CASE STUDY

How TrueCar modernized the way they manage and deploy Terraform at scale

spacelift

TrueCar

340
Company size

96
Engineering team size

AWS, Terraform
Stack

With Spacelift's help, TrueCar devised an IaC solution tailored to their infrastructure strategy while adding critical visibility and guardrails.

Spacelift enabled a modern, GitHub flow approach with direct VCS integration, webhooks, and drift detection for visibility and control.

Spacelift policies make it easy for TrueCar to control how and when stacks are planned and applied.

Summary

Automotive digital marketplace TrueCar needed help aligning their infrastructure-as-code (IaC) strategy with their cutting-edge use of technology elsewhere in the company. An archaic, largely manual approach to IaC was hampering development, causing multiple bugs, and intensifying developers' stress levels. Supported by Spacelift's laser-sharp UI clarity, minutely detailed documentation, and exceptional flexibility, TrueCar has transformed how it manages and deploys Terraform at scale today.

TrueCar, Inc. is a leading automotive digital marketplace that lets auto buyers and sellers connect to its nationwide network of Certified Dealers.

Software developer Yongjie Lim spoke to us about the SRE team at TrueCar and their intensive efforts to transform their IaC strategy.

The challenge for TrueCar

Yongjie gets straight to the point. "Our IaC strategy was sorely in need of improvement."

The SRE team had tried and failed with different approaches to IaC in a long history of using Terraform. Initially, they configured key infrastructure concerns in separate repositories under one Terraform organization in source control and used a barebones pull request workflow consisting solely of a peer review and a basic deployment workflow. Unsurprisingly, this prompted persistent issues with drift, state inconsistencies between branches, and other configuration headaches that stymied new projects.

The SRE team tried a monorepo approach next. They separated infrastructure pieces by folder in one large repository, further subdividing each piece by AWS account deployment (sa, staging, prod, etc). Instead of using pull requests, they were expected to commit configuration changes to master and deploy them immediately. As Yongjie recalls, "this might occasionally prevent stray configurations from sticking on some month-old branch, but it still prompted similar issues when configuration and actual resources did not match. A module reference might not have been updated in source control but deployed out locally. Or code might be committed and never rolled out." The monorepo made Terraform difficult to work through, with long-term drift making updating particularly challenging.

Why TrueCar chose Spacelift

The situation had come to a head by the end of 2022. Exasperated and exhausted, TrueCar's SRE team tried several IaC deployment solutions. And then they discovered Spacelift — specifically Spacelift's documentation.

"The docs were really good, ridiculously detailed," recalls Yongjie. As they explored the platform further, they were equally impressed by the clarity of the user interface and the flexibility of the product itself. With Spacelift's help, TrueCar was able to devise an IaC solution tailored to their infrastructure strategy while adding much-needed visibility and guardrails to the process.

TrueCar's Spacelift experience

Part (i)

The first step was to manage the monorepo.

So just how did they do it? As Yongjie explains, "we manage our Terraform in a consolidated GitHub monorepo, with each folder defining a project (some infrastructure concern, such as VPC, SAML, etc). Each project is then further subdivided into deployment folders that each represent a deployment of the project to a given AWS account and/or region." Here is an example of what the structure might look like:

```
└── vpc
    ├── README.md
    ├── dev
    │   ├── main.tf -> ../main.tf
    │   ├── state.tf
    │   └── terraform.tfvars
    ├── main.tf
    ├── prod
    │   ├── main.tf -> ../main.tf
    │   ├── state.tf
    │   └── terraform.tfvars
    ├── qa
    │   ├── main.tf -> ../main.tf
    │   ├── state.tf
    │   └── terraform.tfvars
    └── staging
        ├── main.tf -> ../main.tf
        ├── state.tf
        └── terraform.tfvars
```

Each deployment has its own dedicated vars and state file, while (usually) sharing a symlinked configuration file that references the module(s) required for the project.

To adopt the monorepo onto the platform, the TrueCar team created a central, administrator Spacelift stack responsible for managing Spacelift resources for each deployment for every project in the monorepo. Yongjie outlines how the stack works:

- **Step 1**
  First, they wrote a script that combs through the monorepo, generating a settings file with a configuration block for each possible deployment.
- **Step 2**
  Then they slotted that script into a before_init hook on the admin stack. Spacelift offers a comprehensive lifecycle hook system that enables you to run arbitrary code before and/or after key stages in the lifecycle of a Terraform deployment. TrueCar leveraged the before_init hook to run the script and dynamically generate the settings file at runtime, before any Terraform plan was kicked off.
- **Step 3**
  Finally, they configured their administrator stack to run whenever a change was made to their monorepo via the project root setting, ensuring that the monorepo and Spacelift resources are always kept up-to-date.

Part (ii)

The next challenge of adapting the monorepo to Spacelift was controlling how and when Stacks were planned and applied. Spacelift makes it easy with policies, allowing TrueCar to define these complex decisions with policy as code.

