KubeCon CloudNativeCon India 2026

The Death of the YAML-Engineer

Engineering **"Invisible"** Platforms with Crossplane and Score

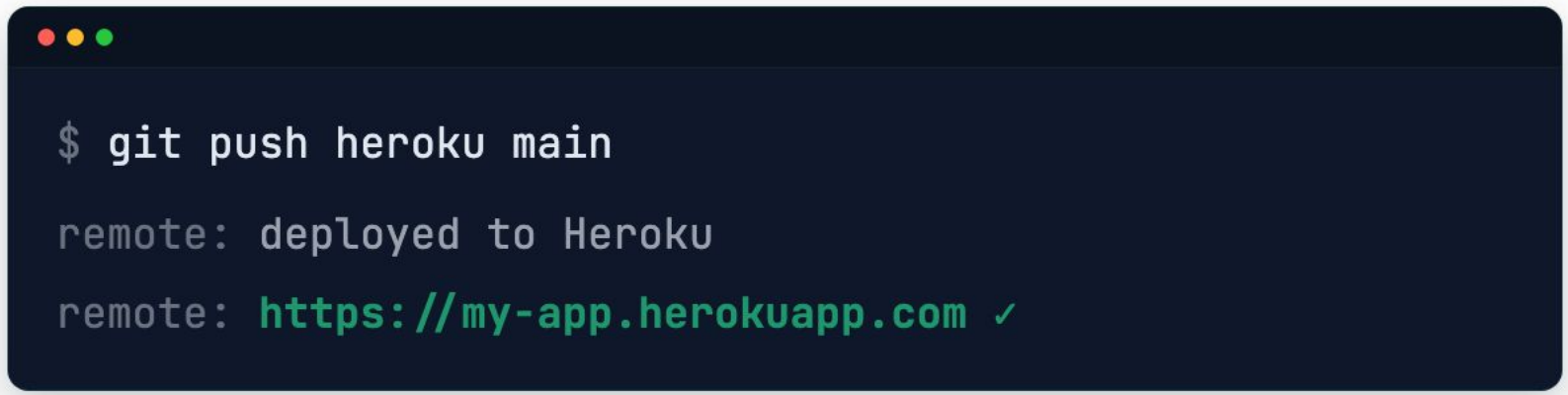
Abhinav Sharma

SRE, KodeKloud

Mumshad Mannambeth

Founder, KodeKloud

The YAML Tax



That's how easy it should be.

Fifty lines of mine. Eight hundred of theirs.

MY CODE · 50 lines

app.py

```python
# app.py — what the developer actually wrote
from flask import Flask, jsonify

app = Flask(__name__)

@app.route("/health")
def health():
    return jsonify(status="ok")

@app.route("/orders/<id>")
def get_order(id):
    return jsonify(id=id, items=[])

# ...50 lines total
```

THEIR CONFIG · ~800 lines

values.yaml

```yaml
    fsGroup: 1001
  resources:
    requests:
      cpu: 250m
      memory: 256Mi
  service:
    annotations:
      nginx.ingress/rewrite: "/"
      cert-manager.io/issuer: "le"
metrics:
  enabled: false
  serviceMonitor:
    enabled: false
tls:
  enabled: false
  autoGenerated: true
networkPolicy:
  enabled: true
  allowExternal: true
ingress:
```

Nothing catches this by default

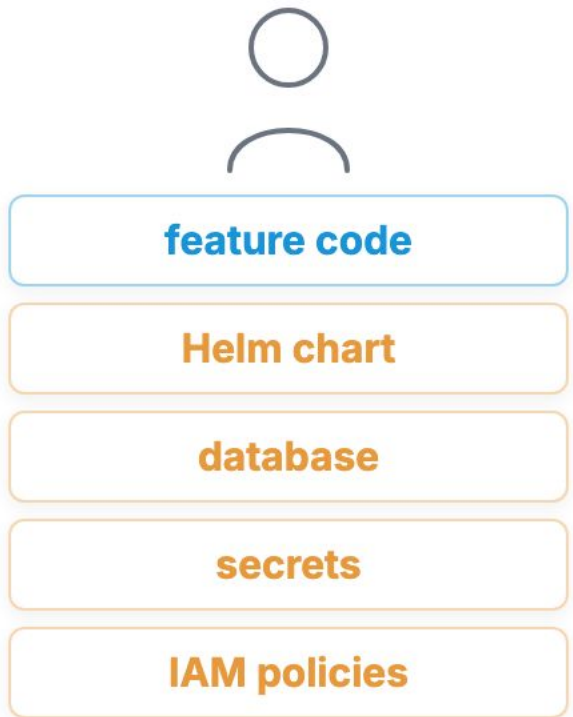
```
# my-values.yaml — override
perstistence.size: 100Gi

# what Helm actually applied
persistence.size: ~~100Gi~~ 8Gi  ← silent default
```

```
$ helm install  deployed successfully ✓
```

