anynines

# 10 Years of Automating PostgreSQL. A Recap.

# Introduction



anynines

# 10 Years

🥳🎂🎉

# 15 actually!

# This Talk

# This Talk

- Chronological summary
- Different stages of automating PostgreSQL
- How the technological zeitgeist impacted automation over time
- Learnings (as an individual as well as an organization)
- How to scale-out operations from dozens to thousands of machines

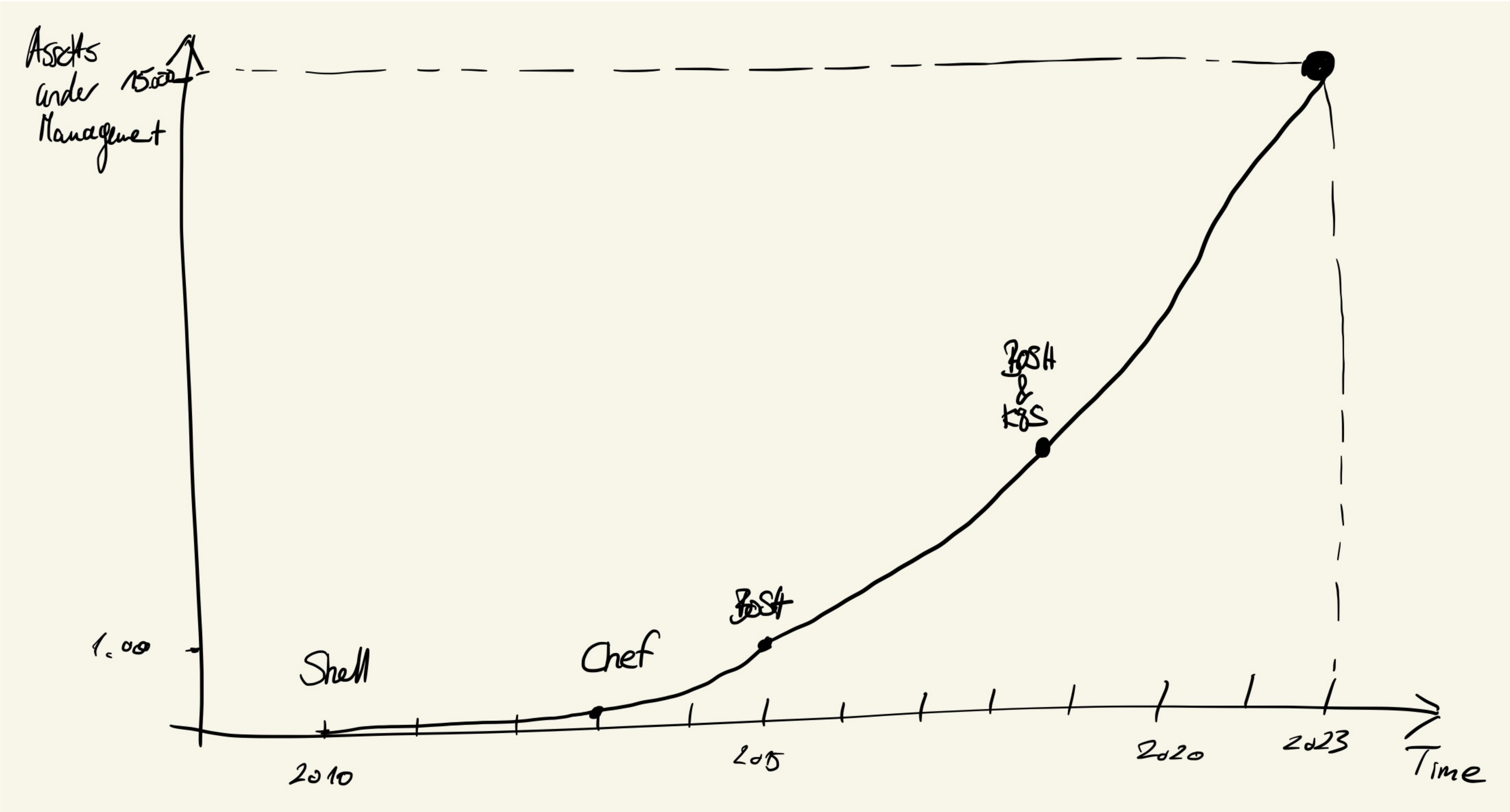


anynines

# The Perspective

# Aspects of Interest

- **Infrastructure**: physical, VM-hosts, virtual infrastructures.
- **Automation technology**: imperative, declarative.
- **Lifecycle mgmt. coverage of automation**: CRUD service instances, service bindings, backups & restore, configuration, upgrades, high-availabilty, etc.
- **Assets under management**: Physical / virtual machines && Pods / containers.
- **Operational responsibility**: Platform operator &&|| Application developer
- **PostgreSQL at the time**: HA & cluster management, upgrades, security, …


Assets under Management
1500
Shell
Chef
BOSH
BOSH & K8S
2010
2015
2020
2023
Time

# Imperative Automation

**Stage 1**: Physical servers
~2008-2009

# Physical Machines

# (Physical) Servers & Shell Scripts

- Best performance per €/$?
- Servers are dimensioned with respect to peak loads and are idle 90% of the time.
- Building clusters requires flexible data centers allowing the wiring of private networking.
- Often servers have contract lifetimes of several years (off-the-shelf DCs often not flexible enough) .
- Dependency to the data center staff for some tasks → Delays


# Shell Scripts

# (Physical) Servers & Shell Scripts

- Shell scripts
- **Coverage**: Supporting repetitive tasks
- Mostly manually managed OS
- OS provided software packages

# (Physical) Servers & Shell Scripts

- Striving for a maximum uptime as fixing failed servers requires manual intervention and takes hours to fully recover.
- Failures have strong impact on the Sysop's sleep.
    - Long TTR
- Manually executed tasks come with a human error rate

anynines

# (Physical) Servers & Shell Scripts

- PostgreSQL
  - Looking for ways to make PostgreSQL highly available
    - Blocklevel || filesystem level replication, e.g. GlusterFS
    - Sync and async replication
    - Pacemaker
    - …

# **Stage 2**: Virtual servers
# ~2009-2011

anynines

# Virtual Machines

# (Physical) Servers & Shell Scripts

- Automanually managed Xen & XVM hosts
- Increased hardware utilization
- Lower barrier of entry: virtual clusters
- VMs ➞ More machines to manage ➞ More automation needed



anynines

# Chef

# (Physical) Servers & Shell Scripts

- Configuration management
- Centralized cookbooks
  - Better reusability of code
  - Less code duplicity
- More efficient that shell scripts

# (Physical) Servers & Shell Scripts

- PostgreSQL
  - Async replication has proven to be the best all-round approach.
  - Pacemaker and, later, repmgr
  - Shared HA-PostgreSQL cluster to lower the barrier of entry (VM app servers + shared virtual or physical DB server) vs. dedicated virtual DB-servers.



anynines

# (Physical) Servers & Shell Scripts

- Limitations
  - **State drift!** Manual intervention necessary although (theoretically) covered by automation.
  - Increasing efforts for maintenance, refactoring and network management become limiting factors.
  - The team's utilization increased.
  - Training new team members was hard as, despite of the cookbooks, still a lot of knowledge was necessary to operate the application systems.



anynines

# The Game Changer

On-demand provisioning of dedicated PostgreSQL instances based on declarative automation.

