Running Postgres-as-a-Service In Kubernetes

@LukasFittl

@LukasFittl

Why (not) run Postgres in Kubernetes?

Consistency - Kubernetes manages all workloads, including databases

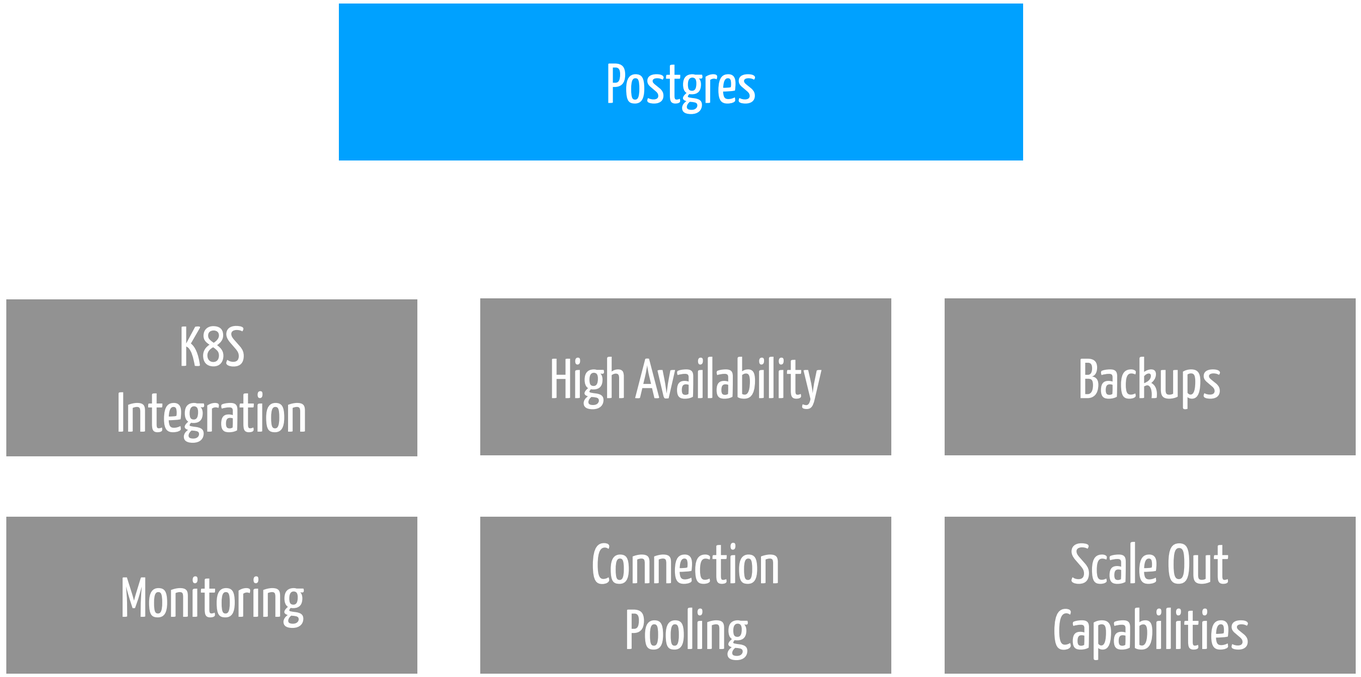
Better Portability - Consistent deployment experience across clouds, instead of relying on cloud-specific APIs

Low Latency - Co-locating Compute and Database, allows certain workloads to perform better

High Effort - Running anything in Kubernetes is complex, and databases are worse

How to deploy Postgres in Kubernetes

Postgres


Postgres
K8S Integration
High Availability
Backups
Monitoring
Connection Pooling
Scale Out Capabilities

Zalando Postgres Operator

zalando / postgres-operator

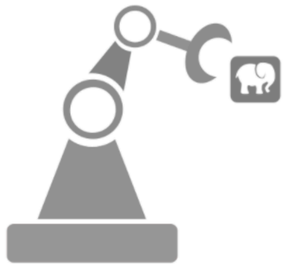
Watch 44 | Unstar 1.1k | Fork 265

Code | Issues 139 | Pull requests 37 | Actions | Security 0 | Insights

83 lines (66 sloc) | 3.92 KB | Raw | Blame | History

Postgres Operator

build passing | coverage 20% | go report A+ | godoc reference | GolangCI A+



Crunchy Data PostgreSQL Operator

CrunchyData / **postgres-operator**

Watch 44 | Star 1.1k | Fork 185

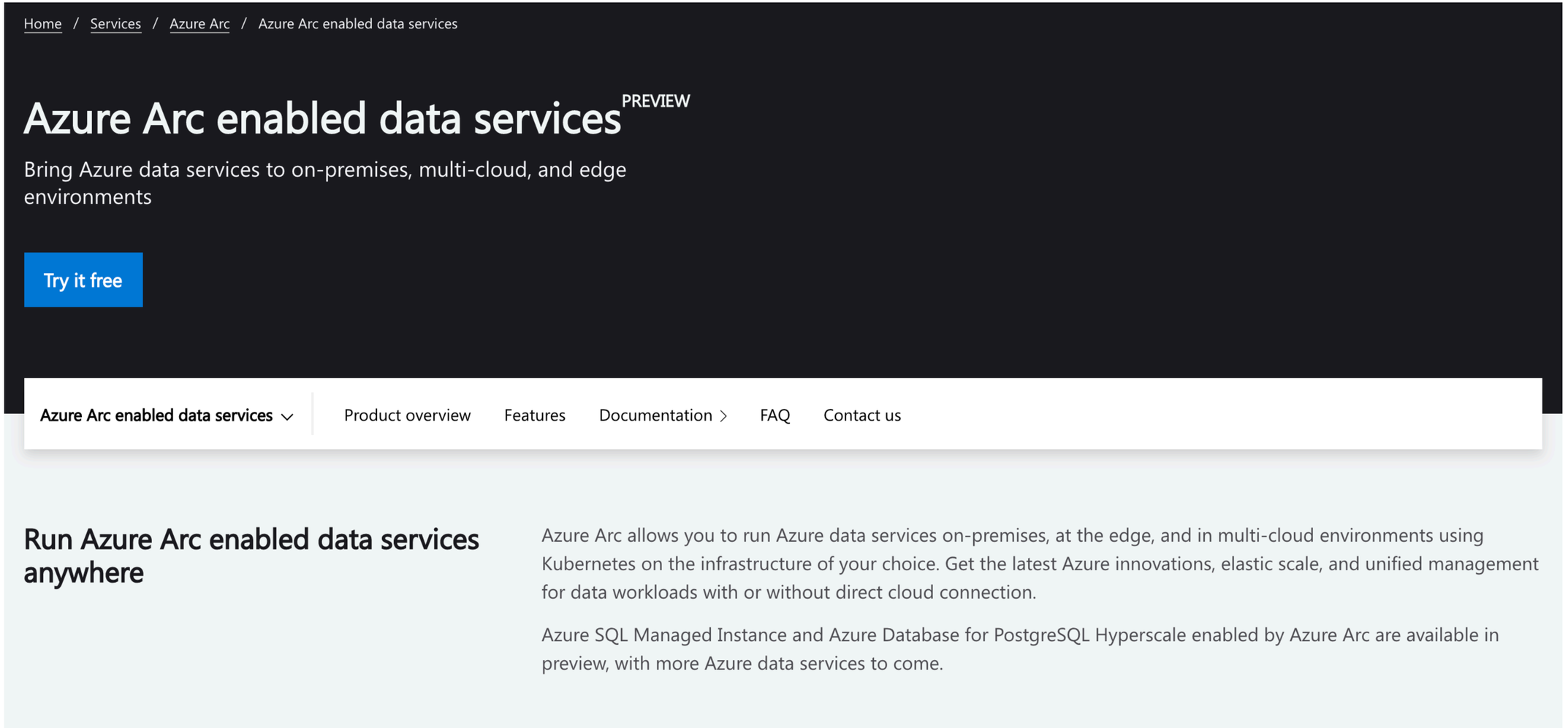
Code | Issues 47 | Pull requests 6 | Actions | Projects 0 | Wiki | Security 0 | Insights

