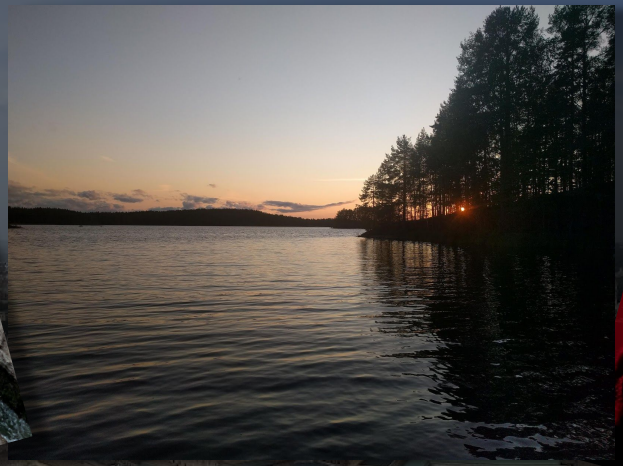
NOVATEC

Hacking Terraform

Engineer your Migration to IaC the Smart Way

11.03.2020, DevOps Gathering 2020

Constantin Weißer | iSibnZe



Infrastructure & Automation Lead at Novatec

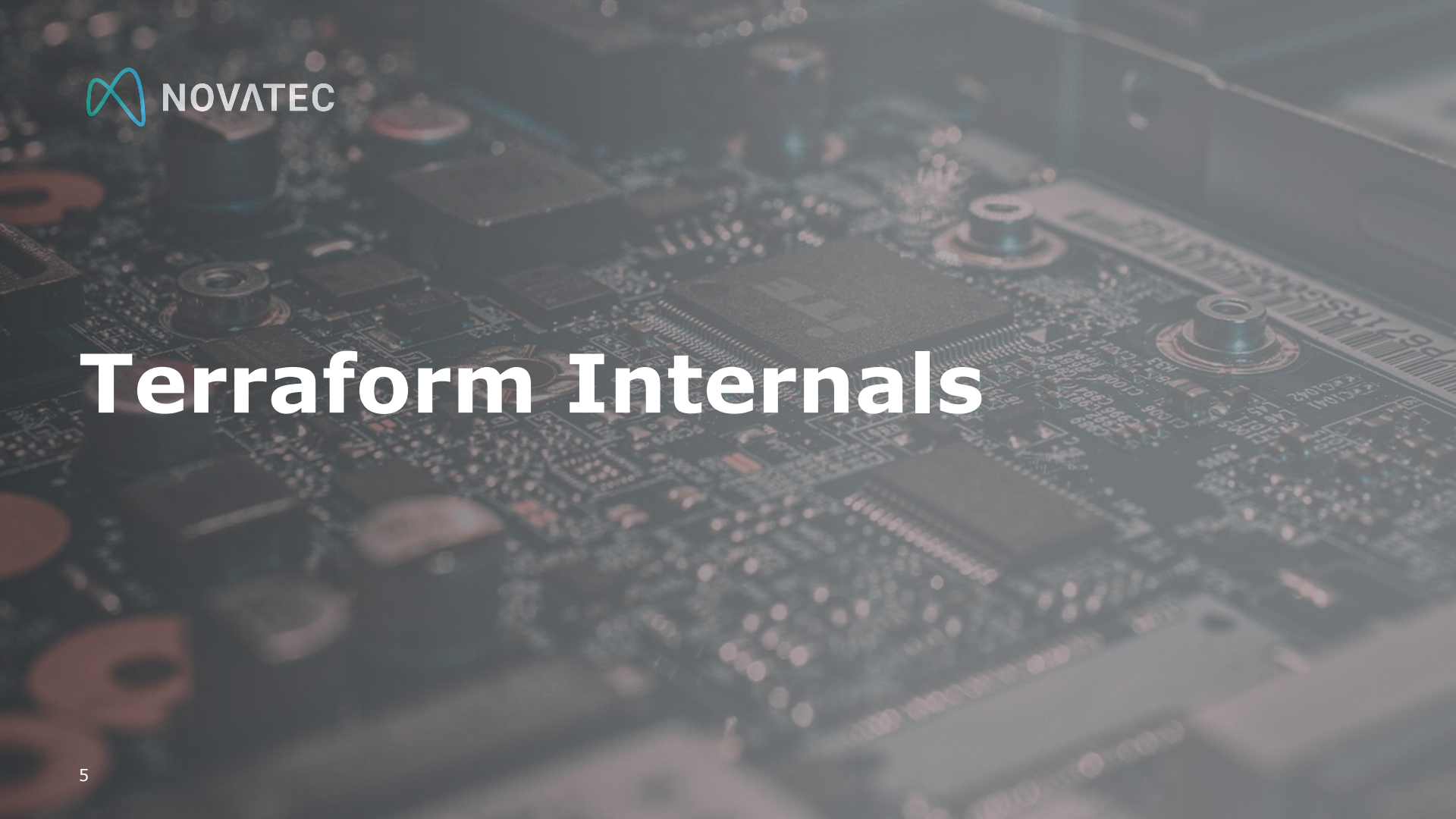
Consultant, Trainer, Traveller, Music Addict


Working on Infrastructure
has changed … a lot!