# Declarative Automation



anynines

CLOUD FOUNDRY BOSH™

Kubernetes

OPEN SERVICE BROKER API™

# Shift in operational responsibility: App devs operate their own databases.

OPEN SERVICE BROKER API™

Kubernetes CRDs

# **Stage 3**: Virtual infrastructures ~2012-2023

# Virtual Infrastructures

The programmable data center

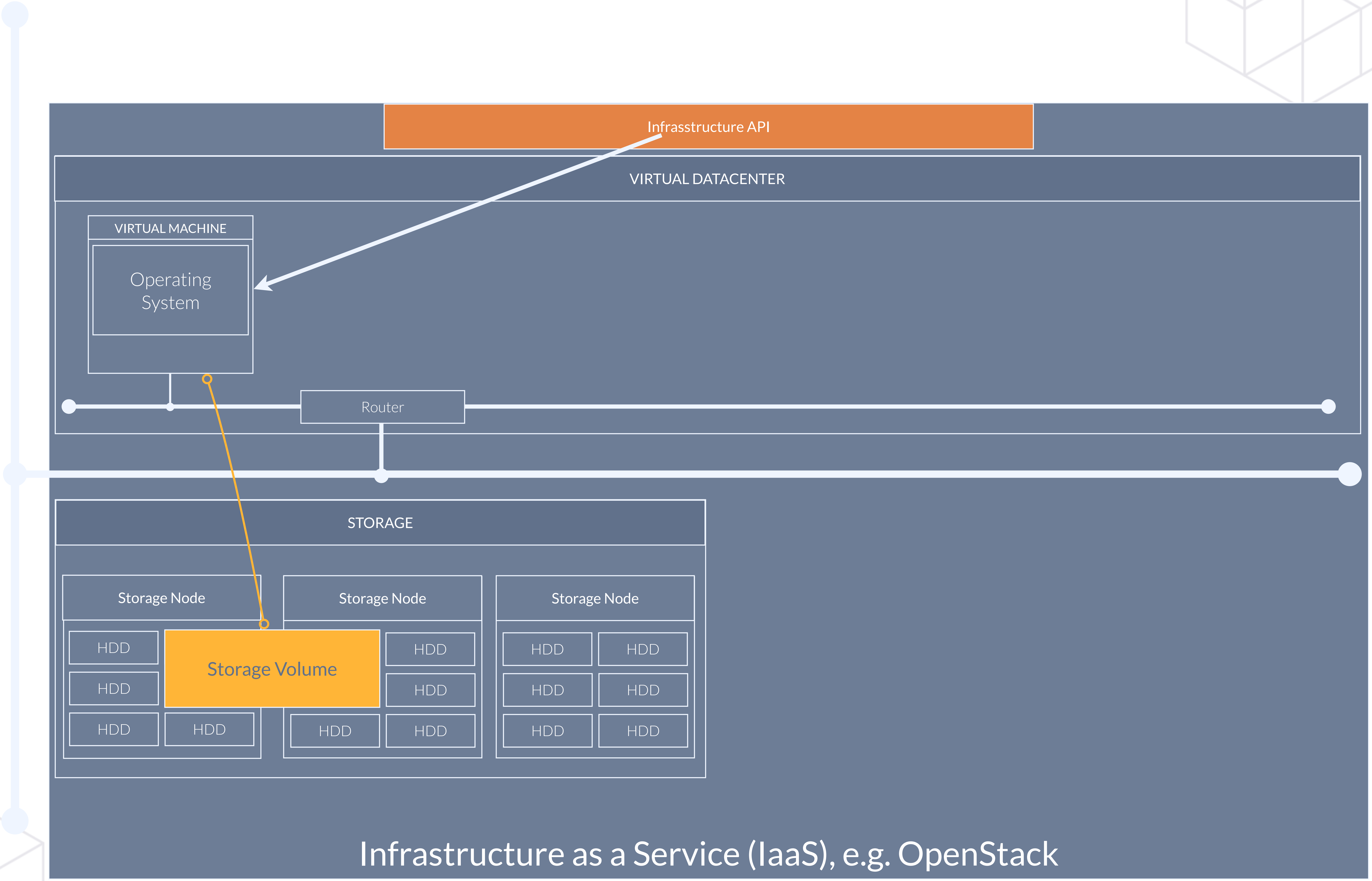
# Dealing with State

# Where to store state?

🔑

Store state on a **remotely attached block device** = persistent disk.

anynines



Infrasstructure API
VIRTUAL DATACENTER
VIRTUAL MACHINE
Operating System
Router
STORAGE
Storage Node
Storage Node
Storage Node
HDD
HDD
HDD
HDD
Storage Volume
HDD
HDD
HDD
HDD
HDD
HDD
HDD
HDD
HDD
HDD
Infrastructure as a Service (IaaS), e.g. OpenStack


anynines

The data lifecycle has been decoupled from the VM lifecycle
⇒ The VM becomes disposable.

# Ephemeral VM, persistent disk.

CLOUD FOUNDRY

BOSH™

# Predictable & repeatable deployments. No state-drift.

# Virtual Infrastructures

- PostgreSQL
  - Sync and Async Streaming replication
  - Repmgr and Patroni
  - Logic backups and PITR backups

# **Stage 4**: Container infrastructures ~2015-2023

# Kubernetes

# Kubernetes

- Declarative automation
- ~ Infrastructure abstraction API
- API standardization & Open framework for automation
- Container isolation & Noisy neighbor issues
- Often: VM automation underneath.

# Container Infrastructures

- PostgreSQL
  - Sync and Async Streaming replication
  - Patroni as a cluster manager

# The Future



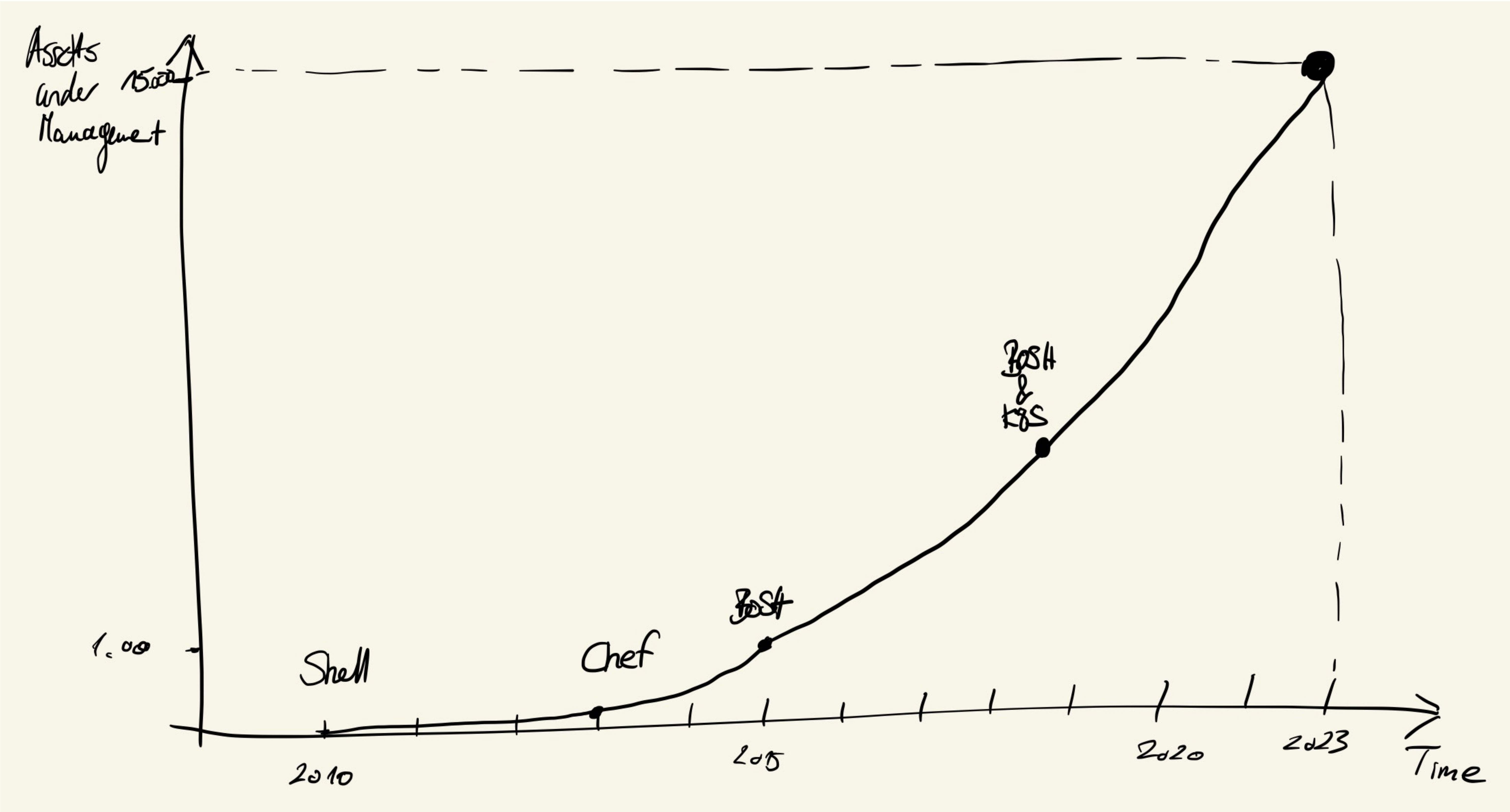
anynines

# Summary & Conclusion



anynines

| Infrastructure | Automation Paradigm | Automation Technology | Operated by | Coverage | Machines / Devop |
|---|---|---|---|---|---|
| Physical machines | Imperative | Shell scripts | Sysop | Simple repetitive tasks | A few dozen |
| (Semi-) Manually Managed VM Hosts | Imperative | Chef | Devop | Parts of the lifecycle. Devops centric. | A few hundred |
| Virtual Infrastructure | Declarative | BOSH | Automation: Devop Database: App Dev | Full lifecycle management | Thousands to ten thousands |
| Virtual or Physical Infrastructure | Declarative | Kubernetes & K8s Add-ons | Automation: Devop Database: App Dev | Full lifecycle management | Thousands to ten thousands |