their data model = your interface

No clean boundary



The fix is a contract

developer

`declares what`

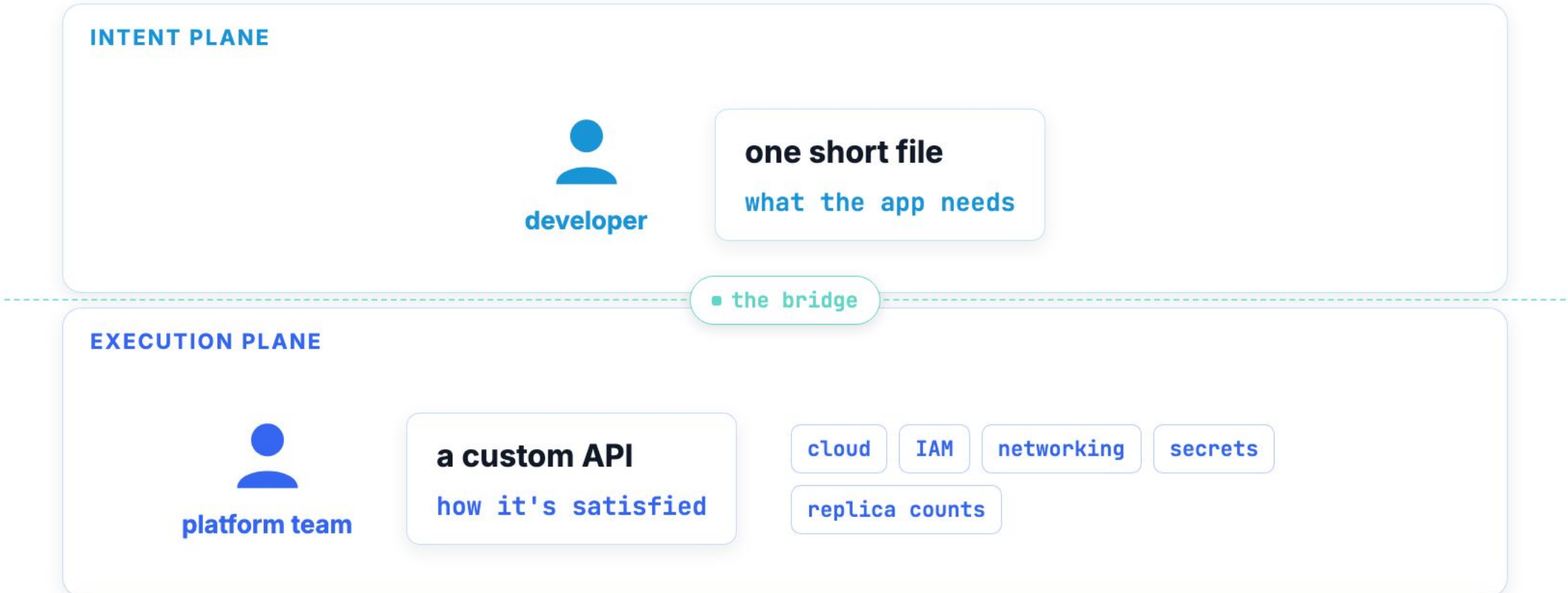
CONTRACT

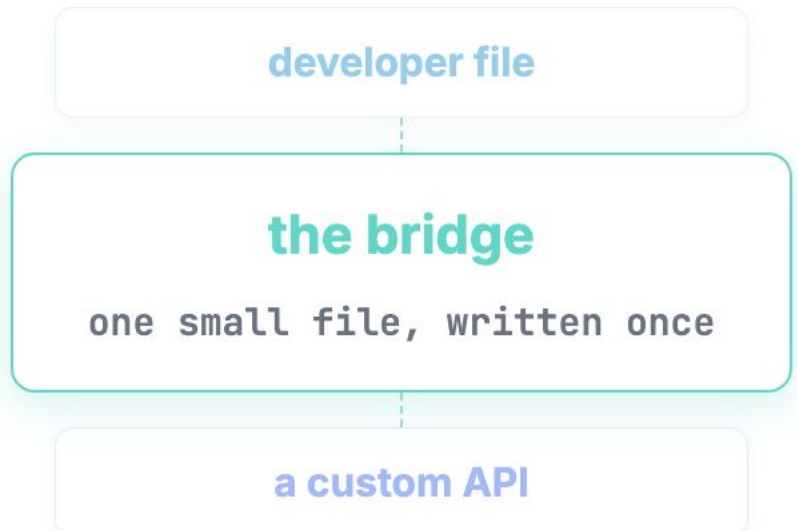
platform team

`handles how`

The Invisible Platform

Two planes. One contract.




developer file
the bridge
one small file, written once
a custom API

THE PRINCIPLE

The developer writes `type: postgres` and gets a connected database — without knowing its hostname, password, or where it runs.

