Terraform Enterprise

Intro to Sentinel

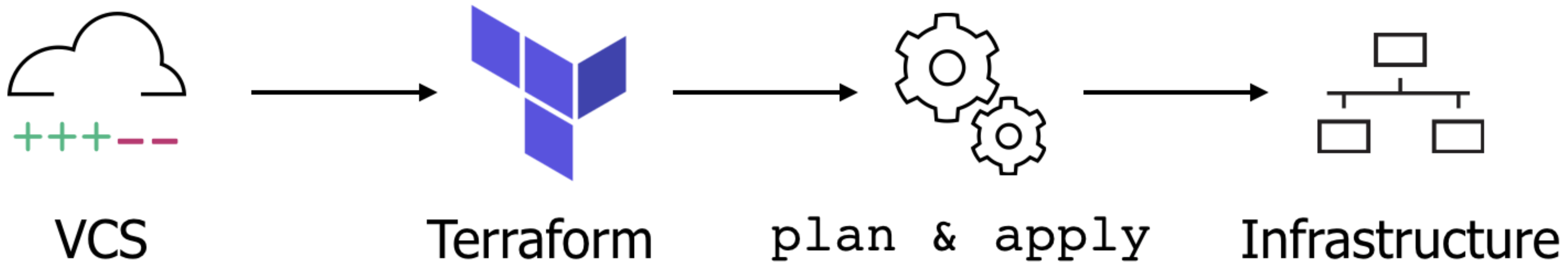
HashiCorp

Agenda

- Imports and Mock Data
- Writing Policies for Terraform Enterprise
- The Sentinel CLI and Policy Testing
- Policy Sets & the Terraform Enterprise workflow

Policy-as-Code with Sentinel

Terraform application workflow



Policy-as-Code

- Treat policies as applications
- Store in version control
- Automate enforcement and review
- Automate logic testing
- Proactive vs. reactive

Policy Use Cases

Security Standards

- Require all S3 buckets to use the private ACL
- Restrict roles the AWS provider can assume
- Forbid or allow only certain resources, providers or data sources

Audit Tracking

- Enforce explicit ownership in resources
- Review an audit trail for Terraform Enterprise operations

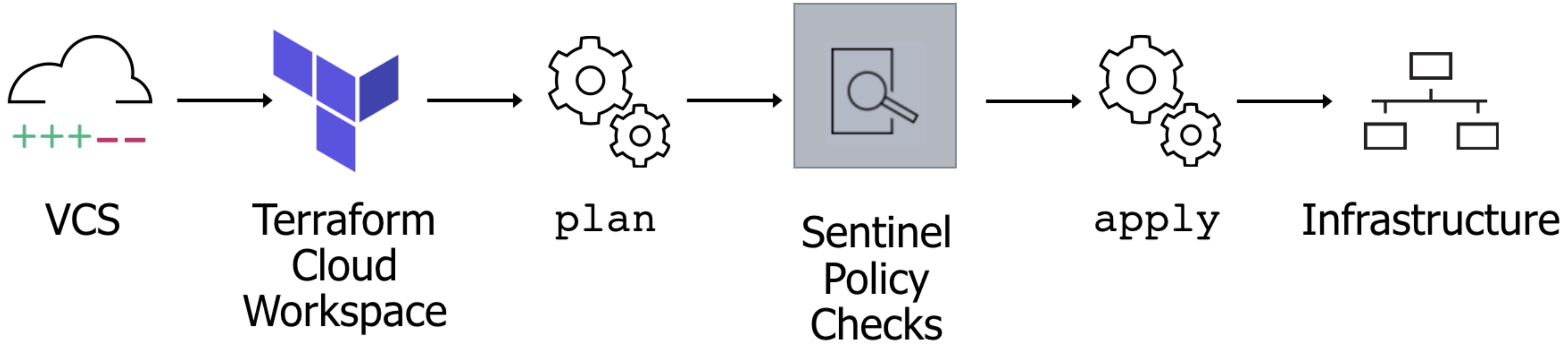
Resource Restriction

- Limit the size of VMs and clusters for cost
- Enforce mandatory tagging on resources built with Terraform
- Restrict modules to your organizations Private Module Registry

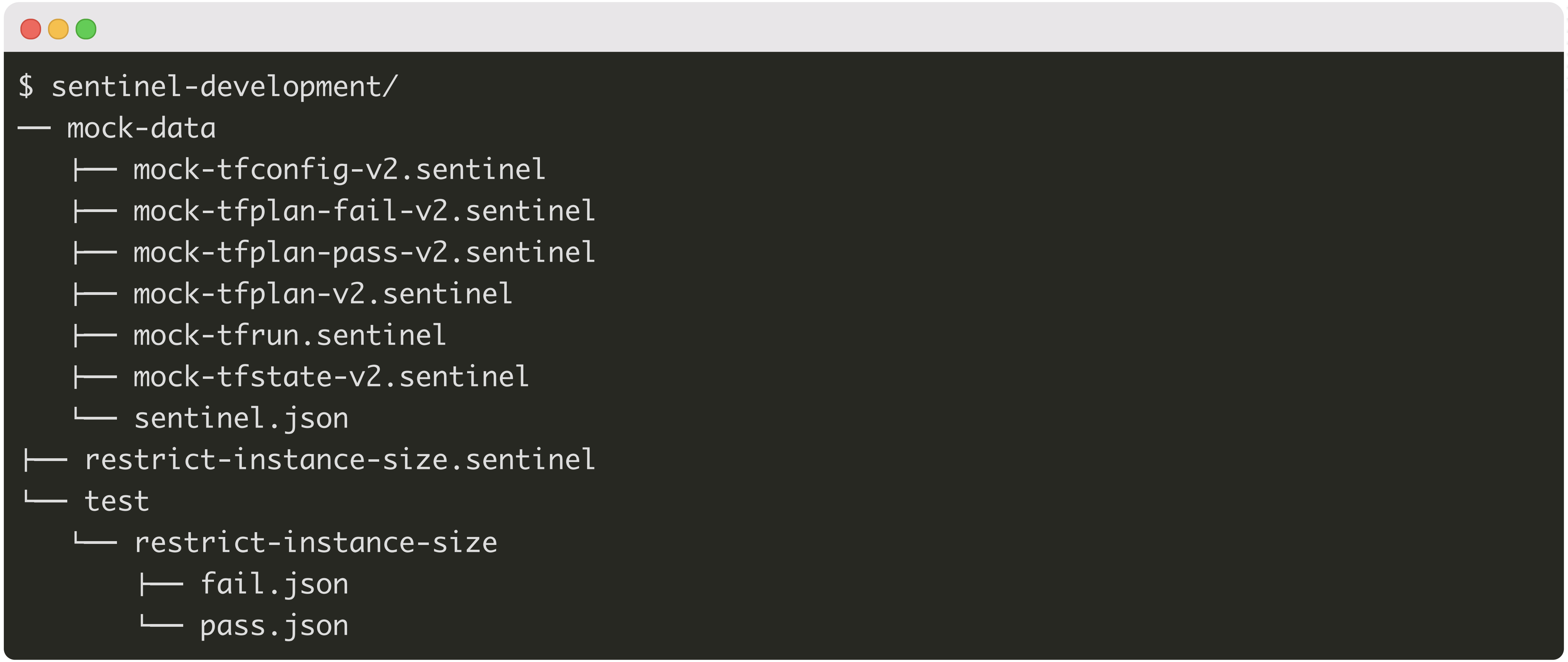
Example policy requirements

Problem:	Your organization is over-provisioning EC2 instances
Scope:	Dev Terraform Workspace
Solution:	Limit maximum compute size to `t2.micro`

Terraform application workflow with Sentinel



Sentinel Development File Structure



```
$ sentinel-development/
── mock-data
   ├── mock-tfconfig-v2.sentinel
   ├── mock-tfplan-fail-v2.sentinel
   ├── mock-tfplan-pass-v2.sentinel
   ├── mock-tfplan-v2.sentinel
   ├── mock-tfrun.sentinel
   ├── mock-tfstate-v2.sentinel
   └── sentinel.json
├── restrict-instance-size.sentinel
└── test
   └── restrict-instance-size
       ├── fail.json
       └── pass.json
```

Mock Data

HashiCorp

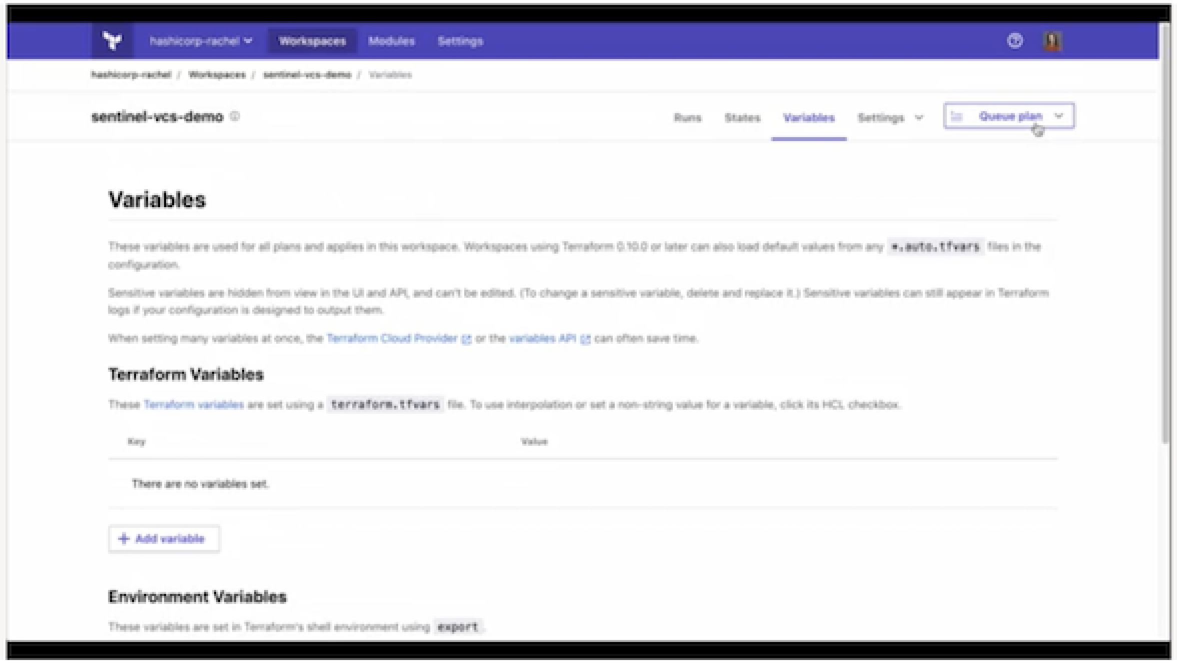
Mock Data for Sentinel Policies

- Generated from existing Terraform configurations
- Exposes data in a Terraform plan, state, and configuration—including sensitive values
- Requires specific permissions to access

https://www.terraform.io/docs/cloud/sentinel/mock.html

Plan and download mocks

- Queue a plan in the UI or `terraform plan` in CLI
- Download Sentinel mocks when plan finishes



Variables

These variables are used for all plans and applies in this workspace. Workspaces using Terraform 0.10.0 or later can also load default values from any `*.auto.tfvars` files in the configuration.

Sensitive variables are hidden from view in the UI and API, and can't be edited. (To change a sensitive variable, delete and replace it.) Sensitive variables can still appear in Terraform logs if your configuration is designed to output them.

When setting many variables at once, the Terraform Cloud Provider or the variables API can often save time.

Terraform Variables

These Terraform variables are set using a `terraform.tfvars` file. To use interpolation or set a non-string value for a variable, click its HCL checkbox.

Key	Value
There are no variables set.	

