Functional infrastructure (as code)

VIRTUS LAB

About me

Łukasz Biały

Principal Software Engineer
@ VirtusLab

Scala, FP, Distributed Systems

VIRTUSLAB

Agenda

- a quick introduction of infrastructure as code domain
- architecture of Pulumi, the programmable infra-as-code tool
- overview of Besom, the purely functional Scala SDK for Pulumi
- some comparisons and future directions

The long way through the infrastructure as code tooling

2005 - 2023

Genesis

- the need to reduce manual operations in administration of large fleets of servers
- the need to enforce homogeneity and consistency between environments

Modus operandi

- describe the desired state of your infrastructure
- the tool changes the world to suit your desires
- state of transformations is tracked somehow by the tool to detect manual interventions and drift

The timeline

PUPPET

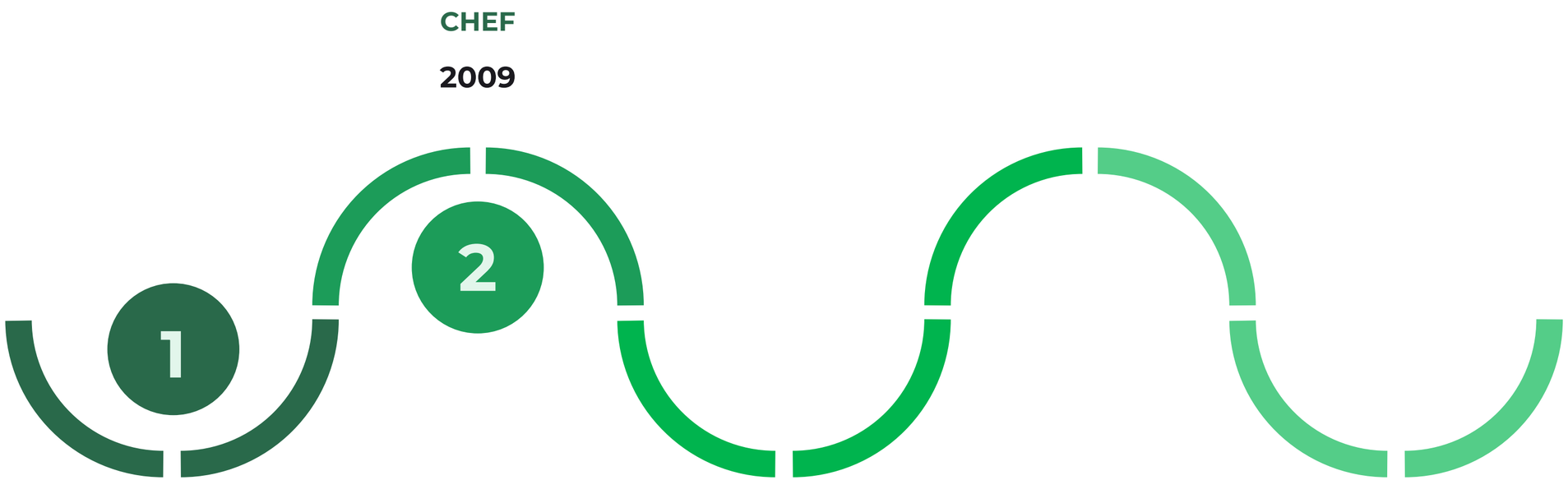
2005


```
class { 'apache':
  default_vhost => false,
  default_mods => false,
  mpm_module => 'prefork',
}

include apache::mod::php

apache::vhost { 'example.com':
  port    => '08',
  docroot => '/var/www/html',
}
```

Puppet

Let’s solve the problem by introducing a weird DSL.

The timeline

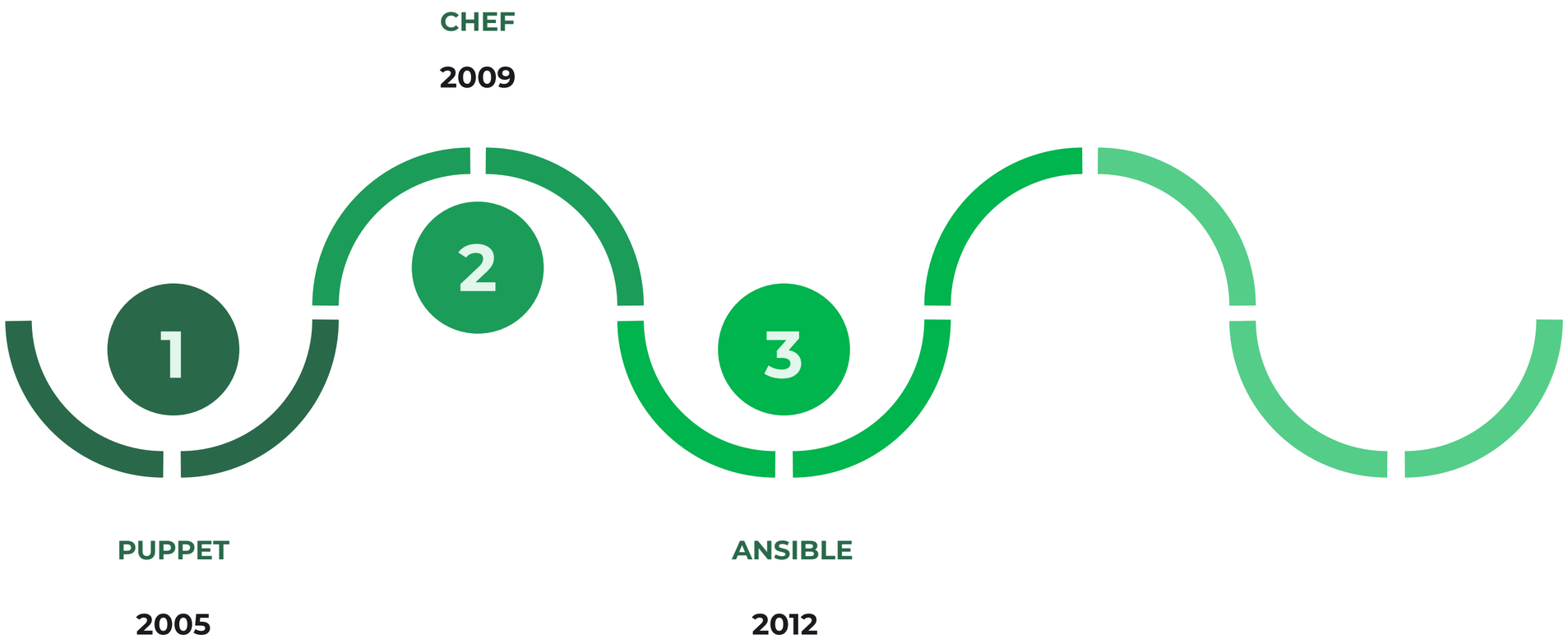


```
node.default['main']['doc_root'] = "/vagrant/web"

execute "apt-get update" do
  command "apt-get update"
end

apt_package "apache2" do
  action :install
end

service "apache2" do
  action [ :enable, :start ]
end
```