Assets under Management
1500
Shell
Chef
BOSH
BOSH & K8S
2010
2015
2020
2023
Time

# Summary

- Virtualization, EVM-PD
- Declarative automation
- Increased automation friendliness of PG



anynines

# Thank You

anynines

a9s Data Services

a9s Elasticsearch

a9s LogMe

# Questions?
# @anynines
# @fischerjulian

a9s Redis

a9s MySQL

a9s MongoDB

a9s PostgreSQL

a9s RabbitMQ

anynines

# Thank You!

a9s Data Services

a9s Elasticsearch

a9s LogMe

a9s Redis

a9s MySQL

a9s MongoDB

a9s PostgreSQL

a9s RabbitMQ

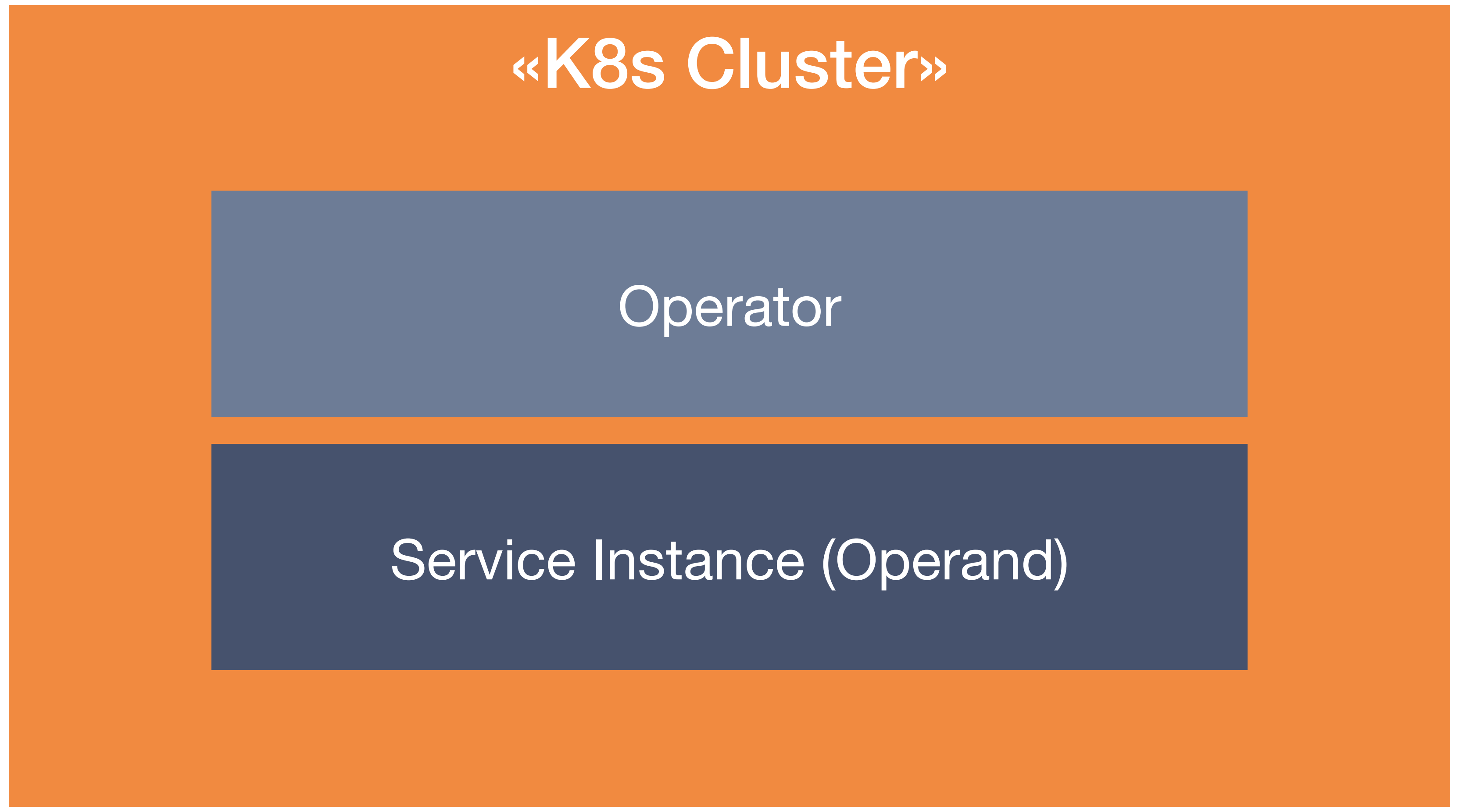
anynines

# Data Service Automation

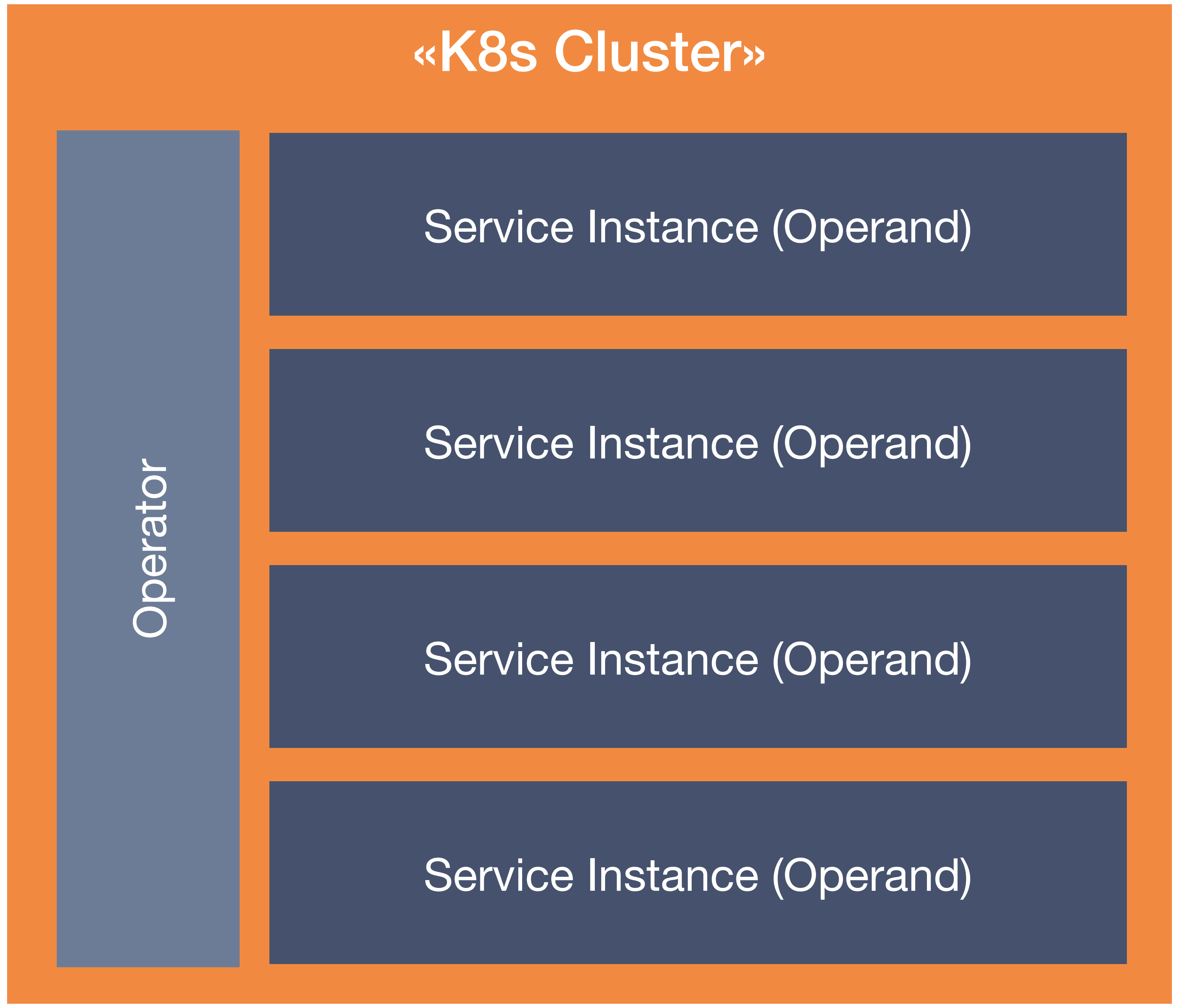


anynines

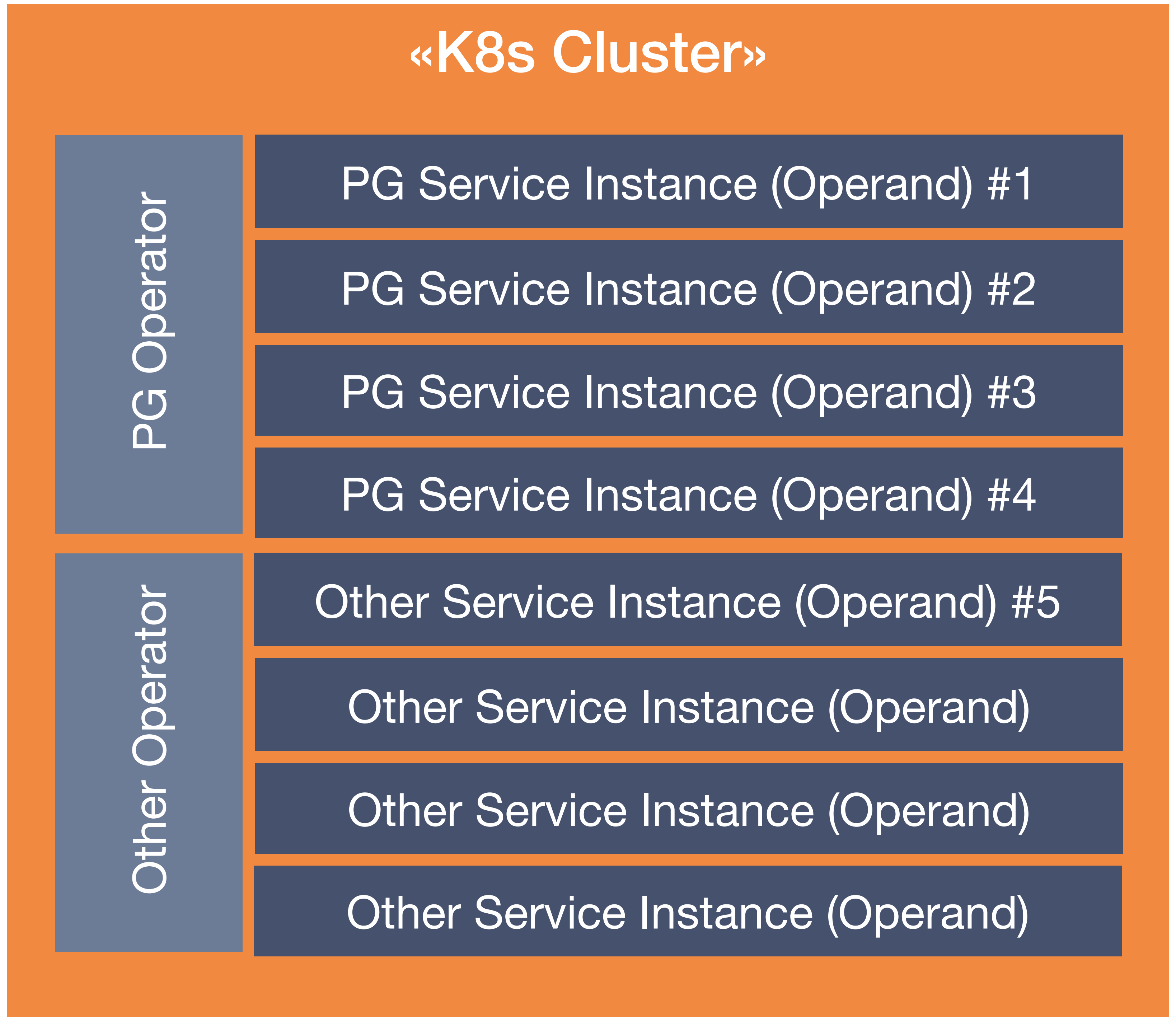
"Fully **automating** the entire **lifecycle** of a wide range of **data services** to run on cloud-native **platforms** across **infrastructures** at **scale**."



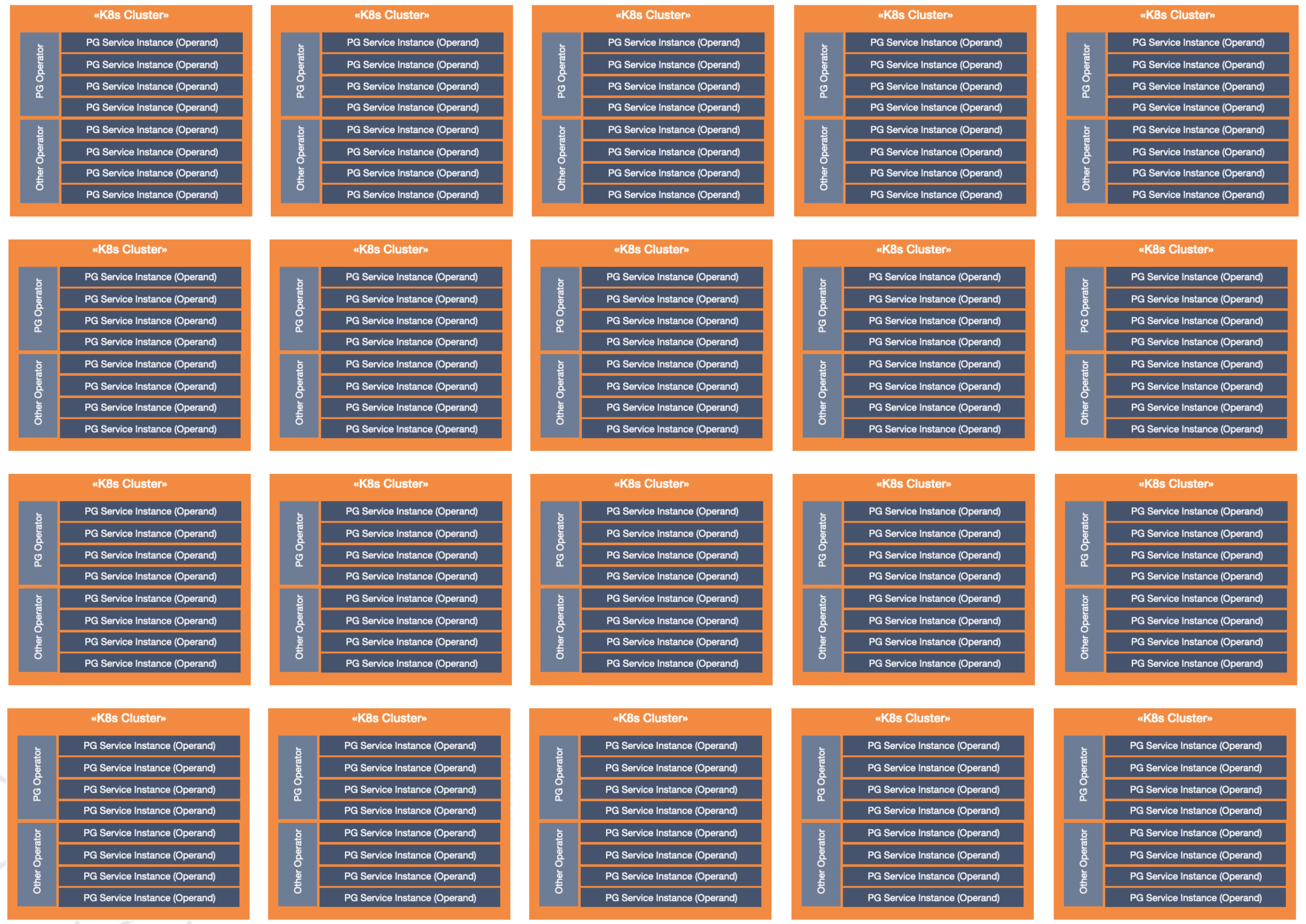

A single K8s cluster
with a single service instance
managed by a single Operator.




A single K8s cluster
with multiple service-instances
managed by a single Operator.




A single K8s cluster
with multiple service-instances
managed by a multiple Operators.




# Many K8s clusters each with multiple service-instances managed by a multiple Operators.

# 100s or 1000s of data service instances!



anynines

# Scale Matters!




anynines

# Each data service instance matters!



anynines

# Methodology



anynines

# Principles

# Principles

- Know your target audience. **Requirements** and desired **qualities**.
- Choose your data services, wisely. Be aware of open source licenses.
- Strive for full lifecycle automation.
- On-demand provisioning of dedicated service instances.
- **Rebuild failed instances instead of fixing them**.
- Design for scalability.