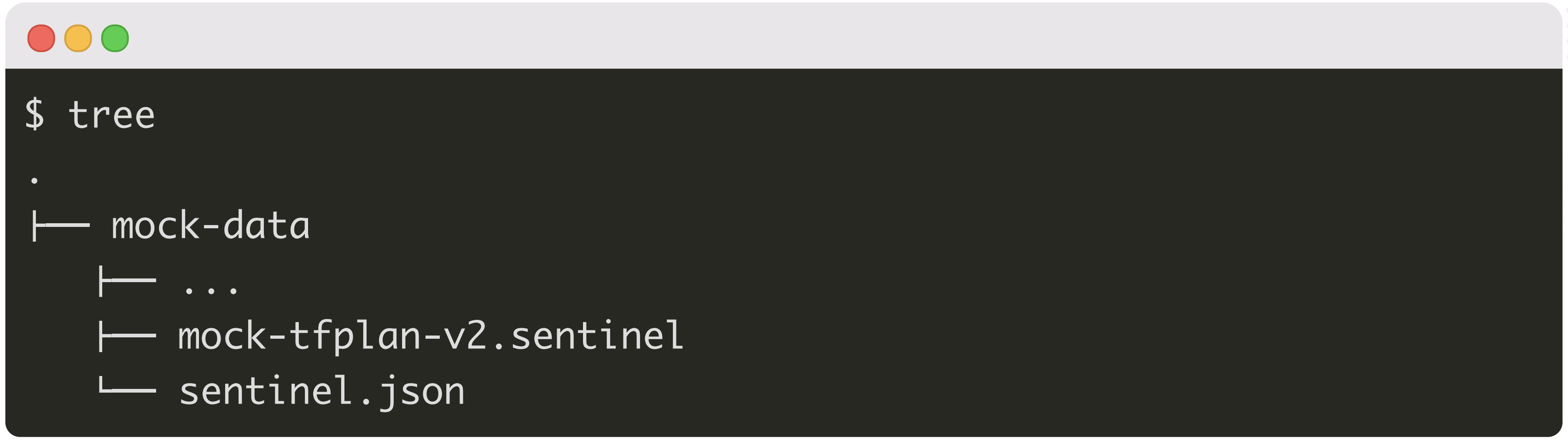
Environment Variables

These variables are set in Terraform's shell environment using `export`.

[How to download Sentinel Mocks from TFE?]

Mock data structure

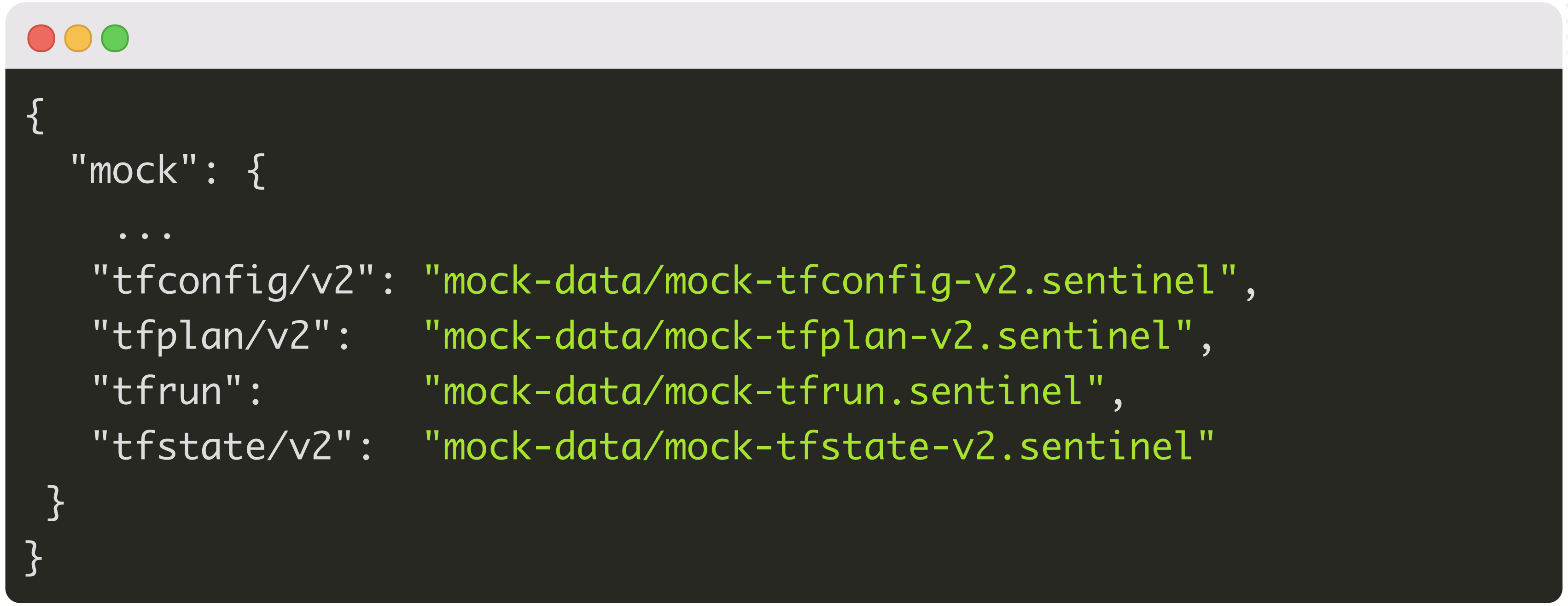
Mock data `zip` file contents



```
$ tree
.
├── mock-data
    ├── ...
    ├── mock-tfplan-v2.sentinel
    └── sentinel.json
```

Sentinel mock paths

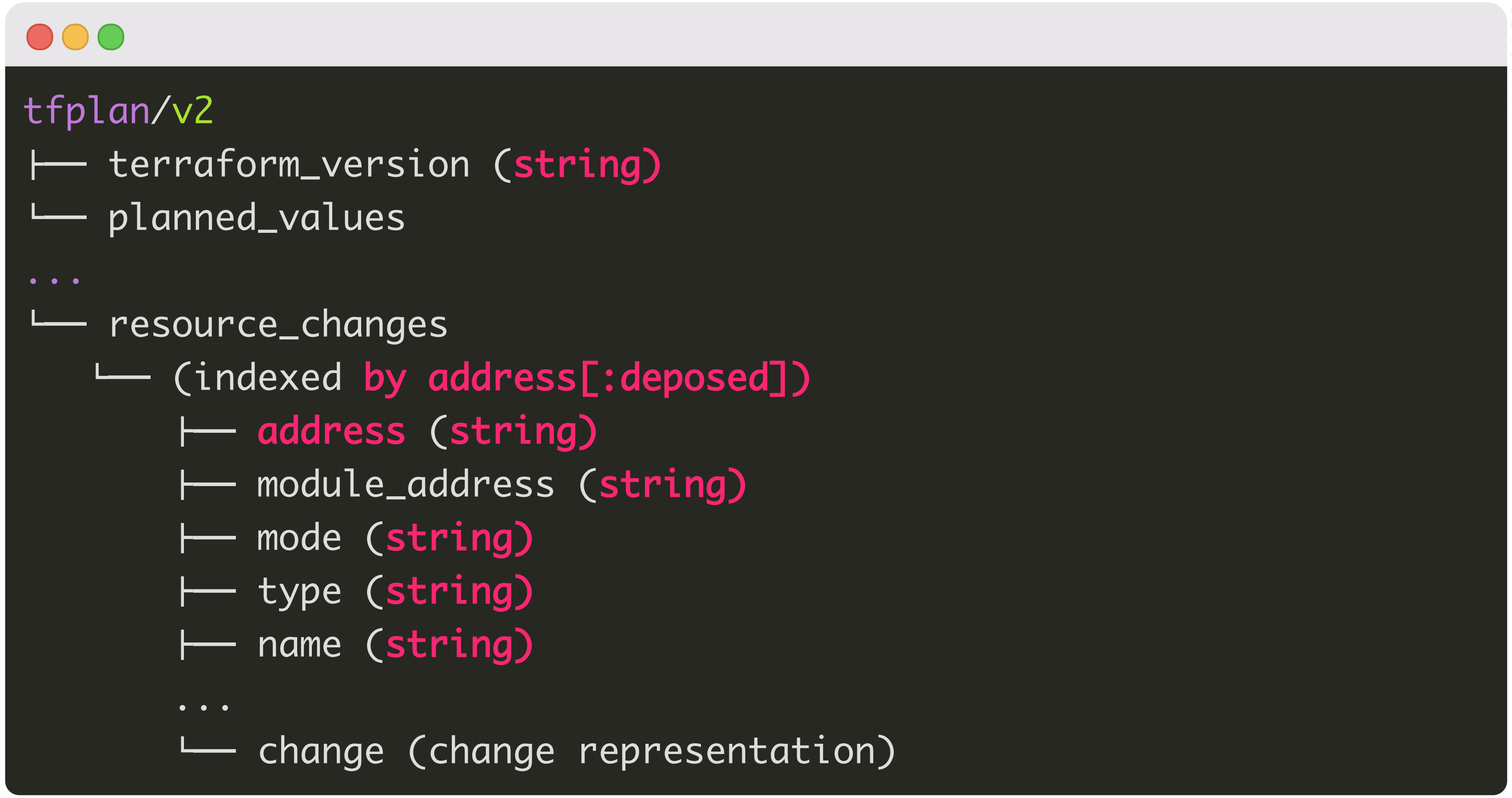
Terraform Enterprise generates `sentinel.json` automatically



```
{
  "mock": {
    ...
   "tfconfig/v2": "mock-data/mock-tfconfig-v2.sentinel",
   "tfplan/v2":   "mock-data/mock-tfplan-v2.sentinel",
   "tfrun":       "mock-data/mock-tfrun.sentinel",
   "tfstate/v2":  "mock-data/mock-tfstate-v2.sentinel"
 }
}
```

Mock data types

Data types contained in `mock-tfplan-v2.sentinel`



```
tfplan/v2
├── terraform_version (string)
└── planned_values
...
└── resource_changes
    └── (indexed by address[:deposed])
        ├── address (string)
        ├── module_address (string)
        ├── mode (string)
        ├── type (string)
        ├── name (string)
        ...
        └── change (change representation)
```

Writing Sentinel Policies for Terraform Enterprise

Import data

Terraform imports	Import use cases	
tfplan	Resource size restriction	☒
tf-config	Required modules + Provider restriction	✓
tfrun	Cost Estimation + Organization	📋
tfstate	Version validation	✓

https://docs.hashicorp.com/sentinel/intro/getting-started/imports/

Imports in policies

Similar to libraries or external plugins

- External data for policy decisions
- Standard and product specific imports
- Mock data imports from Terraform Enterprise