The Problem

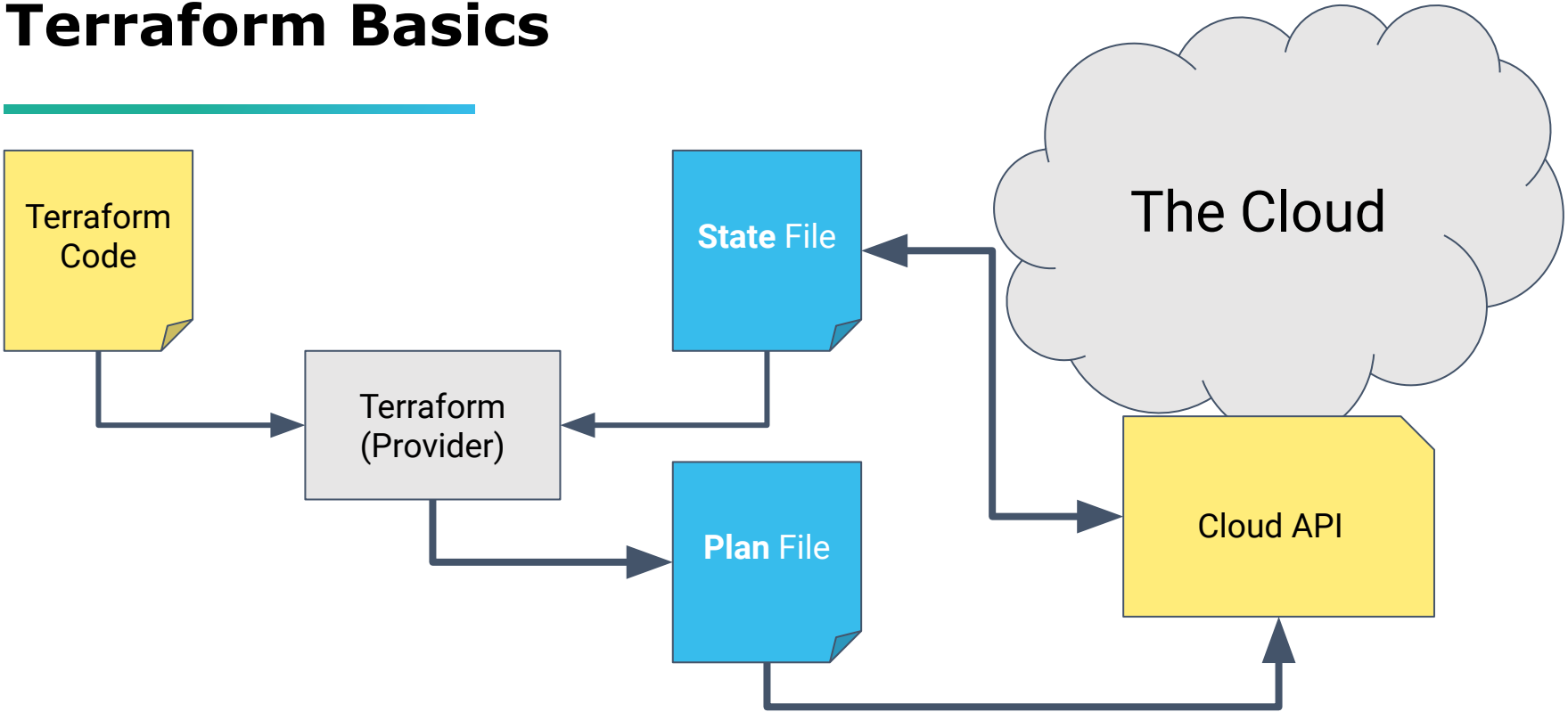
- Productive infrastructure was provisioned without Terraform
- Other IaC tool was used (e.g. proprietary)
- Multiple code bases need to merge
- …

Now, you want to migrate to Terraform!