Chef

Let’s solve the problem by using a Ruby DSL.

The timeline

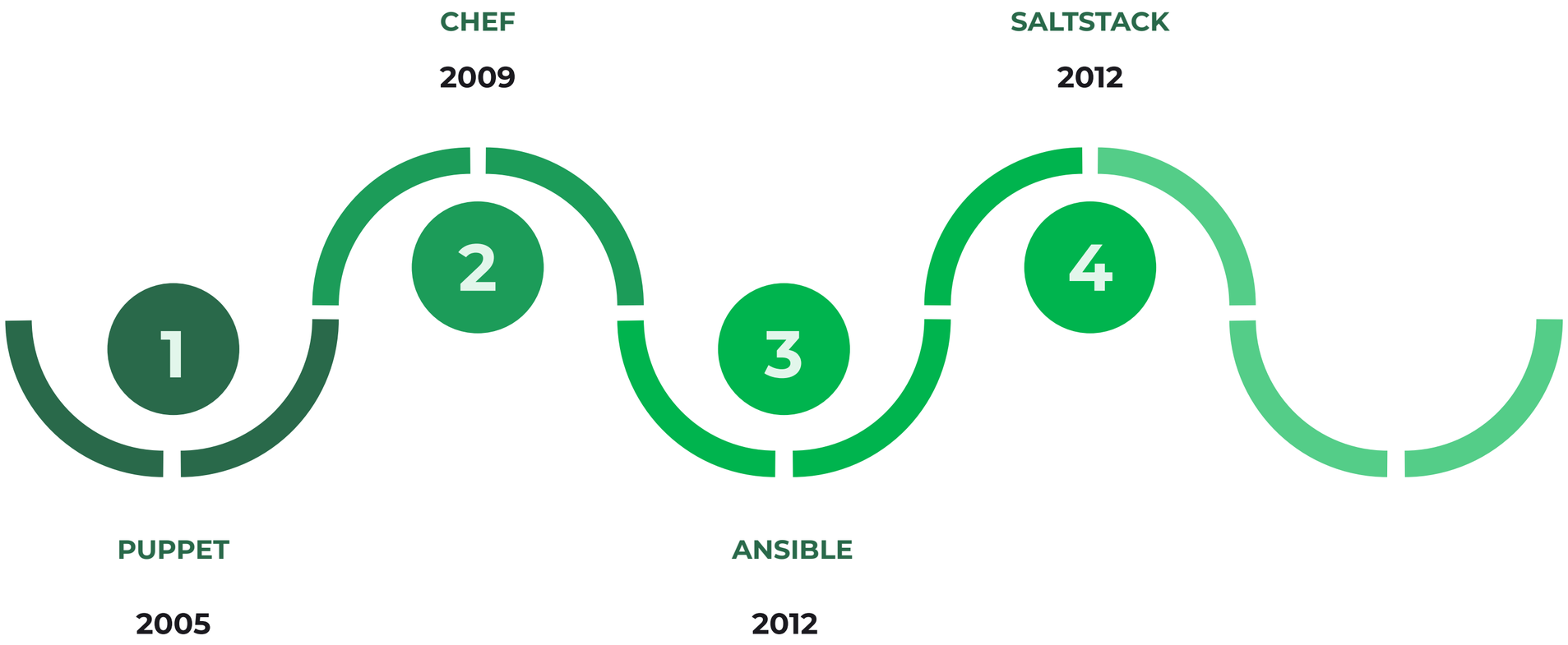


```
---
- hosts: apache
  sudo: yes
  tasks:
    - name: install apache2
      apt: name=apache2 update_cache=yes state=latest
    - name: enabled mod_rewrite
      apache2_module: name=rewrite state=present
      notify:
        - restart apache 2
  handlers:
    - name: restart apache2
      service: name=apache2 state=restarted
```

Ansible

Let’s solve the problem by introducing a YAML DSL (with Python plugins).