204 lines (122 sloc) | 11.6 KB | Raw | Blame | History

Crunchy Data PostgreSQL Operator

go report A

PostgreSQL Hyperscale - Azure Arc



Postgres

K8S Integration

High Availability

Backups

Monitoring

Connection Pooling

Scaling Capabilities

K8S Integration

CLIs, Operators & API Servers

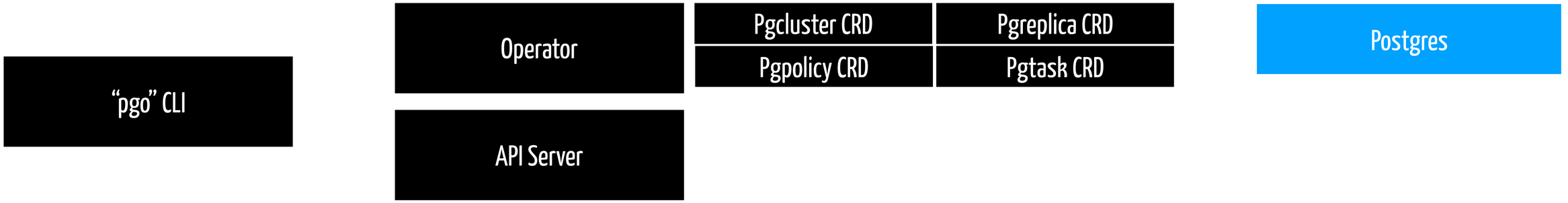
Namespace handling

Storage

Zalando Postgres Operator

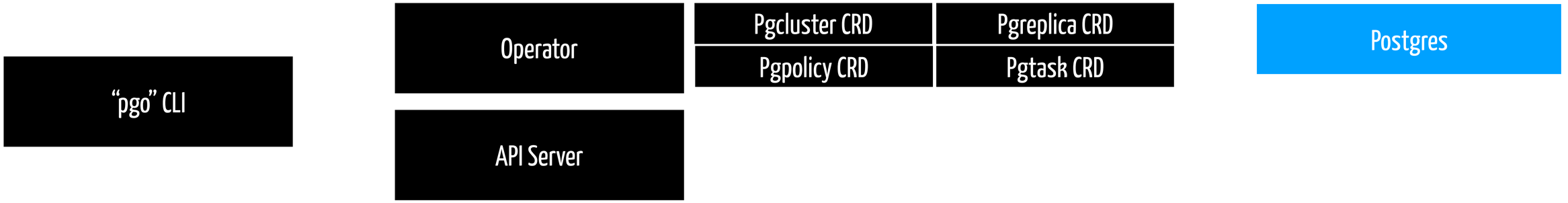
Zalando Postgres Operator

Crunchy Data PostgreSQL Operator



Zalando Postgres Operator

Crunchy Data PostgreSQL Operator



PostgreSQL Hyperscale - Azure Arc

Zalando Postgres Operator

```
kind: postgresql
metadata:
  ...
spec:
  databases:
    foo: zalando
  numberOfInstances: 2
  postgresql:
    version: "12"
  preparedDatabases:
    bar: {}
  teamId: acid
  users:
    foo_user: []
    zalando:
    - superuser
    - createdb
  volume:
    size: 1Gi
```

Crunchy Data PostgreSQL Operator

```
kind: Pgcluster
metadata:
  ...
spec:
  ArchiveStorage:
    ...
  BackrestStorage:
    ...
  PrimaryStorage:
    ...
  ReplicaStorage:
    ...
  WALStorage:
    ...
  ...
  clustername: hippo
  database: hippo
  exporterport: "9187"
  limits: {}
  name: hippo
  namespace: pgo
  podAntiAffinity:
    default: preferred
  port: "5432"
  primaryhost: hippo
  replicas: "0"
  resources:
    memory: 128Mi
  rootsecretname: hippo-postgres-secret
  syncReplication: null
  tablespaceMounts: {}
  tlsOnly: false
```

PostgreSQL Hyperscale - Azure Arc

```
kind: DatabaseService
metadata:
  ...
spec:
  docker:
    ...
  engine:
    type: Postgres
    version: 12
  monitoring:
    ...
  scale:
    shards: 2
  scheduling:
    default:
      resources:
        requests:
          memory: 256Mi
  service:
    port: 5432
    type: NodePort
  storage:
    volumeSize: 1Gi
```

Namespace handling

Crunchy Data' Postgres Operator **Namespace Modes**:

dynamic:
Operator can create, delete, update any namespaces and manage RBAC.
Operator requires ClusterRole privilege.

readonly:
Namespaces need to be pre-created and RBAC pre-configured.
Operator requires ClusterRole privilege.