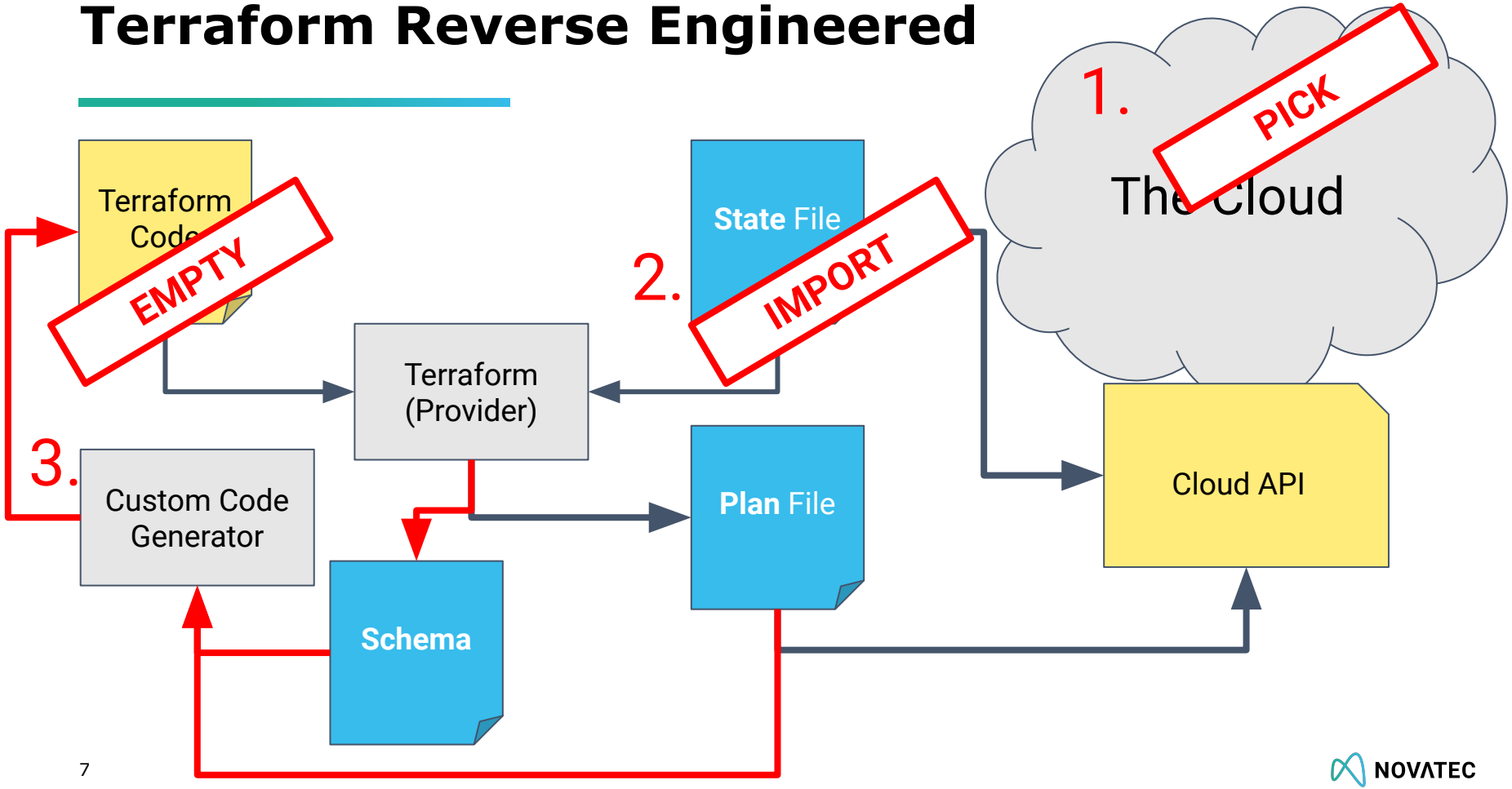


Terraform Internals

Terraform Basics

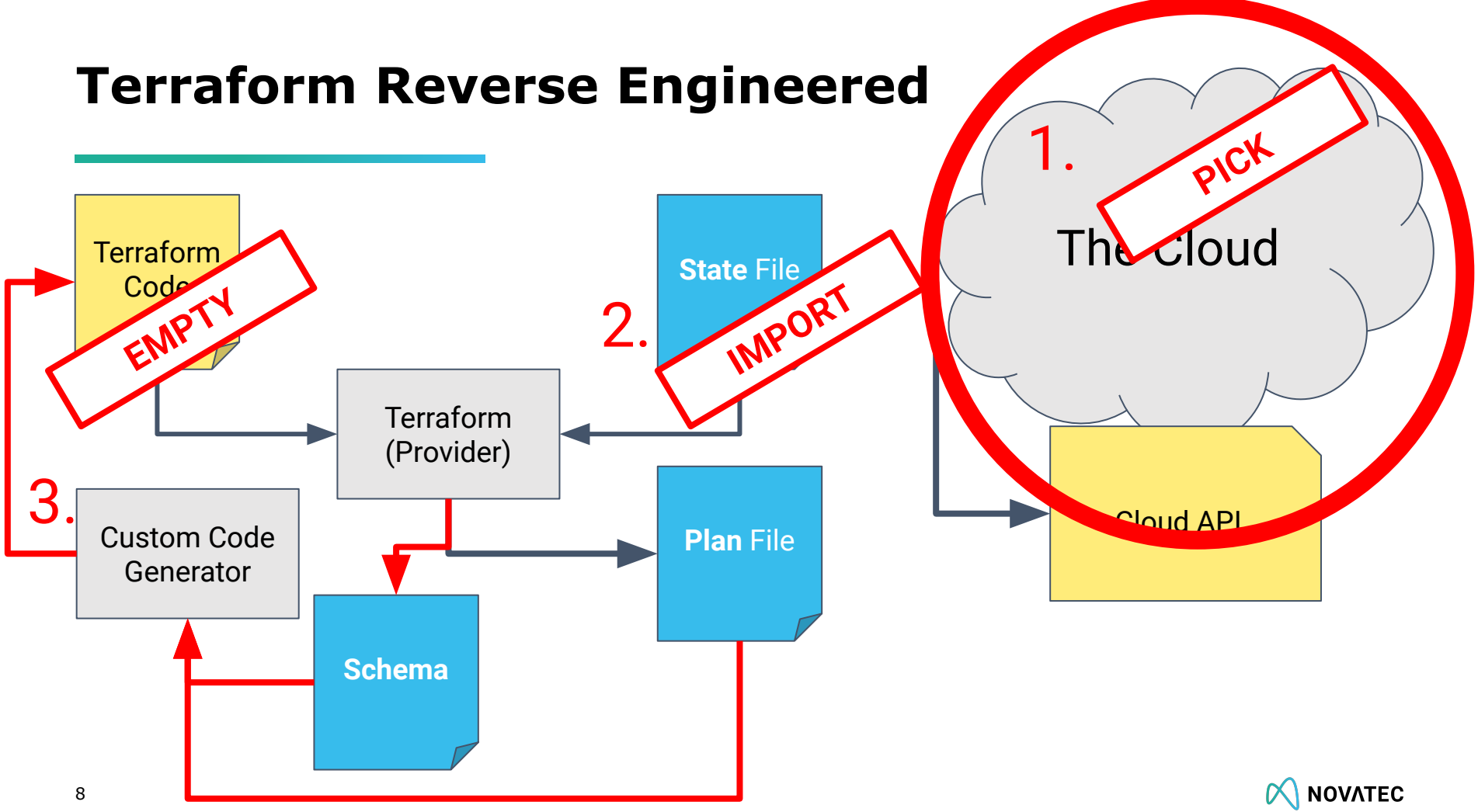


Terraform Reverse Engineered



NOVATEC

Terraform Reverse Engineered



NOVATEC


NOVATEC
Hacker’s Setup

Choose your Weapons!

- **Kotlin + JUnit**

  Run single functions without coding control flow

- Embedded bash snippets + JSON

  Communication Terraform ⇔ Kotlin

- JSONPath + Kotlin built-ins

  Examine and transform data

Merging the Good and the Ugly

```