The timeline

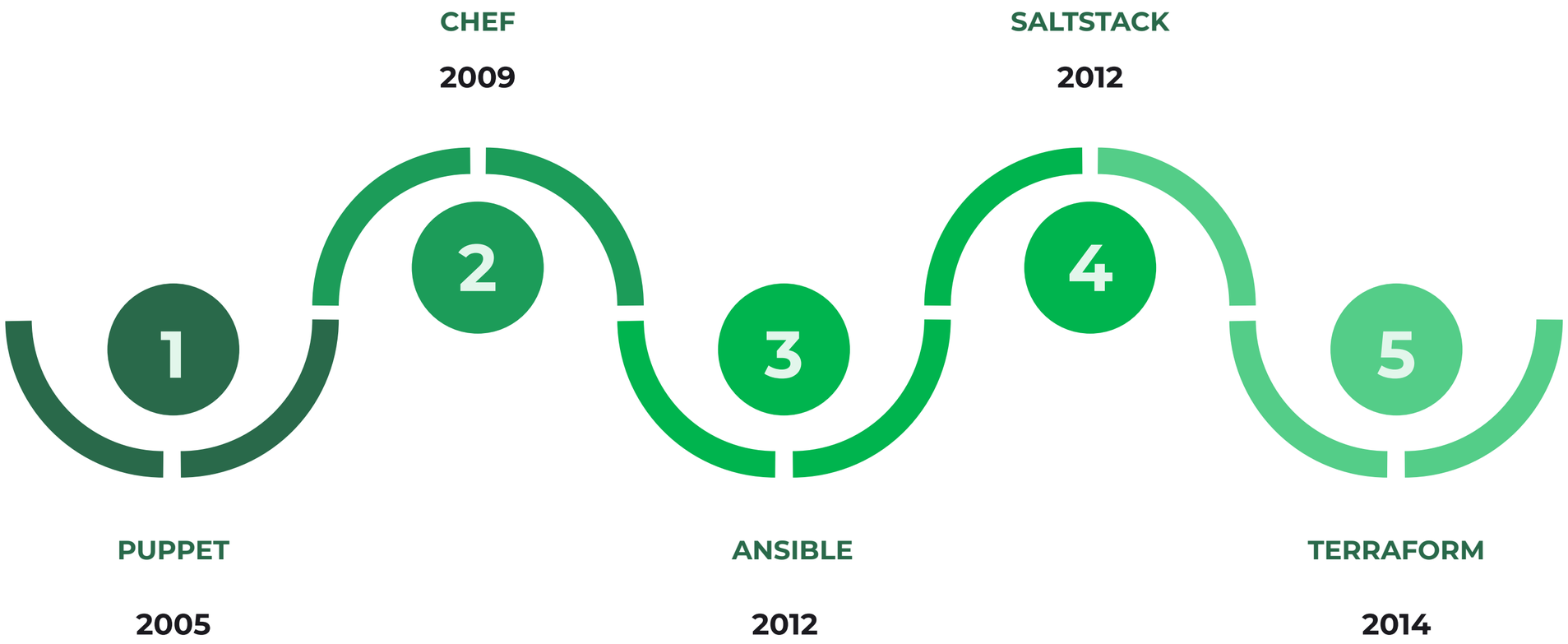


```
apache2:
  pkg.installed

apache2 Service:
  service.running:
    - name: apache2
    - enable: True
    - require:
      - pkg: apache2

/etc/apache2/conf-available/tune_apache.conf:
  file.managed:
    - source: salt://files/tune_apache.conf
    - require:
      - pkg: apache2

Enable tune_apache:
  apache_conf.enabled:
    - name: tune_apache
    - require:
      - pkg: apache2
```


SaltStack

Let’s solve the problem by introducing yet another YAML DSL.

The timeline



```
resource "aws_instance" "web" {
  ami                  = "ami-005e54dee72cc1d00"
  instance_type        = var.instance_type
  key_name             = var.instance_key
  subnet_id            = aws_subnet.public_subnet.id
  security_groups      = [aws_security_group.sg.id]

  user_data = <<-EOF
  #!/bin/bash
  sudo apt update -y
  sudo apt install apache2 -y
  EOF

  tags = {
    Name = "web_instance"
  }
}
```

Terraform

Let’s solve the problem by introducing yet another weird DSL.

```yaml
---
# https://kubernetes.io/docs/concepts/storage/volumes/#configmap
apiVersion: v1
kind: Pod
metadata:
  name: volumes-configmap-pod
spec:
  containers:
    - command:
        - sleep
        - "3600"
      image: busybox
      name: volumes-configmap-pod-container
      volumeMounts:
        - name: volumes-configmap-volume
          mountPath: /etc/config
  volumes:
    - name: volumes-configmap-volume
      configMap:
        name: volumes-configmap-configmap
        items:
          - key: game.properties
            path: configmap-volume-path
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: volumes-configmap-configmap
data:
  game.properties: |
    enemies=aliens
    lives=3
    enemies.cheat=true
```

```yaml
      - operator: In
        scopeName: PriorityClass
        values: ["medium"]
---
apiVersion: v1
kind: ResourceQuota
metadata:
  name: resource-quotas-quotas-pods-low
spec:
  hard:
    cpu: "5"
    memory: 10Gi
    pods: "10"
  scopeSelector:
    matchExpressions:
      - operator: In
        scopeName: PriorityClass
        values: ["low"]
# https://kubernetes.io/docs/concepts/storage/volumes/#local
apiVersion: v1
kind: PersistentVolume
metadata:
  name: volumes-local-persistent-volume
  labels:
    pv: local
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  storageClassName: hostpath
  local:
    path: /mnt/disks/ssd1
  nodeAffinity:
    required:
      nodeSelectorTerms:
        - matchExpressions:
```

