

WHITEPAPER

CI/CD Pipelines for Infrastructure as Code

A reference architecture for Terraform, Ansible and CloudFormation pipelines on AWS, with no stored cloud credentials and one account per environment.

PUBLISHED BY
ZSoftly Technologies Inc.

DATE
September 2026

VERSION
2.0

AUDIENCE
Platform, DevOps and
security teams

CONTENTS

Executive summary	Page 2
Architecture overview	Page 3
Authentication without stored secrets	Page 4
Account model and environments	Page 5
Pipeline design	Page 6
Foundation, state and shared modules	Page 7
Platforms, caching and operations	Page 8
Rollout plan and how ZSoftly can help	Page 9
Sources	Page 10

Vendor facts in this paper were checked against AWS, GitHub, GitLab, Jenkins and HashiCorp documentation on 29 September 2026. The companion repository is at github.com/zsoftly/iac-cicd-reference.

EXECUTIVE SUMMARY

Treat infrastructure changes like code changes, with tighter controls

Infrastructure as code lets you review and repeat every change to your cloud. The pipeline that applies that code is now one of the most privileged systems you own. It can create, change and delete production resources. This paper shows how to build that pipeline so it holds no long-lived credentials, cannot touch the wrong account, and leaves an audit trail for every change.

The problem

- **Stored keys.** Many pipelines still hold AWS access keys as CI secrets. Those keys do not expire and are hard to rotate. Anyone with read access to the CI settings gets the keys.
- **One key for everything.** A single admin key often deploys to every environment, so a mistake in a test job reaches `prd`.
- **Shared state without locks.** Two runs against the same Terraform state at the same time corrupt it.
- **Copy-paste modules.** Each team writes its own VPC and IAM code, so reviews repeat and defaults drift.

The approach

The CI platform proves its identity to AWS with a short-lived OpenID Connect (OIDC) token. A minimal role accepts that token and can do one thing: assume a deploy role in the target account. Each environment lives in its own account. CloudFormation StackSets create the roles and the Terraform state backend in every account, including accounts added later. Pipelines run only when code is ready for review, and production changes need a manual approval and a change request.

No stored keys

OIDC tokens and role sessions that expire within the hour.

Account per environment

A fault in one account stays in that account.

Locked, versioned state

S3 backend with native locking and object versioning.

Three platforms

Patterns for GitHub Actions, GitLab CI and Jenkins.

Outcomes

- No AWS access keys in any CI system, so there are none to leak or rotate.
- Every change reaches AWS through a reviewed pull request and a recorded role session in CloudTrail.
- New accounts are ready for deployments as soon as they join the right organizational unit.
- Fewer wasted pipeline runs, and faster reviews because teams use pinned, shared modules.

Who should read this

Platform and DevOps engineers will use every chapter. **Security teams** should read the authentication and account chapters on pages 4 and 5. **Engineering leaders** should read this summary and the rollout plan on page 9.

CHAPTER 1

Architecture overview

The design separates three things that most pipelines mix together: who the pipeline is, what it is allowed to change, and where the record of the change lives.

