Cyral

WHITE PAPER

How to get Away from Privilege Management Toil with Security As Code

Contents

Overview

Security as Code turns manual security practices into automated code so that security testing and reviews are baked into your organization's everyday workflow. In contrast to manual security practices that were compartmentalized and guarded by specific teams, the Security as Code framework puts these practices on the record where they can be reviewed and audited by a wider team, and it builds them into the workflow so they're treated as first-class operational steps that can't be skipped.

Policy as Code is the part of this framework that automates the workflows for user and data access authorization. Under a Policy as Code approach, manual policy practices are replaced with "code" (policy files readable by your team and by the systems that enforce the policies) so that the right people on your team can request, approve, and publish policy changes instantly.

Security for Infrastructure as Code

In your continuous integration/continuous deployment pipeline ("CI/CD pipeline"), Security as Code means automating security actions using the same tools your team uses to roll out infrastructure changes—from pull request to deployment to production, and every step along the way. This process moves security design and testing to an earlier stage in the development workflow, ensuring your team can deliver a more secure product, faster.

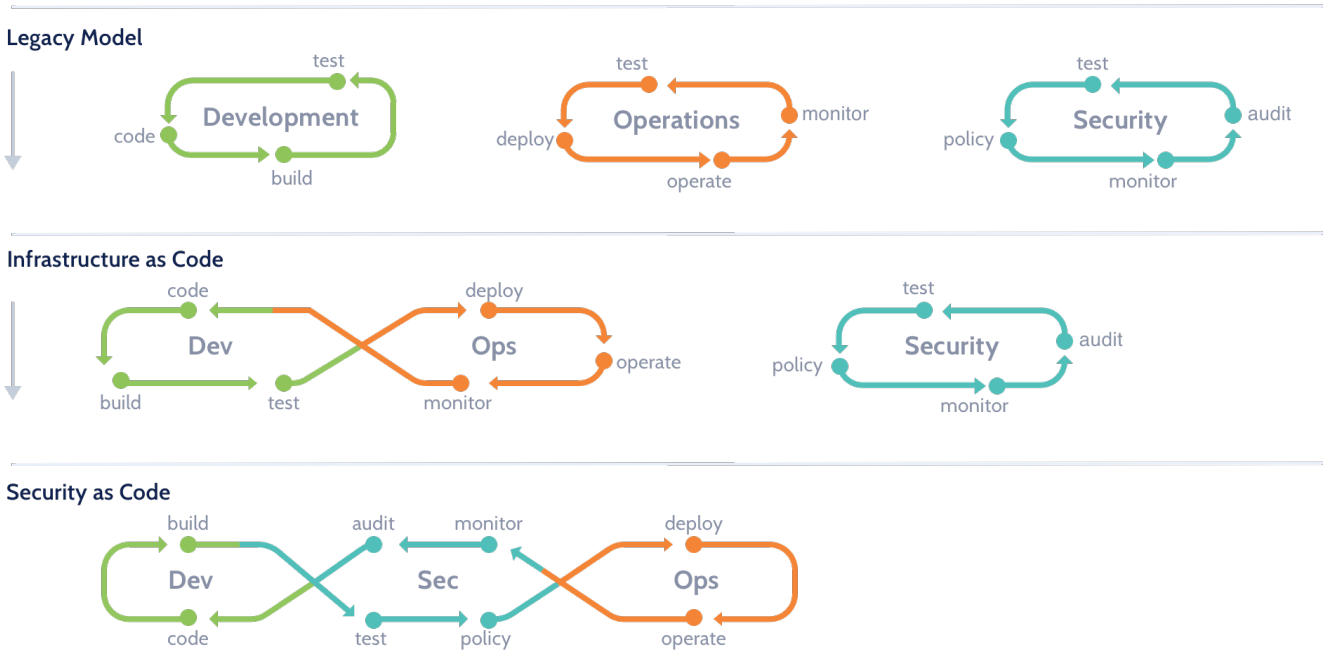


Figure 1: Progression from siloed development, ops, and security to "as code" collaboration

In this white paper, we'll show that the key to Security as Code is to automate where possible and look for ways to implement security checks throughout the build process. Security as Code and Policy as Code help to reduce toil, scale your team, and provide visibility to the rest of the organization so that you can move fast and stay secure.