https://www.terraform.io/docs/cloud/sentinel/import/index.html

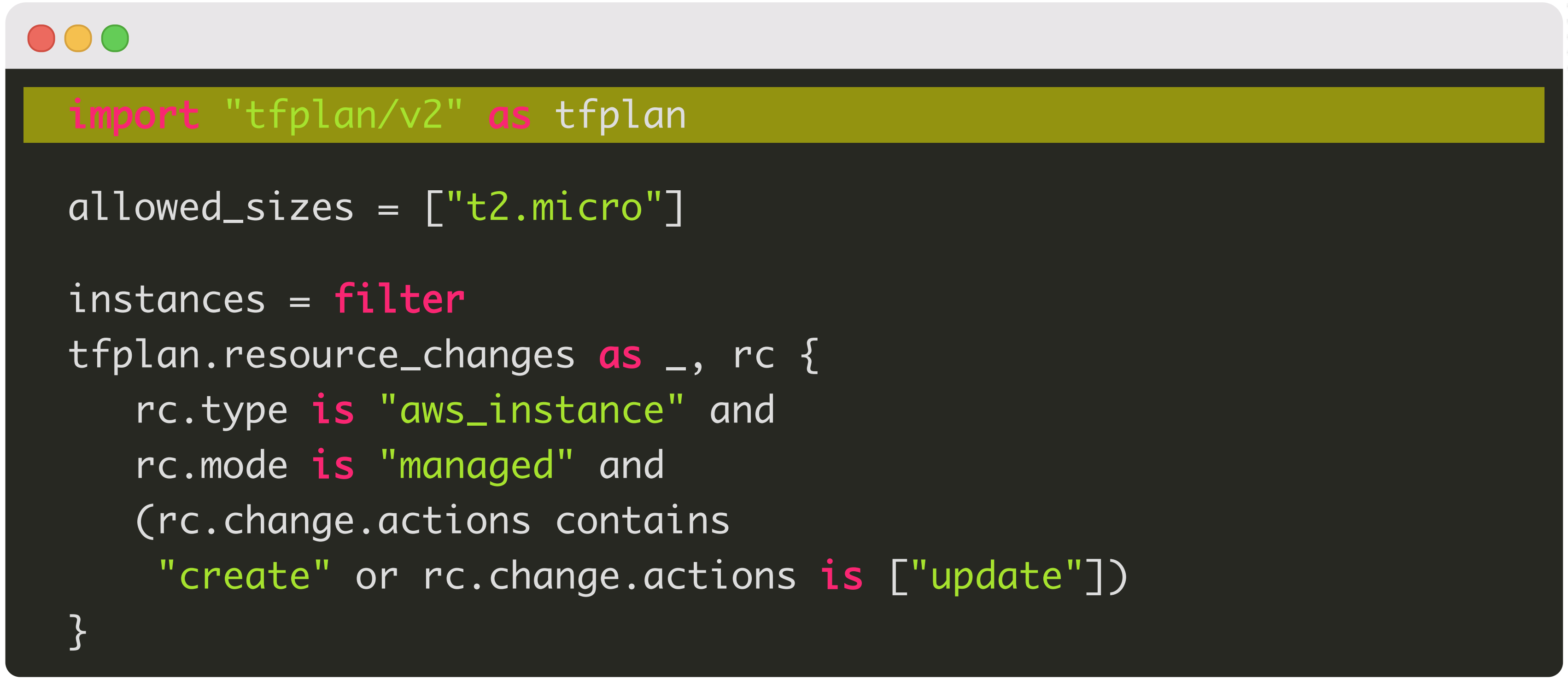
Example Scenario

Our Dev environment bills are high but utilization is very low.

Conclusion: Resources are being over-provisioned.

Example Scenario

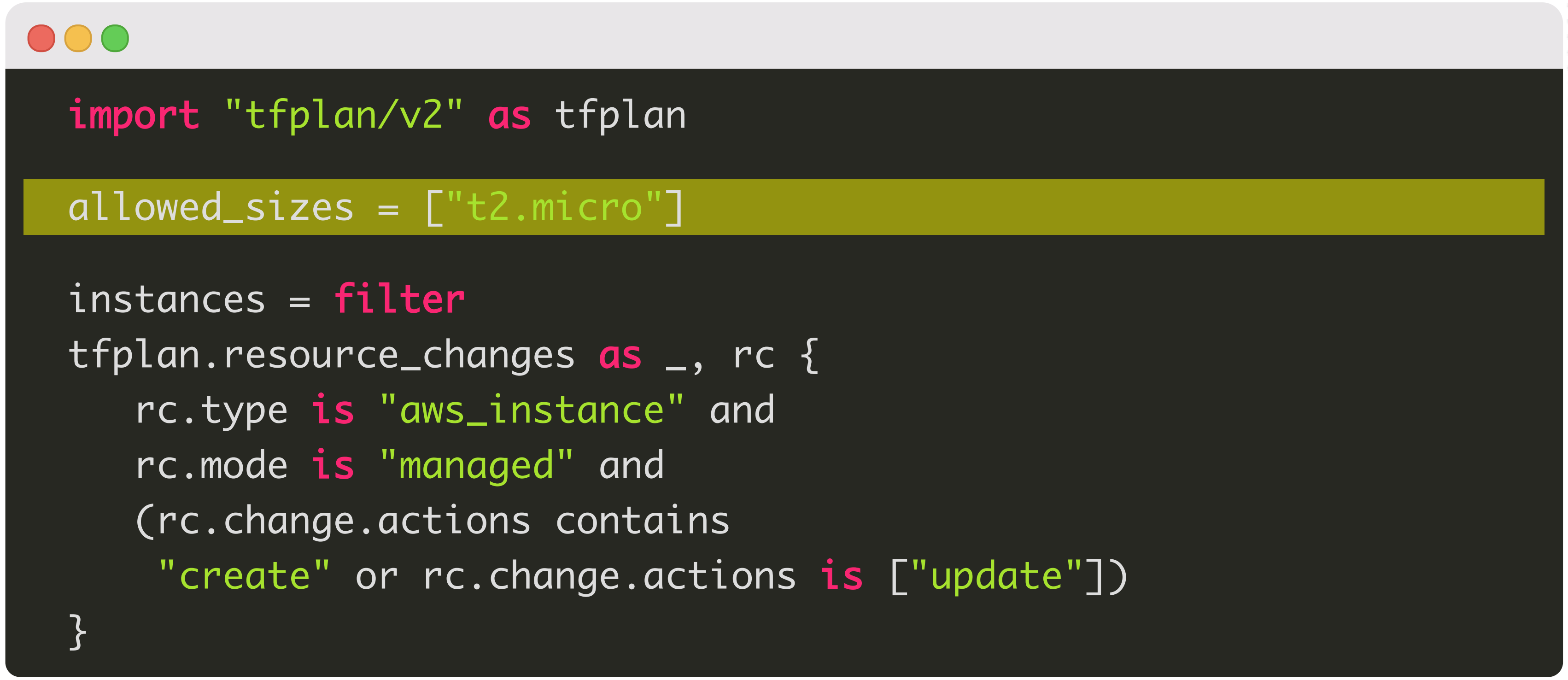
Importing data for your policy



```
import "tfplan/v2" as tfplan

allowed_sizes = ["t2.micro"]

instances = filter
tfplan.resource_changes as _, rc {
   rc.type is "aws_instance" and
   rc.mode is "managed" and
   (rc.change.actions contains
    "create" or rc.change.actions is ["update"])
}
```

Example Scenario

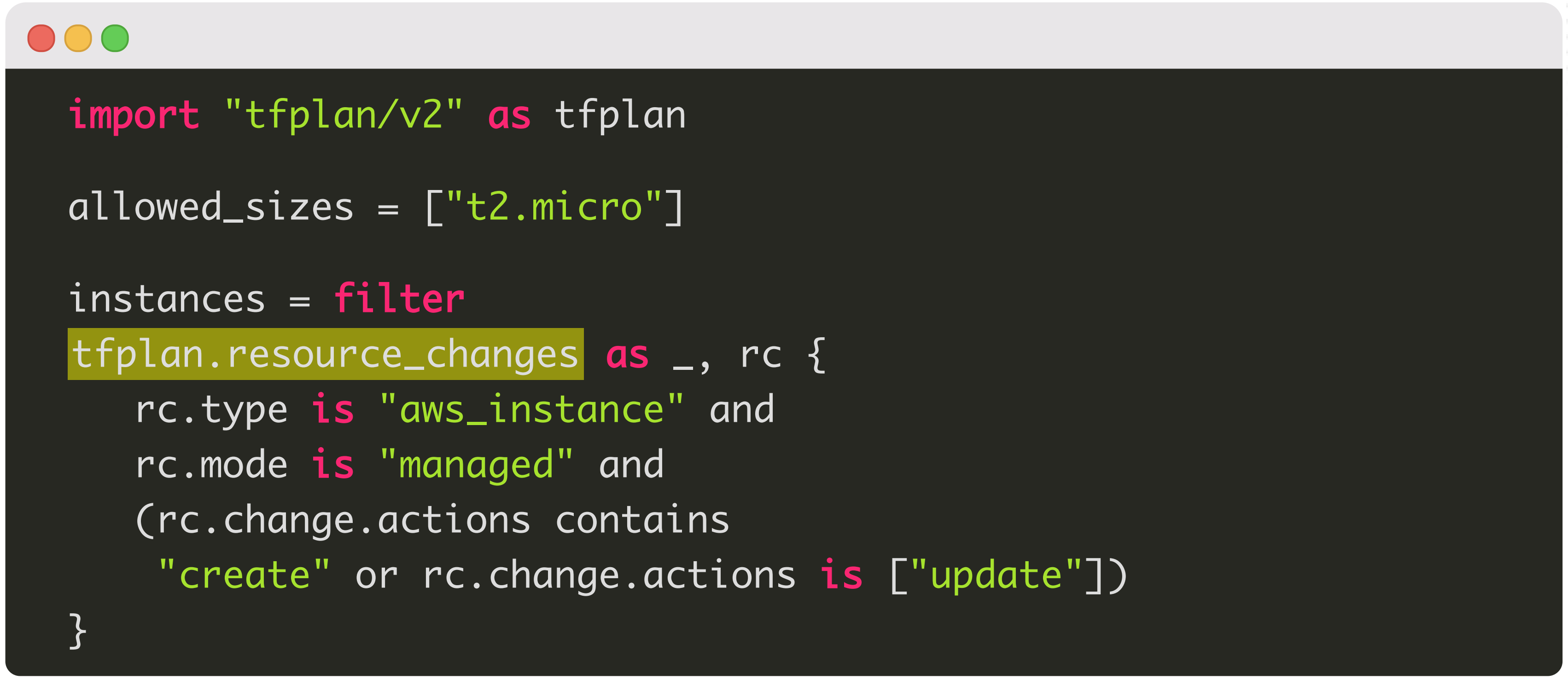
Defining and filtering parameters in your policy



```
import "tfplan/v2" as tfplan

allowed_sizes = ["t2.micro"]

instances = filter
tfplan.resource_changes as _, rc {
   rc.type is "aws_instance" and
   rc.mode is "managed" and
   (rc.change.actions contains
    "create" or rc.change.actions is ["update"])
}
```

Example Scenario

Using imported data to get information about resources



```
import "tfplan/v2" as tfplan

allowed_sizes = ["t2.micro"]

instances = filter
tfplan.resource_changes as _, rc {
   rc.type is "aws_instance" and
   rc.mode is "managed" and
   (rc.change.actions contains
    "create" or rc.change.actions is ["update"])
}
```

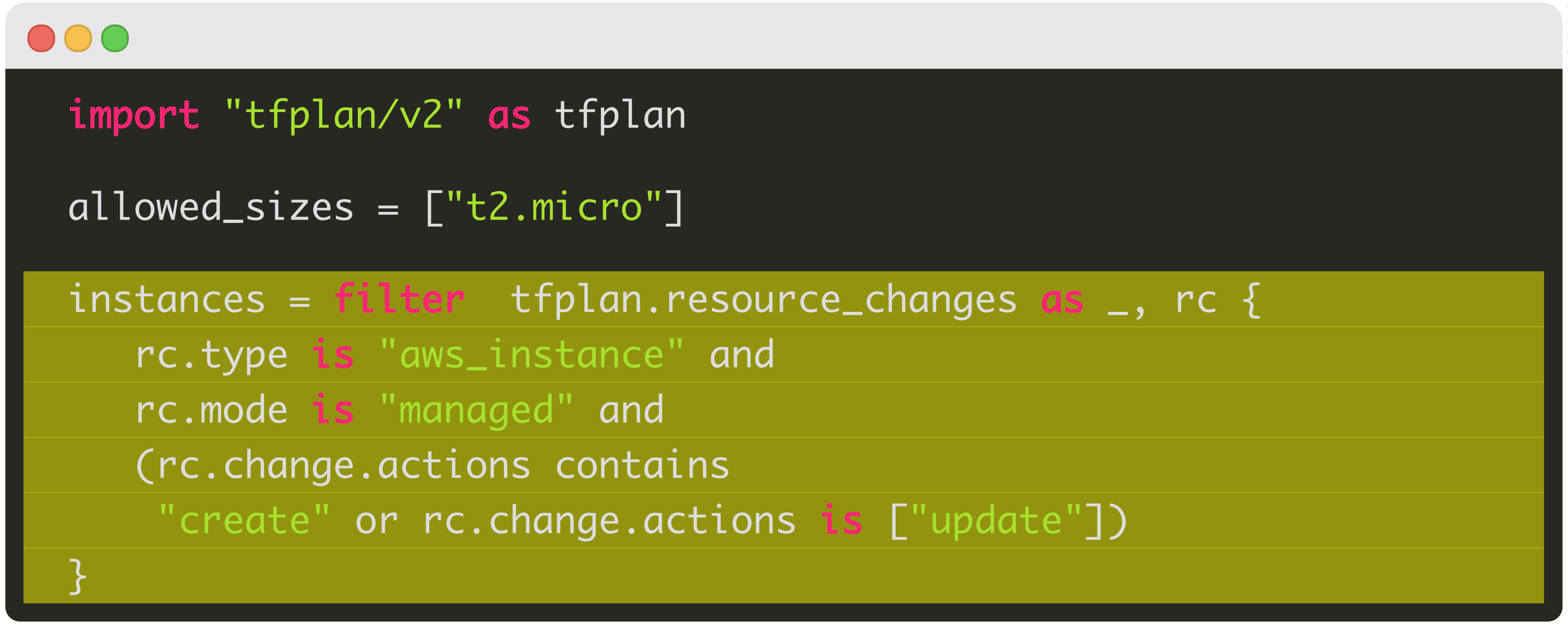
Finding data in import collections

```
resource_changes = {
  "aws_instance.web": {
    "address": "aws_instance.web",
    "change": {
      "actions": [ "create", ],
      "after": {
        ...
        "instance_type: "t2.micro",
        ...
      }
      "mode":             "managed",
      "provider_name":    "aws",
      "type":             "aws_instance",
   },
}
```

```
import "tfplan/v2" as tfplan

allowed_sizes = ["t2.micro"]

instances = filter tfplan.resource_changes
as _, rc {
  rc.type is "aws_instance" and
  rc.mode is "managed" and
  (rc.change.actions contains "create" or
   rc.change.actions is ["update"])
}
...
```