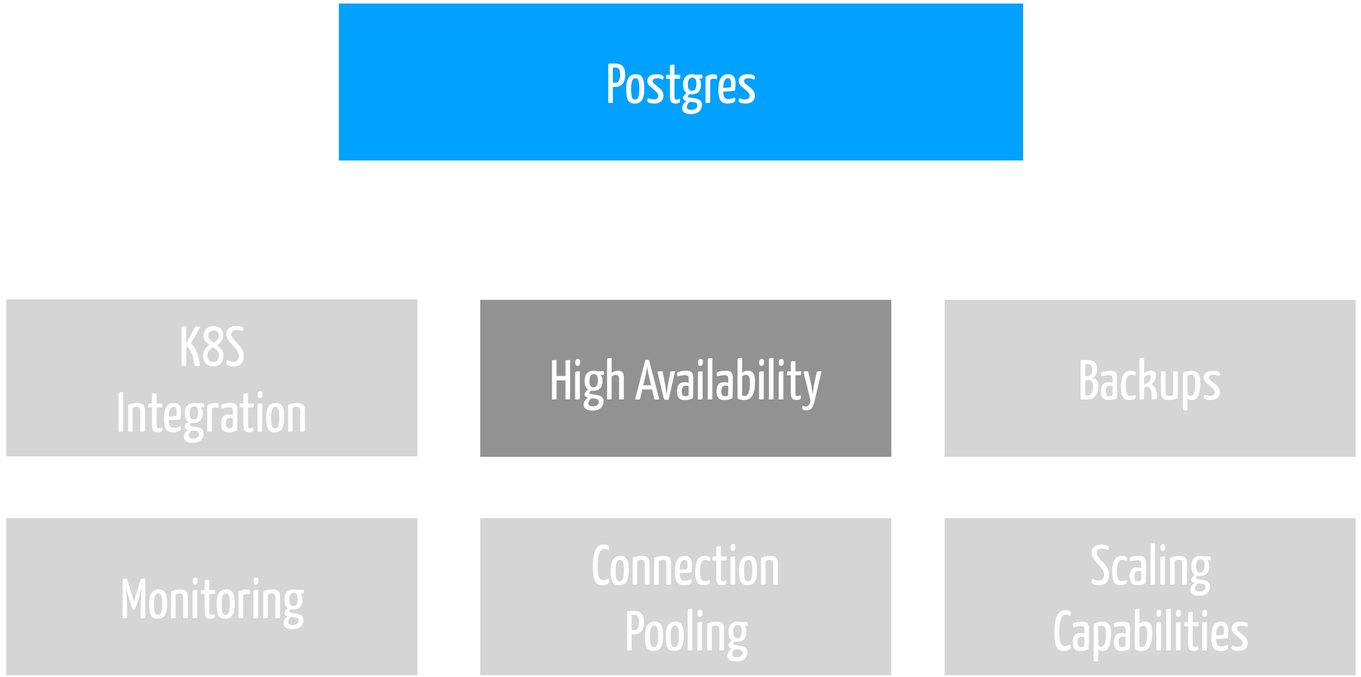
disabled:
Deploy to single namespace, no ClusterRole privilege required.

Storage

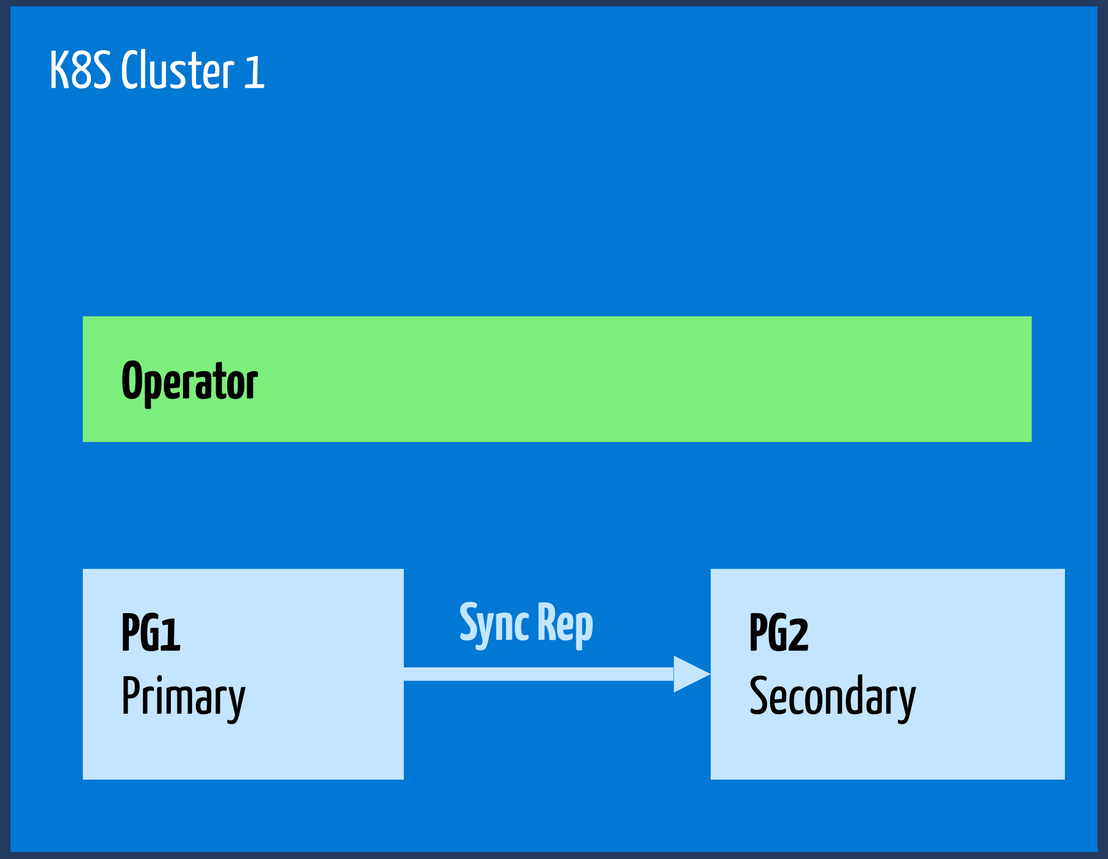
Postgres | Persistent Volume | Persistent Volume Claim | Network Storage

Generally, expect Persistent Volume Claims (PVCs) to be utilized for the database storage.