The Bad Old Days

In most organizations, user permissions and grants have been handled by IT and/or security teams by clicking through a series of screens. Even a simple access request sets in motion a confusing and cumbersome sequence of steps the user and team must navigate.

1. This process is often kicked off by an informal request from the user to gain the access they need.
2. They will then be redirected to submit a formal ticket to an IT support request queue.
3. The ticket is then routed to a request queue to be triaged by Level-1 support staff. Often this ticket may need additional approvals from the user's manager if her manager wasn't included in the initial request.
4. The support staff must consult a playbook, if one exists, to properly triage the ticket and decide what action needs to be taken.
5. The ticket is then reassigned to the responsible manager, and the user must wait until her request gets the proper approval.
6. Once the ticket has been approved, additional approval will be needed by security and or compliance to determine whether the requester, given her job role, deserves access.
7. Finally, a number of days after the initial request is submitted, the administrator of the service is assigned the ticket. They handle the ticket, granting the user access. Oftentimes, even this final step is done manually by clicking through a number of screens and either adding the user and assigning individual permissions or, at best, assigning her to a group with the necessary permissions.

This process is long, cumbersome, and incredibly time-consuming for all involved. More often than not, users are over-permissioned, and the approvals represent only rubber-stamp actions, not thorough vettings. Also, permission decisions are only revisited when an organizational change or audit forces a review. This opens security gaps due to policy creep, role changes, misconfigurations, and lags in enforcement.

If this sounds familiar, and if some of these steps are part of your current workflow, then it's worth adopting a more evolved security and access management process. By moving to continuous and automated security, risk, and compliance management, organizations can do away with historic, inefficient workflows and limit gaps in their security posture.

Policy as Code as a Tenet of Security as Code

From startups to large organizations, handbook-based policy management rarely scales well and is often applied in a non-uniform way. Policy as Code addresses this by codifying policies, providing visibility, and enforcing them automatically. By adopting Policy as Code, an organization forces itself to translate its policy decisions into code that enforces decisions in the same way, every time.

Under Policy as Code, policies are standardized into records readable by both the people who manage them and the systems that enforce them. Policies written in formats like YAML and JSON allow this, because they're consistent enough for an authorization system to enforce and simple enough for security personnel to read.

With machine-readable policies, authorization can be offloaded to security software, reducing the toil necessary to constantly monitor and maintain one-off requests, freeing development teams to focus on core features.

With policies stored consistently and globally, realtime auditing of access grants is possible for the first time. Security teams can monitor and review authorization decisions and policy changes, allowing the entire company to move faster without breaking core security and compliance requirements.

Benefits of Policy as Code

Policy as Code reduces toil for IT and security teams by helping to automate workflows that are typically done manually.

- You simplify onboarding of new employees, removal of access for departed employees, and rollout of new tools and technology.
- You no longer need to traverse different systems to track down the full policy documentation that may have been stored in your standard tracking system and in other communication systems. Full tracking and history is now visible instantly in the organization's version control system, where the responsible teams can view every policy and permission change.
- You reduce the toil of approving and denying requests. By codifying your decisions, those who grant access no longer need to perform the legwork to determine if someone should get access or not. Permissions can be auto-provisioned at exactly the level each user or group needs, and no more.
- Policy as Code takes the guesswork out of granting permissions and applies your policies uniformly across every request, from any application, tool, or user. Once Policy as Code is integrated with your organization's directory service and groups, it becomes possible for the security and IT team to implement true role-based access.