Example Scenario

Defining and filtering parameters in your policy



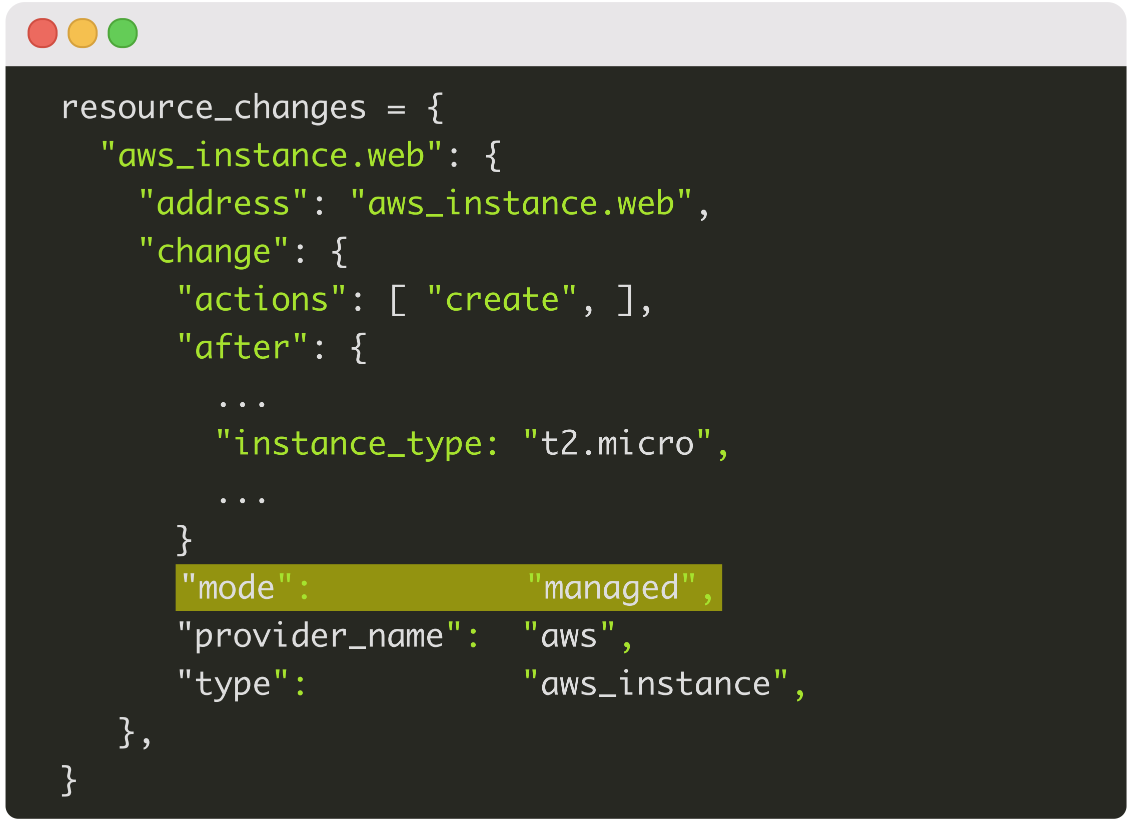
```
import "tfplan/v2" as tfplan

allowed_sizes = ["t2.micro"]

instances = filter  tfplan.resource_changes as _, rc {
    rc.type is "aws_instance" and
    rc.mode is "managed" and
    (rc.change.actions contains
     "create" or rc.change.actions is ["update"])
}
```

Filter by resource type

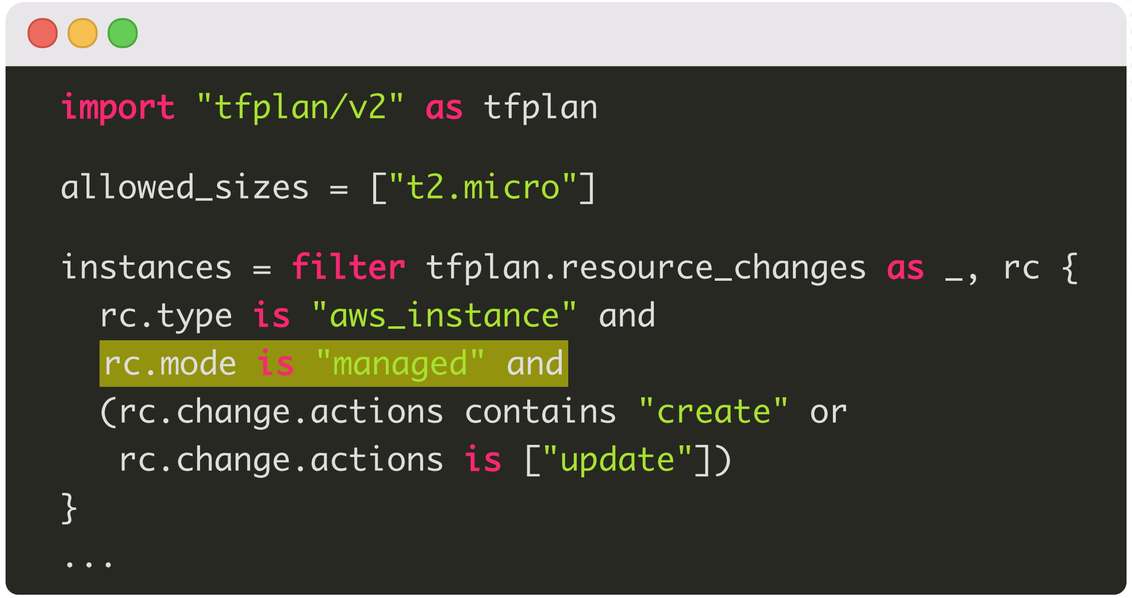
```
resource_changes = {
  "aws_instance.web": {
    "address": "aws_instance.web",
    "change": {
      "actions": [ "create", ],
      "after": {
        ...
        "instance_type: "t2.micro",
        ...
      }
      "mode":           "managed",
      "provider_name":  "aws",
      "type":           "aws_instance",
   },
}
```

```
import "tfplan/v2" as tfplan

allowed_sizes = ["t2.micro"]

instances = filter tfplan.resource_changes as _, rc {
  rc.type is "aws_instance" and
  rc.mode is "managed" and
  (rc.change.actions contains "create" or
   rc.change.actions is ["update"])
}

...
```

Filter by resource mode



```
resource_changes = {
  "aws_instance.web": {
    "address": "aws_instance.web",
    "change": {
      "actions": [ "create", ],
      "after": {
        ...
        "instance_type: "t2.micro",
        ...
      }
      "mode":              "managed",
      "provider_name":   "aws",
      "type":              "aws_instance",
  },
}
```



```
import "tfplan/v2" as tfplan

allowed_sizes = ["t2.micro"]

instances = filter tfplan.resource_changes as _, rc {
  rc.type is "aws_instance" and
  rc.mode is "managed" and
  (rc.change.actions contains "create" or
   rc.change.actions is ["update"])
}
...
```

Filter-by-action

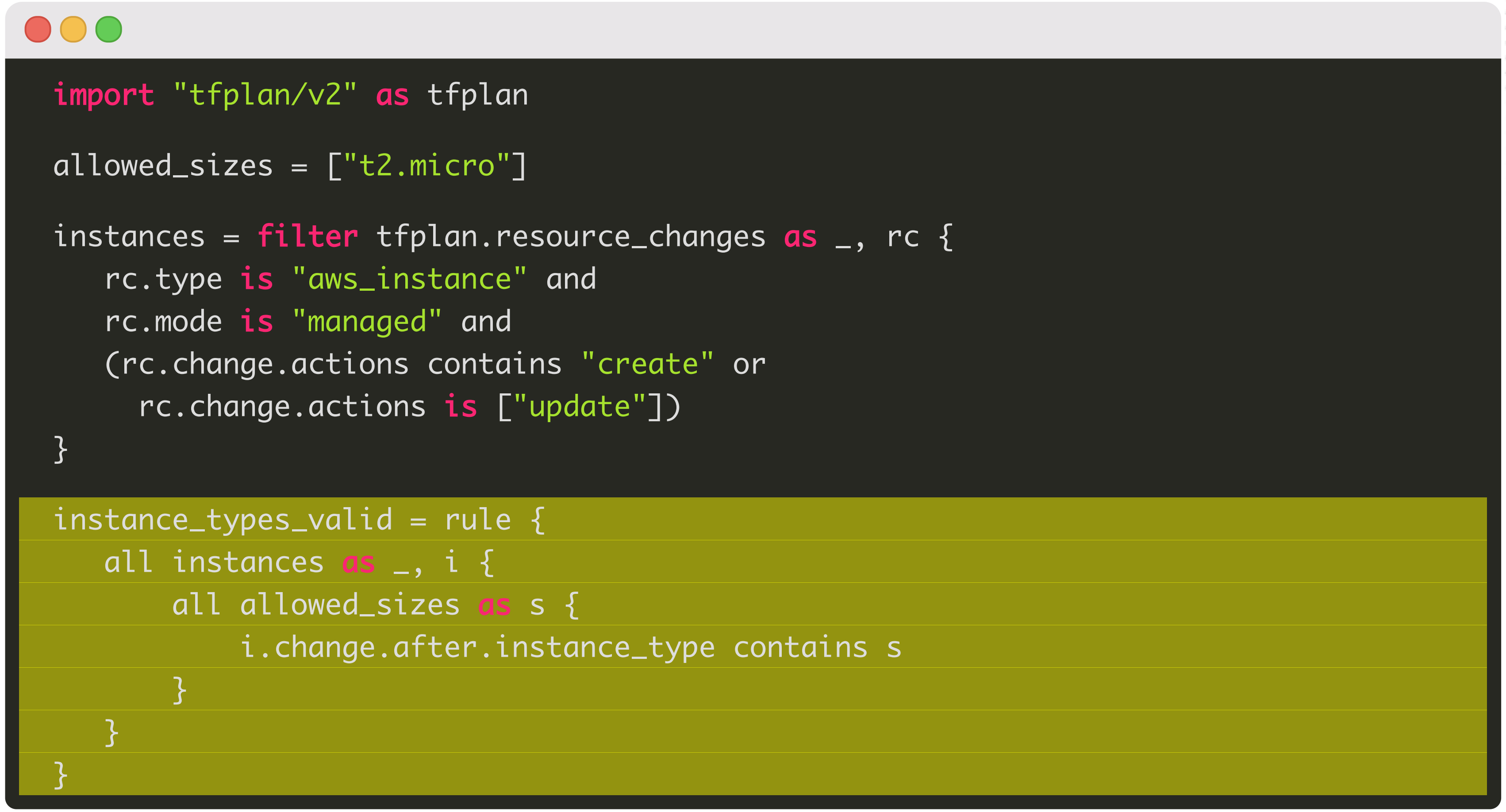
```
resource_changes = {
  "aws_instance.web": {
    "address": "aws_instance.web",
    "change": {
      "actions": [ "create", ],
      "after": {
        ...
        "instance_type: "t2.micro",
        ...
      }
      "mode":             "managed",
      "provider_name":   "aws",
      "type":             "aws_instance",
   },
}
```

```
import "tfplan/v2" as tfplan

allowed_sizes = ["t2.micro"]

instances = filter tfplan.resource_changes as _, rc {
  rc.type is "aws_instance" and
  rc.mode is "managed" and
  (rc.change.actions contains "create" or
   rc.change.actions is ["update"])
}
...
```