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deployments-simple-deployment-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: deployments-simple-deployment-app
  template:
    metadata:
      labels:
        app: deployments-simple-deployment-app
    spec:
      containers:
        - name: busybox
          image: busybox
          command:
            - sleep
            - "3600"
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-rewrite
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target
spec:
  rules:
    - http:
        paths:
          - path: /rewritepath
            pathType: Prefix
            backend:
              service:
                name: testsvc
                port:
```

Perfectly understandable and trivial to maintain

Find the bug

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deployments-simple-deployment-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: deployments-simple-deployment-app
  template:
    metadata:
      labels:
        app: deployments-simple-deployment-app
    spec:
      containers:
        - name: nginx
          image: nginx
          ports:
            - containerPort: 80
```

```yaml
---
apiVersion: v1
kind: Service
metadata:
  name: deployments-simple-deployment-service
spec:
  selector:
    app: deployment-simple-deployment-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
```

Issues

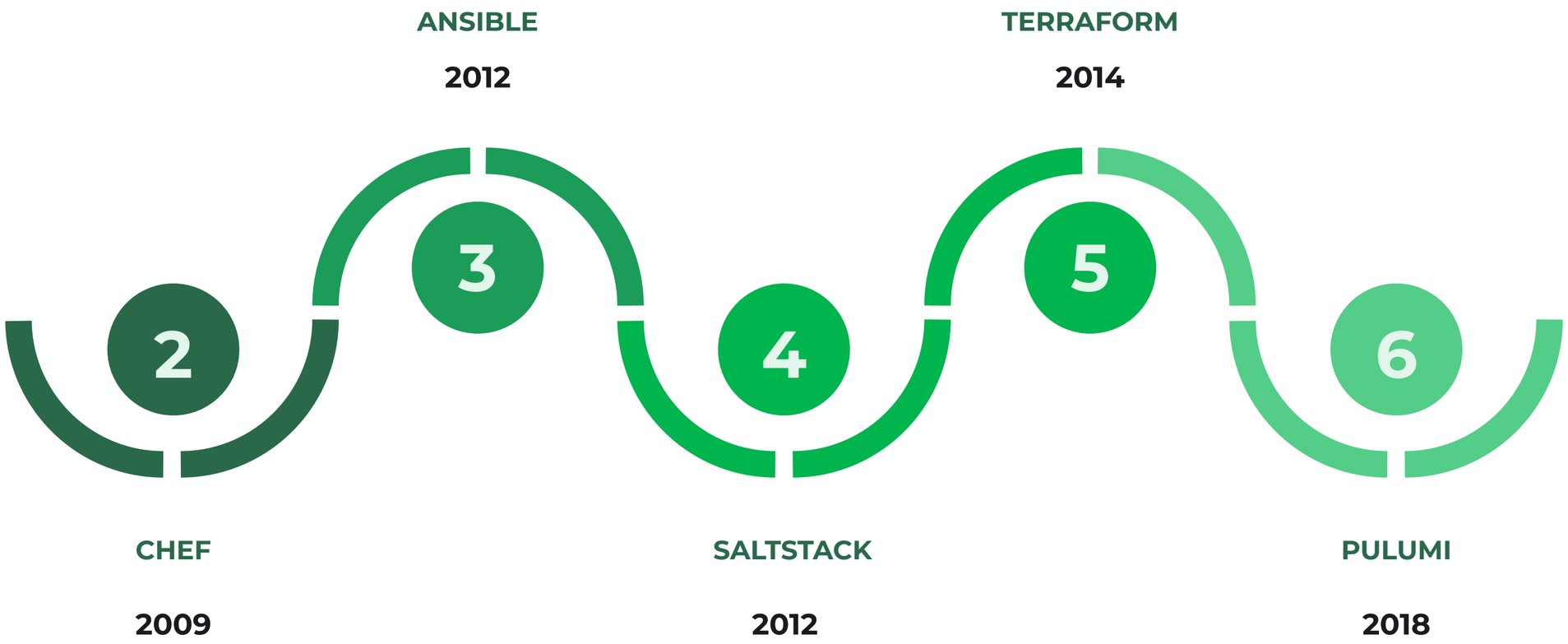
- everything is stringly typed!
- yaml in particular has a huge mistake potential
- it’s declarative (that’s sort of good) but in the same time things can get borked in runtime (so not like SQL) and expressivity is very limited

Problems for large scale projects

- yaml & bash do not look like a good way to build stable platforms
- kubernetes is *lenient* at best when it comes to verifying if manifests make any sense
- could we do better?

Pulumi

The timeline



What's new?

*"Using Pulumi, you author cloud programs using **your favorite language**, spanning low-level infrastructure-as-code to highly productive and modern container- and serverless-powered applications."*