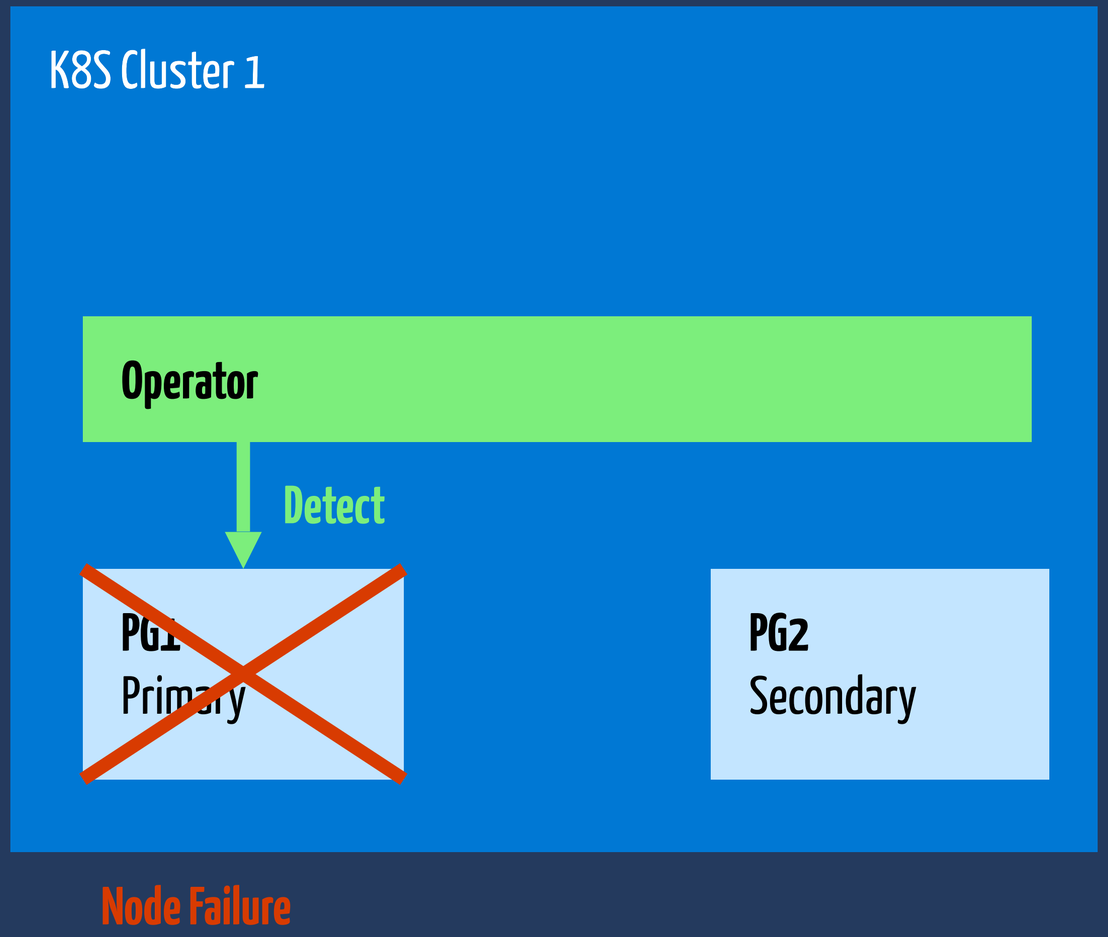
Crunchy Data PostgreSQL Operator also supports table spaces, to utilize different storage types within the same database server (be careful when using it)


Postgres
K8S Integration
High Availability
Backups
Monitoring
Connection Pooling
Scaling Capabilities

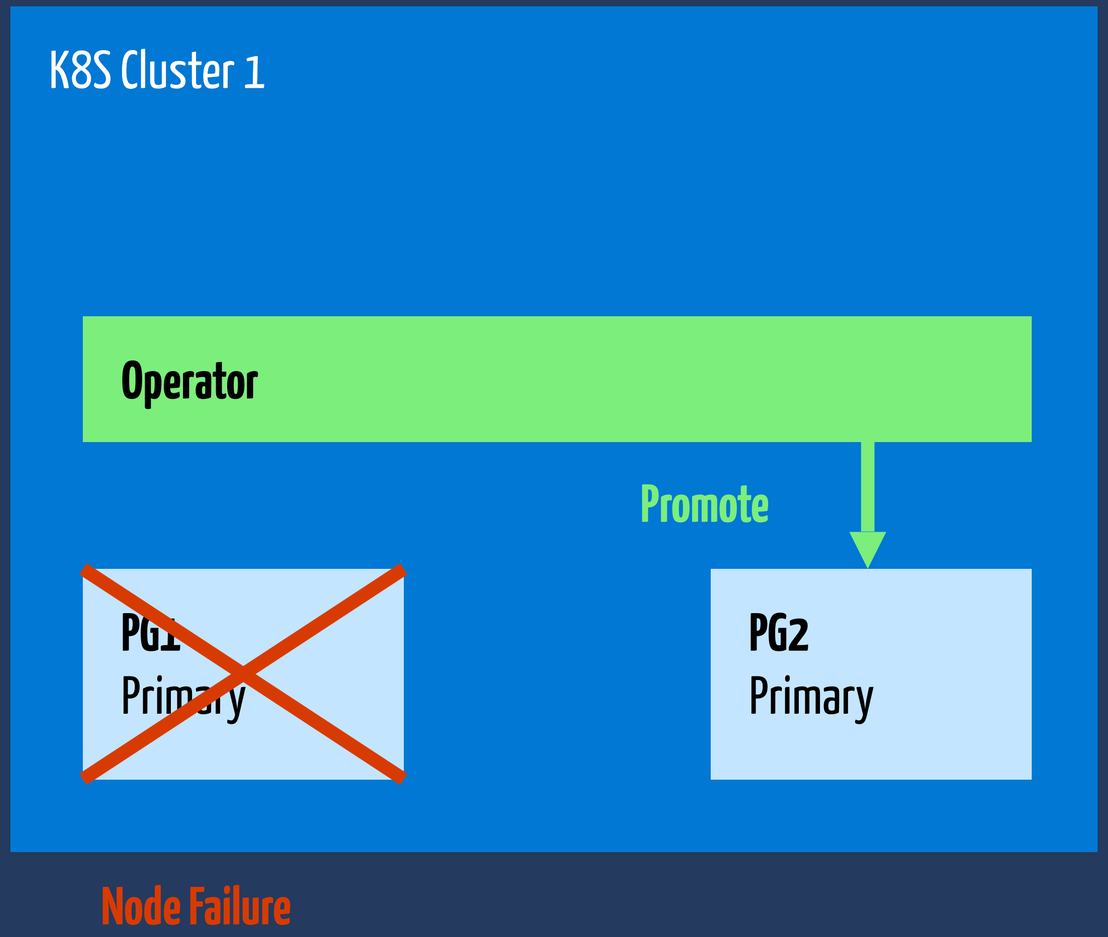
Scenario 1: Automated Failover within a K8S cluster