Creating rules

Boolean expressions

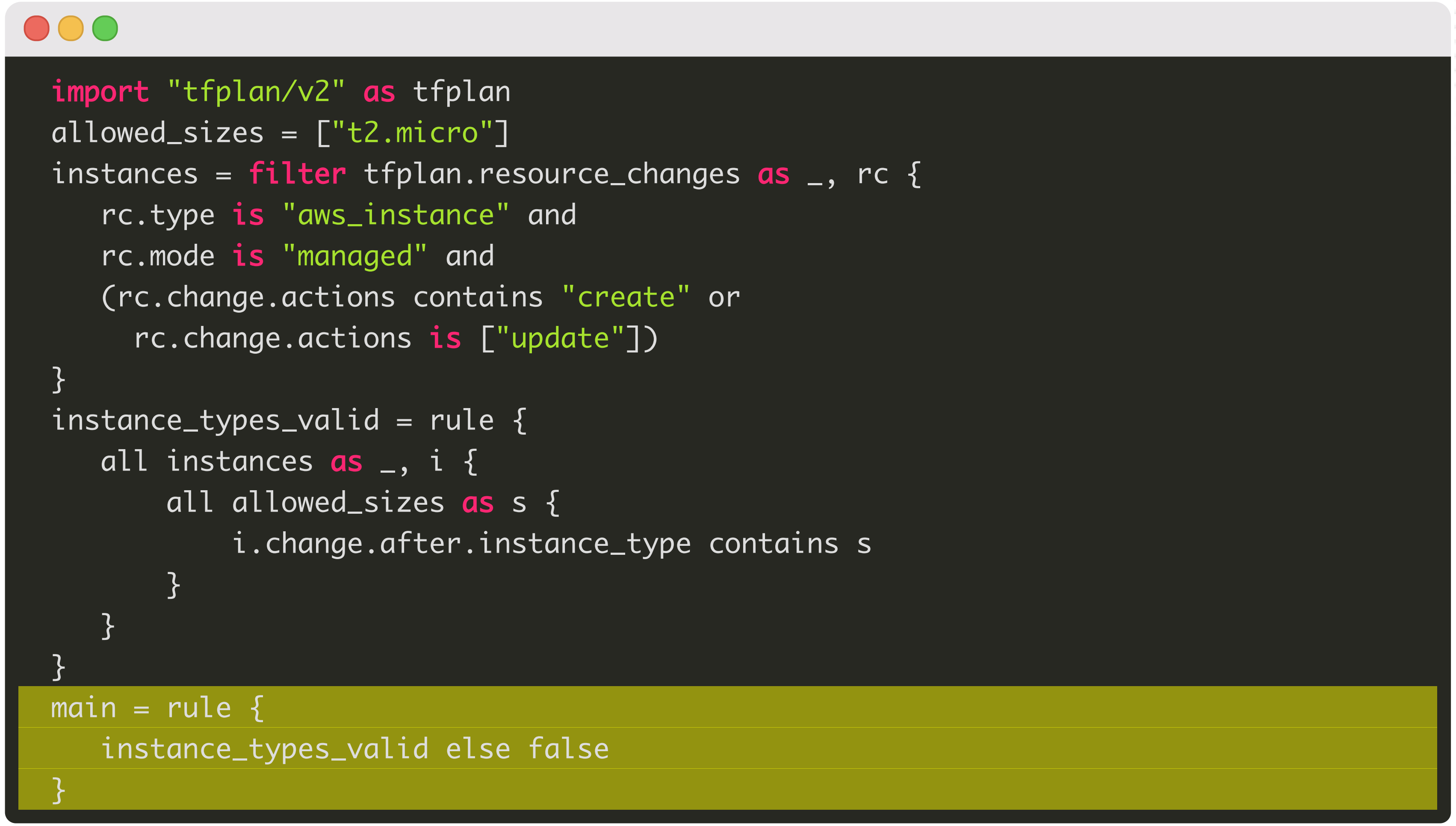
- `true`: pass
- `false`: fail



```
import "tfplan/v2" as tfplan

allowed_sizes = ["t2.micro"]

instances = filter tfplan.resource_changes as _, rc {
   rc.type is "aws_instance" and
   rc.mode is "managed" and
   (rc.change.actions contains "create" or
     rc.change.actions is ["update"])
}

instance_types_valid = rule {
   all instances as _, i {
       all allowed_sizes as s {
           i.change.after.instance_type contains s
       }
   }
}
```

Evaluate for requirements

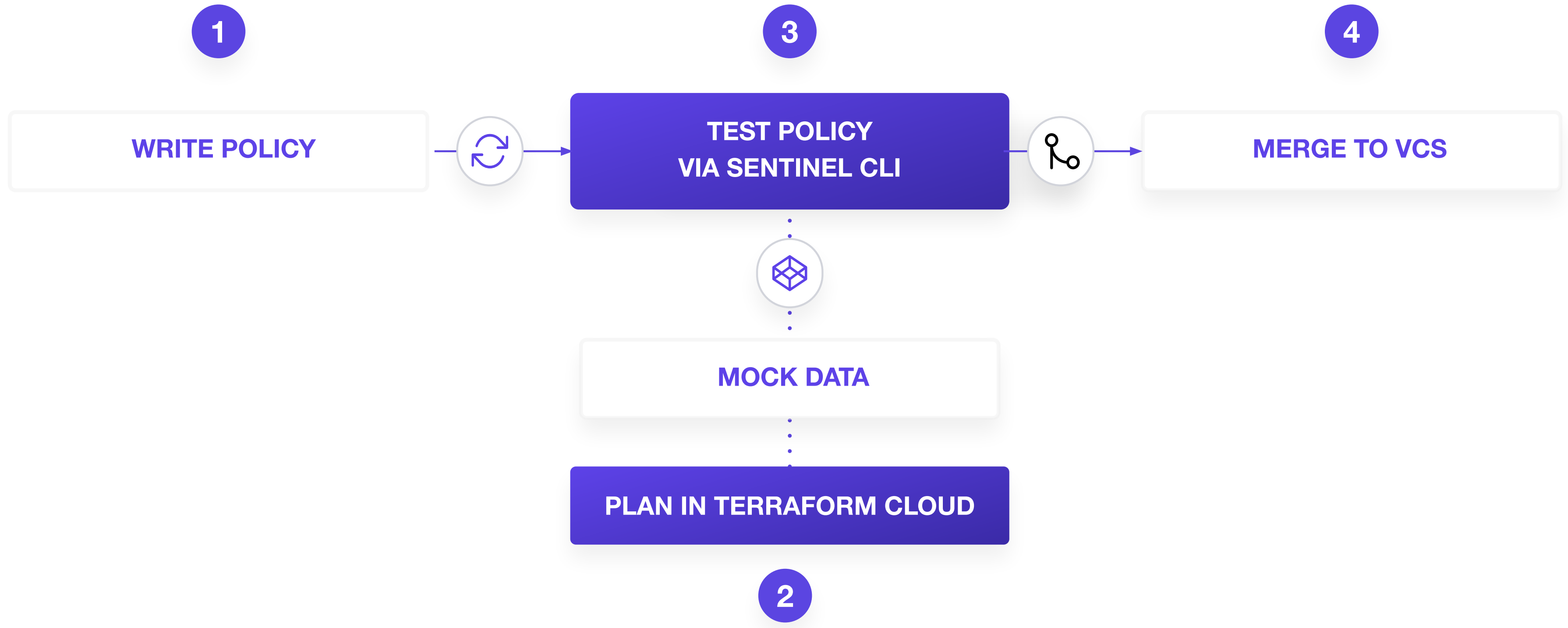
Determine the result of the policy



```
import "tfplan/v2" as tfplan
allowed_sizes = ["t2.micro"]
instances = filter tfplan.resource_changes as _, rc {
   rc.type is "aws_instance" and
   rc.mode is "managed" and
   (rc.change.actions contains "create" or
     rc.change.actions is ["update"])
}
instance_types_valid = rule {
   all instances as _, i {
       all allowed_sizes as s {
           i.change.after.instance_type contains s
       }
   }
}
main = rule {
   instance_types_valid else false
}
```

The Sentinel CLI and Policy Testing

Sentinel Development



Test-driven policy development

Why write tests for Sentinel policies?

- Policies and infrastructure are prone to change for refactoring.
- Writing tests ensures that when changes happen, your policy will still work as intended.
- Writing tests instills confidence in the policy process