Why “invisible”

WHAT THE DEVELOPER SEES

one short file

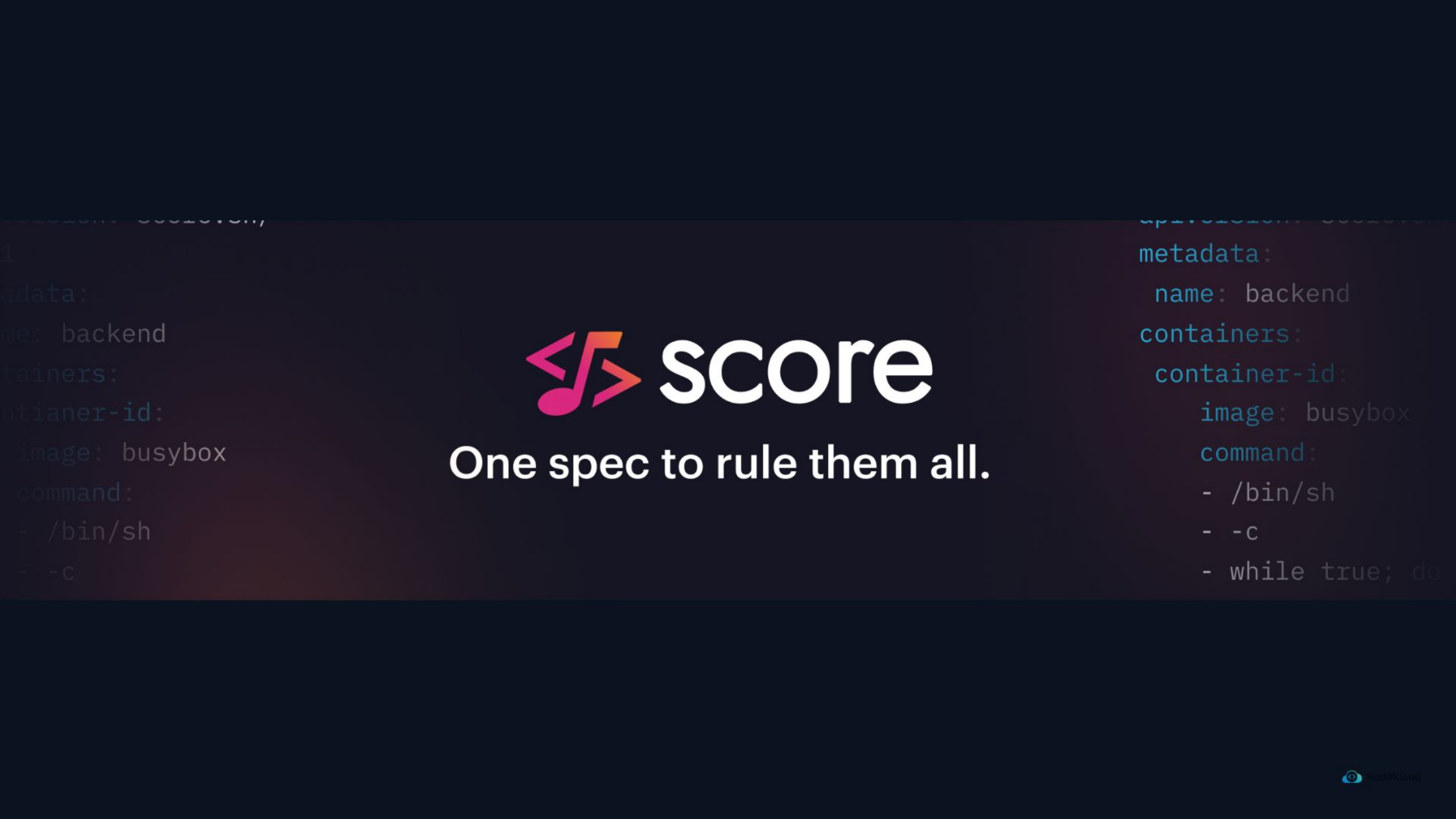
`twenty-five lines`

the developer never looks down

THE PLATFORM, STILL THERE

`a custom API` `cloud` `IAM` `networking` `secrets` `backups` `replica counts`

from where the developer sits, it just works


score
One spec to rule them all.
metadata:
name: backend
containers:
container-id:
image: busybox
command:
- /bin/sh
- -c
- while true; do

Same app. Five hundred lines, or twenty-five.

