Assignment 4: Deploy Gitea in a Custom VPC with Multiple EC2 Instances

Due Date: **10 April 2026**

Overview

In this assignment, you will extend your earlier single-instance Gitea deployment and your FastAPI-based REST server by building a small multi-instance architecture inside a custom AWS VPC.

Your goal is to deploy Gitea and a small internal audit/event API behind an `nginx` reverse proxy using at least three EC2 instances:

- a public-facing reverse proxy instance
- a backend Gitea instance in a private subnet
- a backend audit/event API instance in a private subnet

The purpose of this assignment is to show how cloud deployment involves placing components into a network architecture with explicit trust boundaries, controlled routing, and least-privilege security rules.

Target Architecture

Your deployment must follow this basic pattern:

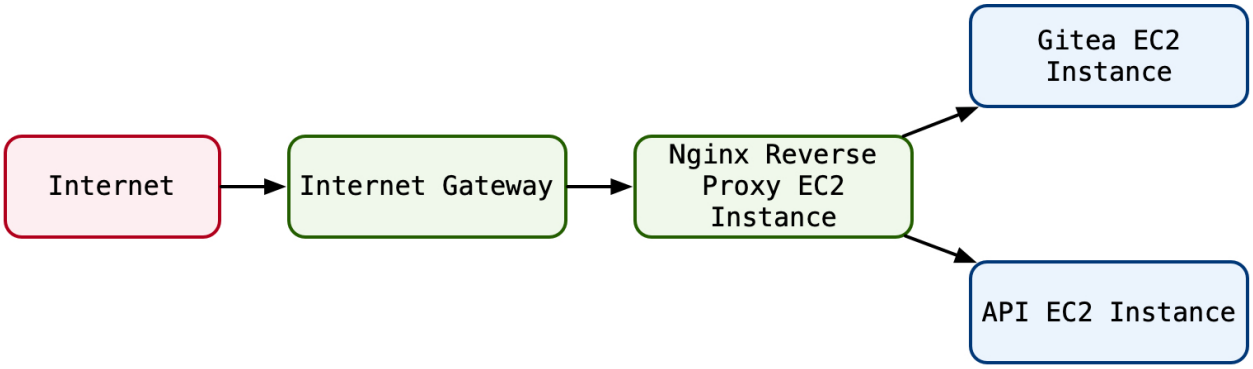

Figure 1: Flow of inbound traffic for this assignment.

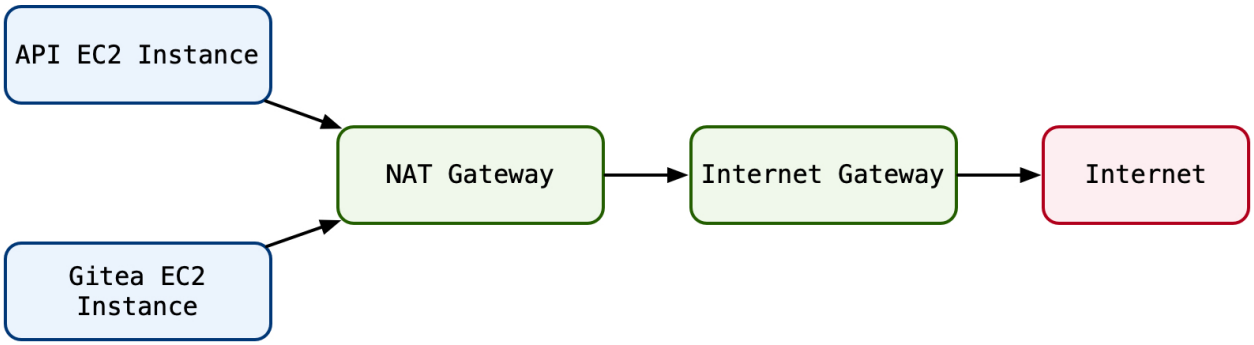

Figure 2: Flow of outbound traffic from the private backend instances through the NAT gateway to the internet.

The required version of this assignment uses three instances. You may optionally extend the design further with a separate database in a private subnet (for extra credit), but that is not required.

