TUTORIAL

How to Secure AWS EC2 Instances with IAM Roles and Security Groups

Learn how to secure EC2 instances by replacing static credentials with IAM roles and tightening network exposure with security groups. This tutorial covers prerequisites, implementation, validation, and operational checks so you can apply the pattern safely before production.

Virtualization - July 21, 2026

Why this matters operationally

The practical problem is simple: EC2 instances are often launched with too much network exposure and with credentials that are hard to rotate safely. That combination increases the blast radius of a compromise, makes access reviews harder, and creates avoidable drift between what the instance should be able to do and what it actually can do.

This tutorial shows how to secure EC2 instances by using IAM roles for instance identity and security groups for network control. By the end, you will be able to remove long-lived credentials from an instance, restrict inbound and outbound paths to only what the workload needs, and verify the effective access path before you move the configuration into production. If you are building a fresh environment, pair this workflow with How to Set Up AWS Virtualization with Secure EC2 Networks so the network baseline is already aligned with your instance hardening plan.

What you will build

The finished state should look like this:

- The EC2 instance has an IAM role attached through an instance profile.
- The application or administration workflow uses temporary credentials from the role instead of static access keys on disk.
- The instance security group allows only required inbound traffic from approved sources.
- Outbound traffic is limited to the destinations the workload actually needs.
- Validation steps confirm both identity-based access and network filtering behave as intended.

If you already have an EC2 instance with sensitive data, this pattern should be treated as a hardening change, not just a launch-time preference. A related pattern is covered in Hardening Amazon EC2 with IAM Roles and Security Groups, but this tutorial focuses on the workflow and verification you need to apply it safely.

Prerequisites and stop-here checks

Before you change anything, confirm that the instance and account context can support the approach.

Prerequisites

You need:

- Permission to create or update IAM roles, instance profiles, and security groups.
- An EC2 instance you can test without risking production outage.
- A clear list of instance-side AWS actions the workload must perform, such as reading from object storage, publishing logs, or decrypting data.
- A network access matrix that identifies which ports, protocols, and source ranges are actually required.

Stop-here-if warnings

Stop here if any of the following are true:

- You do not know what AWS API actions the instance needs.
- The instance uses a static access key embedded in code or environment files and you cannot yet replace it.
- You do not know whether inbound traffic is initiated by users, a load balancer, a bastion host, or another service.
- You have not confirmed whether the workload relies on broad outbound internet access.

Those gaps are not minor documentation issues. They are the exact places where insecure defaults hide. If you continue without resolving them, you risk either breaking the workload or leaving a false sense of protection in place.

Prepare the access and network model

Goal

Define the minimum permissions and the minimum reachable network surface before making changes.

Action

Break the problem into two separate controls:

- Identity control with IAM roles
- List the AWS actions the instance needs.
- Map those actions to the smallest practical policy scope.
- Decide whether the instance needs read-only access, write access, or a narrow service integration.
- Network control with security groups
- Identify required inbound ports and source ranges.

- Identify required outbound destinations and ports.
- Decide whether the instance should be directly reachable at all, or reachable only through a load balancer, VPN, or jump path.

If the instance is part of a larger virtualization build-out, it is worth validating the surrounding network design first. The guidance in AWS Virtualization Security Hardening for EC2 and EBS Encryption is useful when the instance also depends on encrypted storage and tightly controlled launch-time access.

Expected output

You should have a short, reviewable access plan that answers:

- What the instance can call in AWS.
- What the instance can receive on the network.
- What the instance can send outbound.
- What traffic should be denied by default.

Validation

A good test here is whether another engineer could read the plan and infer the intended blast radius. If the plan says ?allow SSH from anywhere? or ?allow all outbound traffic? without a documented justification, the design is not finished.

Common failure

The most common mistake is combining identity and network design into a single vague rule set. That makes later troubleshooting harder because a denied API call and a denied port both look like ?the instance is broken,? even though they are different controls.

Attach an IAM role to the instance

Goal

Give the instance an identity that can obtain temporary credentials without embedding long-lived keys on disk.

Action

Create an IAM role with only the permissions the workload needs, then attach it to the EC2 instance through an instance profile.

A typical workflow is:

- Create a role trusted by EC2.
- Attach a scoped policy.
- Associate the role with the instance.
- Remove any static access keys from the instance filesystem, user data, environment variables, or configuration management templates.

Example trust policy for EC2:

Example of a narrowly scoped permissions policy for a workload that only needs to read objects from one bucket prefix:

Expected output

The instance should be able to request temporary credentials automatically, and the workload should use those credentials without any application code change if it already relies on the default credential chain.

Validation

Run a metadata and identity check from the instance. On a Linux instance, you can verify the role is attached and credentials are available through the instance metadata service:

Then confirm the caller identity through the AWS CLI if it is installed and configured to use instance credentials:

You should see an identity associated with the instance role, not a personal user access key.

Common failure

The most common failure is role attachment without policy precision. That creates a new credential source but does not actually reduce privilege. Another frequent issue is leaving older static keys in place, which can silently bypass the new role and make validation misleading.

Tighten inbound access with security groups

Goal

Reduce the instance?s reachable attack surface to only the ports and sources that are operationally required.