- Joe Duffy, ex-MS-Midori, CEO of Pulumi

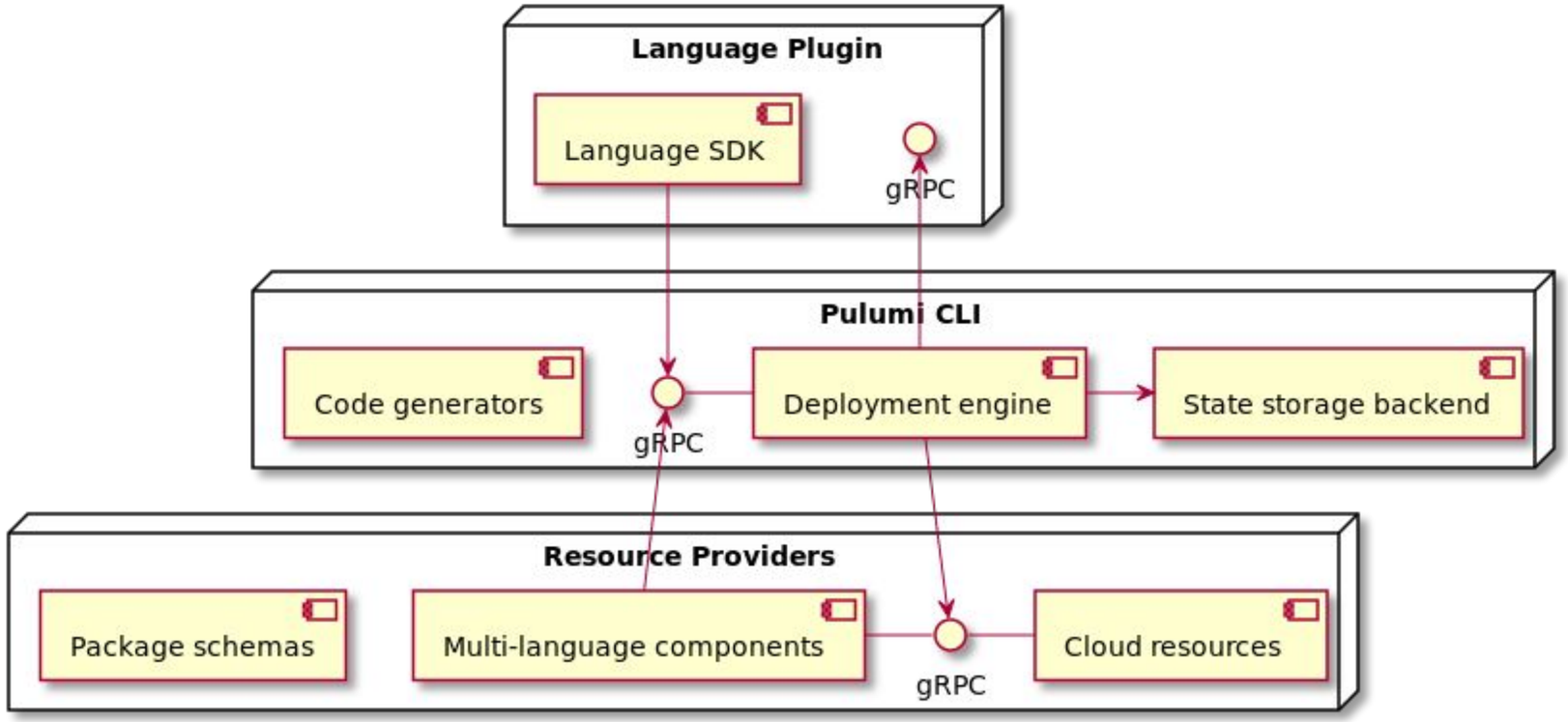
How does it look?

```
const pulumi = require("@pulumi/pulumi");
const aws = require("@pulumi/aws");

let size = "t2.micro";

// Get the id for the latest Amazon Linux AMI
let ami = aws.ec2.getAmi({
    filters: [
        { name: "name", values: ["amzn-ami-hvm-*-x86_64-ebs"] },
    ],
    owners: ["137112412989"], // Amazon
    mostRecent: true,
}, { async: true }).then(result => result.id);

// create a new security group for port 80
let group = new aws.ec2.SecurityGroup("web-secgrp", {
    ingress: [
        { protocol: "tcp", fromPort: 22, toPort: 22,
          cidrBlocks: ["0.0.0.0/0"] },
        { protocol: "tcp", fromPort: 80, toPort: 80,
          cidrBlocks: ["0.0.0.0/0"] },
    ],
});
```

How does it look?

```
// create a simple web server using the startup script for the
instance
let userData =
  `#!/bin/bash
   sudo apt update -y
   sudo apt install apache2 -y`;

let server = new aws.ec2.Instance("web-server-www", {
    tags: { "Name": "web-server-www" },
    instanceType: size,
    // reference the group object above
    vpcSecurityGroupIds: [ group.id ],
    ami: ami,
    userData: userData // install apache web server
});

exports.publicIp = server.publicIp;
exports.publicHostName = server.publicDns;
```

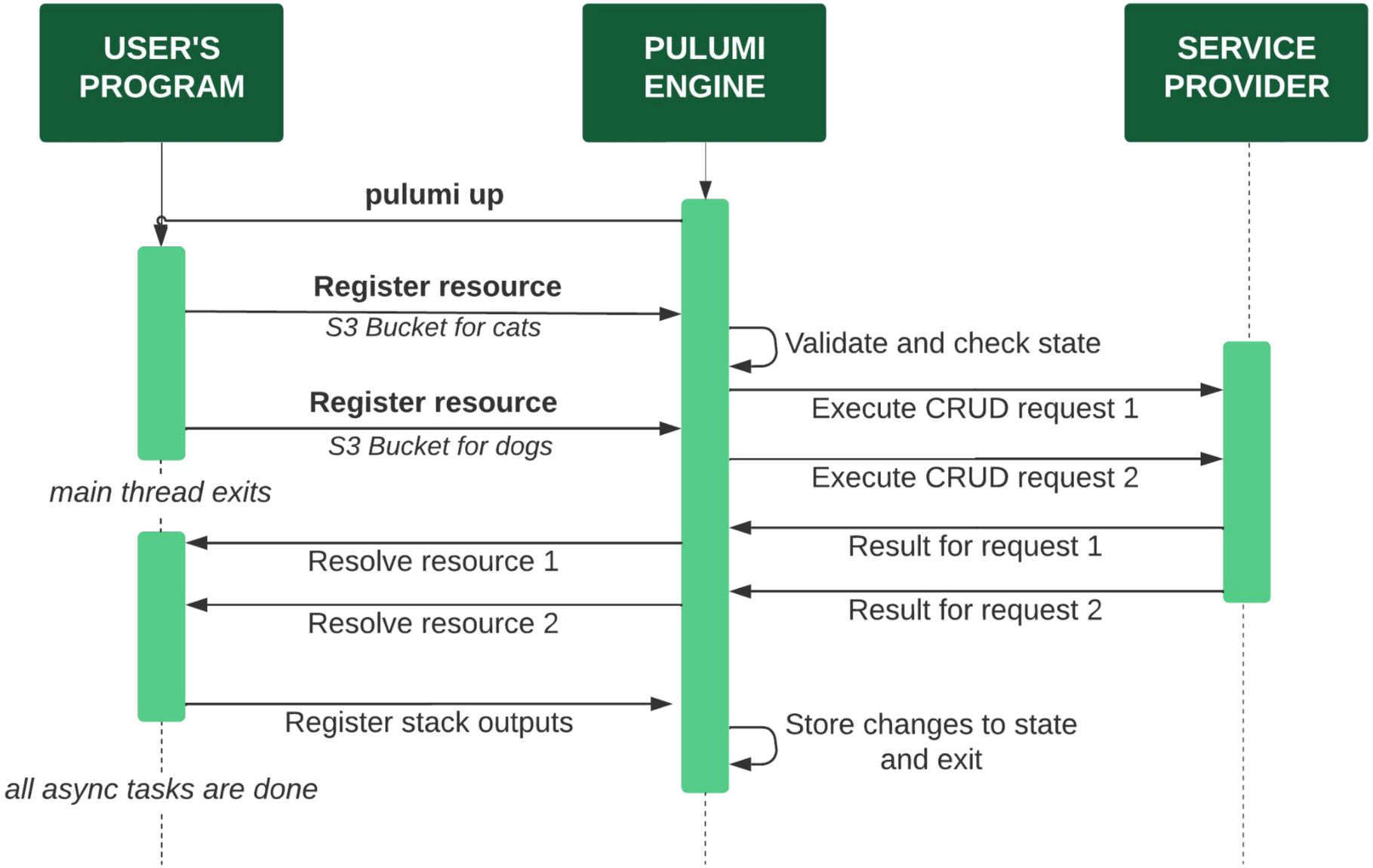
How does it work?