The first policy TrueCar applies to each stack is a push policy. Push policies allow control over what code commits generate plans for a given stack in Spacelift, with rules based around a fairly comprehensive event schema. TrueCar needed a policy that could trigger a Spacelift plan for a deployment whenever any files in its path changed, as well as triggering whenever a change was made to shared files in the deployment's project subdirectories:

```
package spacelift

track {
    input.push.branch == input.stack.branch
}

propose {
    affected
}

ignore {
    input.push.tag != ""
}

# delimiter for file path
delimiter := "/"

# generate all affected files directory names
# use a combination of set and array comprehension
# see: https://www.openpolicyagent.org/docs/latest/policy-language/#comprehensions
affected_dirnames := {x | x = {dirname |
        path := input.pull_request.affected_files[_]
        t := trim(path, delimiter)
        d := split(t, delimiter)
        dirname := concat(delimiter, array.slice(d, 0, count(d) - 1))
}[_]}

# generate stack directory names
stack_dirnames := {x | x = {dirname |
        paths := split(trim(input.stack.project_root, delimiter), delimiter)
        path := paths[_]
        dirname := concat(delimiter, array.slice(paths, 0, i + 1))
}[_]}

# if at least one affected dir exists in stack dir, then this rule becomes true else false
affected {
        affected_dirnames[_] == stack_dirnames[_]
}

# https://docs.spacelift.io/concepts/policy#sampling-policy-inputs
sample { true }
```

Yongjie points out that "Spacelift also allows sampling policy evaluations (configured with the last block in the policy above), with a robust policy tester that allows us to iterate on this policy with live event data!"

The second policy TrueCar applied to each stack is a plan policy. Plan policies control how plans are deployed — whether they wait for manual confirmation or they autodeploy. Using a similarly exhaustive list of event attributes from the result of the **terraform plan**, the team was easily able to configure their stacks to automatically deploy on merges, while requiring confirmation on manual plans kicked off by a human operator (for triaging use cases).

```
package spacelift

warn["Manually triggered runs require confirmation"]{
  not is_null(input.spacelift.run.triggered_by)
  not startswith(input.spacelift.run.triggered_by, "api:")
}

sample { true }
```

Warn, or require confirmation, for any runs that were triggered by an individual user. This policy additionally has a stipulation for allowing graphql api-triggered runs to autodeploy, which TrueCar leverages for other automation workflows involving Spacelift stacks.

Part (iii)

The next step was to improve the process. TrueCar's old process had inefficiencies stemming from a system that encouraged pushes to master without proper rules and reviews in place. "Spacelift helped us move to a modern, GitHub flow approach with direct VCS integration, webhooks, and drift detection that gave us the visibility and control we needed over our infrastructure," Yongjie points out.

- **VCS integration**
  Spacelift provides first-class integration with several VCS providers out of the box – in TrueCar's case, this was GitHub Enterprise (GHE). With the GHE integration, Spacelift automatically generates commit statuses and issue comments for each commit's Spacelift-executed terraform plan, enabling reviewers to quickly evaluate changesets without leaving the pull request page, or investigate proposed modifications in detail by following status links to the Spacelift run.
- **Webhooks**
  Webhooks are a first-class feature for Spacelift stacks, sending stack notifications to any number of configured HTTP endpoints. Spacelift generates webhook events for every step in a stack's overall run. TrueCar captures any failures and posts them to a Slack channel for manual remediation. "Eventually, we could easily envision a system where we integrate runs in a SlackOps fashion, notifying on each stage of a project's deployment and even controlling apply confirmations from Slack directly!" says Yongjie.
- **Drift detection**
  To audit the state of their infrastructure, TrueCar uses Spacelift's drift detection feature to monitor which systems have drifted out of sync. Spacelift enables schedule defining, periodically running plans against a given stack and checking to see if any resources have drifted. From there, stacks can either be left in a confirmation state to resolve the drift or configured to auto-resolve the drift by applying the drift detection plan. The TrueCar team opted to simply generate a report of stacks that have drifted using Spacelift's graphql API, giving them the flexibility to triage drift on their own time.

Spacelift's impact on TrueCar

Yongjie is very clear about the transformation Spacelift has delivered for TrueCar: "Spacelift provided us with the flexibility, clarity, and features we needed to bring our IaC management in line with best practices, with project management and history, auto-deployment, and policies to control our infrastructure the way that works for us. And we haven't even touched on the more advanced features that could fit a team's use case!"

The platform is moving fast: The TrueCar team recently configured a newly-released custom overview for its stacks and stopped through a detailed changeset breakdown of a significant pull request with the platform's improved diff visualizer.

Yongjie urges other organizations to investigate the platform. "Give Spacelift a try, or run through the docs – you may find that Spacelift is the platform your company needs to modernize your IaC strategy!"