Learning Objectives

By completing this assignment, you should be able to:

1. Create a custom VPC with at least one public subnet and one private subnet.
2. Explain the roles of an internet gateway, NAT gateway, route table, subnet, and security group.
3. Launch multiple EC2 instances into specific subnets.
4. Restrict direct internet access to backend services.
5. Configure an `nginx` reverse proxy to forward requests to different backend services.
6. Build a small REST API and add request logging to it.
7. Verify connectivity and diagnose common networking failures.
8. Document a small cloud architecture clearly.

Requirements

You must:

- create a custom VPC rather than using the default VPC
- use the AWS `VPC only` workflow rather than `VPC and more` so that you create the supporting network components yourself
- create at least one public subnet and one private subnet
- attach an internet gateway to the VPC
- create a NAT gateway in the public subnet

- configure the public subnet route table with `destination 0.0.0.0/0` and `target Internet Gateway`
- configure the private subnet route table with `destination 0.0.0.0/0` and `target NAT Gateway`
- ensure the private subnet does not provide direct inbound internet exposure
- launch at least three EC2 instances
- place the `nginx` reverse proxy instance in the public subnet with a public IP address
- place the Gitea instance in the private subnet with no public IP address
- place the audit/event API instance in the private subnet with no public IP address
- create two security groups, one for the public `nginx` instance and one for the private backend instances, with rule sets that satisfy the security and access requirements listed below
- install and run Gitea on the backend instance
- build and run a REST API on the audit/event instance
- configure the REST API to log requests to a file
- configure the `nginx` reverse proxy instance to forward traffic to both backend services

Security and Access Expectations

Plan your security groups deliberately. You may use either:

- one security group for `nginx` and one shared security group for both private backend instances
- one security group for `nginx`, one for Gitea, and one for the audit/event API

Do not create extra ad hoc security groups unless you can clearly explain why they are needed.

Your security groups must satisfy all of the following minimum requirements:

Public `nginx` Instance

- inbound `TCP 80` from `0.0.0.0/0`
- inbound `TCP 22` only from a tightly restricted source such as your own IP address
- no rule opening Gitea port `3000` or API port `5000` directly to the internet

Private Gitea Instance

- no public IP address
- no inbound rule allowing `0.0.0.0/0` to port `3000`
- no inbound rule allowing `0.0.0.0/0` to port `22`

- inbound Gitea traffic on port `3000` allowed only from the `nginx` instance security group
- SSH allowed only from the `nginx` instance security group

Private Audit/Event API Instance

- no public IP address
- no inbound rule allowing `0.0.0.0/0` to port `5000`
- no inbound rule allowing `0.0.0.0/0` to port `22`
- inbound API traffic allowed only from the `nginx` instance security group
- SSH allowed only from the `nginx` instance security group

Outbound traffic may use the default allow-all egress rule.

Your finished deployment must demonstrate both of these outcomes:

- users can access Gitea through the proxy instance's public IP address or DNS name
- users can send API requests through the proxy instance to the audit/event API
- users cannot directly access the Gitea web interface on the backend instance from the public internet
- users cannot directly access the audit/event API from the public internet
- the private backend instances can reach the internet for package installation and updates through the NAT gateway

Recommended Design Constraints

Use the following values:

- VPC CIDR block: `10.0.0.0/16`
- public subnet CIDR block: `10.0.1.0/24`
- private subnet CIDR block: `10.0.2.0/24`
- NAT gateway location: public subnet
- Gitea internal listening port: `3000`
- audit/event API internal listening port: `5000`
- public reverse proxy port: `80`

The reverse proxy should forward incoming HTTP requests from port `80` to the backend Gitea instance on port `3000` using the backend instance's private IP address.

Use a path-based routing design such as:

- `/` → Gitea instance

- `/api/` → audit/event API instance

Software Choices

You may use either of the following for Gitea:

- Docker
- a non-Docker installation

Use `nginx` as the reverse proxy for this assignment.

For the audit/event API, use FastAPI.

NAT Gateway Note

The private-subnet Gitea instance and the private audit/event API instance must not be directly reachable from the internet, but they still need outbound internet access to install packages, download software, or pull container images.

For that reason, your architecture must include a NAT gateway in the public subnet. The private subnet should send default outbound traffic to the NAT gateway. This allows outbound internet access from the private backend instances without making those instances publicly reachable.