~~Deployment~~ ~~StatefulSet~~ ~~Service~~ ~~Secret~~ ~~PVC~~

THE OLD INTERFACE

photo-api/templates/*.yaml

```
# ------------------------------------------------------------
apiVersion: {{ template "photo-api.apiVersion" . }}
kind: Deployment
metadata:
  name: {{ include "photo-api.fullname" . }}-1
  namespace: {{ .Release.Namespace }}
  labels:
    app.kubernetes.io/name: {{ include "photo-api.name" .
    app.kubernetes.io/instance: {{ .Release.Name }}
    helm.sh/chart: {{ include "photo-api.chart" . }}
  annotations:
    checksum/config: {{ include (print $.Template.BasePat
spec:
  replicas: {{ .Values.replicaCount | default 3 }}
  selector:
    matchLabels:
      app.kubernetes.io/name: {{ include "photo-api.name"
  template:
    metadata:
```

THE CONTRACT

score.yaml

```
apiVersion: score.dev/v1b1
metadata:
  name: note-taker
containers:
  app:
    image: abhinav332/note-taker-app:latest
    variables:
      POSTGRES_HOST: ${resources.db.host}
      POSTGRES_PORT: ${resources.db.port}
service:
  ports:
    web: { port: 8989, targetPort: 8989 }
resources:
  db:
    type: postgres
    params: { size: small }
```

Shaped like the platform, or the app?

PLATFORM NOUNS

a Helm chart is shaped like Kubernetes

templates/deployment.yaml

```yaml
kind: Deployment
metadata:
  name: {{ include "note-taker.name" . }}
spec:
  replicas: {{ .Values.replicaCount }}
---
kind: Service
spec:
  ports: [ { port: 8989 } ]
---
kind: ConfigMap
data:
  APP_ENV: production
```

APP NOUNS

a score.yaml is shaped like your app

score.yaml

```yaml
apiVersion: score.dev/v1b1
metadata:
  name: note-taker
containers:
  app:
    image: abhinav332/note-taker-app
service:
  ports:
    web: { port: 8989 }
resources:   # dependencies
  db:
    type: postgres
```

Five fields. Three do the work.

what to run — REQUIRED — RUNTIME

what to publish — NETWORK

what the platform provides — DEPENDENCIES

score.yaml

```yaml
apiVersion: score.dev/v1b1
metadata:
  name: note-taker
containers:
  app:
    image: abhinav332/note-taker-app
    variables:
      POSTGRES_HOST: ${resources.db.host}
      POSTGRES_PORT: ${resources.db.port}
service:
  ports:
    web: { port: 8989 }
resources:
  db:
    type: postgres
```

developer: what → platform: how

Five fields. Three do the work.

score.yaml

```yaml
apiVersion: score.dev/v1b1
metadata:
  name: note-taker
containers:
  app:
    image: abhinav332/note-taker-app
    variables:
      POSTGRES_HOST: ${resources.db.host}
      POSTGRES_PORT: ${resources.db.port}
service:
  ports:
    web: { port: 8989 }
resources:
  db:
    type: postgres
```

all below the boundary now

NOT IN YOUR FILE

- ~~✗ database hostname~~
- ~~✗ database password~~
- ~~✗ which cloud / region~~
- ~~✗ replica count~~
- ~~✗ IAM role ARNs~~
- ~~✗ ingress annotations~~

What you do with the file

You don't deploy `score.yaml`. You run a Score CLI on it.

```
$ score-compose generate score.yaml
```

compose.yaml

```
# abbreviated — real output is longer
services:
  db:
    image: postgres:16
    ports: [ 5432 ]
    # ...volumes, environment, networ
```

```
$ score-k8s generate score.yaml
```

manifests.yaml

```
# abbreviated — each kind has a full
kind: StatefulSet
kind: Service
kind: Secret
# ...metadata, spec, replicas, volume
```

```
$ score-cloud? generate score.yaml
```

```
? unknown
```

a real, managed cloud database

Same file. The provisioner changes.

score.yaml

```yaml
apiVersion: score.dev/v1b1
containers:
  app:
    variables:
      DB_HOST: ${resources.db.host}
resources:
  db:
    type: postgres
```