Test-driven policy development

How to write tests

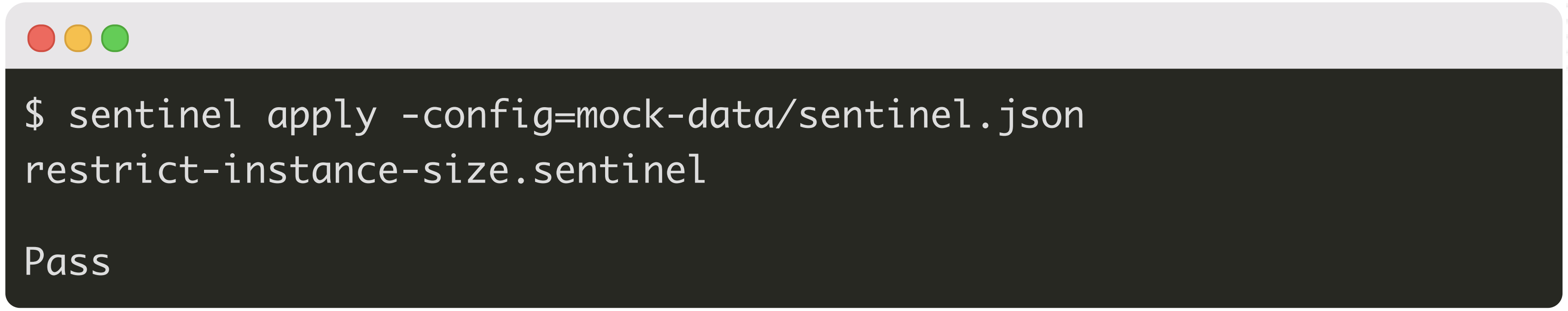
1. Edit data for failing parameters

2. Confirm passing parameters

3. Observe trace data for main rule evaluation in the Sentinel CLI

Sentinel CLI

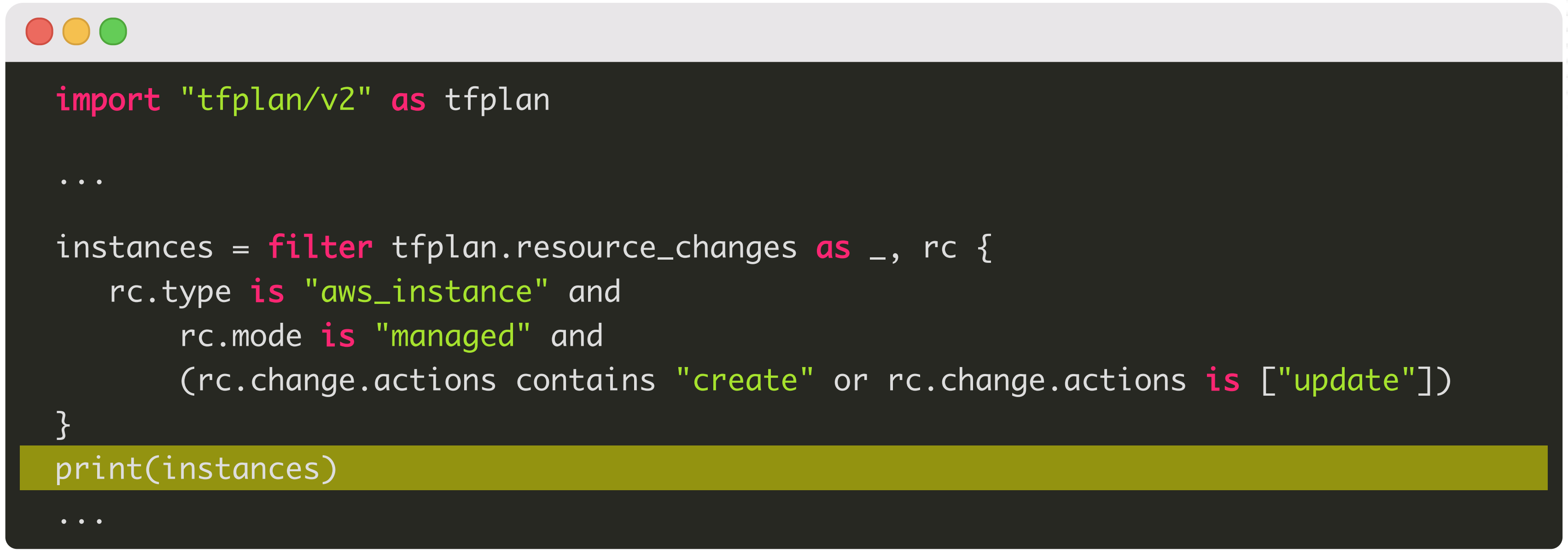
Test policies locally with mock data



```
$ sentinel apply -config=mock-data/sentinel.json
restrict-instance-size.sentinel

Pass
```

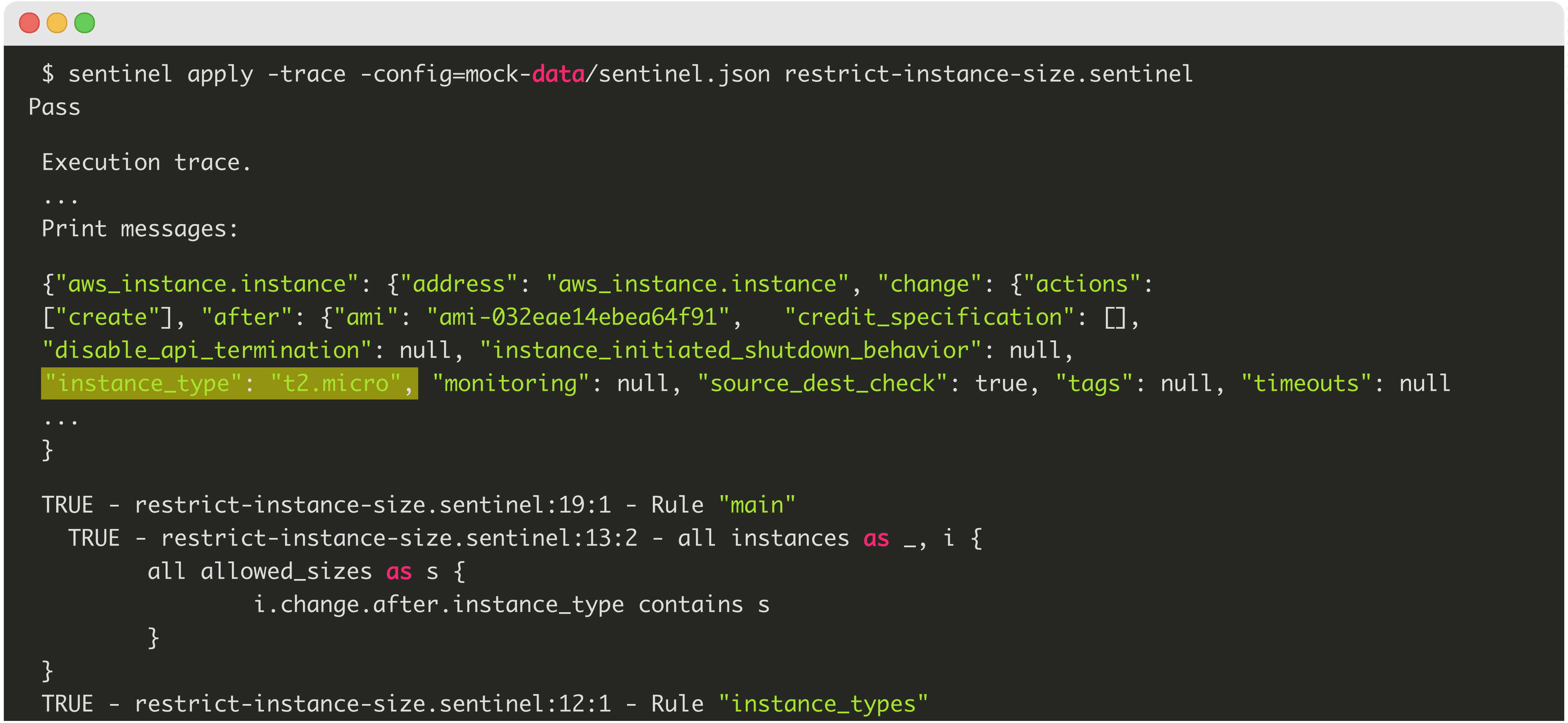
Print

Show filtered data for testing and troubleshooting



```
import "tfplan/v2" as tfplan

...

instances = filter tfplan.resource_changes as _, rc {
    rc.type is "aws_instance" and
        rc.mode is "managed" and
        (rc.change.actions contains "create" or rc.change.actions is ["update"])
}
print(instances)
...
```

Print (cont.)



```
 $ sentinel apply -trace -config=mock-data/sentinel.json restrict-instance-size.sentinel
Pass

 Execution trace.
 ...
 Print messages:

 {"aws_instance.instance": {"address": "aws_instance.instance", "change": {"actions":
 ["create"], "after": {"ami": "ami-032eae14ebea64f91",   "credit_specification": [],
 "disable_api_termination": null, "instance_initiated_shutdown_behavior": null,
 "instance_type": "t2.micro", "monitoring": null, "source_dest_check": true, "tags": null, "timeouts": null
 ...
 }

 TRUE - restrict-instance-size.sentinel:19:1 - Rule "main"
   TRUE - restrict-instance-size.sentinel:13:2 - all instances as _, i {
         all allowed_sizes as s {
                 i.change.after.instance_type contains s
         }
 }
 TRUE - restrict-instance-size.sentinel:12:1 - Rule "instance_types"
```

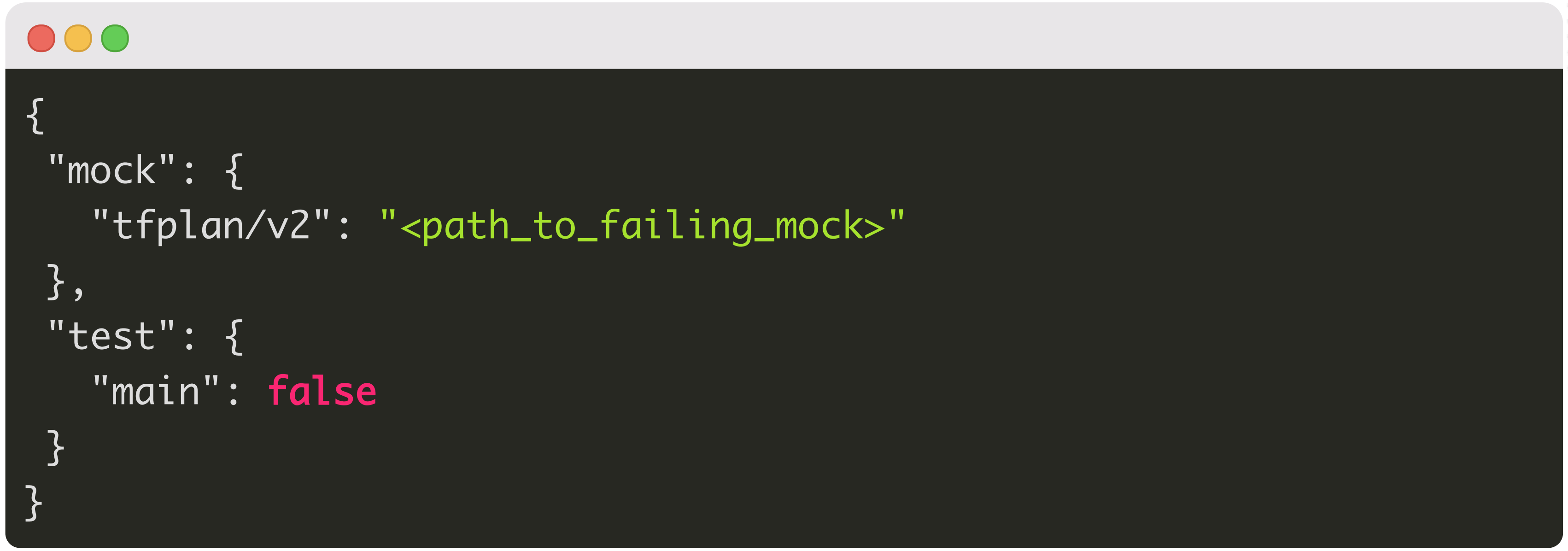
A passing test

A passing test in your testing directory for each policy

```
{
 "mock": {
   "tfplan/v2": "<path_to_passing_mock>"
 },
 "test": {
   "main": true
 }
}
```