Cost note:

- You have a budget of `750 instance-hours per month`.
- A NAT gateway is not part of that EC2 instance-hour budget and may create additional AWS charges.
- Shut down or delete resources you are not actively using, and clean up your deployment promptly after you finish the assignment.

Backend Access Note

Private-subnet backends are not normally reachable directly from the internet. If you need administrative access to a private instance during setup, consider whether a `jump host` pattern would fit this architecture.

Your submission should make it clear how you handled backend administration without exposing the private instances directly to the public internet.

Audit/Event API Requirements

Your audit/event API must expose at least the following endpoints:

- `GET /api/health`
- `GET /api/events`
- `POST /api/events`

The API may store events in memory, in a flat file, or in another simple local format unless your instructor requires a database.

The `POST /api/events` endpoint must accept JSON. A simple payload such as the following is sufficient:

```
{
  "event_type": "push",
  "repo": "team1/project",
  "user": "alice"
}
```

Your API must log requests to a file. Each log entry should include at least:

- timestamp
- HTTP method
- request path
- response status code
- client IP address or forwarded client IP
- a short request body summary for `POST` requests

Task Sequence

Part 1: Build the Network

- create a custom VPC using the AWS `VPC only` workflow
- create one public subnet and one private subnet in that VPC
- attach an internet gateway to the VPC after the VPC has been created
- create a NAT gateway in the public subnet after the public subnet and internet gateway are in place
- create and associate route tables after the subnets, internet gateway, and NAT gateway have been created

Part 2: Launch the Instances

- launch a reverse proxy EC2 instance in the public subnet
- launch a Gitea EC2 instance in the private subnet

- launch an audit/event API EC2 instance in the private subnet
- assign security groups with least-privilege rules

Part 3: Configure the Software

- install `nginx` on the public instance
- install and run Gitea on the backend instance
- build and run the audit/event API on the second backend instance
- configure API request logging
- configure the reverse proxy to forward requests to the correct backend private IP address based on the URL path

Part 4: Validate the Architecture

- confirm that the proxy instance can reach both backend instances
- confirm that external users can reach Gitea through the proxy
- confirm that external users can reach the audit/event API through the proxy
- confirm that the backend services are not directly reachable from the internet

Part 5: Submit Evidence

- architecture diagram
- screenshots and command output
- network summary

Deliverables

Submit the following:

1. Architecture Diagram

Include a simple diagram that shows:

- the VPC
- the public subnet
- the private subnet
- the NAT gateway
- the `nginx` reverse proxy instance

- the Gitea instance
- the audit/event API instance
- the internet gateway
- the main traffic paths

2. Network Configuration Summary

Provide a short writeup that lists:

- the VPC CIDR block
- the subnet CIDR blocks
- the route table design
- the NAT gateway placement
- the security groups and their inbound rules
- the URL path routing design in `nginx`

3. Deployment Evidence

Provide screenshots or terminal output showing:

- all three EC2 instances running
- the created subnets and security groups
- the NAT gateway created in the public subnet and referenced by the private route table
- the Gitea service running on the backend instance
- the audit/event API running on its backend instance
- the API log file receiving entries
- the `nginx` reverse proxy configuration
- successful browser access through the proxy

4. Connectivity Verification

Include command output for tests such as:

- `curl` from the proxy instance to the backend Gitea instance
- `curl` from the proxy instance to the audit/event API instance
- `curl` from your local machine to the public proxy endpoint
- `curl` or `POST` from your local machine to the public `/api/` endpoint

- package installation, image pull, or other outbound access from at least one private backend instance
- evidence that direct public access to the backend Gitea port fails
- evidence that direct public access to the audit/event API port fails

Optional Extensions

If you complete the base assignment early, you may attempt one or more of the following:

- register a domain or subdomain and configure DNS
- add HTTPS with Let's Encrypt
- place MariaDB or PostgreSQL on a fourth EC2 instance in a private subnet

Submission Checklist

Before submitting, verify that you have included all of the following:

- architecture diagram
- VPC and subnet summary
- route table summary
- security group summary
- proof that all three instances are running
- proof that Gitea works through the reverse proxy
- proof that the audit/event API works through the reverse proxy
- proof that direct public access to both backend services is blocked
- proof that API requests are being logged