laptop

$ score-compose generate

a Docker Postgres container

```
DB_HOST=db
```

Kubernetes cluster

$ score-k8s generate

StatefulSet + Service + Secret

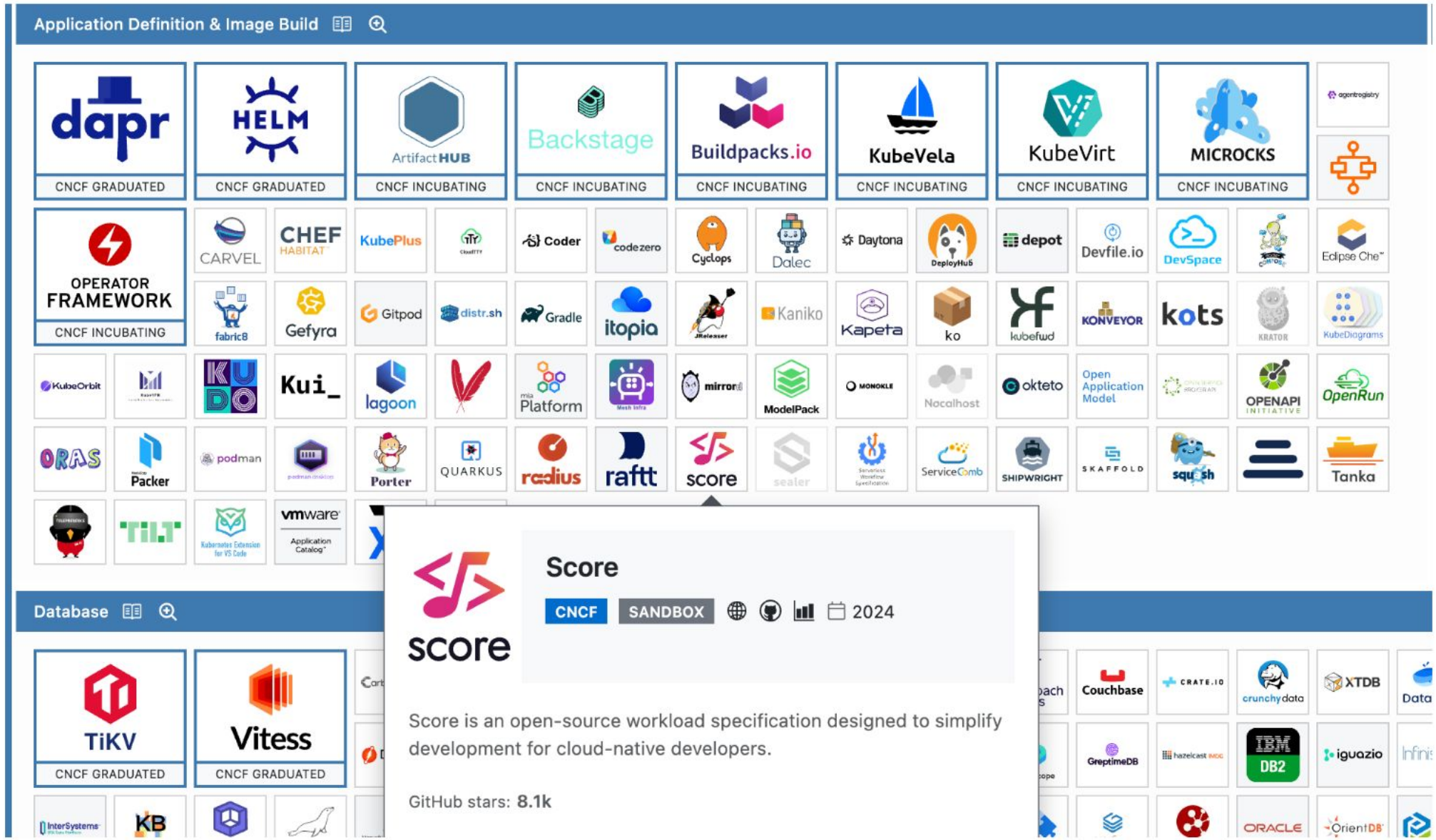
```
DB_HOST=note-taker-db.…
```

the cloud ?

$ score-cloud? generate

a real, managed cloud database

```
which provisioner?
```


Application Definition & Image Build
dapr
CNCF GRADUATED
HELM
CNCF GRADUATED
Artifact HUB
CNCF INCUBATING
Backstage
CNCF INCUBATING
Buildpacks.io
CNCF INCUBATING
KubeVela
CNCF INCUBATING
KubeVirt
CNCF INCUBATING
MICROCKS
CNCF INCUBATING
OPERATOR FRAMEWORK
CNCF INCUBATING
CARVEL
CHEF HABITAT
KubePlus
Coder
codezero
Cyclops
Dalec
Daytona
DeployHub
depot
Devfile.io
DevSpace
Eclipse Che
fabric8
Gefyra
Gitpod
distr.sh
Gradle
itopia
Kaniko
Kapeta
ko
kubefwd
KONVEYOR
kots
KRATOR
KubeDiagrams
KubeOrbit
KUDO
Kui_
lagoon
mia Platform
mirrord
ModelPack
MONOKLE
Nocalhost
okteto
Open Application Model
OPENAPI INITIATIVE
OpenRun
ORAS
Packer
podman
Porter
QUARKUS
radius
raftt
score
sealer
ServiceComb
SHIPWRIGHT
SKAFFOLD
squash
Tanka
TiLT
vmware Application Catalog
Database
TiKV
CNCF GRADUATED
Vitess
CNCF GRADUATED
Couchbase
CRATE.IO
crunchy data
XTDB
GreptimeDB
hazelcast
IBM DB2
iguazio
InterSystems
KB
ORACLE
OrientDB
score
Score
CNCF
SANDBOX
2024
Score is an open-source workload specification designed to simplify development for cloud-native developers.
GitHub stars: 8.1k

Provisioning - The How

What Score actually does

Score produces YAML. Something else has to make it real.

Same shape. Different target.