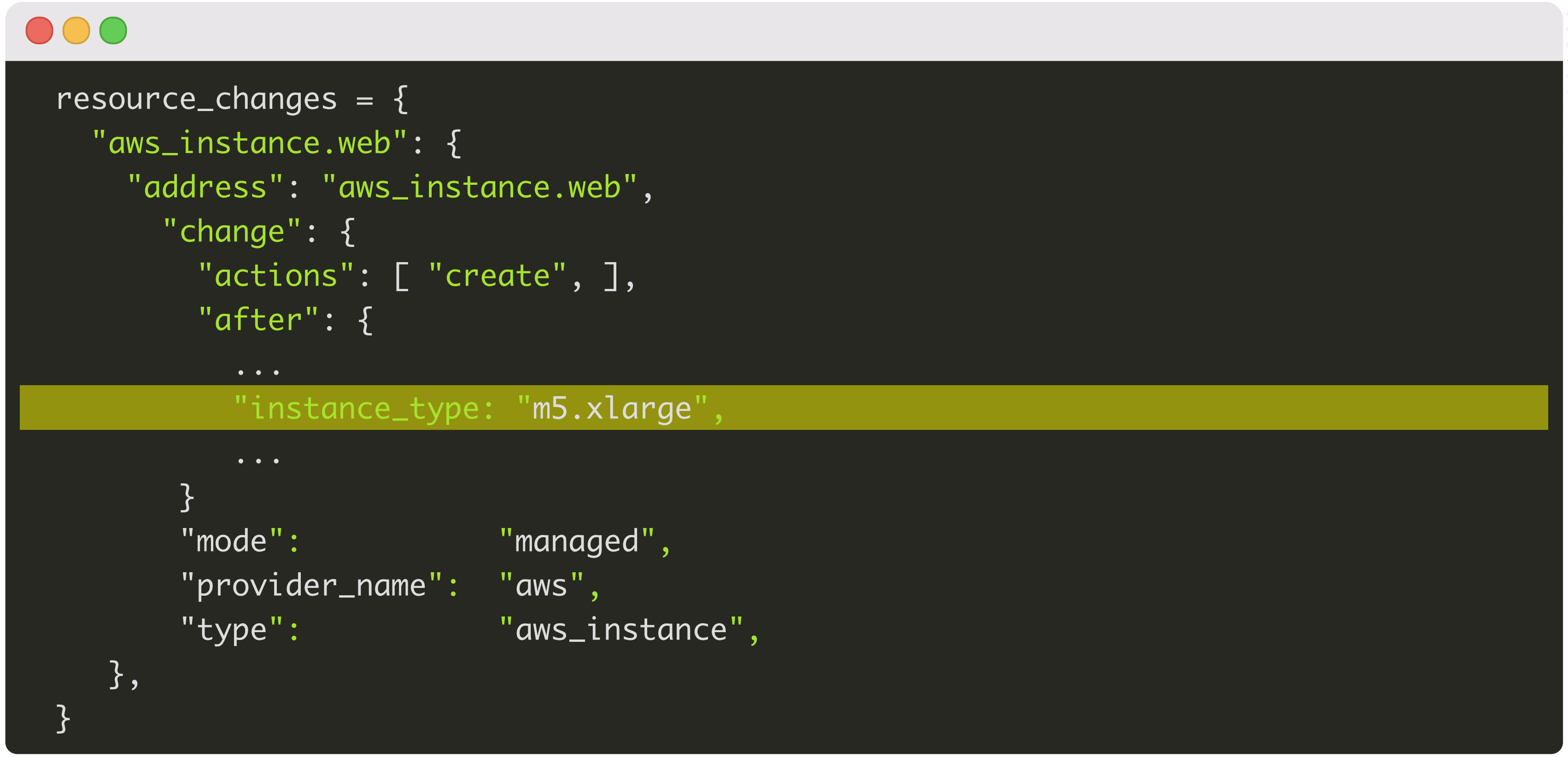
A failing test

Passing the edited mock data to Sentinel and ensuring the main rule will evaluate it as false.



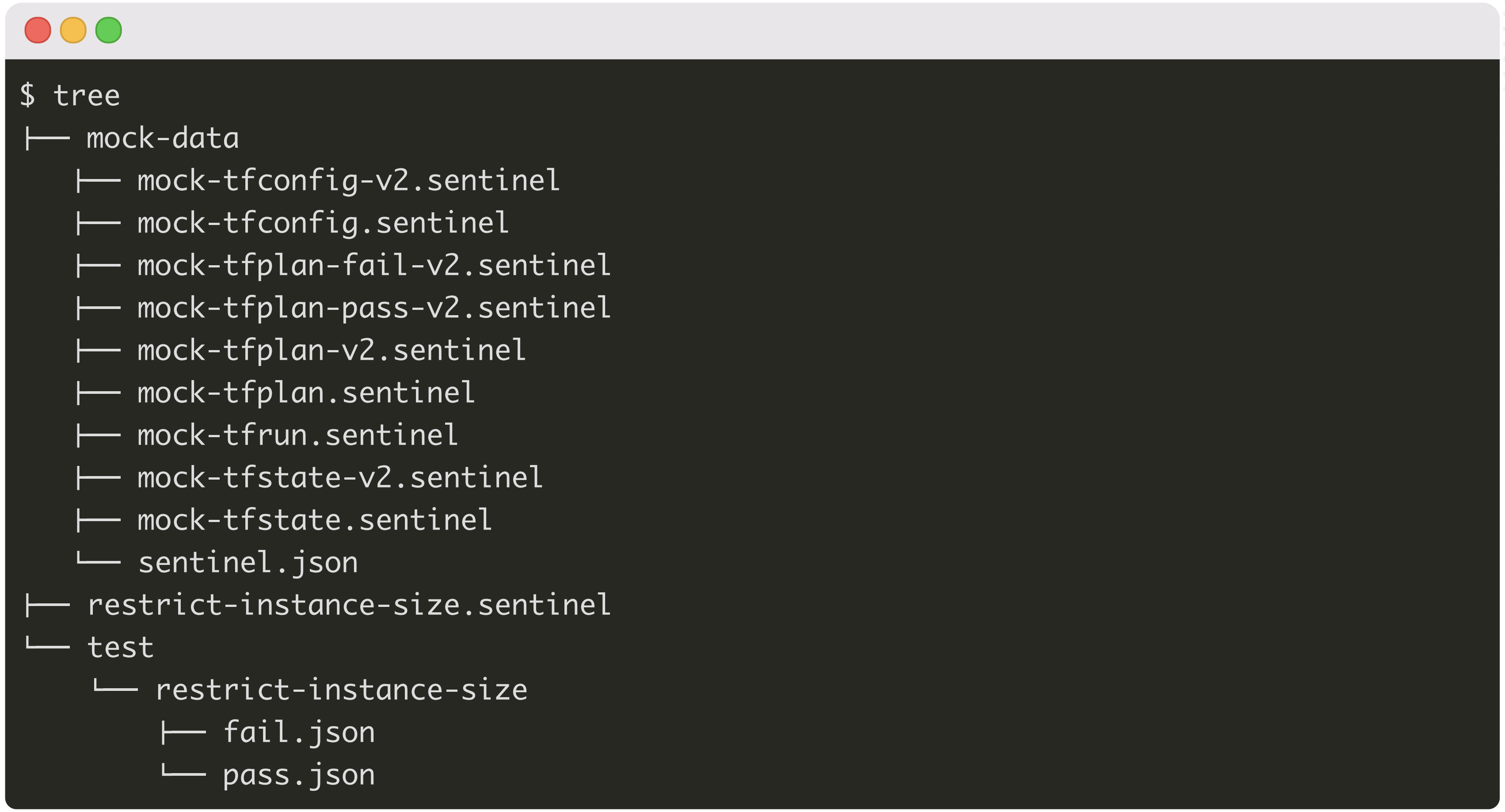
```
{
 "mock": {
   "tfplan/v2": "<path_to_failing_mock>"
 },
 "test": {
   "main": false
 }
}
```

Editing mock data for failures



```
resource_changes = {
  "aws_instance.web": {
    "address": "aws_instance.web",
      "change": {
        "actions": [ "create", ],
        "after": {
          ...
          "instance_type: "m5.xlarge",
          ...
       }
       "mode":            "managed",
       "provider_name":   "aws",
       "type":            "aws_instance",
   },
}
```

Test data structure



```
$ tree
├── mock-data
    ├── mock-tfconfig-v2.sentinel
    ├── mock-tfconfig.sentinel
    ├── mock-tfplan-fail-v2.sentinel
    ├── mock-tfplan-pass-v2.sentinel
    ├── mock-tfplan-v2.sentinel
    ├── mock-tfplan.sentinel
    ├── mock-tfrun.sentinel
    ├── mock-tfstate-v2.sentinel
    ├── mock-tfstate.sentinel
    └── sentinel.json
├── restrict-instance-size.sentinel
└── test
    └── restrict-instance-size
        ├── fail.json
        └── pass.json
```

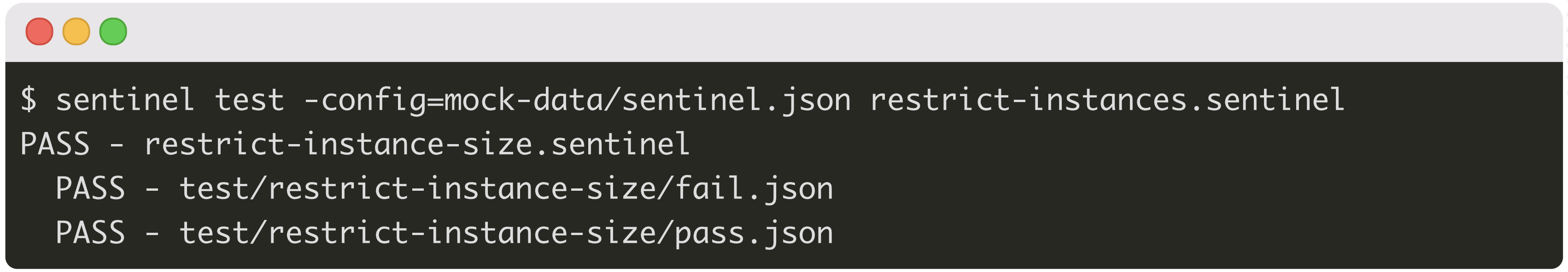
Example test file

`test/restrict-instance-size/pass.json`

```json
{
  "mock": {
    "tfplan/v2": "../../mock-data/mock-tfplan-pass-v2.sentinel"
  },
  "test": {
    "main": true
  }
}
```

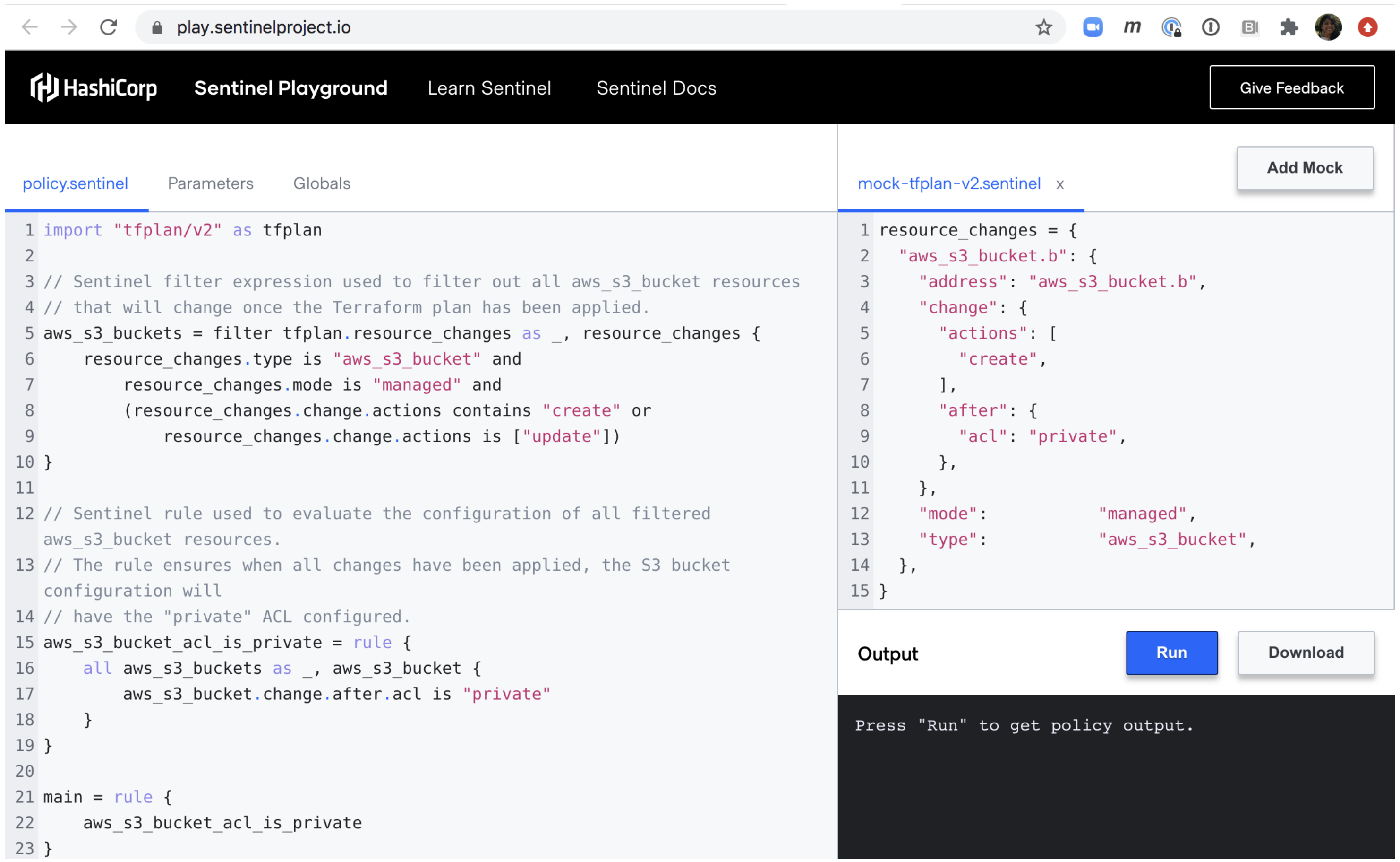
Running tests in the CLI

The test command checks for passing & failing scenarios.



```
$ sentinel test -config=mock-data/sentinel.json restrict-instances.sentinel
PASS - restrict-instance-size.sentinel
  PASS - test/restrict-instance-size/fail.json
  PASS - test/restrict-instance-size/pass.json
```

