target** — you must know the instance ID (and that the instance is in a state that accepts association). Waiting until the instance is running avoids race conditions and ensures the association succeeds.

- **Operational safety** — doing it late lets you verify instance health and that the instance profile, security groups, and user-data completed successfully before exposing the instance via a public, static IP.