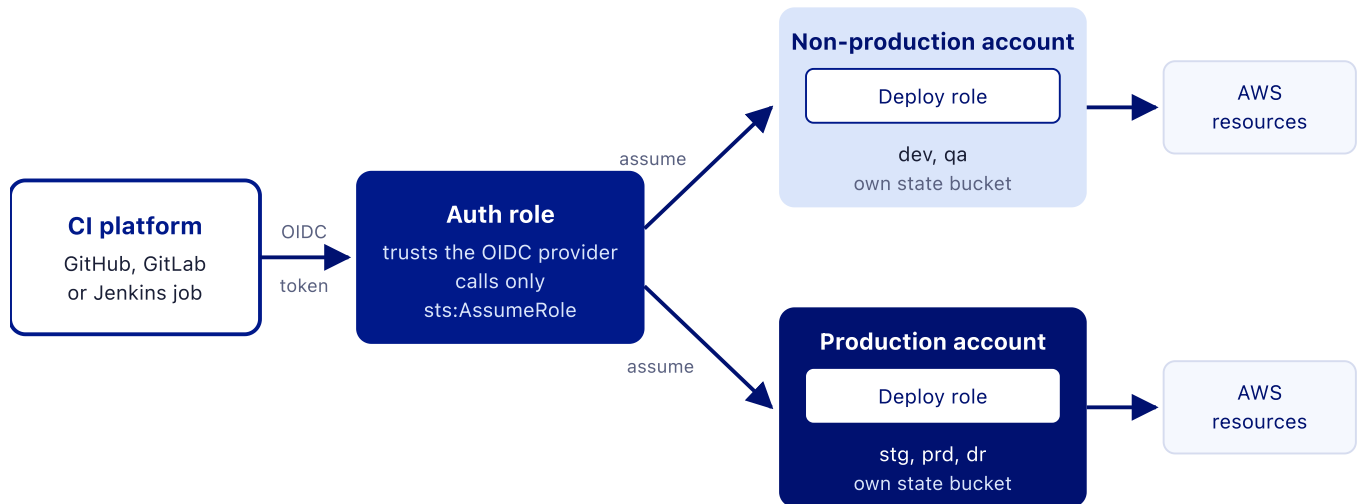


Figure 1. The role chain. The auth role proves identity. The deploy role in each account holds the permissions. Every role session is short-lived and recorded in CloudTrail with the repository, branch or environment from the token.

Design principles

- **No stored secrets.** OIDC federation for hosted runners, instance profiles with IMDSv2 for self-hosted runners on EC2.
- **Authentication apart from authorisation.** The auth role cannot change infrastructure. The deploy role cannot be reached from outside the chain.
- **One account per blast radius.** Non-production and production never share an account.
- **Automatic onboarding.** StackSets put roles and state backends into every account in a target OU.
- **Predictable order.** Numbered prefixes keep environments in promotion order in every tool.
- **Platform-neutral.** The same pattern works on GitHub Actions, GitLab CI and Jenkins.

Three questions, three separate answers

Question	Answered by	Covered on
Who is the pipeline?	A signed OIDC token from the CI platform, checked by the IAM OIDC identity provider [1][3]	Page 4
What may it change?	The deploy role in one target account, reached only through the auth role	Page 5
Where is the record?	The pull request, the CloudTrail role session and a versioned state file [7]	Pages 6 and 7

Table 1. The separation this design keeps, and where each part is covered.

The companion repository, github.com/zsoftly/iac-cicd-reference, implements the two-account version with CloudFormation templates, scripts and platform notes [10]. The OU layout on page 5 extends it.

CHAPTER 2

Authentication without stored secrets

Each job asks its CI platform for a signed OIDC token. AWS STS checks the token against an IAM OIDC identity provider and returns temporary credentials for the auth role [1][3]. The auth role then assumes the deploy role in the target account. No access key exists at any step.

Scope the trust policy tightly

The trust policy decides which repositories and environments are allowed to use the role. A wildcard such as `repo:example-org/*` lets any repository in the organisation deploy. Match the exact repository and, for production, the exact CI environment. GitHub puts the environment name in the `sub` claim as `repo:ORG/REPO:environment:NAME` [1].

```
# Auth role in the production account (CloudFormation)
AssumeRolePolicyDocument:
  Statement:
    - Effect: Allow
      Principal:
        Federated: !Ref GitHubOidcProviderArn    # token.actions.githubusercontent.com
      Action: sts:AssumeRoleWithWebIdentity
      Condition:
        StringEquals:
          token.actions.githubusercontent.com:aud: sts.amazonaws.com
          token.actions.githubusercontent.com:sub: repo:example-org/infra-live:environment:70-prd

# Deploy role: trusts only the auth role
AssumeRolePolicyDocument:
  Statement:
    - Effect: Allow
      Principal:
        AWS: !Sub arn:aws:iam::${AuthAccountId}:role/cicd-auth-role
      Action: sts:AssumeRole
```

Know the limits

- **One-hour sessions.** AWS limits a chained role session to one hour, whatever the role's own maximum [2]. Split long applies into smaller stacks, or re-assume the role between steps.
- **Workflow permission.** GitHub Actions only issues the token when the job has `permissions: id-token: write` [1].
- **GitLab.** The `id_tokens` keyword issues OIDC tokens on the Free, Premium and Ultimate tiers. The `sub` claim carries the project path and ref [4].
- **Jenkins.** The OpenID Connect Provider plugin gives builds a signed, time-limited token for the same flow [5]. Agents on EC2 use an instance profile instead.

Why two roles and not one

The auth role changes rarely and has one permission. Every repository, runner type and platform can share the same deploy role behind it, so you review one permission set per account. If an attacker obtains an auth session, it expires within the hour and reaches only the deploy roles that trust it by ARN.