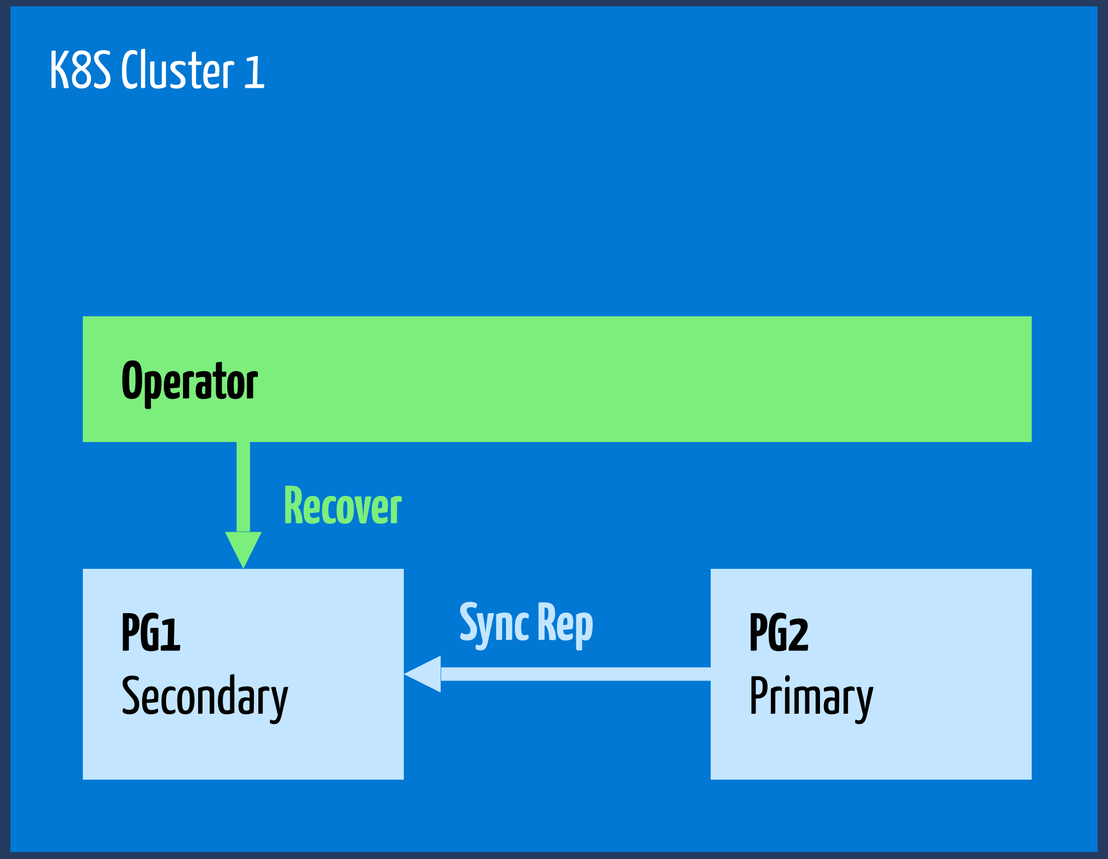
Scenario 1: Automated Failover within a K8S cluster



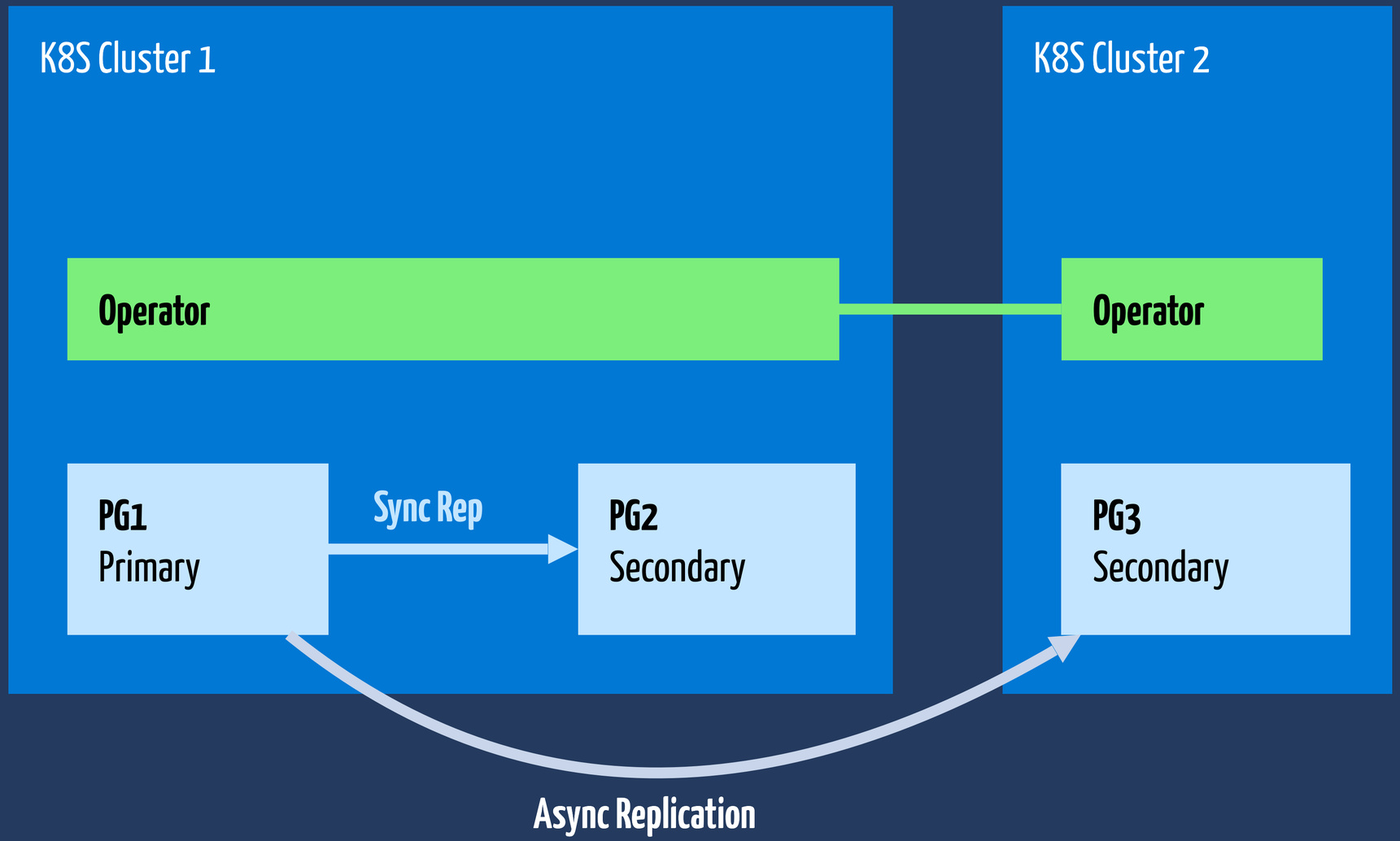
Scenario 1: Automated Failover within a K8S cluster



