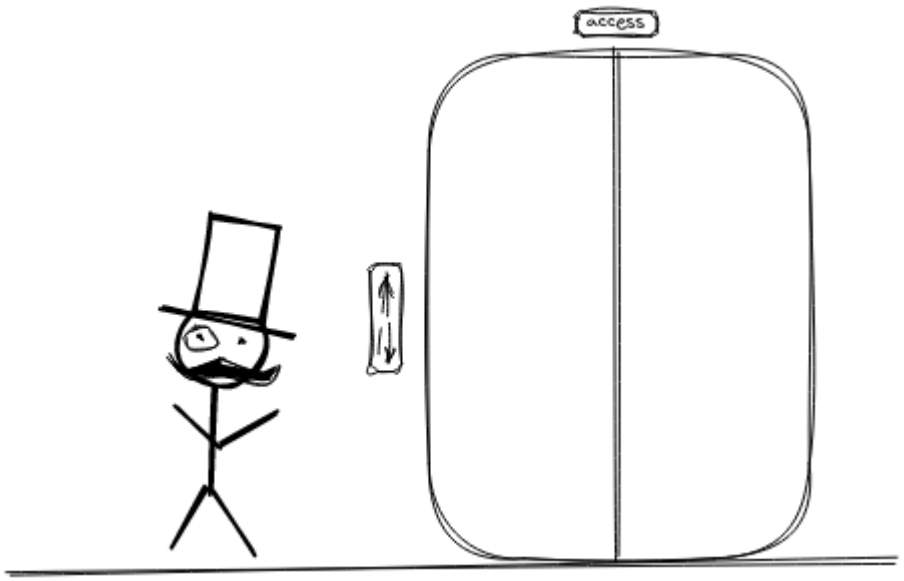
medium.com

The Access Elevator 🛗 - Oda Product & Tech - Medium

Torvald Lekvam

8-9 minutes

This is a short story about how a small system for **on-demand production access** has greatly matured our security and software development habits. Our engineers now have ~zero write access to production, but if they really need it, they can get what they need via auto-grant/peer review policies or break glass procedures.



A developer making a sketch of a developer using the Access Elevator

Motivation

Being a relatively young company, there are still lots of traces of

that energizing startup vibe among our software developers. High change velocity, few rules, and hackathons are examples of the good ones. But not everything is straight up sustainable. A perk from the very start, maybe something the reader can relate to, is the *lingering production admin access*. Bashing into a production shell for some short and sweet data mutations here and there. It speeds things up. While it's convenient, there are many reasons why engineers should not have default write access to production. For us, *distrust* is luckily not one of them; Oda has a remarkable high-trust environment among its engineers. Let me highlight what strive towards:

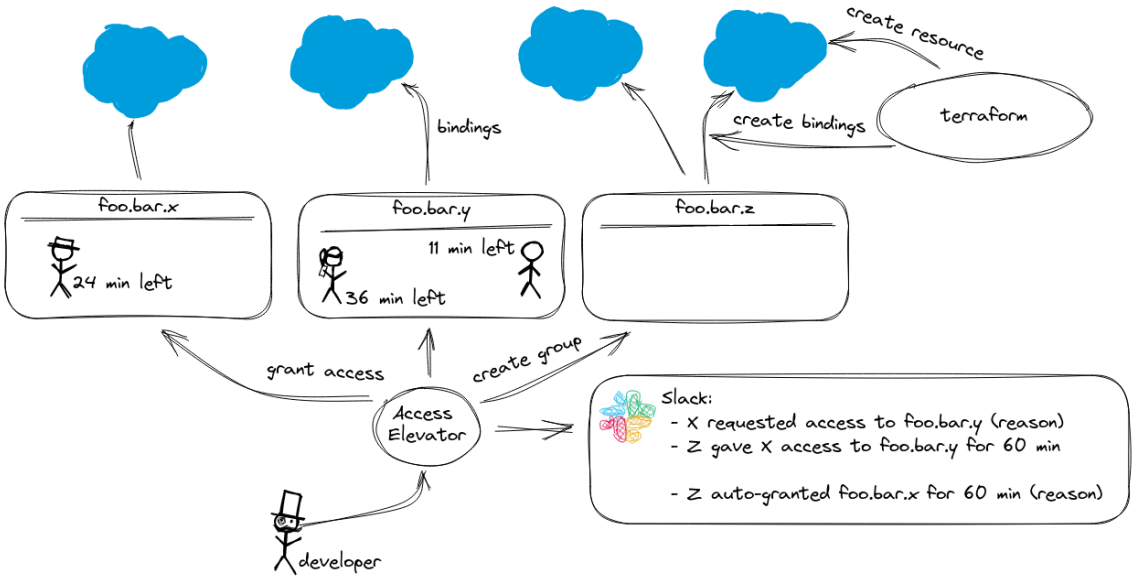
- Foremost, we want to **prevent human mistakes**. As I hope to touch upon in a future blog post, a small terminal whoopsie error can take down an entire fulfillment centers for hours. 🏭💥
- **Reducing the attack surface**. The industry is in constant fear of its privileged accounts being compromised. No better way to mitigate this threat than making the assets close to worthless.
- **No dangling privileged permissions**. It's GDPR heaven; the burden of regular permission reviews is almost eliminated.
- **Audit trails and transparency**. Not only can you see in retrospect who had access to what and when, we also want to prompt our developers with the *why*. This also works proactively, because people need to actually articulate their rationale.