# Principles

- Operational model first, automation second.
- Be a **backup**/**restore** hero.
- Solve issues on the **framework** level, fine-tune data service specifically.
- **Test** code. **Test** service instances. **Test** desired and undesired behavior.
- Provide meaningful default configuration values. Except custom config parameters.

# Principles

- Don't touch upstream code, except for …
- Master **release management**
- **Deliver releases** into target environments quickly
- Collect feedback from users (e.g. through support)
- Provide meaningful documentation. Better documentation, less support.

# Data Service Automation with Kubernetes



anynines

# Ways to Implement an „Operator“



anynines

# Data Service Automation with K8s

- Kubernetes CRDs + Custom Controllers
- Operator SDK
- KUDO

# Stages of Development



anynines

# Data Service Automation with K8s

- Operational Model - Level 1: What a sysop/DBA would do.
- Operational Model - Level 2: Containerization, YAML + `kubectl`
- Operational Model - Level 3: Operator
- Operational Model - Level 4: Operator Lifecycle Management



anynines

# CRDs

```yaml
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  # name must match the spec fields below, and be in the form: <plural>.<group>
  name: pgs.ds.a9s.io
spec:
  # group name to use for REST API: /apis/<group>/<version>
  group: ds.a9s.io
  # list of versions supported by this CustomResourceDefinition
  versions:
    - name: v1
      # Each version can be enabled/disabled by Served flag.
      served: true
      # One and only one version must be marked as the storage version.
      storage: true
      schema:
        openAPIV3Schema:
          type: object
          description: Yeah! Science!
          properties:
            spec:
              type: object
              required: ["replicas"]
              properties:
                postgresVersion:
                  type: string
                  # pattern: major.minor.patchlevel or major.minor > determine patchlevel automatically
                  default: "12.2"
                # postgresPlugins:
                #   type: array
                replicas:
                  type: integer
                  # pattern: 2n+1
                  minimum: 1
                  default: 1
  # either Namespaced or Cluster. Namespaced as data service instances should belong to a namespace.
  scope: Namespaced
  names:
    # plural name to be used in the URL: /apis/<group>/<version>/<plural>
    plural: pgs
    # singular name to be used as an alias on the CLI and for display
    singular: pg
    # kind is normally the CamelCased singular type. Your resource manifests use this.
    kind: PostgreSQL
    # shortNames allow shorter string to match your resource on the CLI
    shortNames:
      - pg
      - pgs
```



anynines

```yaml
apiVersion: ds.a9s.io/v1
kind: PostgreSQL
metadata:
  name: pg-1
spec:
  postgresVersion: "12.2"
  replicas: 3
```

# K8s CRDs

- CRD = Custom Resource Definition
- Introduce custom data structures to Kubernetes
- Kubernetes provides an endpoint for managing these objects
- Kubernetes provides persistency by storing them in its etcd.

# Controllers

```go
// [...]

// add adds a new Controller to mgr with r as the reconcile.Reconciler
func add(mgr manager.Manager, r reconcile.Reconciler) error {

  // Create a new controller
  // [...]

  // Watch for changes to primary resource Memcached
  err = c.Watch(&source.Kind{Type: &cachev1alpha1.Memcached{}}, &handler.EnqueueRequestForObject{})
  if err != nil {
    return err
  }

  // TODO(user): Modify this to be the types you create that are owned by the primary resource
  // Watch for changes to secondary resource Pods and requeue the owner Memcached
  err = c.Watch(&source.Kind{Type: &corev1.Pod{}}, &handler.EnqueueRequestForOwner{
    IsController: true,
    OwnerType:    &cachev1alpha1.Memcached{},
  })

  // [...]
}
```



anynines

```go
func (r *ReconcileMemcached) Reconcile(request reconcile.Request)
(reconcile.Result, error) {
  reqLogger := log.WithValues("Request.Namespace", request.Namespace,
"Request.Name", request.Name)
  reqLogger.Info("Reconciling Memcached")

  // Fetch the Memcached instance
  instance := &cachev1alpha1.Memcached{}

 err := r.client.Get(context.TODO(), request.NamespacedName, instance) //
Retrieve the object

  if err != nil {
    if errors.IsNotFound(err) {
      // Request object not found, could have been deleted after reconcile
request.
      // Owned objects are automatically garbage collected. For additional
cleanup logic use finalizers.
      // Return and don't requeue
      return reconcile.Result{}, nil
    }
    // Error reading the object - requeue the request.
    return reconcile.Result{}, err
  }

  // Define a new Pod object (similar to a YAML Spec)
  pod := newPodForCR(instance)

  if err := controllerutil.SetControllerReference(instance, pod, r.scheme);
err != nil {
    return reconcile.Result{}, err
  }
```

```go
  // Check if this Pod already exists
  found := &corev1.Pod{} // Empty Pod object

  err = r.client.Get(context.TODO(), types.NamespacedName{Name: pod.Name,
Namespace: pod.Namespace}, found)

  // If an error occurs and in particular the error is of the type NotFound then
we know the Pod doesn't exist.
  if err != nil && errors.IsNotFound(err) {
    reqLogger.Info("Creating a new Pod", "Pod.Namespace", pod.Namespace,
"Pod.Name", pod.Name)

    // Create the secondary objects ... in this case a single pod.
    err = r.client.Create(context.TODO(), pod)
    if err != nil {
      return reconcile.Result{}, err
    }

    // Pod created successfully - don't requeue
    return reconcile.Result{}, nil
  } else if err != nil {
    return reconcile.Result{}, err
  }

  // Pod already exists - don't requeue
  reqLogger.Info("Skip reconcile: Pod already exists", "Pod.Namespace",
found.Namespace, "Pod.Name", found.Name)
  return reconcile.Result{}, nil
}
```

# K8s Controllers

- Read custom resource object specifications
- Translate **primary resources** into a set of **secondary resources**.
- E.g. a **PostgreSQL** resource into a **Service** and a **StatefulSet**.
- Watches the primary spec for changes.
- Ensures secondary resources to comply to the desired state of the primary's spec.



anynines

# Common Pitfalls

- Underestimate complexity and effort
- Insufficient coverage of essential lifecycle operations
- Too little robustness, observability and predictability
- Applying automation that doesn’t fit the context

# What Organizations Want

- Expose lifecycle operations using Kubernetes Custom Resources (CRDs)
- **On-Demand Provisioning of Dedicated Service-Instances**
- Allow **configuration updates**
- Provide **monitoring** of health and status
- Infrastructure-agnostic
- Runs on different Kubernetes flavors.
- Authentication with dedicated user for each application accessing the DSI



anynines

- **Horizontal** 2n+1 DSI **scalability**: 1, 3, 5 ….
- Automatic failure detection and **fail-over**. Self-healing to recover degraded clustered service instances.
- Host-anti-affinity. Support for **multiple AZs**.
- **Vertical** DSI **scalability**: replace small pods with larger pods with even larger pods, …
- Provide **backup and restore** capabilities with the ability to create backup schedules.



anynines

- Stream backups to external object stores.
- Allow **choosing data service versions**.
- Documentation.
- **Encryption at rest and encryption at transit**.
- …