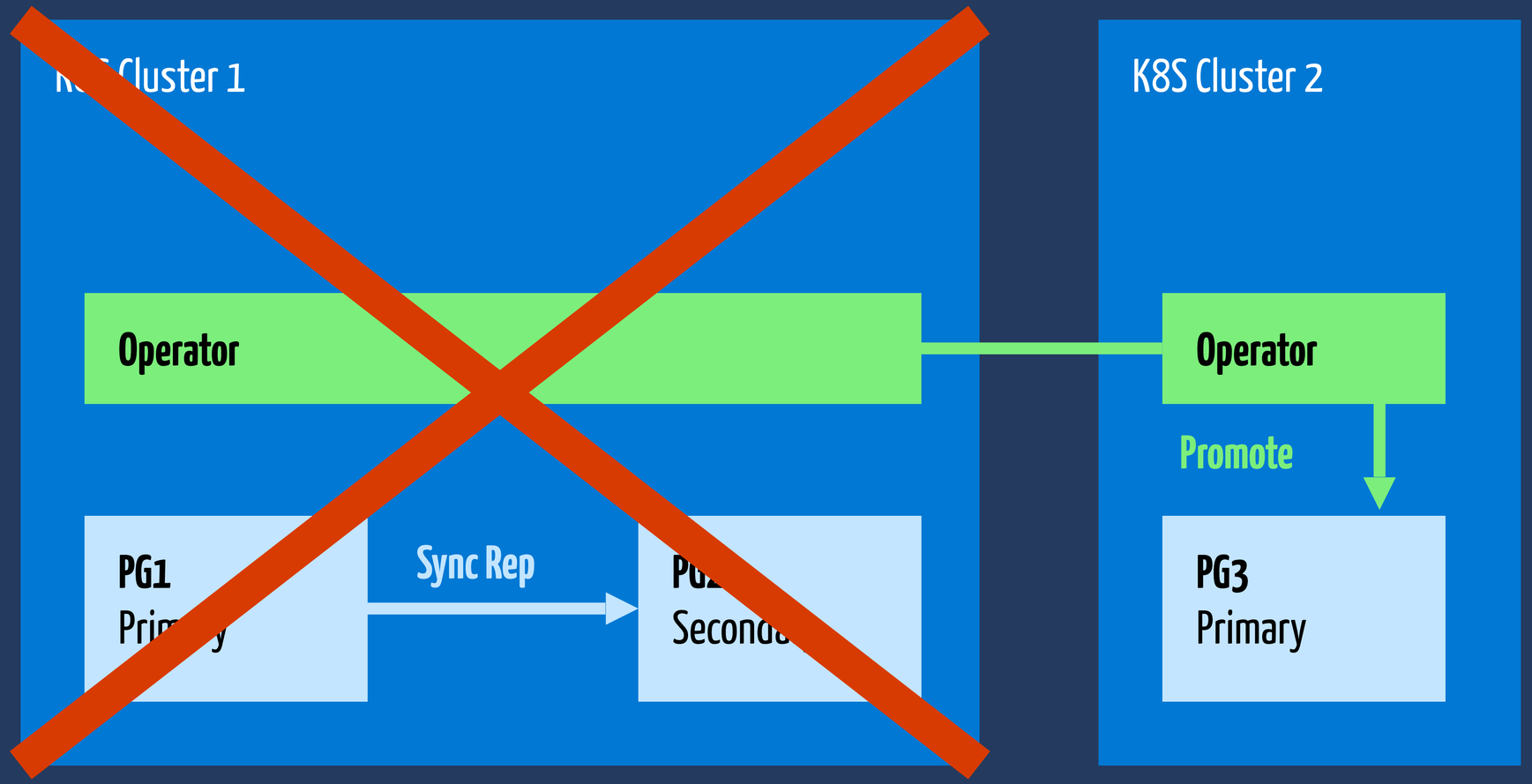
Scenario 1: Automated Failover within a K8S cluster