CHAPTER 3

Account model and environments

Start with two accounts and grow into an OU layout when you have more teams or stricter audit needs. Both use the same roles and templates.

Two accounts, for smaller teams

A non-production account holds `dev` and `qa`. A production account holds `stg`, `prd` and `dr`. Each account has its own auth role, deploy role and state bucket. This is what the companion repository builds.

OU layout, for larger estates

A Platform OU (PLT) holds a runner account with the only auth role. A Workloads OU (WKL) splits into non-production and production OUs with one account per environment. Deploy roles in every WKL account trust the PLT runner role.

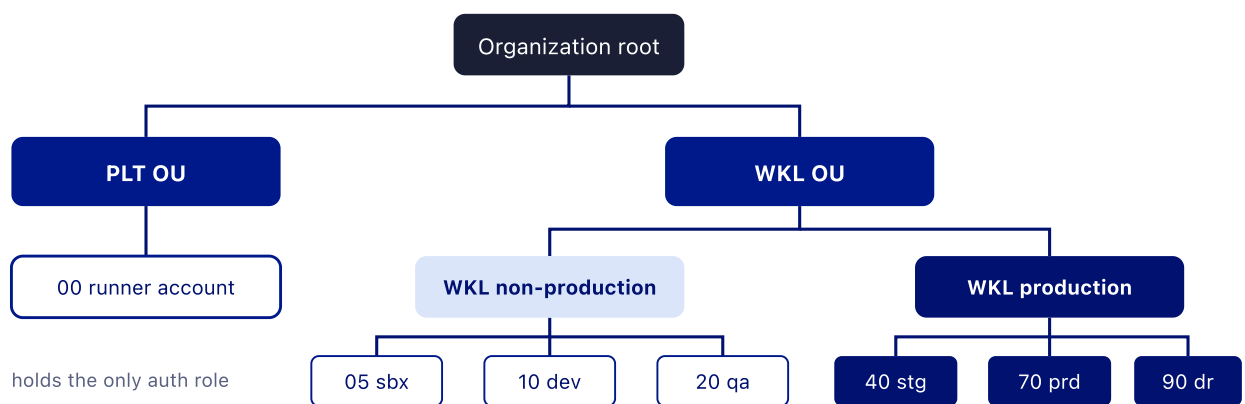


Figure 2. OU layout. The runner account authenticates. Workload accounts hold deploy roles and state.

Numbered environment prefixes

Alphabetical order puts `dr` before `prd` and `qa` after it. Numbered prefixes keep environments in promotion order in folder listings, pipeline stages and dashboards. Gaps leave room to add an environment later, such as a canary stage at 75.

Prefix	Environment	OU	Region	Change request
00	runner	PLT	Primary	No
05	sbx (sandbox)	WKL non-production	Primary	No
10	dev	WKL non-production	Primary	No
20	qa	WKL non-production	Primary	No
40	stg	WKL production	Primary	No
70	prd	WKL production	Primary	Yes
90	dr	WKL production	Second region	No

Table 2. Environments in promotion order. The companion repository uses the labels `qat` and `prod` for the qa and prd environments.

CHAPTER 4

Pipeline design

Run pipelines when code is ready for review

A pipeline that runs on every push to every branch spends most of its time on work in progress. A developer who pushes ten commits before opening a pull request starts ten runs, and each `terraform plan` calls the AWS APIs of every account it touches. Run the pipeline on these events only.

- **Pull request opened or updated.** The code is ready for review.
- **Push to the main branch.** The code is merged.
- **Manual trigger.** Someone asked for a run on purpose.
- **Skip** pushes to feature branches with no open pull request.
- **Filter by path** so a change to Ansible roles does not plan Terraform, and the reverse.
- **Measure** run counts for a month before and after. The saving depends on how your team commits.

Four stages

- 1 **Validate**, automatic. Format check, lint and syntax validation: `terraform fmt -check`, `terraform validate`, `ansible-lint`, `cfn-lint` and a Trivy configuration scan.
- 2 **Plan**, automatic. Produce the plan and post the summary to the pull request. Save the plan file as an artifact so the deploy stage applies exactly what was reviewed.
- 3 **Deploy**, manual. Apply the saved plan to one environment. Allow only one deploy per environment at a time.
- 4 **Destroy**, manual and restricted to administrators. Protected by an approval and never available for `prd` from a pull request.