Action

Create or update the instance security group with explicit inbound rules. Examples:

- Allow SSH only from a controlled management network, not the public internet.
- Allow application traffic only from a load balancer security group.
- Allow admin ports only from a bastion or VPN range.
- Deny everything else by omission.

A practical rule set might look like this conceptually:

- TCP 22 from a management CIDR or bastion security group.
- TCP 443 from a load balancer security group.
- No other inbound rules.

If the instance does not need direct operator access, do not add an SSH rule at all. In many environments, Session Manager or another controlled access path is a better operational choice than exposing port 22.

Expected output

The instance should only accept inbound traffic that matches an explicit business or operational requirement.

Validation

From an authorized source, confirm the intended port is reachable. From an unauthorized source, confirm it is not. The goal is not just that the application works; the goal is that everything else fails closed.

You can validate by checking the effective security group rules on the instance and, from a separate host, attempting to connect to a denied port. If the denied port is reachable, the rule set is too broad or another path is open.

Common failure

The most common mistake is allowing management access from large CIDR ranges because it is convenient during setup. Another is attaching the wrong security group to the instance or to an attached load balancer and assuming the problem is on the instance itself.

Restrict outbound traffic to what the workload needs

Goal

Prevent the instance from reaching arbitrary external destinations unless the workload requires it.

Action

Review outbound rules with the same discipline as inbound rules. The default allow-all outbound posture is convenient, but it is not always appropriate for sensitive workloads. Decide whether the instance needs:

- Access to internal application tiers.
- Access to AWS service endpoints.
- Access to package repositories or update mirrors.
- Access to a proxy or egress gateway.

Then define outbound rules that match those requirements. If the workload can use private endpoints or internal services, prefer those over general internet egress.

Expected output

The instance should be able to reach only the destinations required for operation, patching, logging, and telemetry.

Validation

Check the application paths that depend on outbound access. For example, if the workload writes logs or reads objects, verify those specific calls succeed. Then test a disallowed destination or port to confirm the security group blocks it.

Common failure

Outbound restrictions often break silently because the workload retries and logs only a generic timeout. If you change egress rules, validate package updates, external API calls, DNS resolution, and any bootstrap dependencies in one pass.

Verify the effective access path

Goal

Confirm that both identity and network controls work together as intended.

Action

Test the instance from three angles:

- Identity test
- Confirm the instance can call only the AWS APIs its role allows.
- Attempt a denied AWS action and verify it fails with an access error.
- Inbound network test
- Confirm allowed ports are reachable only from approved sources.
- Confirm denied ports are unreachable.
- Outbound network test
- Confirm required destinations are reachable.
- Confirm disallowed destinations are blocked.

A useful pattern is to document one positive and one negative test per control. That makes validation reproducible and easier to review during audits or change approvals.

Expected output

You should be able to demonstrate that the instance has a working temporary identity and a constrained network footprint.

Validation

A simple acceptance checklist is:

- aws sts get-caller-identity returns the instance role identity.
- Allowed AWS API actions succeed.
- Denied AWS API actions fail.
- Allowed inbound traffic reaches the application.
- Denied inbound ports are closed.
- Required outbound services work.
- Unapproved outbound destinations are blocked.

Common failure

A common trap is validating only the happy path. A role that can read the expected bucket but also has broader access is still too permissive. A security group that permits the service port but also leaves SSH open to a wide CIDR is still too exposed.

Operational follow-up after deployment

Goal

Keep the security posture intact after the initial hardening pass.

Action

Add the hardening checks to routine operations:

- Review attached IAM policies when the application changes.
- Review security group rules when the instance role, subnet, or load balancer changes.
- Remove stale keys, stale rules, and stale exceptions.
- Re-test after patching, AMI replacement, or instance replacement.

If your environment already uses infrastructure as code, codify the role, instance profile, and security group rules there so drift is visible in review. If not, at minimum record the intended policy scope and network rules in your change process.

Expected output

The instance remains secure after maintenance, scaling, or replacement events, rather than only being secure on the day it was launched.

Validation

Use periodic checks to compare the deployed configuration to the documented baseline. Confirm that:

- The instance still has the expected role.
- No extra security group rules were added.
- No static keys reappeared in deployment artifacts.
- Outbound and inbound behavior still matches the approved matrix.

Common failure

The most common operational failure is drift. A temporary troubleshooting rule is added and never removed, or a new application feature starts using a broader permission set than the original role allowed. If you do not revalidate after change, the security design gradually disappears.

Final production checklist

Before you treat the configuration as production-ready, confirm these items:

- The instance uses an IAM role, not static access keys.
- The role grants only the actions the workload requires.
- Inbound security group rules are limited to approved sources.
- Outbound rules are limited to required destinations where practical.
- Negative tests confirm denied actions fail as expected.
- The configuration is documented and repeatable.

If all of those checks pass, you have a defensible baseline for securing EC2 instances with IAM roles and security groups. The key operational advantage is not just that the instance is locked down; it is that you can prove which identity it has, which network paths are open, and what should fail before the workload reaches production.

Use this guidance together with Azure VM right-sizing to connect the workflow with related operational context already available on the site.