Scenario 2: Disaster Recovery to another K8S cluster



Scenario 2: Disaster Recovery to another K8S cluster



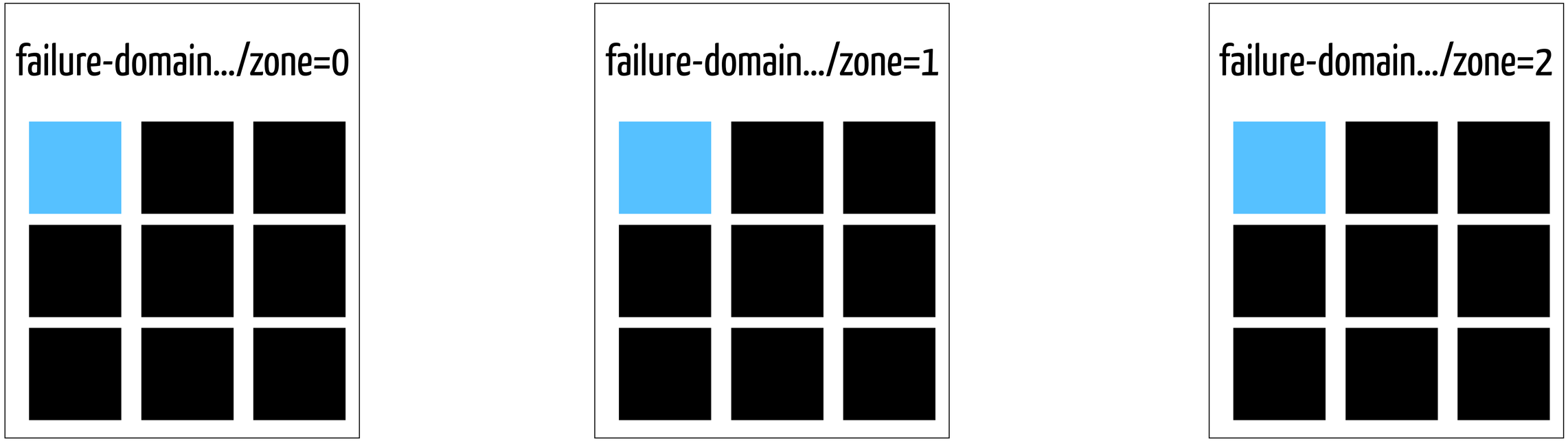
Large-Scale Data Center Failure

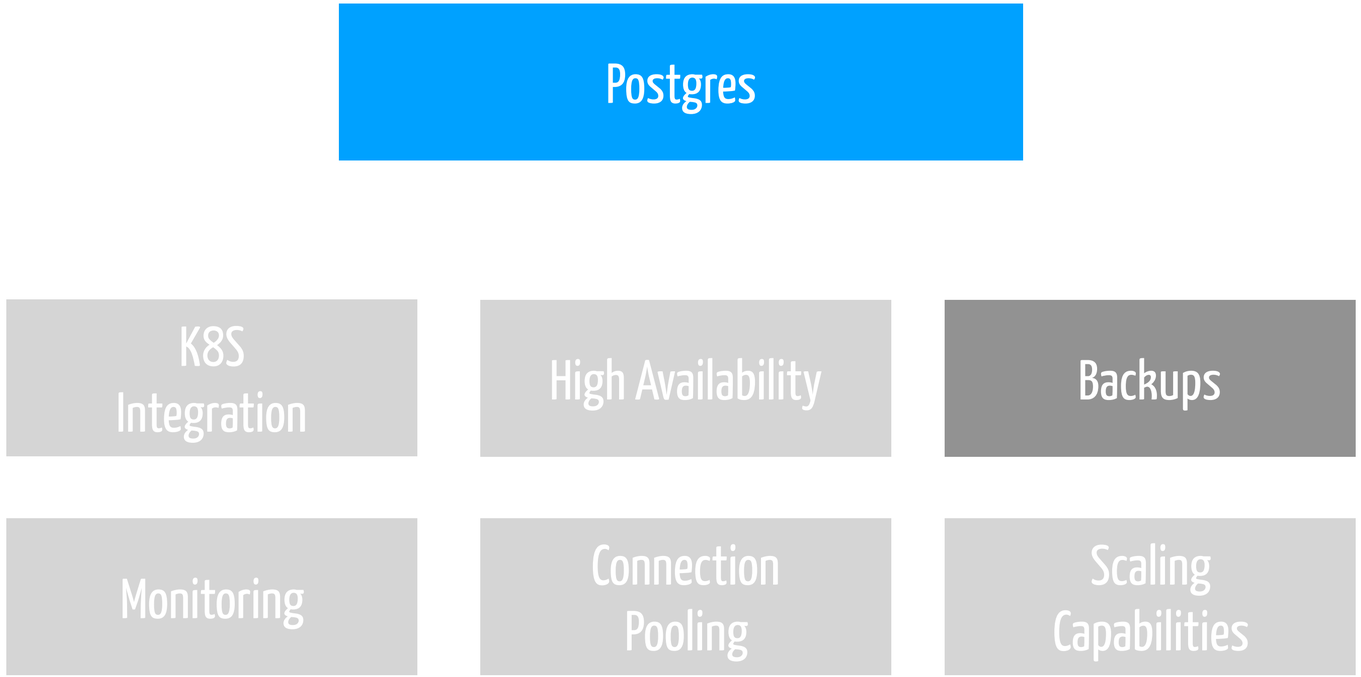
High Availability

	HA within Same K8S Cluster	HA across K8S Clusters
Zalando Postgres Operator	Built-In	Manual
Crunchy Data PostgreSQL Operator	Built-In	Manual
PostgreSQL Hyperscale - Azure Arc	Built-In	Manual

Pod Anti-Affinity

label: failure-domain.beta.kubernetes.io/region=westus2




Postgres
K8S Integration
High Availability
Backups
Monitoring
Connection Pooling
Scaling Capabilities

Backups

	Local Volume Backups	Point-in-time-Restore	Offsite Backups
Zalando Postgres Operator	n/a	Built-in (wal-e)	Built-in (wal-e)
Crunchy Data PostgreSQL Operator	Built-in (pgBackRest)	Built-in	Built-in (Amazon S3)
PostgreSQL Hyperscale - Azure Arc	Built-in	Built-in	Built-in (K8S Volume Mount)