bash {
"""
    echo 'What is a type system????'
    exit 42
"""
}
```

$format is not a Bash Variable!

```
val format = "+%H:%M:%S"
println(bashCaptureOutput {"date $format"}) // 17:09:25
```

Toolkit

- `bash{""}`

  Run a bash script and redirect stdout/stderr to parent process
- `bashCaptureOutput{""}`

  Run a bash script and capture output as a string
- `bashCaptureJson{""}`

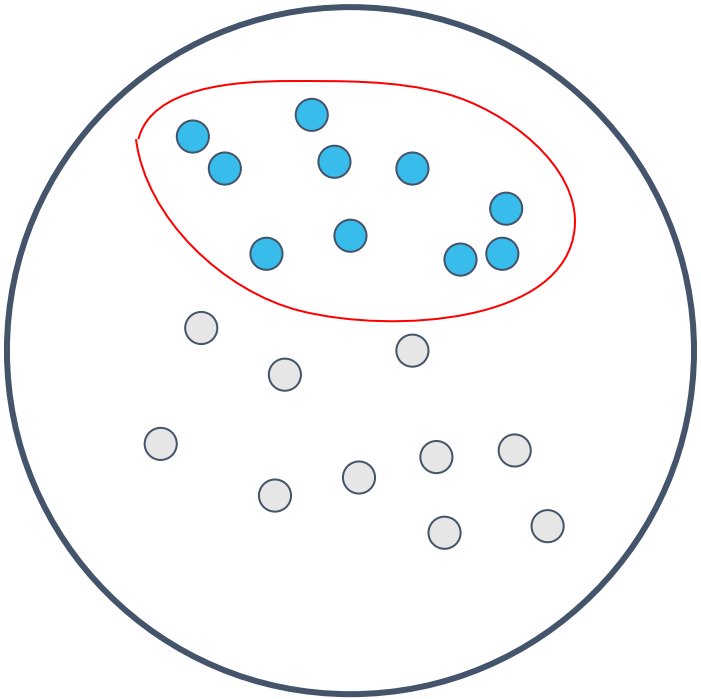
  Run a bash script and parse output as **JSON** into Kotlin types