# The Long Life of a Service Instance

# Data Service Automation

Install Operator

Update Operator

Update Operator Configuration

Delete Operator

Create Service Instance

Delete Service Instance

Add log sink

Add metrics sink

Add alerting rule

Vertical scale-up

Horizontal scale-out

Patch-level upgrade

Minor upgrade

Major upgrade

Change configuration setting

**Create service-binding**

**Delete service-binding**

Create a backup schedule

Create backup

Restore backup

Enable (Postgresql) extension

Disable (Postgresql) extension

Network Delay Fluctuation

Network Bandwidth Fluctuation

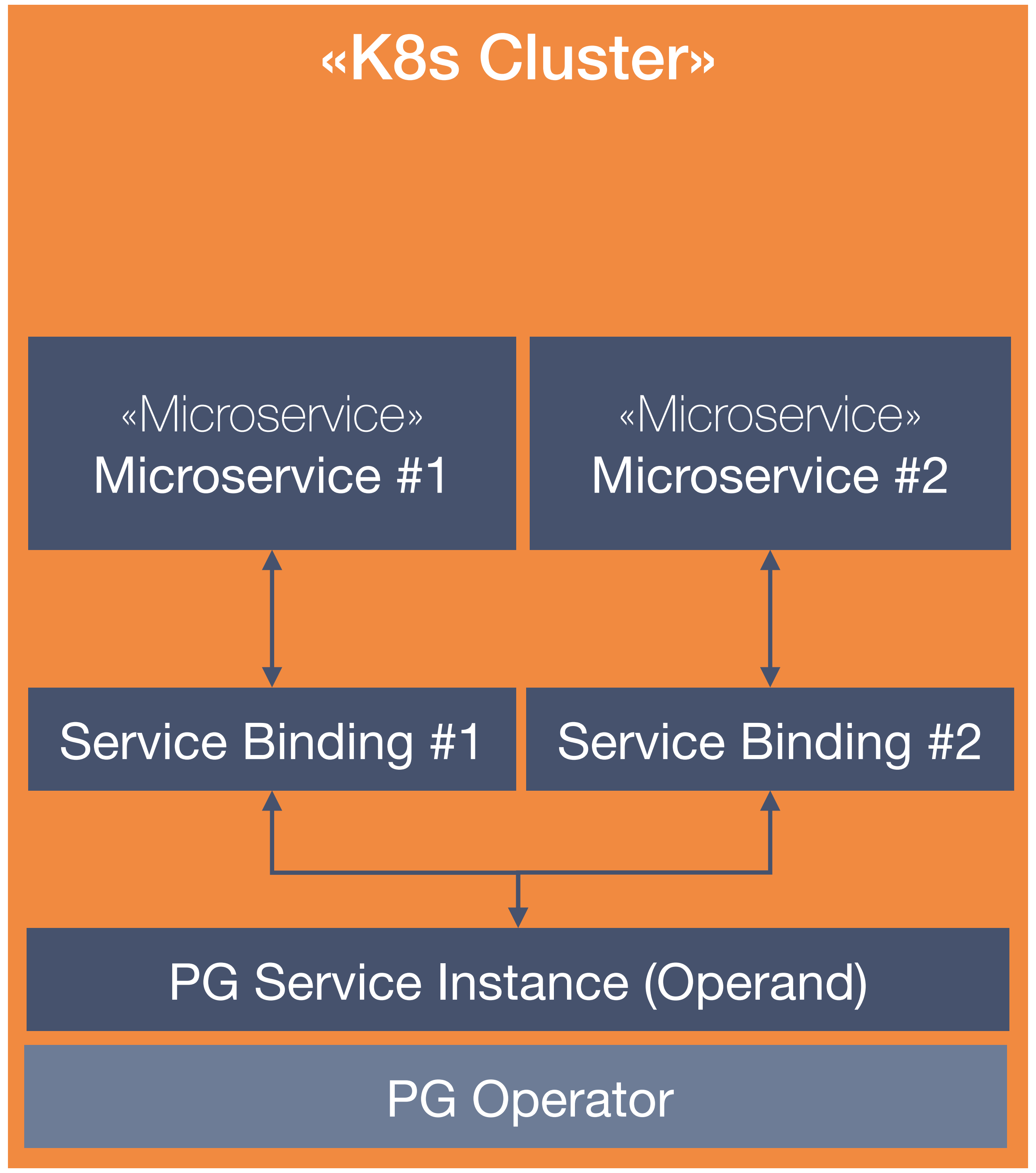
Network Partitioning

Availability Zone Failure

Kubernetes Node Failure

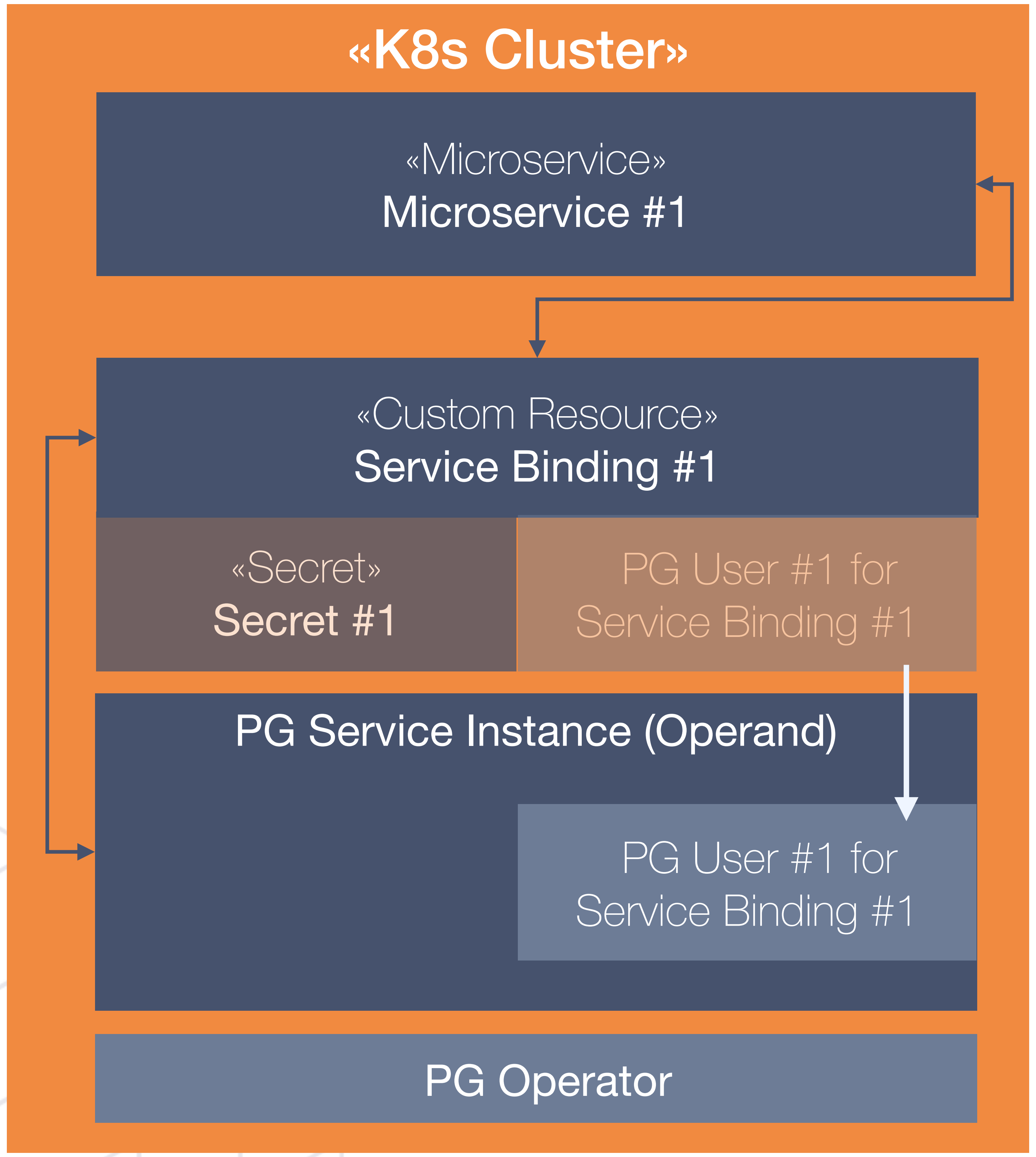
# Service Bindings

# Service Bindings




A Service Binding represents the connection between an app and a data service instance.

# Service Bindings




A Service Binding comprises a Kubernetes Secret as well as a user in the managed data service, e.g. a PostgreSQL user.