How does it *really* work though?

A small example

```
import * as pulumi from "@pulumi/pulumi";
import * as aws from "@pulumi/aws";

// Create S3 buckets
const catsBucket = new aws.s3.Bucket("cats", {
    acl: "public-read",
});

const dogsBucket = new aws.s3.Bucket("dogs", {
    acl: "public-read",
});

// Export the URLs of the buckets
export const catsUrl: Output<string> =
  catsBucket.websiteEndpoint;

export const dogsUrl: Output<string> =

  dogsBucket.websiteEndpoint;
```


USER'S PROGRAM
PULUMI ENGINE
SERVICE PROVIDER
pulumi up
Register resource
S3 Bucket for cats
Validate and check state
Register resource
S3 Bucket for dogs
Execute CRUD request 1
Execute CRUD request 2
main thread exits
Result for request 1
Result for request 2
Resolve resource 1
Resolve resource 2
Register stack outputs
Store changes to state and exit
all async tasks are done

Going deeper

- Output[A] type is the main *pseudo*monad in pulumi implementations

- Output[A] is generally implemented in terms of an asynchronous datatype (think: Promise/A+, System.Threading.Tasks.Task, java.util.concurrent.CompletableFuture) wrapping an OutputData structure

Going deeper

- being based on single-write async datatypes means that Output is effectively memoized
- it’s also eager
- while DSL and dry-runs limit what can happen in terms of infrastructure, side effects are generally unconstrained

Cool stuff

- Pulumi gives the user an ability to define custom components built out of provided resources
- Pulumi can be embedded in other apps via AutomationAPI to control infrastructure programmatically
- Pulumi can enforce policies for all managed resources