  Ideally, we want all changes to production to go via properly vetted systems. There is mostly common sense behind that, but Google has some excellent material on the matter if you want more (Google Safe Proxies. Page 37). This ensures that changes are peer reviewed in some form, boundaries are set upfront, and changes become traceable. It might even invite you

to write some tests. It also keeps a healthy pressure on keeping review processes and CI/CD pipelines swift and healthy.

Implementation

We built, what we internally call, the «Access Elevator». There are commercial products that do similar things, but we thought the overall complexity was low enough for us to implement it ourselves. Also, it gave us added flexibility to integrate it to existing workflows such as CLI tooling and Slack.



We (always try to) use what we have, and we already use Google Identity and have lots of Google Groups for most of our organizational ACL needs. The Access Elevator is therefor allowed to create Google Groups and defines a set of *role*s tied to them.

```
@dataclass
 class Role:
 name: str
 description: str
 group: EmailAddr
 reviewers: list[EmailAddr] = field(default_factory=list)
 default_duration_min: int = 60
```

An example might be `dns.admin.odacom@oda-cloud.net` – and under normal circumstances this group has no members. As you can imagine, this group is then bound to admin rights for the DNS zone oda.com via IAM rules specified somewhere in its respective Terraform code, like so:

```
resource "google_project_iam_member"
"access_elevator_dns_admin_odacom" {
  project = local.project_id
  role    = "roles/dns.admin"
  member  = "group:dns.admin.odacom@oda-cloud.net"
}
```

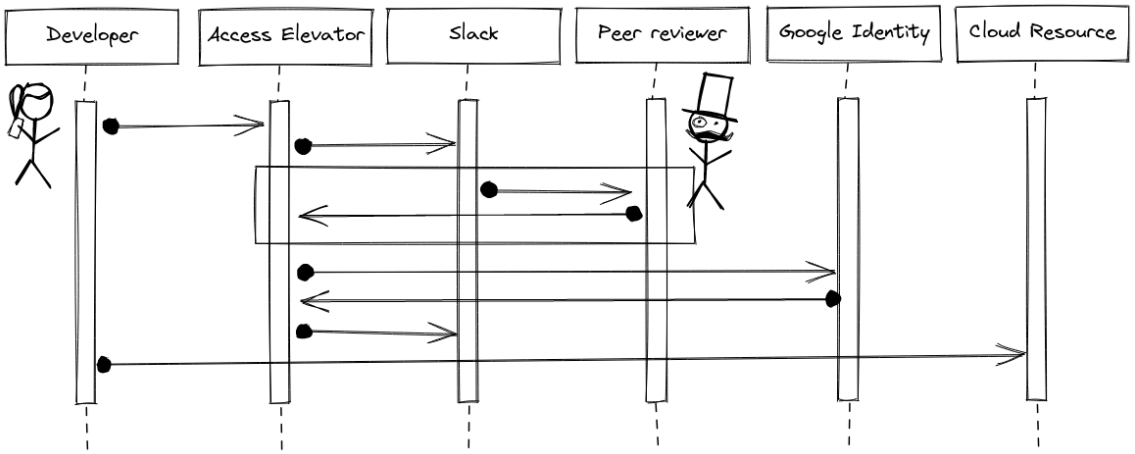
The naming pattern for these roles is `<resource>.<access-level>.<resource-name>`. Other examples could be:

- `project.editor.oda-firewall` which would give you (a slightly custom) *Editor role* in the Google Cloud project `oda-firewall`, or
- `vm.ssh.odacom-db` that would grant you access to Google's Identity-aware Proxy (for SSH) for oda.com's main PostgreSQL instances.