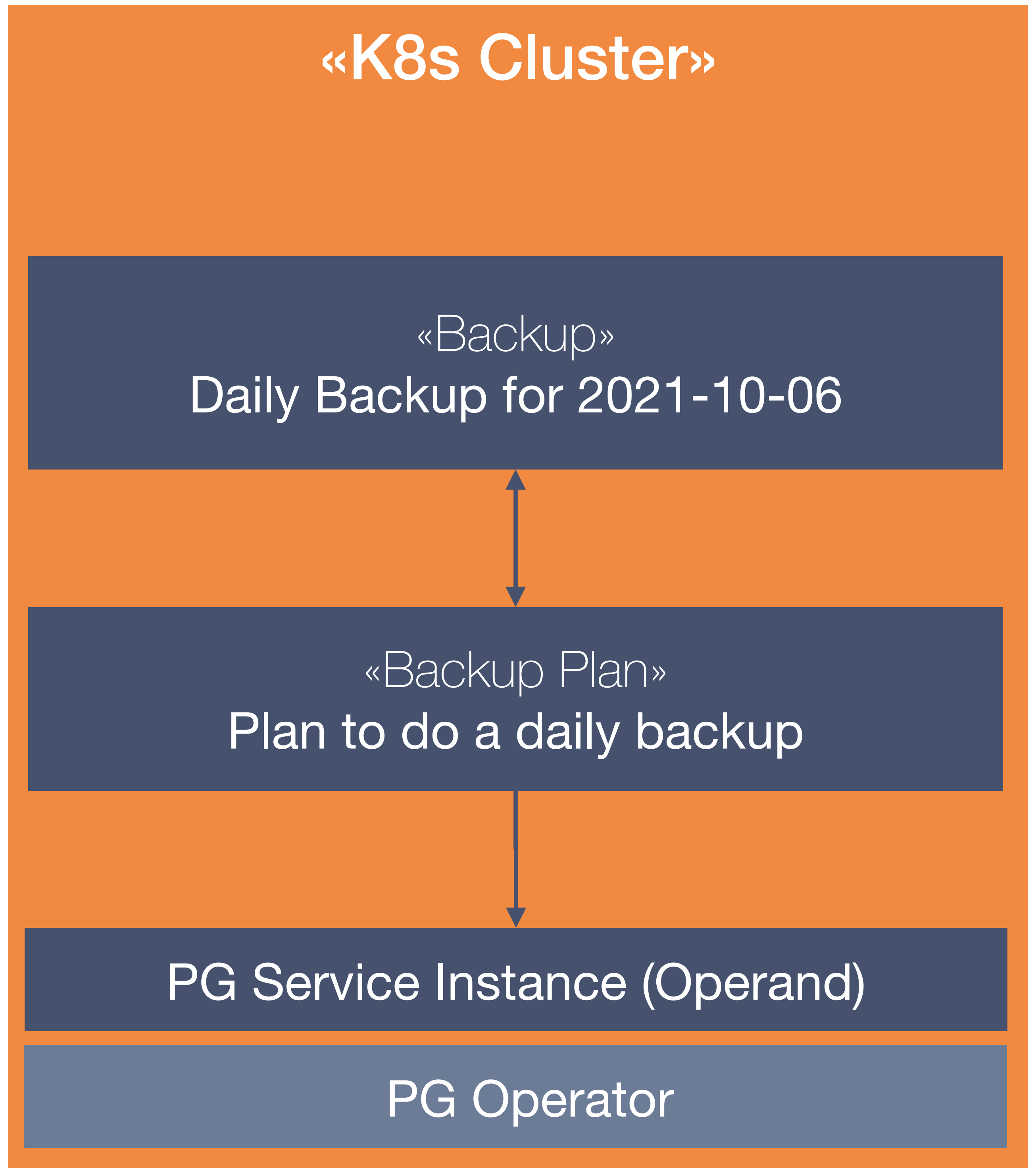
Both user and secret are unique to a particular Service Binding.



anynines

# Backups

# Backups




A single K8s cluster
with multiple service-instances
managed by a multiple Operators.

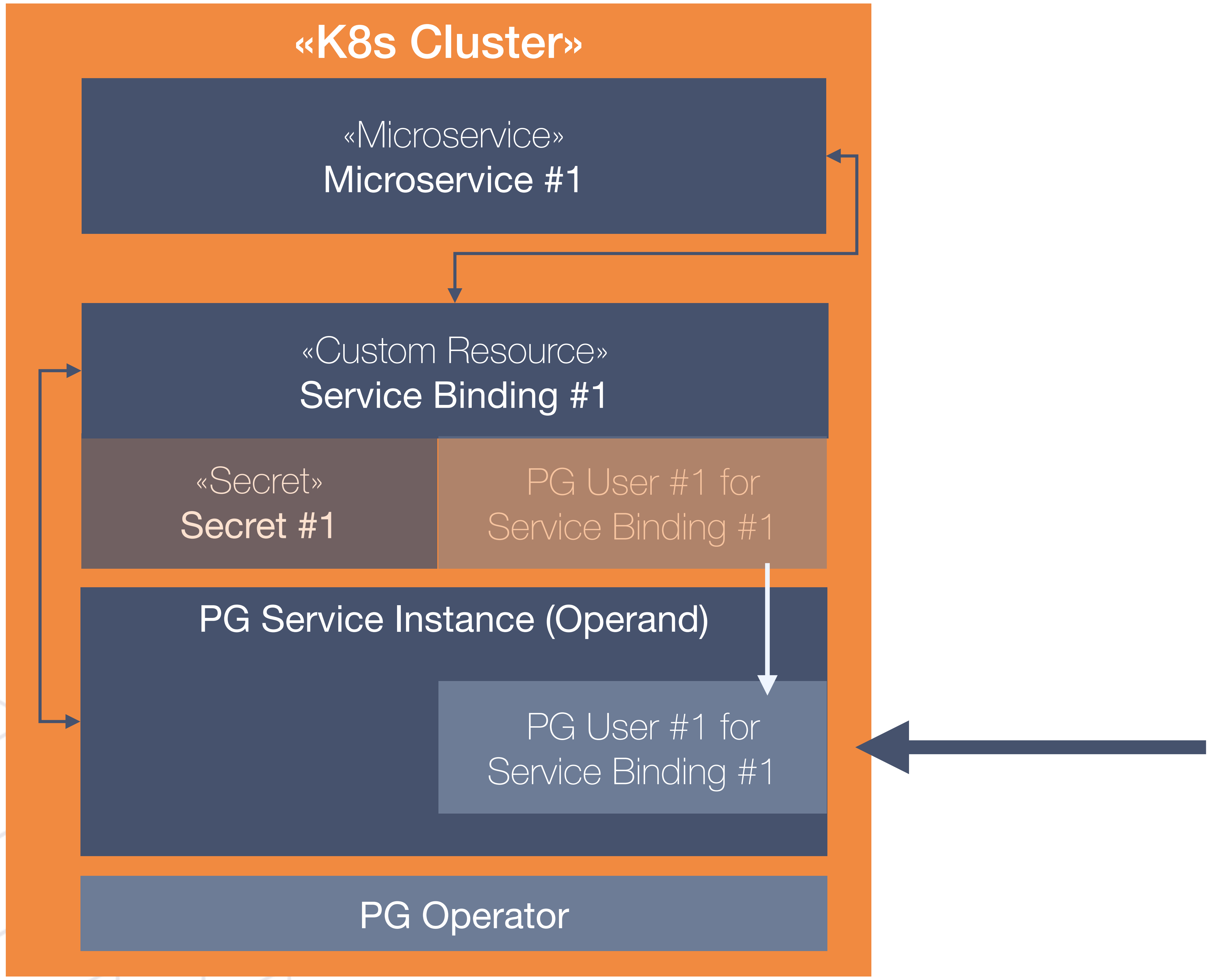
# Technology

# Writing Controllers

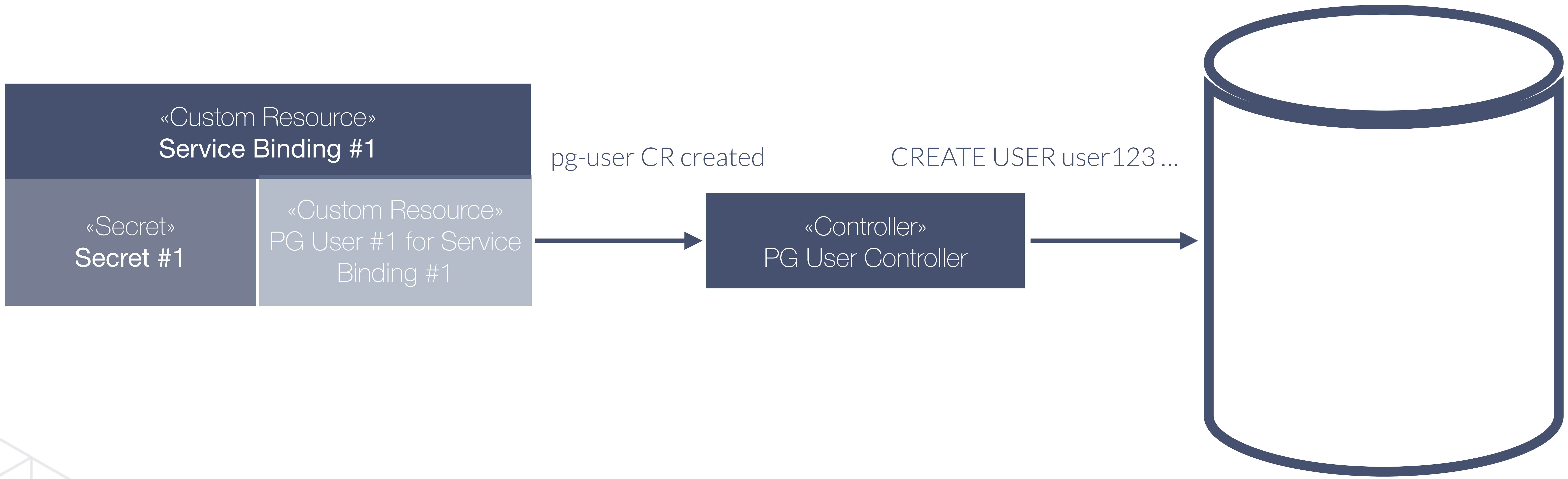
# Reconciling External Resources



anynines




How to reconcile the postgres user?