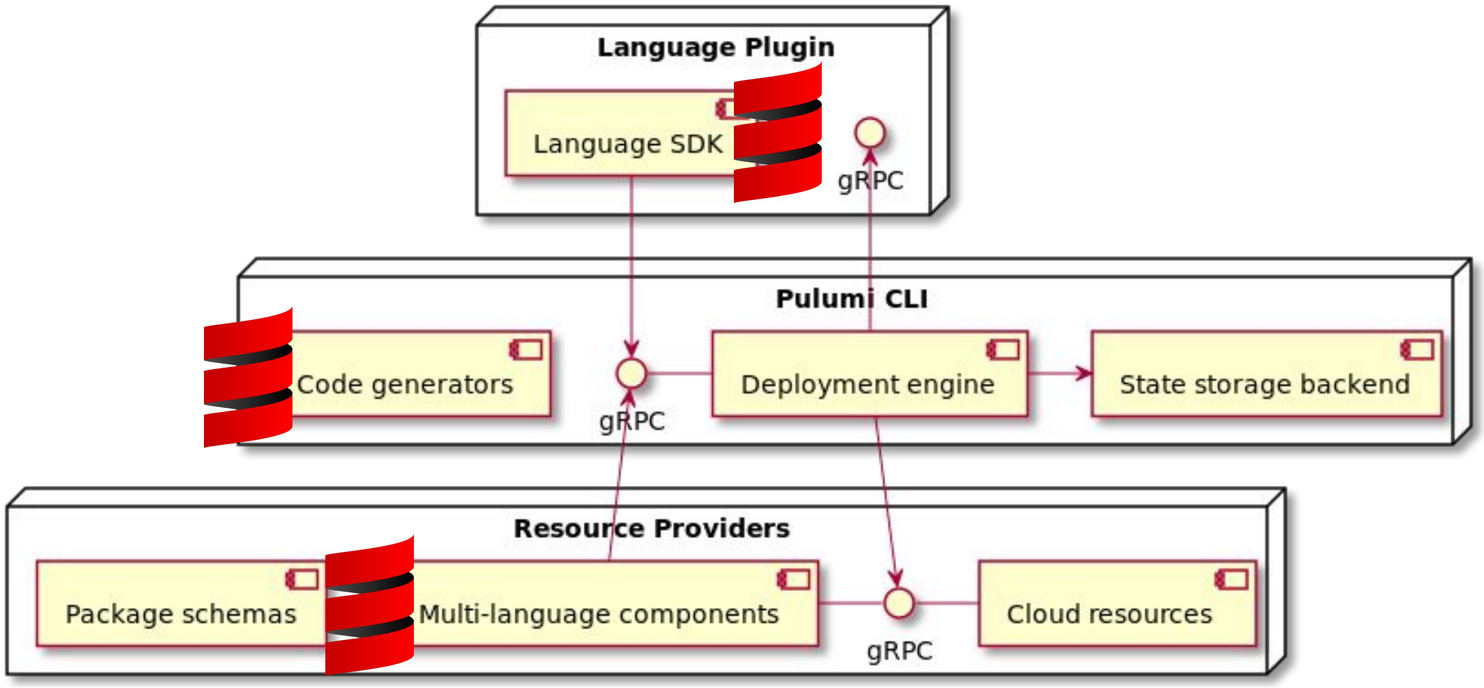
BESOM

Scala SDK for Pulumi

Motivation

- We have a experience with maintenance of cloud deployments
- Current solutions are just too constraining and too brittle
- We’ve built a similar project just for Kubernetes at VL
- We want to manage infra in a language where It Works If It Compiles™

Which pieces of puzzle do we need?



Objectives

- Leverage all of the Scala 3 improvements and niceties
- Support for all Scala ecosystems (just like tAPIr and sttp do)
- Make everything more type-safe
- No footguns!
- Focus on Developer eXperience

Architectural decisions

- Make everything purely functional
- Leverage patterns known to Scala programmers: map, flatMap, sequence (it's *traverse* btw)
- Leverage metaprogramming and generic derivation to minimize the codebase size and avoid bugs but keep things simple in public APIs

How to deal with other effects?

```
enum Result[+A]:
  case Suspend(thunk: () => Future[A])
  case Pure(a: A)
  case Defer(thunk: () => A)
  case Blocking(thunk: () => A)
  case Fail(t: Throwable) extends Result[Nothing]
  case BiFlatMap[B, A](
    r: Result[B],
    f: Either[Throwable, B] => Result[A]
  ) extends Result[A]
  case Fork(that: Result[A]) extends Result[Fiber[A]]
  case Sleep(that: () => Result[A], duration: Long)
```

What is Result?

- a free algebra defining execution operations as data
- each target effect provides a `trait Runtime[F[+_]]` instance that is used to interpret Result at the end of the world
- it's a monad
- it's an equivalent to Free[S, A] where S is fixed algebra describing our operations

How are user programs built?

- Output[A] is defined in terms of Result[OutputData[A]]
- user programs are defined in terms of Outputs
- users can choose any effect library that has an implementation of Result.ToFuture[F] typeclass
- Scala Future, Cats-Effect IO and ZIO are supported OOTB

A taste of the API

```scala
import besom.*, api.aws, aws.s3.*

@main def main = Pulumi.run:
  for
    catsBucket <- aws.s3.bucket("cats", BucketArgs(
            acl = "public-read"
          ))
    dogsBucket <- aws.s3.bucket("dogs", BucketArgs(
            acl = "public-read"
          ))
  yield exports(
    catsUrl = catsbucket.websiteEndpoint,
    dogsUrl = dogsBucket.websiteEndpoint
  )
```

What is different?

- we don’t use constructors, we use functions as resource constructors
- laziness does break away from usual Pulumi evaluation
  - we will warn about dangling resources that were not composed into the program
  - we are thinking about support for opt-in eager evaluation of Result datatype

Features

```scala
val labels = Map("app" -> "nginx")
val appNamespace: Output[k8s.core.v1.Namespace] =
  namespace("liftoff") <<< Resource constructor

val html =
  """<h1>Welcome to Besom:
    |Functional Infrastructure
    |in Scala 3</h1>".stripMargin

val indexHtmlConfigMap: Output[k8s.core.v1.ConfigMap] =
configMap(
  "index-html-configmap",
  ConfigMapArgs(
    metadata = ObjectMetaArgs(
      name = "index-html-configmap",
      labels = labels,
      namespace =
        appNamespace.flatMap(_.metadata).map(_.name.get)
    ),
    data = Map(
      "index.html" -> html
    )
  )
)
```

Lifting

- when user keeps resources as values:

```
val appNamespace: Output[k8s.core.v1.Namespace] =
  namespace("liftoff")
```

- how to get to the name of the namespace?

```
appNamespace.flatMap(_.metadata).map(_.name.get)
```

- lifted syntax:

```
appNamespace.metadata.name.flatten
```

Mechanics of lifting

- we generate these extension methods:

```
extension (on: Output[Namespace])
  def metadata: Output[Metadata] =
    on.flatMap(_.metadata)

extension (om: Output[Metadata])
  def name: Output[Option[String]] =
    om.map(_.name)
```

- so in the end user can just write:

```
appNamespace.metadata.name.flatten
```

Features

```scala
val labels = Map("app" -> "nginx")
val appNamespace: Output[k8s.core.v1.Namespace] =
  namespace("liftoff")

val html =
  """<h1>Welcome to Besom:
    |Functional Infrastructure
    |in Scala 3</h1>".stripMargin

val indexHtmlConfigMap: Output[k8s.core.v1.ConfigMap] =
configMap(
  "index-html-configmap",
  ConfigMapArgs(
    metadata = ObjectMetaArgs(
      name = "index-html-configmap",
      labels = labels,  ↓↓↓ Lifting support
      namespace = appNamespace.metadata.name.flatten
    ),
    data = Map(
      "index.html" -> html
    )
  )
)
```