YAML YOU ALREADY KNOW

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: webapp
spec:
  replicas: 3
  template:
    spec:
      containers:
        - name: webapp
          image: nginx:1.27
```

YAML YOU DON'T — YET

```yaml
apiVersion: s3.aws.upbound.io/v1beta1
kind: Bucket
metadata:
  name: kodekloud-demo-bucket
spec:
  forProvider:
    region: us-east-1
  providerConfigRef:
    name: default
```

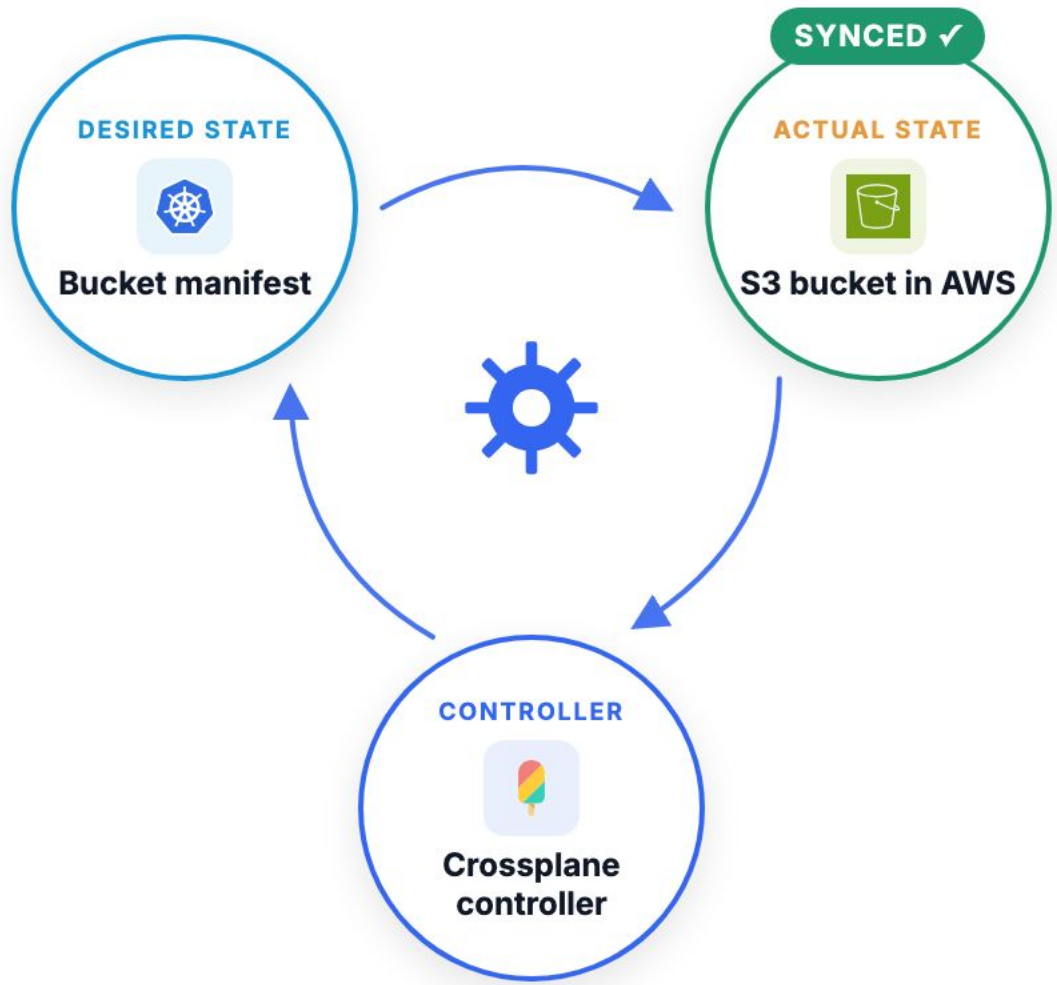

```
$ kubectl apply -f bucket.yaml
bucket.s3.aws.upbound.io/kodekloud-demo-bucket created