Policy as Code gives managers and employees a clear view of access privileges throughout the organization, and reduces uncertainty. Permissions can be reviewed by the right people, anywhere in the organization. When a change is needed, the request can be submitted and reviewed in a standard, well-tracked way. Optionally, employees can self-serve some permissions requests thanks to automatic policies that evaluate the user's role, their group affiliations, and the sensitivity of the resource being requested.

Whether changes are streamlined or self-serve,employees appreciate autonomy and transparency because it cuts the confusion about how or when their request will be reviewed.

Finally, by clearly tracking all permissions changes, an organization has a real time audit log that reduces the work needed to prepare for compliance and certification reviews. With a clearly defined system of record for security policies, approvals are tracked automatically, and there's no longer a need to search multiple systems to get a full picture of access rights.

Implementing Policy as Code for Access Management

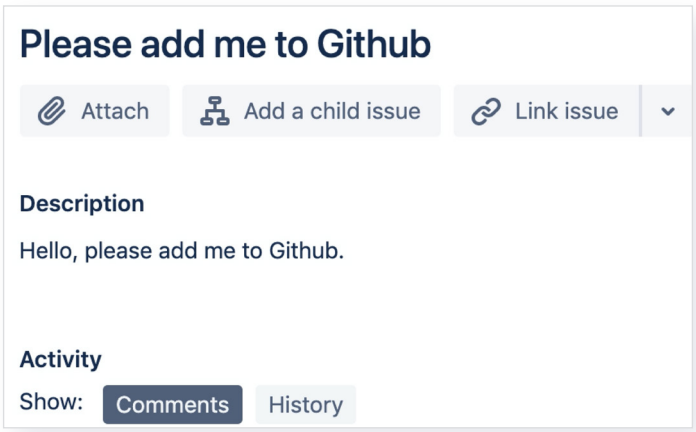
In this section and the ones that follow, we'll look at how an organization can adopt Policy as Code to govern access to sensitive resources. While these practices build on the infrastructure as code approach, they don't require that the organization be on the cutting edge of cloud automation in order to use them. The tools discussed here, like Terraform, offer a low-risk way to begin automating key operations in any organization.

One of the first steps you can take to begin on the road to Policy as Code is to use an infrastructure-as-code tool like Terraform to turn your clicks into code. For example, if you are currently managing one of your SaaS providers like AWS or Github through their console, you can import your identity and access management into Terraform, which looks like:

```
resource "github_repository" "sre-repo" {
  name        = "sre-repo"
  description = "SRE team repo (managed by
Terraform)"
}

resource "github_team" "sre-team" {
  name        = "sre-team"
  description = "SRE team (managed by Terraform)"
  privacy     = "closed"
}

resource "github_team_membership" "sre-team-
membership" {
  team_id  = "${github_team.sre-team.id}"
  username = "SRELead"
  role     = "member"
}

resource "github_team_repository" "sre-team-
repo" {
  team_id    = "${github_team.sre-team.id}"
  repository  = "${github_repository.sre-repo.
name}"
  permission = "push"
}
```