Component API: pulumi-js

```
class MyComponent extends pulumi.ComponentResource {
    constructor(name, opts) { ↓↓↓
        super("pkg:index:MyComponent", name, {}, opts);

        let bucket = new aws.s3.Bucket(`${name}-bucket`,
            {/*...*/}, { parent: this }); <<<
          ↓↓↓
        this.registerOutputs({
            bucketDnsName: bucket.bucketDomainName,
        })
    }
}
```

Component API: Besom

```scala
case class MyComponent(
  bucketDnsName: Output[String]
) extends ComponentResource

def myComponent(name: NonEmptyString)(using Context) =
  component(name, "pkg:index:MyComponent"):
    val bucket = aws.s3.bucket(s"$name-bucket", /*...*/)
    MyComponent(bucket.bucketDomainName)
```

Progress, release

- We are reaching basic feature parity with other Pulumi SDKs rapidly
- Our efforts are directed towards solving issues with provider packages publishing
- We got away with small coverage thanks to Scala and types so... WRITE MORE TESTS
- We assume that first public beta can be released in the end of June or at the start of July

What is the vision?

- We want to closely integrate infra-as-code support with Scala tooling so that it's a seamless experience

```
$ scala infra up ?
```

- We want to build more complex, typed modular blocks that will allow users to define well-typed interfaces between services

... all in Scala.

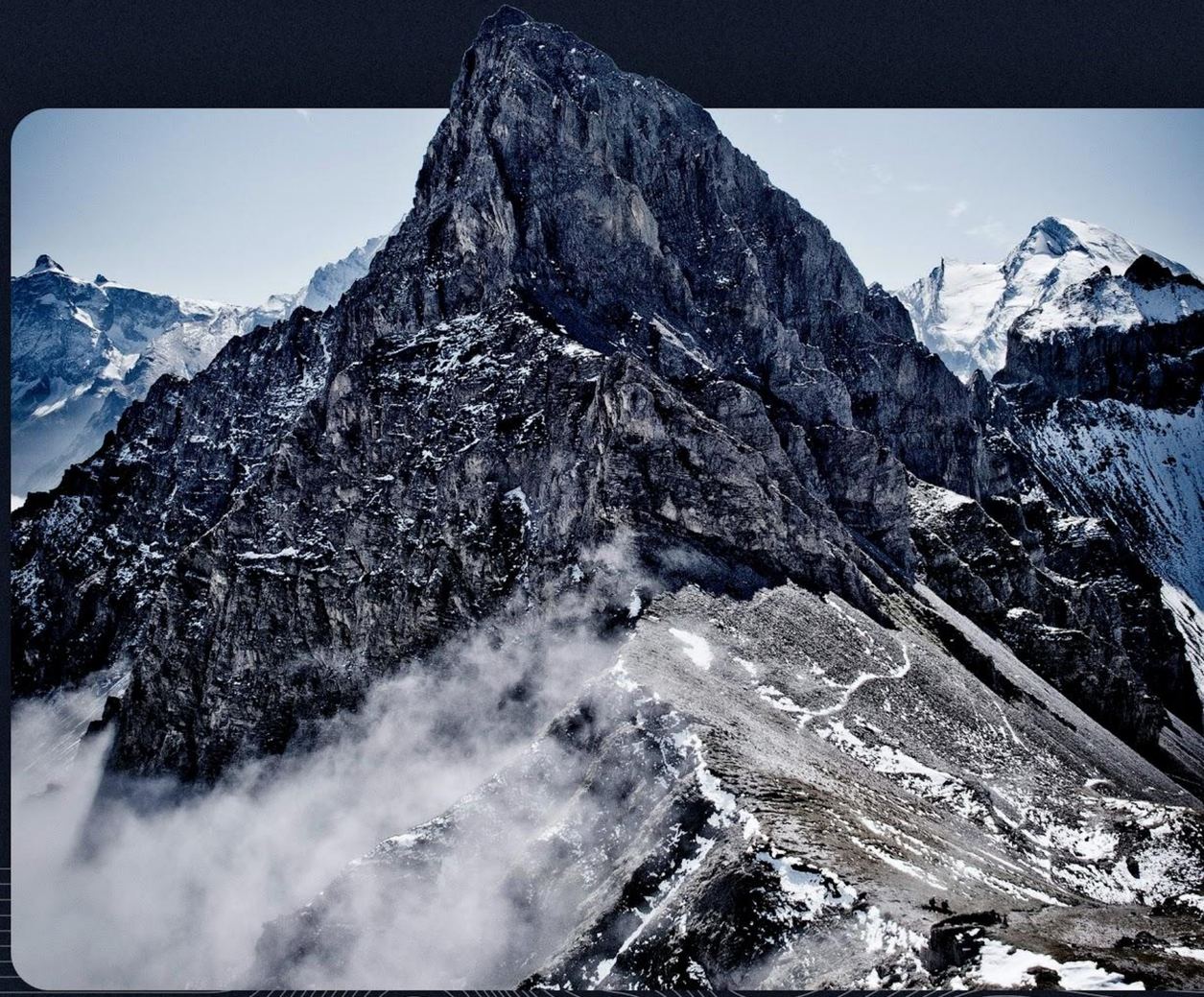
What is the vision?

- A radical idea: typed FAAS prototype - all lambdas defined in one program, typechecked together, built as separate projects and deployed as separate artifacts

- Pulumi already has this as Magic Lambda API for pulumi-js/ts

- We have AWS lambdas (with full working AWS SDK!) compiling to native binaries via Scala-Native and running in sub-15ms range!

Thank you



VIRTUS LAB @ 10 YEARS lambda DAYS 5-6 JUNE 2023