Picking Resources to Import

Picking Resources for Import

- Compose a “shopping list”
- Multiple ways to proceed
  - Compose the list manually
  - Query a cloud API
  - Use a preexisting IaC tool

In the end we need **resource IDs**



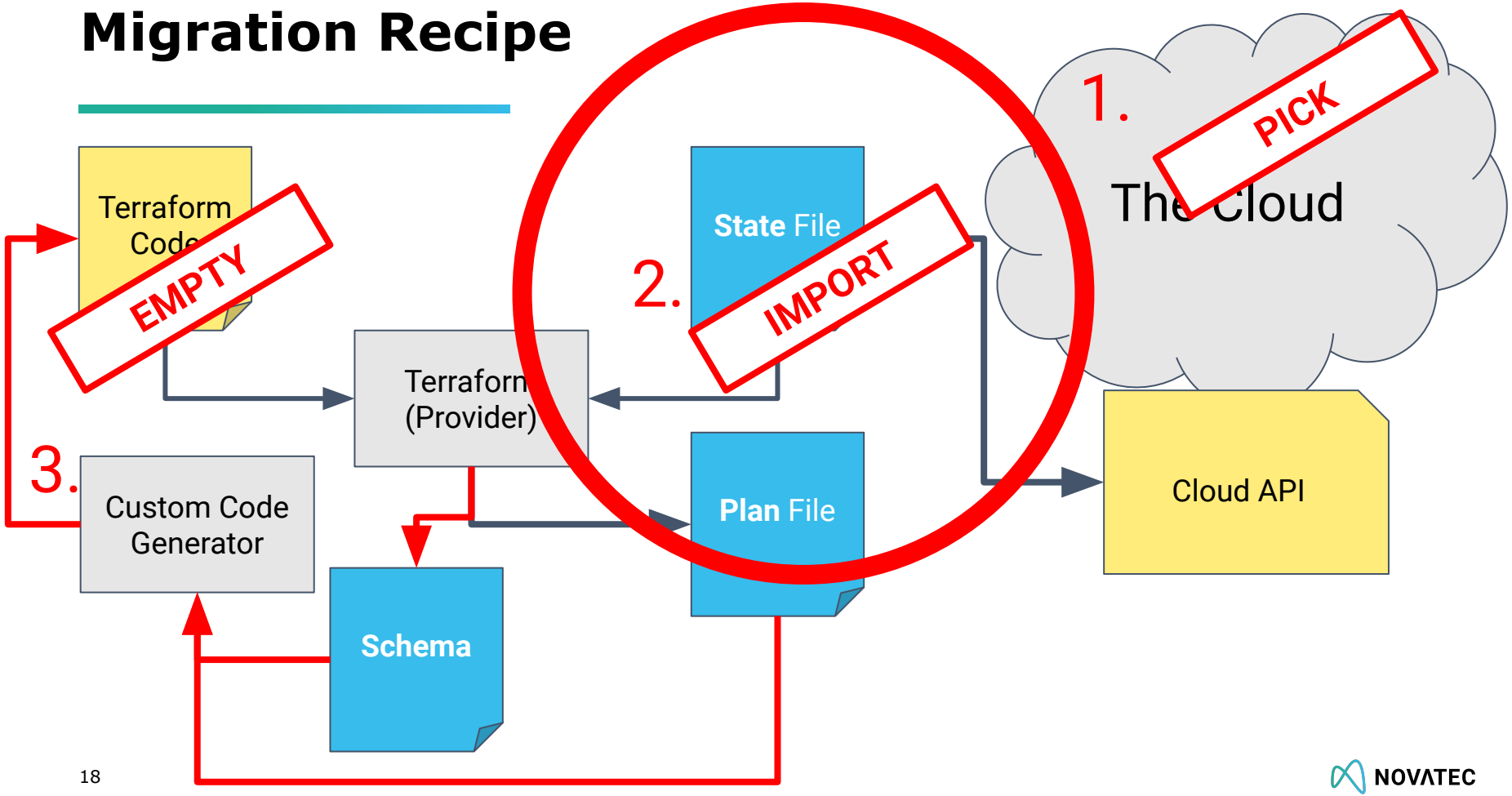
Identifying Resources (2)

```
val importRgs = bashCaptureJson {
"""
    az group list \
        | jq '[ .[] | select (.name | test("^importtest")) ]'
"""
} as List<Map<String, Any?>>
```