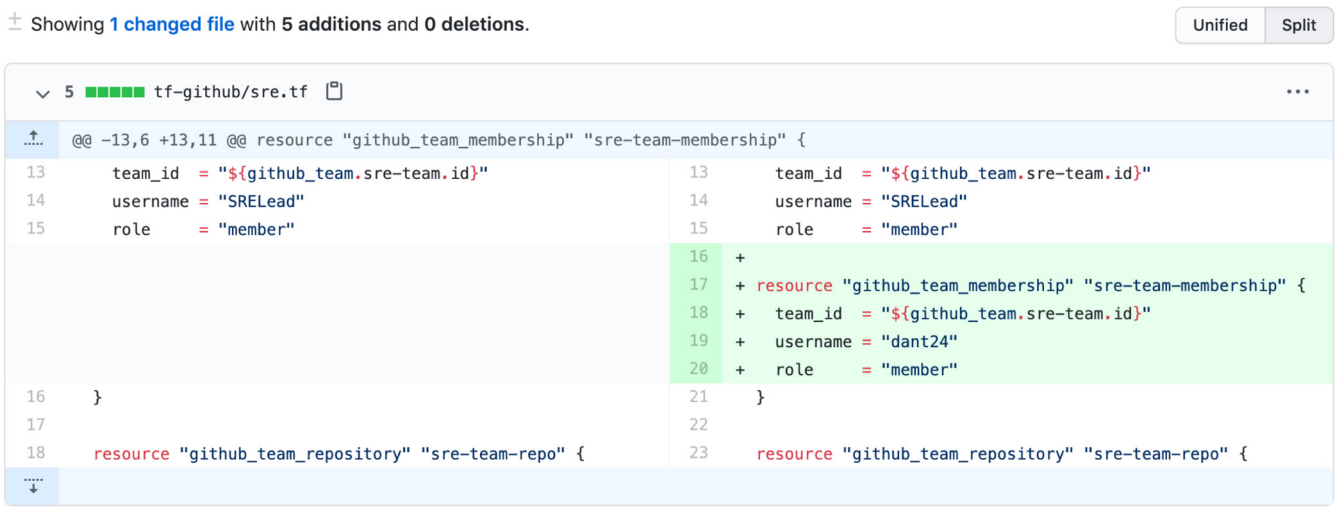
Example: Onboarding an Employee

Historically, when a new employee, in this example, dtobin, joins the organization, a JIRA ticket like the one below will show up in the inbox of the admin:

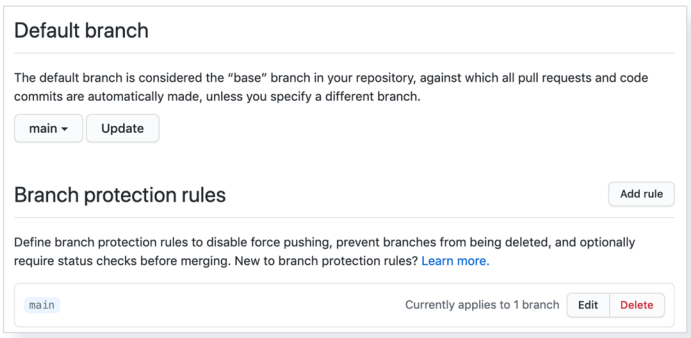


Even at this beginning stage, many things can go wrong, starting with the ticket itself being filed against the wrong person or team, or as is often the case, not having enough information, leading to a tedious back-and-forth between requester and administrator.

Using a Policy as Code framework, the manager of the employee just submits a pull request in the version control system with details as shown in this example using GitHub:



Requests now can follow the standard checks defined in your version control system, including limits on who can change what (“protected branches”) and required approvals. This ensures that each request—like our onboarding request—always goes to the admin in charge.



Thanks to the consistent change controls and approval limits enforced by the version control system, the security and infrastructure teams gain more control and autonomy, and administrators are freed from back-and-forth dialogue related to permissions changes.

Setting Policy

Continuing with our example, the next step in the Policy as Code journey is to make sure privileges for the user dtobin are set up correctly. This can be automated in a number of ways, depending on your environment. In the example below, we’ll show how it would be done using Hashicorp’s Sentinel framework, which unlocks Policy as Code in a Hashicorp-based ecosystem that uses Terraform, Vault, Nomad, and Consul. Sentinel allows you to codify rules like the one shown in the example below. The rules in this example specify that only those site reliability engineers (SREs) who are on call today can make changes to the production environment:

```
import “calendar”

// Get the calendar for anyone in the SRE group for today
sre_calendar = calendar.forgroup(“SRE”).today

//Make sure the user is a member of the SRE group
groupcheck = rule {
  “SRE” in identity.groups
}

//Allow this policy to pass if the user is on call and in the SRE group
main = rule {
  sre_calendar.has_event(“On Call”) and groupcheck
}
```