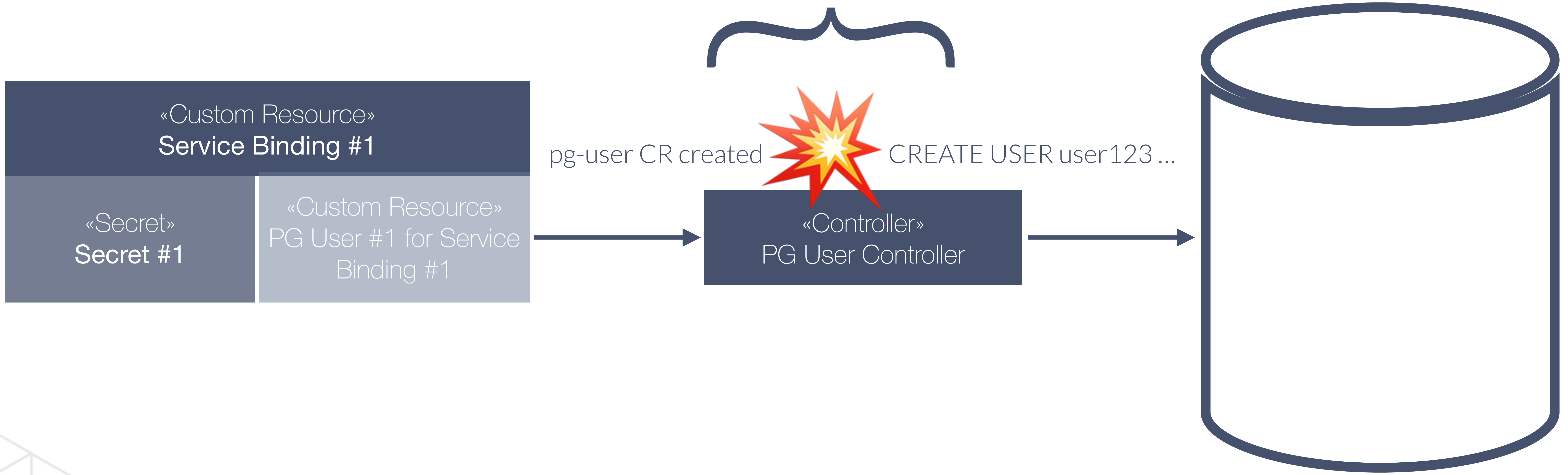
«Custom Resource»
Service Binding #1
«Secret»
Secret #1
«Custom Resource»
PG User #1 for Service Binding #1
pg-user CR created
«Controller»
PG User Controller
CREATE USER user123 ...

# CREATE USER



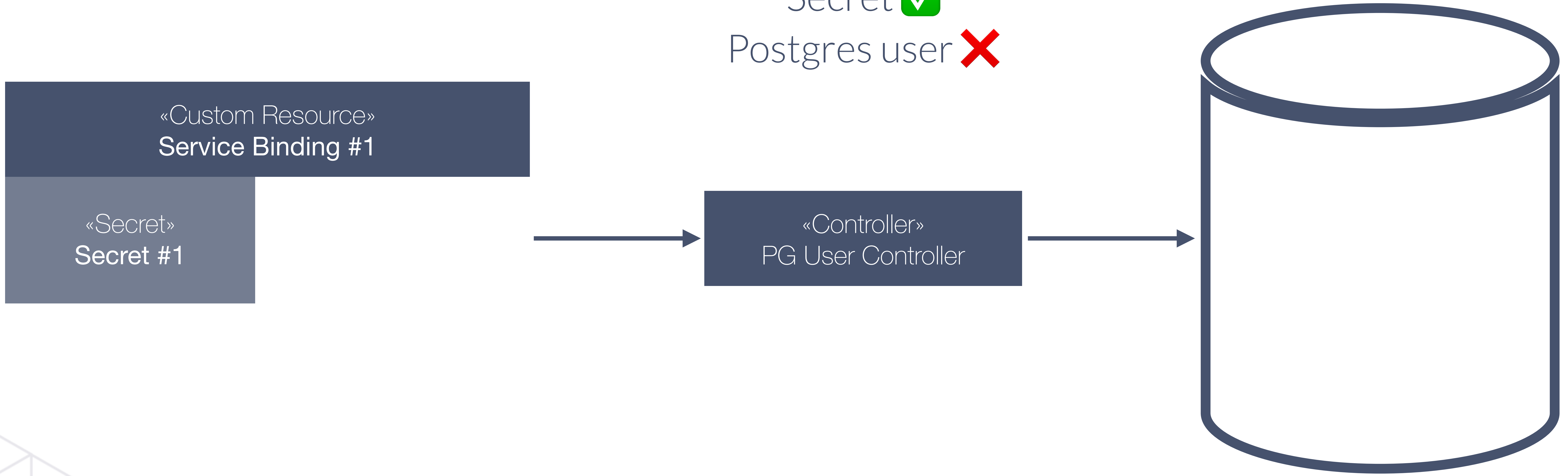
anynines

Careful ⚠️ This is not a transaction.
Atomicity is not guaranteed.



## Inconsistent state.

Secret ✅
Postgres user ❌



Be prepared to
re-reconcile by making
actions idempotent.

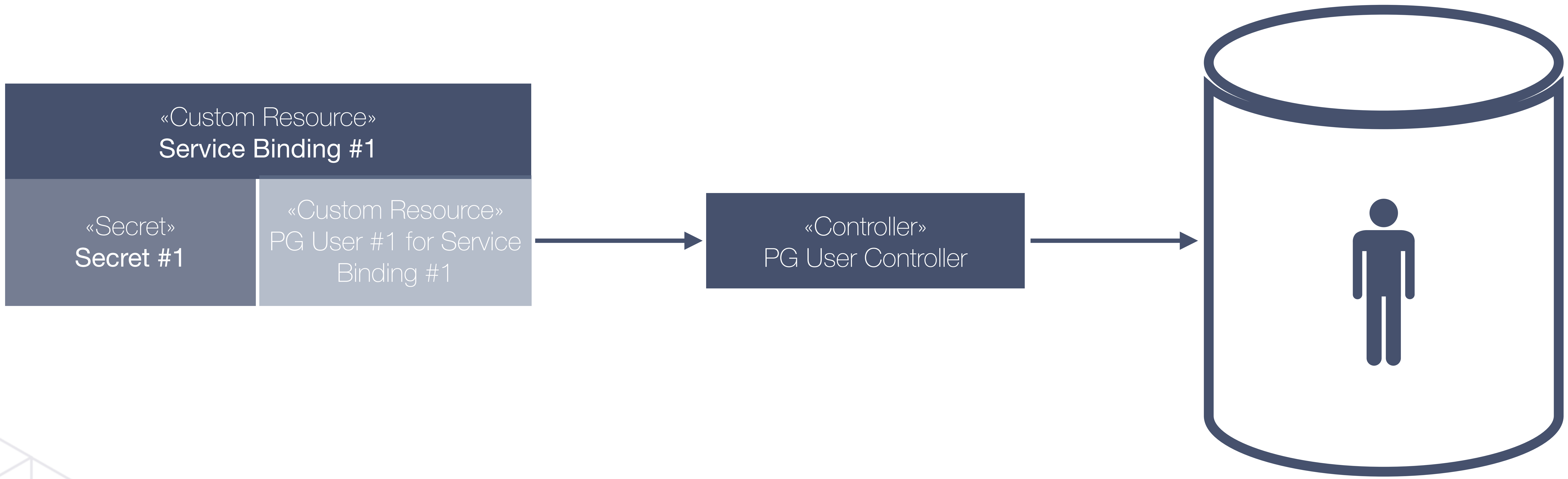


anynines

# CREATE USER
# IF NOT EXISTS



anynines



«Custom Resource»
Service Binding #1
«Secret»
Secret #1
«Custom Resource»
PG User #1 for Service Binding #1
«Controller»
PG User Controller

# Summary

a9s Data Services

a9s Elasticsearch

a9s LogMe

# Questions?
# @anynines
# @fischerjulian

a9s Redis

a9s MySQL

a9s MongoDB

a9s PostgreSQL

a9s RabbitMQ

# Thank You!


a9s Data Services
a9s Elasticsearch
a9s LogMe
a9s MongoDB
a9s PostgreSQL
a9s RabbitMQ