Identifying Resources (3)

```
val importResources = importRgs.flatMap {
    bashCaptureJson { """ az resource list -g "${it["name"]}" """ }
        as List<Map<String, Any?>>
}.plus(importRgs)
```

Migration Recipe



NOVATEC

Mapping to Terraform Types

```
terraform import \
  -allow-missing-config \

  azurerm_resource_group.importest-rg \

  "/subscriptions/.../resourceGroups/rg"
```

Setup a new empty Terraform project

```
val workingDir = createTempDir()
cd(workingDir) {
  file(File(it, "main.tf")) { "provider azurerm {}" }

  bash {"""
        terraform init -no-color
        terraform validate -no-color
  """}
}
```

Mapping

```
fun azureIdToNamedResource(id: String, name: String): String =
    when {
        Regex(".*Microsoft.DBforMySQL/servers/.*$").matches(id) ->
          "azurerm_mysql_server.$name"
        // ... more mappings
        else ->
          throw IllegalArgumentException("Unknown resource type $id")
    }
```

Import

```
importResources.forEach {
    bash {
        """
        terraform import -allow-missing-config "${
            azureIdToNamedResource(
                it["id"] as String,
                it["name"] as String
            )
        }" "${it["id"]}"
        """ }
}
```

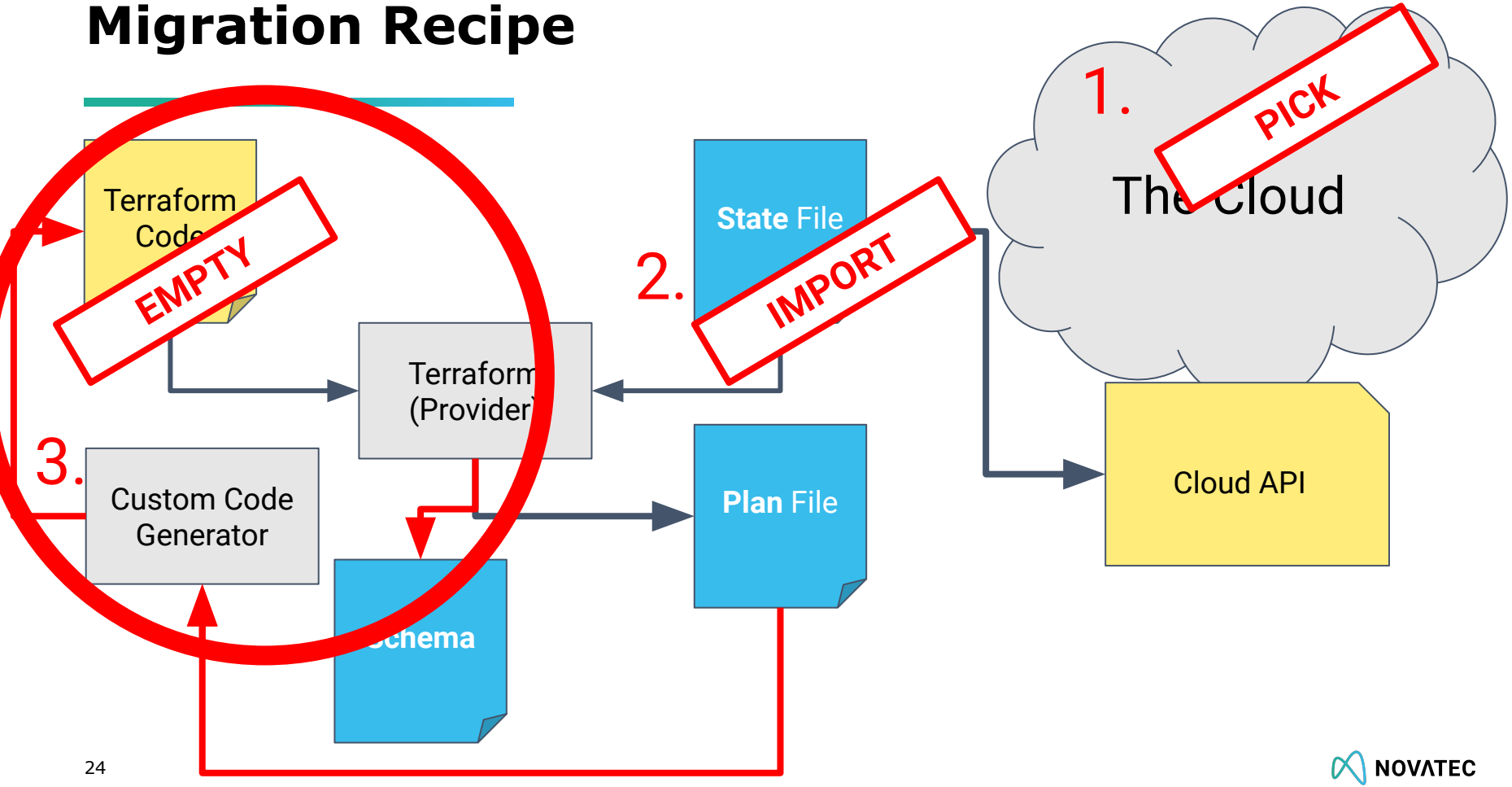
Quick recap: What has happened so far?

