
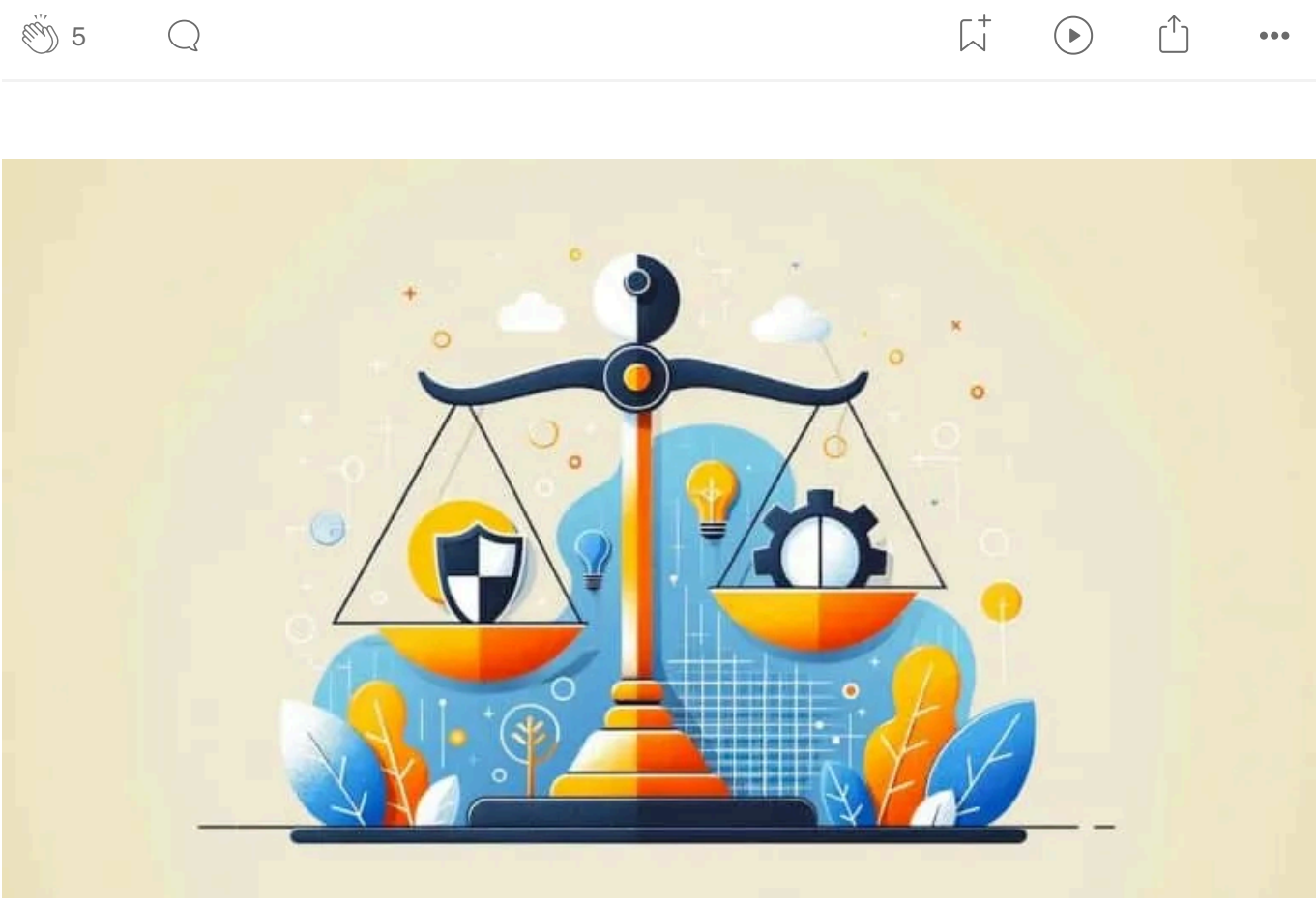


How to secure CI/CD roles without burning production to the ground

 Yan Cui · Follow
Published in theburningmonk.com · 6 min read · Feb 16, 2024



By now, most of us have moved away from using IAM users for CI/CD pipelines. Instead, we'd use dedicated CI/CD roles, one for each pipeline. This forces us to consider *who* can assume this role.