$ kubectl get bucket
NAME                    READY   SYNCED   EXTERNAL-NAME           AGE
kodekloud-demo-bucket   True    True     kodekloud-demo-bucket   47s
```

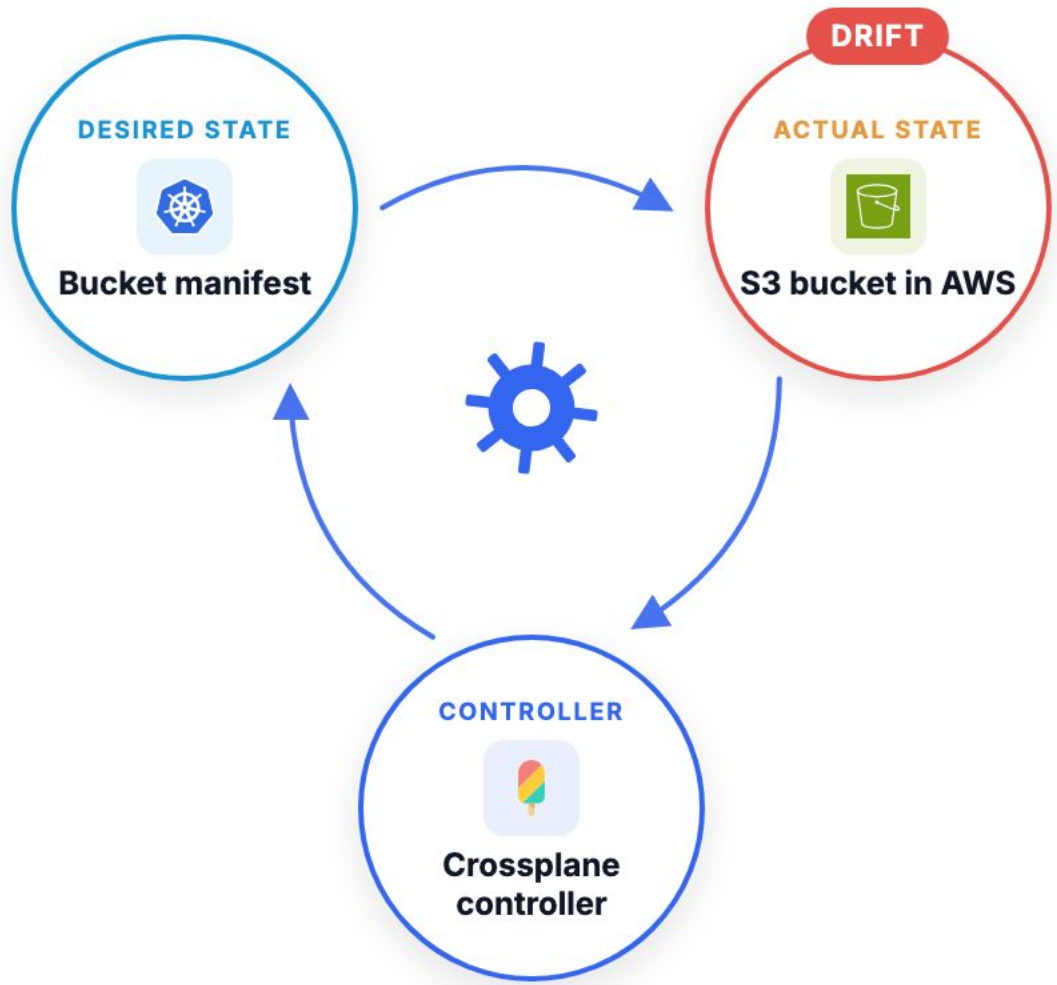
Crossplane

What Kubernetes does for your apps, Crossplane extends to everything else.

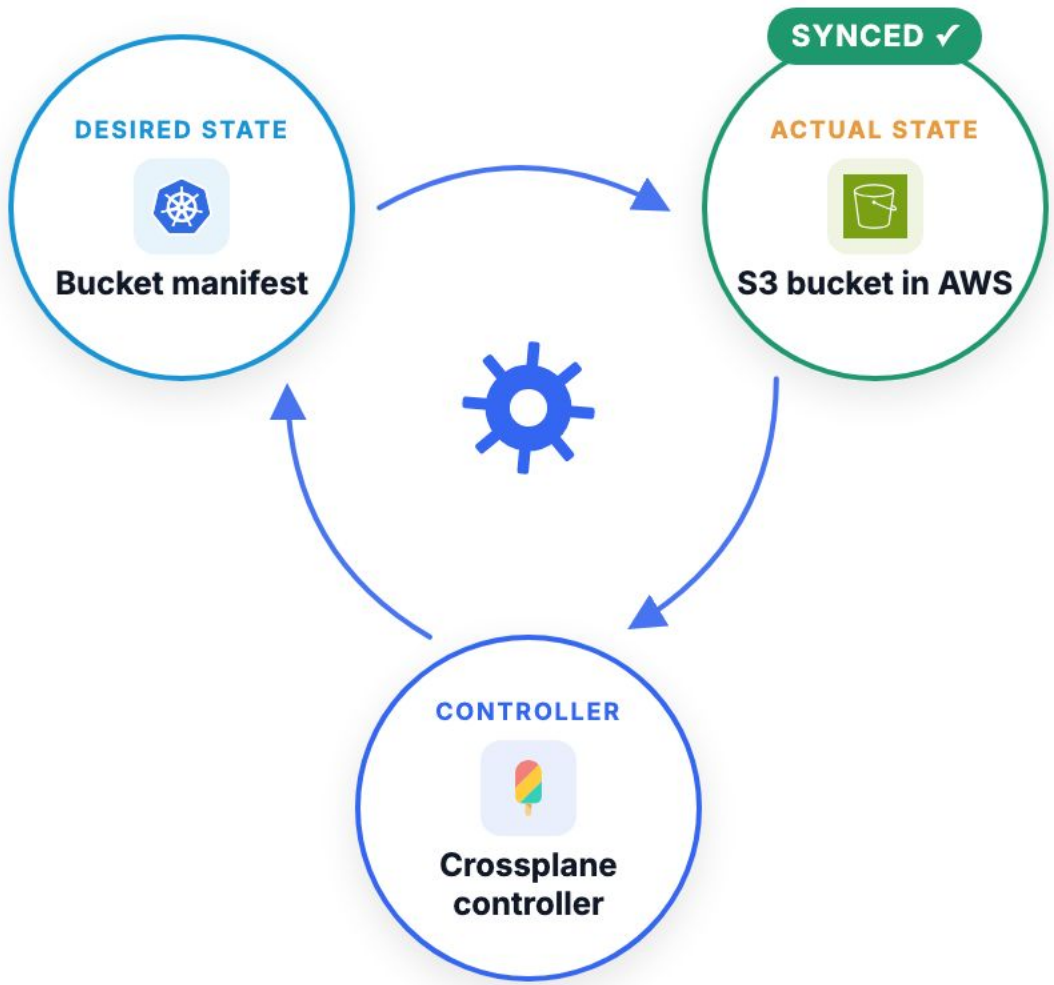
It never stops watching



It never stops watching



It never stops watching



One line. Same command. Different output.

score.yaml