1. Produce our **shopping list**
2. Init terraform in an **empty directory**
3. Import with **-allow-missing-config**

From this point on, we are “**cloud-agnostic**”

(Disclaimer: Some providers still need tending)

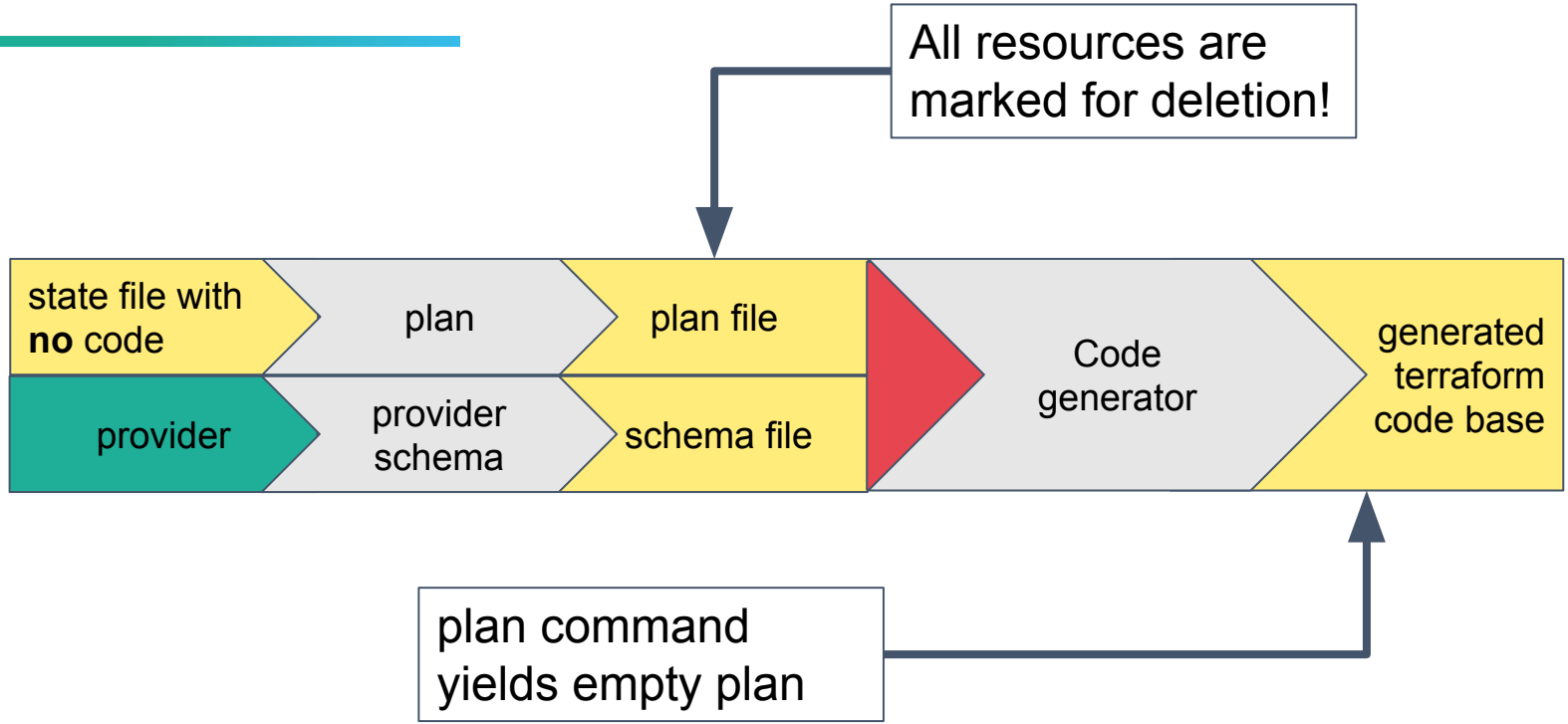
Migration Recipe



NOVATEC

Generating Terraform Code

Universal Code Generator

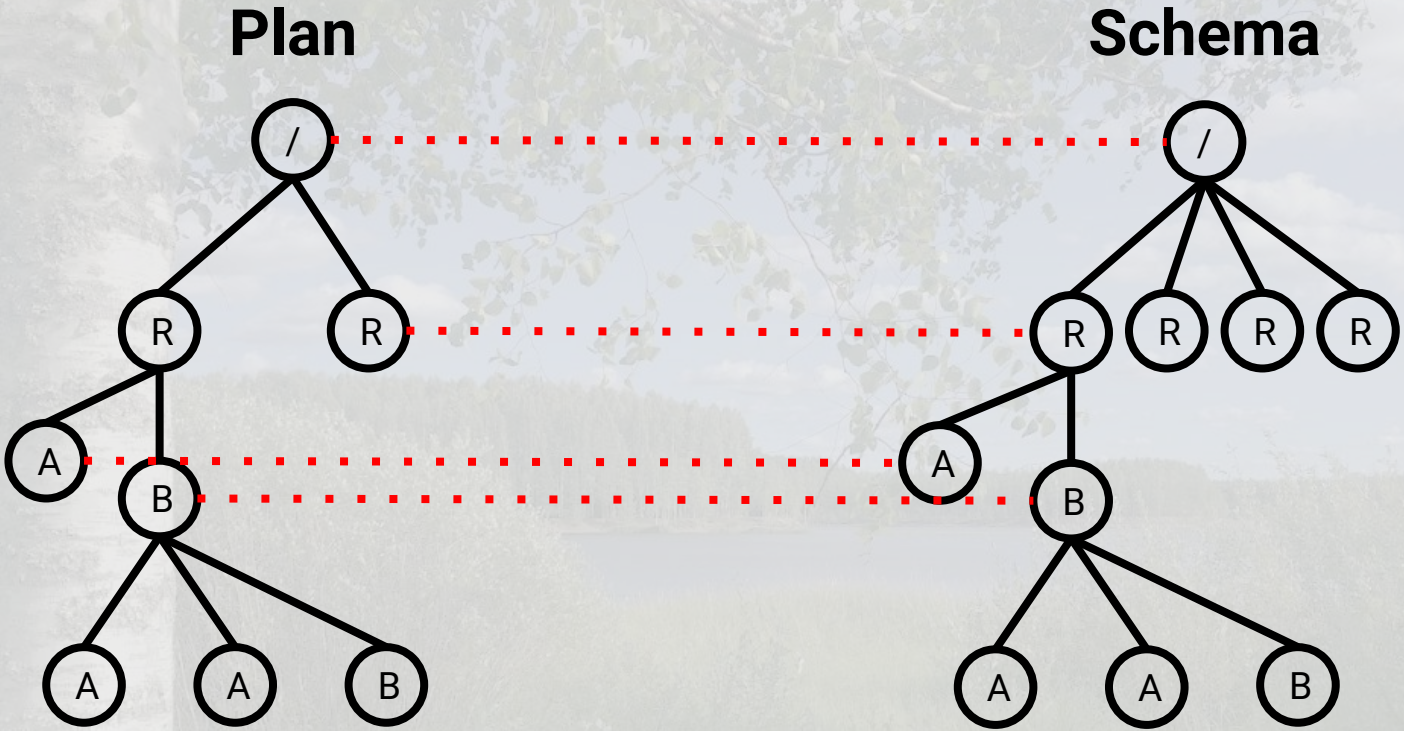


The provider schema has all the metadata we need!

Plan file contains **specified** and **computed** attributes

Read the schema to get the **essentials**!

Post-order tree traversal on the plan file



NOVATEC

Post-order tree traversal on the plan file

Each entry is one of

- Attribute / Single value → Value syntax is **JSON**
- Block → Identical to a Resource (except for the name)

  → **Recurse**

Summary & Observations

- Resource import: easy
- Mapping to terraform types **cloud-specific** (effort!)
- General-purpose code generator in **60 LOC**
- Quality depends on the **provider**
  - Azure schema yields good results
  - AWS schema is puzzling
- Better results with **dedicated code generators**!
  - Specific code for each resource **type**
  - It scales! (Time savings for big migrations)

Learnings

- **Semi-**automate!
- Split import & and code generation!
- Don’t strive for perfection!

Thank you for being here! Questions? Discussions?

Feel free to reach out:

Constantin Weißer | @iSibnZe

constantin.weisser@novatec-gmbh.de

Resources

- **https://github.com/i7c/hackingtf**
- https://www.terraform.io/docs/import/index.html
- https://www.pulumi.com/blog/testing-your-infrastructure-as-code-with-pulumi/