Identity federation is widely supported by 3rd-party providers such as [GitHub Actions](#) [1]. So, no more putting IAM credentials in CI/CD tools and worry that they might be compromised in a security breach [2].

However, attackers can still compromise the pipeline through supply chain attacks. For example, by compromising a Docker image we depend on in our CI/CD pipeline. Or by compromising static analysis tools such as [eslint](#) [3].

So, the question of “**How best to limit CI/CD role's permissions?**” has come up many times during my [Production-Ready Serverless](#) [4] workshop.

Your instinct might be to lock down the CI/CD role to just what it needs. Because we all want to follow the principle of least privilege.

There are different ways to achieve this. Here are two common approaches:

1. Start with a blank slate and add permissions to the role until it's able to deploy your application.
2. Start with the AdministratorAccess policy and allow the role to do everything. Use CloudTrail to track the actions performed by the role over some time. Then use the CloudTrail data to create a tailored set of IAM permissions.

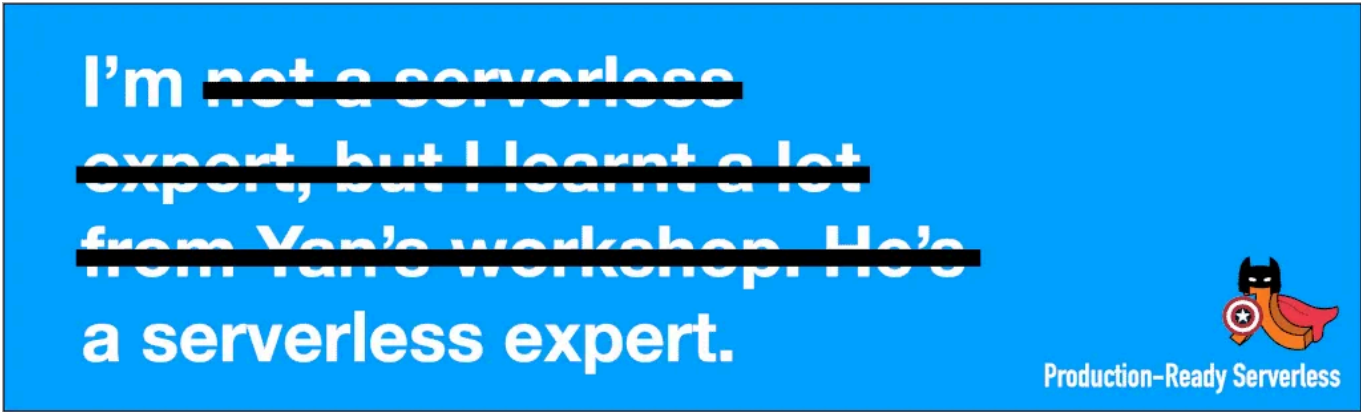
The first approach is very time-consuming and laborious and is best avoided. But in truth, both approaches have significant drawbacks in the

Rollback surprises

For most people, CloudFormation rollbacks don't happen often. Especially for those of us who don't write CloudFormation templates by hand.

So when a deployment fails for the first time, you find out that the CI/CD role is missing a whole bunch of permissions! This can leave you in an awkward position where your stack is stuck in the ROLLBACK_FAILED state. And you are stuck until you resolve the missing permissions.

In essence, crafting a least privileged CI/CD role takes twice as much work as you think. Because you also need to factor in all the permissions for rollbacks.



A tax on productivity

Your architecture is not static. It needs to evolve with your application and your business needs. Every time you introduce a new service, you also need to revise the CI/CD role.

Want to start using OpenSearch? Not before you update the CI/CD role!

Want to start using Bedrock? Not before you ... you get the point.

In organizations where the application team doesn't own the CI/CD role, this productivity tax can be steep. It requires many back-and-forth between teams and hits productivity hard.

And you likely have to do these for every service because every service has its own CI/CD pipeline.

This friction is a constant tax on your productivity. It hampers innovation and delays the adoption of new services into the tech stack. Because introducing a new service means new CI/CD permissions. Who in their right mind wants to deal with all that bureaucracy?

They are not as safe as you think

The reason why we pursue least privileged CI/CD roles is to reduce the blast radius of a security breach. If an attacker compromises our CI/CD pipeline, we want to limit what the attacker can do.