Environment rules

Environment	On a pull request	On main	Approval	Change request
05-sbx, 10-dev	Manual deploy allowed	Manual	Author	No
20-qa	Blocked	Manual	Reviewer	No
40-stg	Blocked	Manual	Reviewer	No
70-prd	Blocked	Manual	Named approver	Yes
90-dr	Blocked	Manual	Named approver	No

Table 3. When each environment can deploy. Only code merged to main reaches qa and above.

The change request gives `prd` an audit trail, a documented rollback plan and stakeholder notice. Pass the change request number into the deploy job as an input and fail the job when it is empty. Deploy to `dr` right after `prd` so the two never drift apart.

CHAPTER 5

Foundation, state and shared modules

Deploy the foundation with StackSets

The roles and state backend must exist before any pipeline can run, so Terraform cannot create them for itself. Use CloudFormation StackSets with service-managed permissions from the management account. With automatic deployment on, CloudFormation adds the stack when an account joins a target OU and removes it when the account leaves [6].

1. OIDC identity provider, to the PLT OU (or to each account in the two-account model).
2. Auth role, to the PLT OU.
3. Deploy role that trusts the auth role ARN, to the WKL OU and its child OUs.
4. Terraform state backend, to the WKL OU.

Once this is in place, a new workload account needs no manual IAM work. Move it into the right OU and it is ready for its first pipeline run.

Terraform state

- **One bucket per account.** Each workload account keeps its own state, keyed by repository and environment, for example `example-org/infra-live/70-prd/terraform.tfstate`.
- **Versioning on.** HashiCorp recommends bucket versioning so you recover state after a bad apply or a deletion [7].
- **Native S3 locking.** Set `use_lockfile = true` in the S3 backend. DynamoDB-based locking is deprecated and will be removed in a future Terraform minor version [7]. Configure both during the move, then drop DynamoDB.
- **Encryption and replication.** Encrypt with SSE-S3 or a KMS key, and replicate the `prd` state bucket to the DR region.

The companion repository's backend template and rollback script still create and use a DynamoDB lock table. Update both when you adopt native S3 locking.

Shared modules, pinned by version

When every team writes its own VPC, IAM and S3 code, reviewers check the same patterns in every repository, and defaults drift. Publish shared modules from one repository, tag releases, and have teams pin a version. A team upgrades when it chooses, and a reviewer can trust a pinned module without reading it again.

```
module "vpc" {  
  source = "git::https://github.com/example-org/tf-modules.git//vpc?ref=v1.4.0"  
  cidr   = "10.20.0.0/16"  
}
```

The same idea applies to Ansible roles in versioned collections and to CloudFormation modules or Service Catalog products. Number Terraform files by layer as well: 00 to 09 for providers and backend, 10 to 19 for identity, 20 to 29 for network, up to 90 to 99 for outputs [10].

CHAPTER 6

Platforms, caching and operations

The same pattern on three platforms

Concern	GitHub Actions	GitLab CI	Jenkins
OIDC token	<code>permissions: id-token: write</code> [1]	<code>id_tokens</code> keyword [4]	OpenID Connect Provider plugin [5]
Skip feature branches	<code>on: pull_request</code> and <code>push</code> to main	<code>workflow: rules:</code>	Multibranch branch filter
Path filter	<code>paths:</code>	<code>rules: changes:</code>	<code>changeset</code> condition
Manual gate	Environment with required reviewers	<code>when: manual</code>	<code>input</code> step
One deploy at a time	<code>concurrency:</code>	<code>resource_group:</code>	<code>lock</code> step (Lockable Resources)

Table 4. Platform features that implement each part of the pipeline.

Cache dependencies, and let the cache expire

Terraform providers, Ansible collections and language packages are downloaded on every run unless you cache them. Point Terraform at a plugin cache with `TF_PLUGIN_CACHE_DIR` [9] and key the cache on the `.terraform.lock.hcl` hash. Key Ansible collections on the `requirements.yml` hash.

- GitHub Actions caches hold 10 GB per repository by default. Owners set a higher limit when needed, and GitHub removes entries not used for 7 days [8].
- For a shared cache across platforms, use an S3 bucket in the runner account with lifecycle rules: 7 days for providers, 3 days for packages, 1 day for plan files.
- Short expiry means security fixes in providers and packages reach your pipelines within days. Hash-based keys stop one branch poisoning another's cache.

Ansible in the same pipeline

Keep one inventory per numbered environment and numbered roles, such as `10-common` for base configuration and `20-security` for hardening. Validate with `ansible-lint`, preview with `ansible-playbook --check --diff` in the plan stage, and apply in the deploy stage under the same approval rules as Terraform.