```yaml
apiVersion: score.dev/v1b1
metadata:
  name: note-taker
containers:
  app:
    image: abhinav332/note-taker-app
    variables:
      POSTGRES_HOST: ${resources.db.host}
      POSTGRES_PORT: ${resources.db.port}
service:
  ports:
    web: { port: 8989 }
resources:
  db:
    type: postgres
    params: { size: small }
    class: crossplane   # the only change
```

```
$ score-k8s generate score.yaml
```

manifests.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
  # 3 replicas of note-taker-app
  POSTGRES_HOST: note-taker-postgres
---
apiVersion: v1
kind: Service
  # exposes note-taker-app on 8989
---
apiVersion: db.kodekloud.io/v1alpha1
kind: DatabaseInstance
  workloadName: note-taker
  engine: postgres
  size: small
```

```
$ kubectl apply -f manifests.yaml
```

Three artifacts. Written once.

INTENT PLANE

`type: postgres · class: crossplane` *the only line the developer writes*

provisioners.yaml · the bridge

EXECUTION PLANE

platform team · writes once

XRD
the developer-facing API

DatabaseInstance

`workloadName` `engine` `size`

COMPOSITION
the recipe

produces

StatefulSet

`Service: note-taker-postgres`

Secret: note-taker-db-secret

storage class · limits · security context · backup window

BRIDGE
provisioners.yaml

the naming rule

```
type: postgres
→ this Composition
```

The bridge is a naming convention

score.yaml

```yaml
apiVersion: score.dev/v1b1
metadata:
  name: note-taker
containers:
  app:
    image: abhinav332/note-taker-app
    variables:
      POSTGRES_HOST: ${resources.db.host}
      POSTGRES_PORT: ${resources.db.port}
service:
  ports:
    web: { port: 8989 }
resources:
  db:
    type: postgres
    class: crossplane
```

→ note-taker-postgres

provisioners.yaml

```yaml
apiVersion: score.dev/v1b1
kind: Provisioner
metadata:
  name: postgres
spec:
  type: postgres
  class: crossplane
  outputs: |
    host: "{{ .SourceWorkload }}-postgres"
    port: 5432
```

The bridge is a naming convention

score.yaml

```yaml
apiVersion: score.dev/v1b1
metadata:
  name: note-taker
containers:
  app:
    image: abhinav332/note-taker-app
    variables:
      POSTGRES_HOST: ${resources.db.host}
      POSTGRES_PORT: ${resources.db.port}
service:
  ports:
    web: { port: 8989 }
resources:
  db:
    type: postgres
    class: crossplane
```

provisioners.yaml

```yaml
apiVersion: score.dev/v1b1
kind: Provisioner
metadata:
  name: postgres
spec:
  type: postgres
  class: crossplane
  outputs: |
    host: "{{ .SourceWorkload }}-postgres"
    port: 5432
```

Demo

From YAML Manager → Platform Architect

YAML MANAGER · TODAY

- Writes Helm charts per service
- Manages infra inside application repos
- Responds to tickets: "provision a database"
- Developer onboarding takes weeks
- Noticed only when things break

PLATFORM ARCHITECT · THE EVOLUTION

- Designs XRDs and Compositions
- Owns the Execution Plane — once
- Developers self-serve via score.yaml
- Developer onboarding: 15 minutes
- Invisible — because everything works

The YAML engineer is not dead. They evolved.

Are you still managing the YAML — or building the platform that made it invisible?

HANDS-ON LAB · KubeCon India 2026

Try the full demo yourself →

SCAN ME

kode.wiki/3QQi2rZ

Score	Crossplane	KodeKloud
score.dev	crossplane.io	kodekloud.com

Abhinav Sharma
github.com/abhi-bhatra

Mumshad Mannambeth
linkedin.com/in/mmumshad