Sentinel Playground



```
import "tfplan/v2" as tfplan

// Sentinel filter expression used to filter out all aws_s3_bucket resources
// that will change once the Terraform plan has been applied.
aws_s3_buckets = filter tfplan.resource_changes as _, resource_changes {
    resource_changes.type is "aws_s3_bucket" and
        resource_changes.mode is "managed" and
        (resource_changes.change.actions contains "create" or
            resource_changes.change.actions is ["update"])
}

// Sentinel rule used to evaluate the configuration of all filtered aws_s3_bucket resources.
// The rule ensures when all changes have been applied, the S3 bucket configuration will
// have the "private" ACL configured.
aws_s3_bucket_acl_is_private = rule {
    all aws_s3_buckets as _, aws_s3_bucket {
        aws_s3_bucket.change.after.acl is "private"
    }
}

main = rule {
    aws_s3_bucket_acl_is_private
}
```

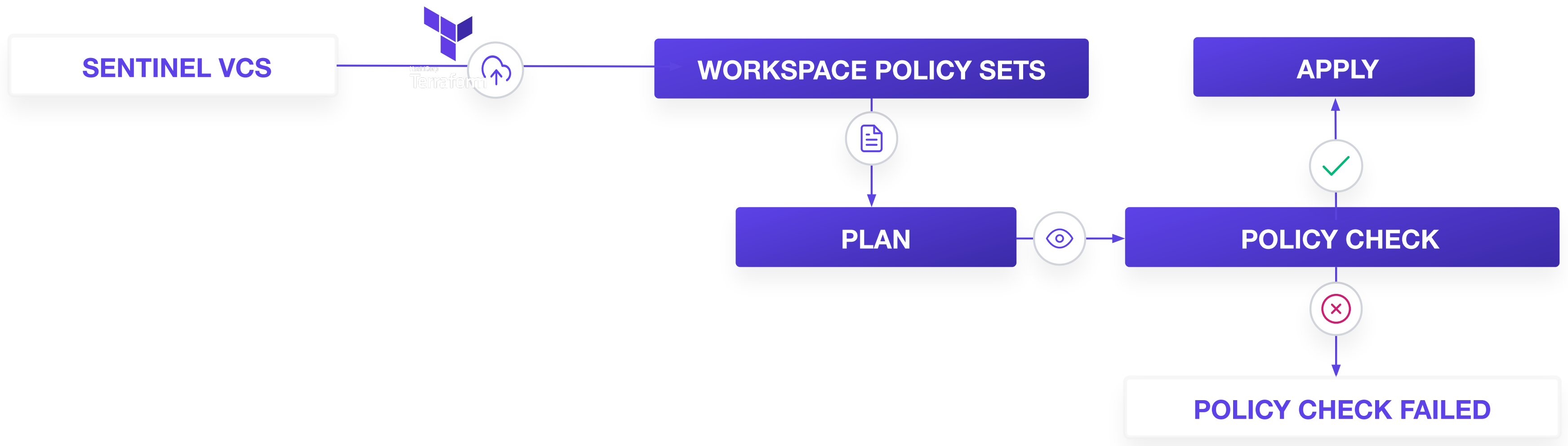
```
resource_changes = {
  "aws_s3_bucket.b": {
    "address": "aws_s3_bucket.b",
    "change": {
      "actions": [
        "create",
      ],
      "after": {
        "acl": "private",
      },
    },
    "mode":           "managed",
    "type":           "aws_s3_bucket",
  },
}
```

```
Press "Run" to get policy output.
```

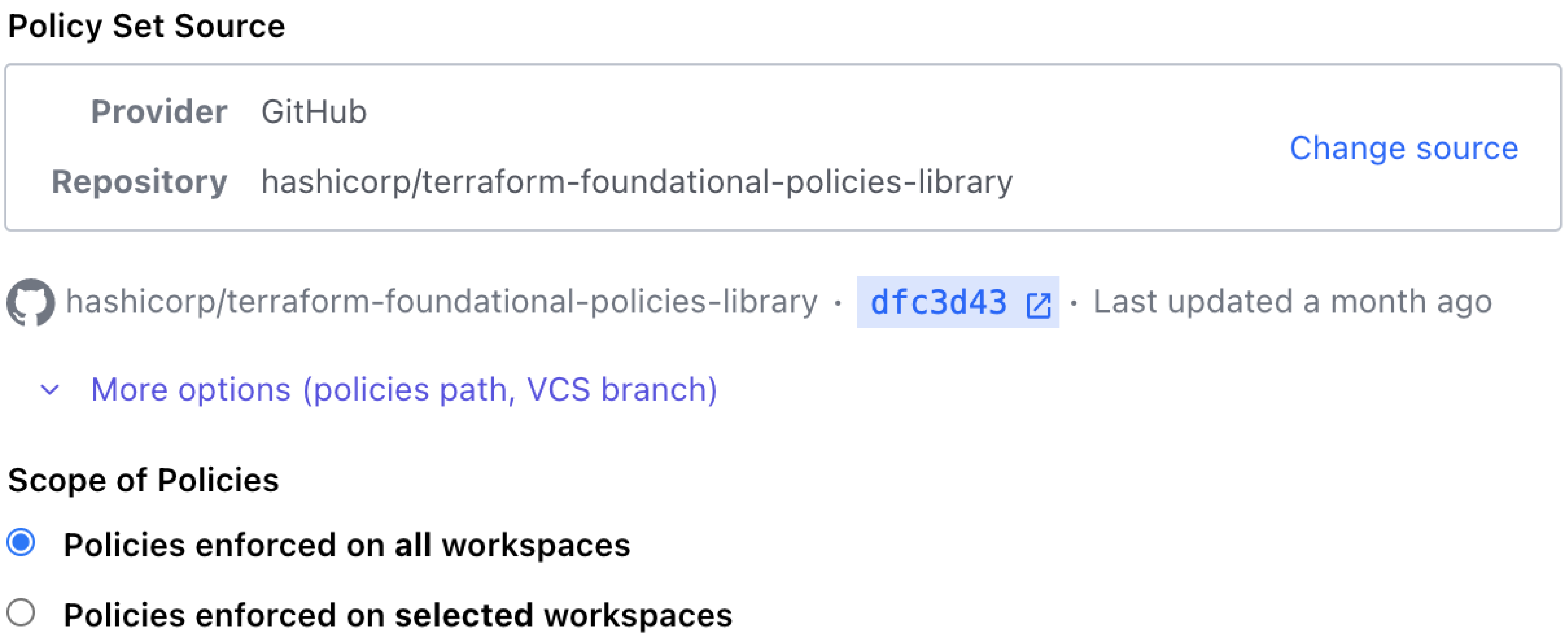
play.sentinelproject.io

Policy Sets and the Terraform Enterprise Workflow

Terraform and Sentinel Implementation



Policy Set Scope



Policy Set Source

Provider	GitHub
Repository	hashicorp/terraform-foundational-policies-library

hashicorp/terraform-foundational-policies-library · `dfc3d43` · Last updated a month ago

Scope of Policies

[✓ **Policies enforced on all workspaces**]

Policies enforced on selected workspaces

Policy enforcement levels

Advisory

- Logged but allowed to pass

Soft Mandatory

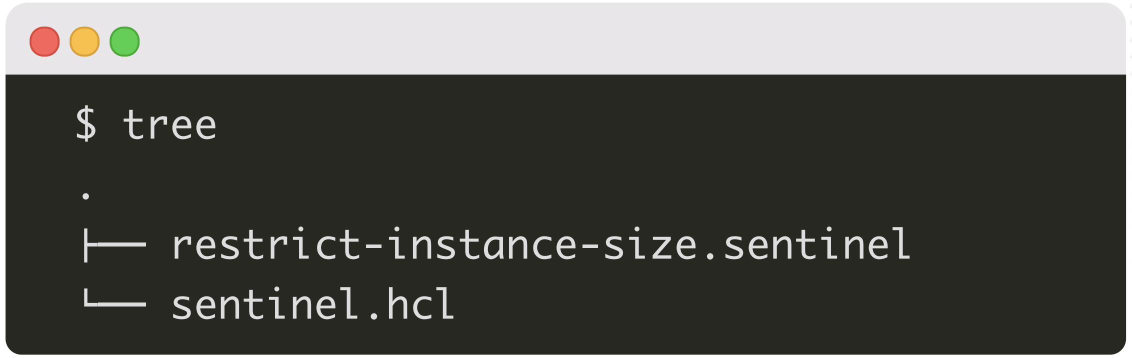
- Teams based permissions for overrides
- Overrides logged for audit if the policy fails

Hard Mandatory

- Default enforcement level
- The policy must pass
- Only way to override is to explicitly remove the policy

Policy repository

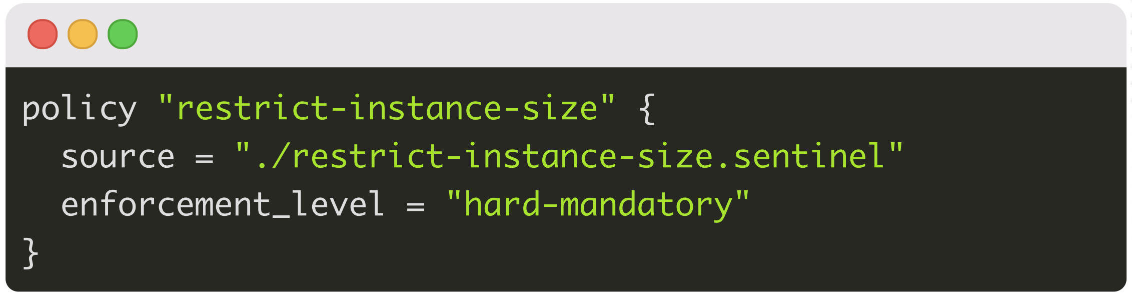
Your policies and the `sentinel.hcl` file must be included in this repo



```
$ tree
.
├── restrict-instance-size.sentinel
└── sentinel.hcl
```

sentinel.hcl

- Each policy Terraform checks in the set
  - Source can be a relative path or HTTP/HTTPS url
- Any modules which need to be made available to policies in the set
- The enforcement level of each policy in the set



```
policy "restrict-instance-size" {
  source = "./restrict-instance-size.sentinel"
  enforcement_level = "hard-mandatory"
}
```

Policy Repo Management

Do

- Create descriptive policy names
- Test before merging
- Employ the principle of least privilege for repository members

Don't

- Push sensitive mock data to VCS
- Merge without testing
- Allow more permissions or users than necessary for your organization

Chapter Summary

- Sentinel is the HashiCorp policy-as-code framework
- Sentinel policies get enforced after the Plan phase
- Policies can have different levels of enforcement
- Policies sets can be applied to an entire organization or specific workspaces
- Imports are necessary to provide sentinel policies with data for enforcement
- Configuration specific mock data can be downloaded from the workspace

Reference links

- Sentinel Getting Started
- Mocking Terraform Sentinel Data
- Sentinel Imports
- Foundational Policies Library
- Sentinel Playground
- Example Policies