However, the CI/CD role likely needs to deploy IAM roles and Lambda functions. In the event of a compromise, a least privileged CI/CD role is not enough to stop the attacker.

Because the attackers can use the CI/CD role to create confused deputies to act on their behalf.

For example, attackers can use the CI/CD role to create IAM roles for them to assume. And they can also create Lambda functions to carry out malicious activities.

We can mitigate these risks to some extent with permission boundaries. But that requires lots more work and it's hard to get right. And again, you have to do it for every CI/CD role.

Prefer a permissive role that is hard to abuse

It's counterproductive to blindly pursue least privileged CI/CD roles. Beyond a certain point, you get diminished return-on-investments.

That is, you have to work much harder to address security risks that are unlikely to happen.

With that in mind, here's my preferred approach to securing a CI/CD pipeline.

Use a separate AWS account for each environment

This is something everyone should do. It insulates environments from each other. If one environment is compromised, it's contained by the account boundary. If an attacker compromises your dev environment, they won't be able to access your production data.

For large organizations, you should go a step further. I recommend having at least one account per team per environment. That way, teams are insulated from other teams' mistakes.

For business critical systems, they should have their own set of accounts too. That is one account per service per environment. It's not always necessary. But it's a good idea to protect these business critical systems from other systems.

Service Control Policies (SCPs)

Use SCPs to deny access to unused resources. These broad strokes include access to unused AWS regions and unused services. This way, you can eliminate large parts of AWS as possible attack surfaces.

For example, if you're not running any EC2 instances, then deny all EC2 activities. That's one less target for crypto-jacking attacks. But hey, [ECS is the new EC2 for crypto-jacking](#) [5]. If you can't disable ECS as a whole, then at least deny access to the regions where you don't have any containers.

Attribute-Based Access Control (ABAC)

[Support for ABAC](#) [6] is improving all the time, although it's still lacklustre in its current form. But many services already support the `aws:ResourceTag/${TagKey}` and `aws:PrincipalTag/${TagKey}` conditions.

These conditions can be used to stop attackers from accessing and creating resources. They can be added to the IAM permissions of the CI/CD role. Or they can be enforced through a permission boundary.

To make them more effective, the CI/CD role should be denied the ability to find out about itself. That way, the attacker can't find out what tags the CI/CD role has and what tags they need to use to create new resources.

Doing so helps us stop the attacker from accessing our resources through the CI/CD role. But they might target the following services to execute malicious code or escalate their privilege:

- Lambda
- EC2
- ECS
- IAM
- CodePipeline
- CloudFormation

Luckily, all these services (and many others) support the `aws:ResourceTag/${TagKey}` condition. We can use this condition to stop the attacker from creating new resources.

A permissive CI/CD role

After doing the above:

- We have removed the attack surface associated with unused regions and services.
- We have limited the blast radius of a compromise to individual teams and services.
- We have made it difficult for attackers to abuse the CI/CD role by requiring correct tags on resources. Which the attacker is not able to figure out easily because the CI/CD role is not allowed to describe itself.

Now we can allow ourselves to have a permissive CI/CD role that is not a pain in the ass to create and maintain! Because the role will be difficult to abuse, and there is a ceiling to what it can actually do.

By "permissive", I don't necessarily mean "AdministratorAccess". But you can give broad read & write permissions to all the services you use. Instead of picking out just the actions you need with a fine comb!

Conclusions

Achieving a balance between security and productivity is critical.

Adopting more permissive CI/CD roles introduces higher risks. But it's counterbalanced with the aforementioned compensating controls and security practices.

You also need continuous monitoring and regular security audits. The SCPs and ABAC settings should be reviewed regularly.

It's equally important to cultivate a security-conscious culture within teams. When everyone is aware of and responsible for security, the CI/CD pipeline becomes more resilient.

Ultimately, the goal is not to eliminate risk but to manage it efficiently. When we combine a permissive role with strong security controls, we can be safe and productive!

Links

[1] [Identity federation for GitHub Actions on AWS](#)

[2] [CircleCi incident report for Jan 4, 2023 security breach](#)

[3] [ESLint: Compromising the Build using Supply Chain Attack](#)

[4] [Production-Ready Serverless workshop](#)

[5] [Tales from the cloud trenches: Amazon ECS is the new EC2 for crypto mining](#)

[6] [Which AWS services support ABAC](#)