The Access Elevator is *stateless*, and it runs in a Google Cloud function outside our primary runtime: Kubernetes. It's nice that it's stateless — it keeps the complexity down, and it's also nice that it runs outside the Kubernetes once internal parts of Kubernetes start fighting over each other and you need some elevated access to fix it. 💥

If a *peer review* is needed (that depends on the Role you ask for, who you are, if you are on-call or not), the review is requested over Slack, tagging the engineer currently on daytime support duties. Upon review, or if there is an auto-grant policy on that

role, the Access Elevator will politely ask Google (over their oddly confusing identity API) to add the developer to the requested group with a fixed expiry timestamp (default 60 min). The developer can then hack in production. 🤠



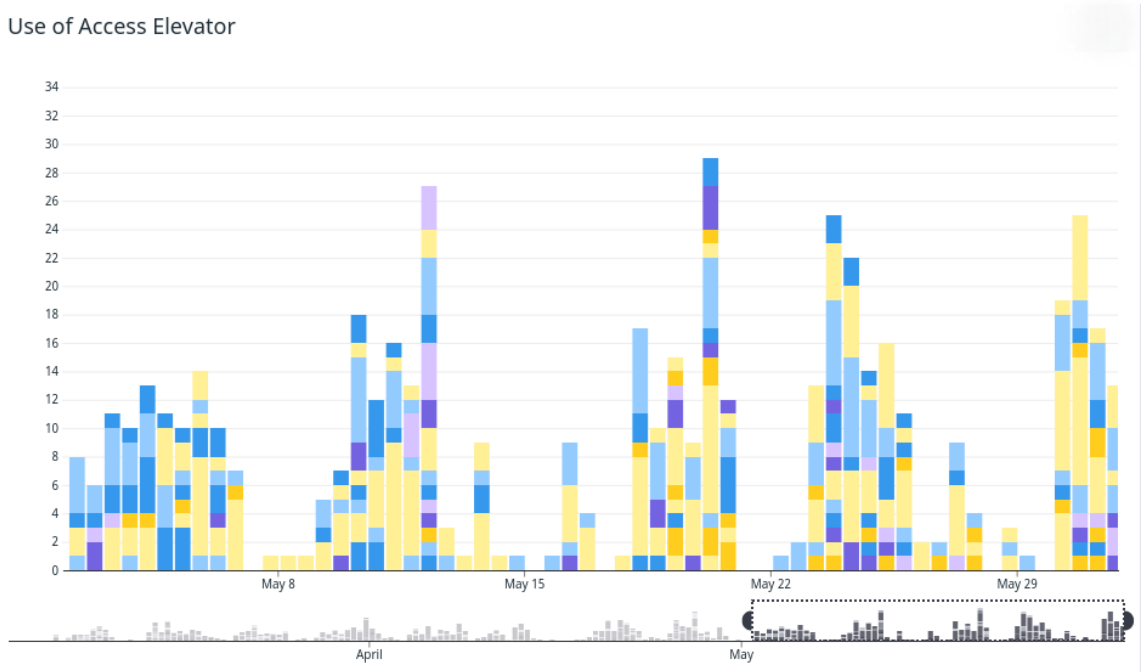
The flow of messages when using the Access Elevator. The box represent a conditional peer-review.

The business logic of the application is about 500 lines of python code, but if you're particularly interested, this is the gist in pseudocode.

Discussion

Since this is just another API, it integrates nicely to our developer CLI toolbox that already does a variety of actions; like `exec`-ing into a running service, fighting a blocking PostgreSQL transaction or inspecting network traffic at one of our fulfillment centers.

In a perfect world, this is only used in break-glass situations. But the world of online groceries is not perfect either – far from so. Most of our groups are still auto-grant, not to rob all developers of their ways of working. But from this tiny control plane, we can now see and understand which patterns and usage exist so that we can make good decisions going forward in terms for security and reliability.



Different roles granted by Access Elevator last 30 days.

Obviously, we want less direct production access. More groups will become subject for peer review over time, which adds a little bit of friction to this conflicting happy path. It's classic nudge theory; at first we can allow for team members to be peer reviewers, but the long game could shift this responsibility solely onto the Reliability Engineers.

We also hope that the required field «reason» nudges people into thinking twice before doing something potentially dangerous. All grants, along with their reasons, are echoed into a company wide and transparent Slack channel. This also gives us the opportunity to uncover known bad patterns of production use, and if applicable, instead redirect people to other and safer methods that might already exist.

Future work

We aspire to the Zero Trust paradigm by always validating requests in our infrastructure, and for HTTP, that means that we inspect every incoming request through a reverse proxy that checks for contextual stuff™. Within that proxy, we enrich

authenticated requests with metadata for internal applications to use, including group memberships. This also includes the groups granted by Access Elevator! At the moment, we only use the Access Elevator for Cloud IAM, but there is also an opportunity here to allow users of our internal applications to do more on-demand access through the same means. Today, a handful of superusers have access to customer data via back-office systems, solely for edge-case purposes. With an access-on-demand system like the Access Elevator, we have evaluated that it's *only* customer service that actually needs access to our customer data on a permanent basis.

We're also looking into how we can add additional authenticating factors, like hardware tokens. This will even further mitigate the threat of compromised accounts. In some cases, where risk is low, we can even use it as a substitute for peer reviewing. But if you want to play in God mode, you might need both!