When things go wrong

- **Access denied on assume.** Compare the token's `sub` claim with the trust policy. It is usually a branch or environment name mismatch.
- **Session expired mid-apply.** The one-hour chain limit applies. Split the stack.
- **Stuck state lock.** Confirm no run is active, then use `terraform force-unlock` with the lock ID.
- **Bad apply corrupted state.** Restore a previous S3 object version. The repository's `95-rollback.sh` lists versions, backs up the current state, asks for explicit confirmation and supports a dry run [10].

CHAPTER 7

An eight-week rollout

Most teams adopt this design in about eight weeks alongside normal work. These are ZSoftly planning estimates for one infrastructure repository and up to six environments. More repositories or a large legacy estate take longer.

Phase	Weeks	Exit criteria
Identity and access	1 to 2	OIDC provider, auth and deploy roles live in every account, access keys removed from CI
State and foundations	2 to 3	StackSets deployed, versioned state buckets with native locking, existing state migrated
Pipeline design	3 to 4	Triggers, path filters, four stages and environment rules working for one repository
Shared modules	4 to 5	First modules published with tags and consumed by pinned version
Validation and testing	5 to 6	Lint, scans and plan review in every pull request, a state rollback rehearsed
Rollout and handover	6 to 8	All environments on the new pipeline, runbooks accepted by your team

Table 5. Rollout phases and the evidence that closes each one.

How ZSoftly can help

ZSoftly Technologies Inc. is a cloud engineering company. We design, build and run cloud platforms and pipelines for clients in Canada, the United States and the United Kingdom. The engineers who design your pipeline are the ones who build it and hand it over.

Assess Review your pipelines, credentials, state and account layout.	Design Agree the account model, roles, triggers and rules in writing.	Build Deliver the phases above as code, with a demonstration at each step.	Operate Run the pipelines with you until your team is comfortable.	Hand over Code, runbooks and access transfer to your team.
--	---	--	--	--

- Your first hour with our engineers is free. Use it to review one pipeline or one account layout.
- You own every template, workflow and module we write, in your own repositories.
- Start from the open-source companion repository with or without us.

Check your readiness before week 1

- Which CI systems hold AWS access keys today, and who can read them?
- Is each environment in its own account, or do some share one?
- Where does Terraform state live, and is it versioned and locked?
- Who approves a `prd` deploy, and where is that approval recorded?
- Which modules are copied between repositories instead of pinned?
- Who will own the pipeline roles after handover?

Talk to an engineer. Book a discovery call at zcp.zsoftly.ca/contact or call +1 (647) 771-1065.

REFERENCES

Sources

All sources were checked on 29 September 2026.

1. GitHub. OpenID Connect reference.
<https://docs.github.com/en/actions/reference/security/oidc>
2. AWS. IAM roles, role chaining.
https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.html
3. AWS. Create an OpenID Connect identity provider in IAM.
https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_providers_create_oidc.html
4. GitLab. Configure OpenID Connect in AWS to retrieve temporary credentials.
https://docs.gitlab.com/ci/cloud_services/aws/
5. Jenkins. OpenID Connect Provider plugin.
<https://plugins.jenkins.io/oidc-provider/>
6. AWS. Enable or disable automatic deployments for StackSets in AWS Organizations.
<https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/stacksets-orgs-manage-auto-deployment.html>
7. HashiCorp. Terraform S3 backend.
<https://developer.hashicorp.com/terraform/language/backend/s3>
8. GitHub. Dependency caching reference.
<https://docs.github.com/en/actions/reference/workflows-and-actions/dependency-caching>
9. HashiCorp. Terraform CLI configuration file, provider plugin cache.
<https://developer.hashicorp.com/terraform/cli/config/config-file>
10. ZSoftly. IaC CI/CD reference repository.
<https://github.com/zsoftly/iac-cicd-reference>

Notices

This paper is for general information. It does not create a warranty or a commitment from ZSoftly Technologies Inc. Timelines are planning estimates. Code samples are examples to adapt and review, not production-ready templates. Amazon Web Services, AWS and AWS CloudFormation are trademarks of Amazon.com, Inc. or its affiliates. GitHub, GitLab, Jenkins, Terraform and Ansible are trademarks of their respective owners.

© 2026 ZSoftly Technologies Inc. Version 2.0, September 2026.