You can have policies on applications and services as well. For example, in Terraform you can prevent the creation of new security groups in AWS that allow blanket ingress access:

```
import “tfplan”

main = rule {
 all tfplan.resources.aws_security_group as _,
instances {
   all instances as _, sg {
    all sq.applied.ingress as ingress {
      ingress.cidr_blocks not contains “0.0.0.0/0”
    }
   }
 }
}
```

This allows developers to spin up applications without worrying about exposing them to the outside world.

In Nomad, you can enforce that you allow only Docker workloads:

```
allowed_drivers = [“docker”]

main = rule {
 all job.task_groups as tg {
  add tg.tasks as t { t.driver in allowed_drivers }
 }
}
```

Another option for Policy as Code is Styra’s Open Policy Agent (OPA), a framework for policy that integrates well with popular cloud stacks. OPA helps to decouple policy from the service itself. OPA policies are written in Rego, a language that allows for declarative, fine-grained control.

For example, you could integrate OPA with an internal IT administration tool and only allow the members of the ITAdmin group to see groups for any user. In the example shown below, we use a centralized OPA server that enforces policy for a number of different applications. For each application, we use the REST interface of the OPA server to access the policy. For the IT administration tool, we create the following OPA policy:

```
package httpapi.authz
import input

#Allow IT admin members to get anyone’s groups.
allow {
 input.method == “GET”
 input.path = [“users”, “groups”, _]
 input.user == itadmin[_]
}

#DTobin is the only member of ITAdmin.
itadmin = [
 “dtobin”,
]
```

Below is sample code from an IT administration web server. The server uses a form to enter the username, and we then split the path that is accessed. For our example, the user will access /users/groups/, and we convert this into JSON that we can send to the OPA server for validation.

```
# Grab basic information. We assume the value, user, is passed on a form.
http_api_user = request.form['user']

# Get the path as a list (removing leading and trailing /)
# Example: "/users/group/" will become ["users", "group"]
http_api_path_list = request.path.strip("/").split("/")
input_dict = {  # create input to hand to OPA
   "input": {
      "user": http_api_user,
      "path": http_api_path_list, # Ex: ["users", "group", "alice"]
      "method": request.method  # HTTP verb, e.g. GET, POST, PUT, ...
   }
}
# ask OPA for a policy decision
rsp = requests.post("http://opaserver.com:8181/v1/data/httpapi/authz",
      json=input_dict)
if rsp.json()["allow"]:
  # HTTP API allowed
else:
  # HTTP API denied
```

Once the above validation is implemented using OPA, only members of the ITAdmin group can see a user's groups in the IT administration tool.

Conclusion

Policy as Code is a great way to reduce the toil and rubber-stamp approvals that often come with relying on handbook-based policies and procedures. By taking cues from and extending Security as Code across your organization, you are increasing visibility and empowering your users to participate in and understand your security and governance policies. Policy as Code removes the inconsistency, overprovisioning, and even the added documentation that may be scattered across numerous communication channels. Your policy documentation is now enshrined in source code with full revision history. Policy as Code and Security as Code both codify decisions and bring about enhanced security, more consistent governance, reduced toil, and increased visibility for the entire organization.

About Cyral

Cyral delivers enterprise data security and governance across all data services such as S3, Snowflake, Kafka, MongoDB, Oracle and more. The cloud-native service is built on a stateless interception technology that monitors all data endpoint activity in real-time and enables unified visibility, identity federation and granular access controls. Cyral automates workflows and enables collaboration between DevOps and Security teams to automate assurance and prevent data leakage. Cyral is venture-backed by Redpoint, A.Capital, Costanoa and SVCI. Follow the company on Twitter at @CyralInc.

cyral.com/tech-talk