Postgres

K8S Integration

High Availability

Backups

Monitoring

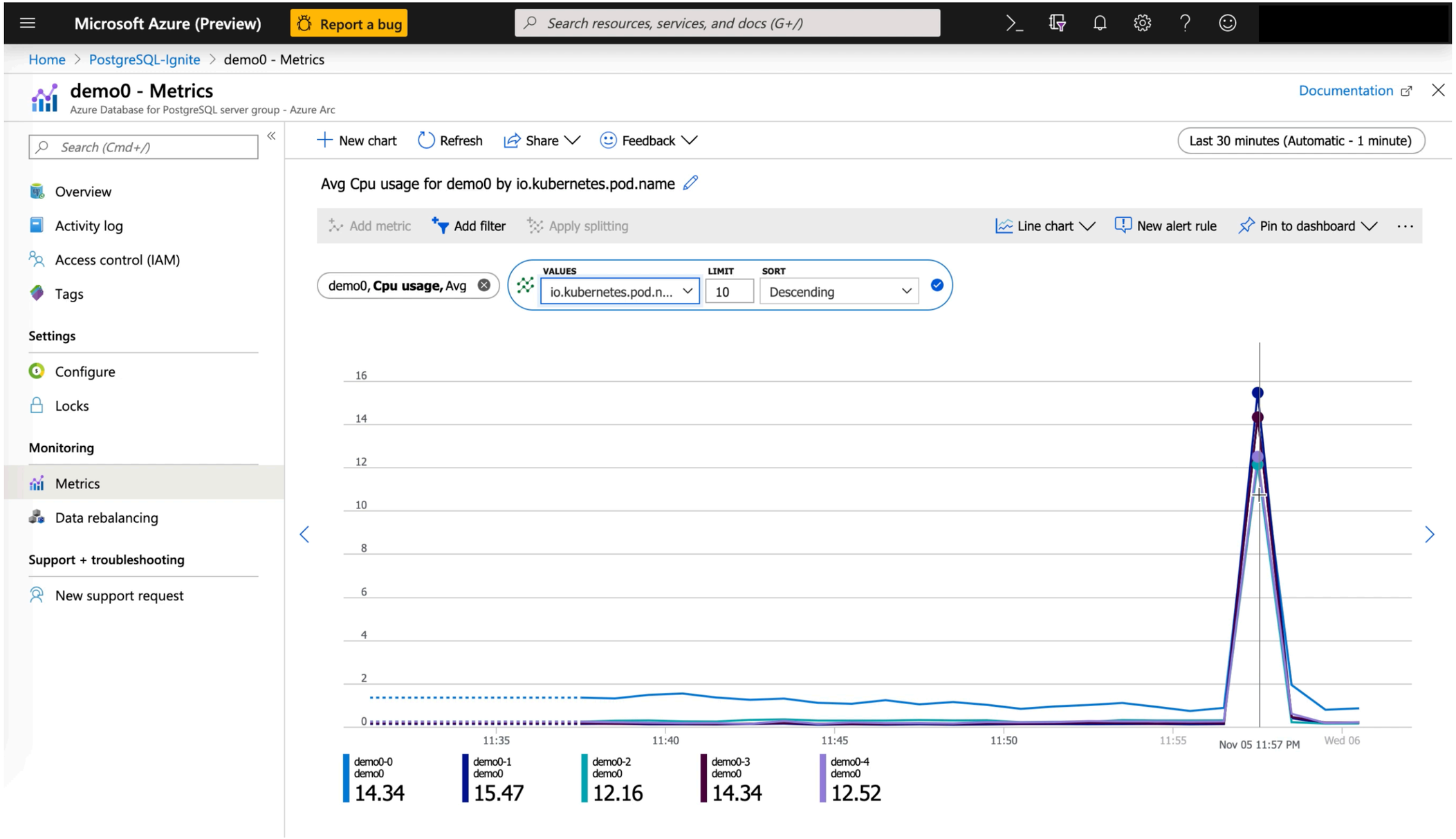
Connection Pooling

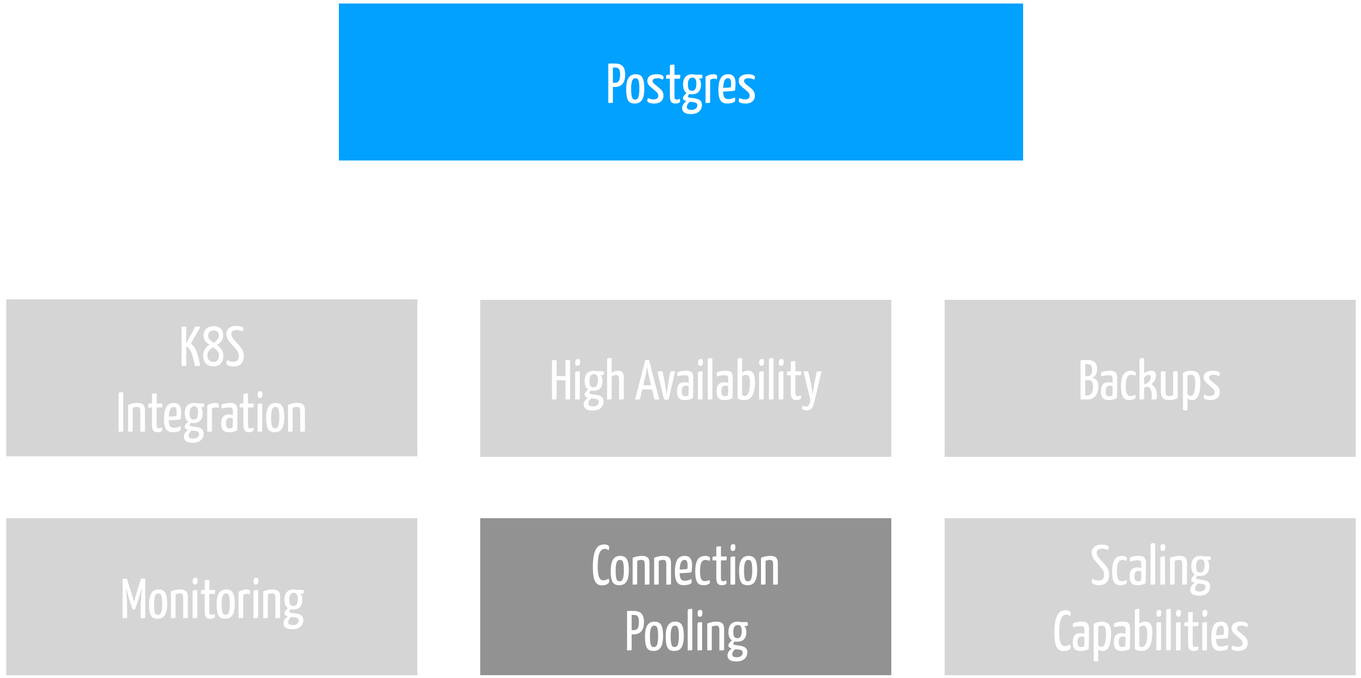
Scaling Capabilities

Monitoring

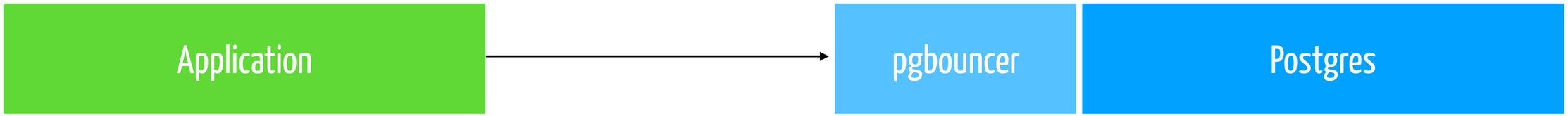
	Metrics	Logs
Zalando Postgres Operator	not built in	not built in
Crunchy Data PostgreSQL Operator	Grafana	Built-in pgbadger
PostgreSQL Hyperscale - Azure Arc	Grafana + Azure Monitor	Kibana + Azure Log Analytics

PostgreSQL Hyperscale - Azure Arc




Postgres
K8S
Integration
High Availability
Backups
Monitoring
Connection
Pooling
Scaling
Capabilities

Connection Pooling



pgbouncer is important for **idle connection scaling** in Postgres

Idle connection in Postgres: 5-10MB
Idle connection in pgbouncer: <1MB

Connection Pooling

	Pgbouncer
Zalando Postgres Operator	Built-in
Crunchy Data PostgreSQL Operator	Built-in
PostgreSQL Hyperscale - Azure Arc	Planned

Postgres

K8S Integration

High Availability

Backups

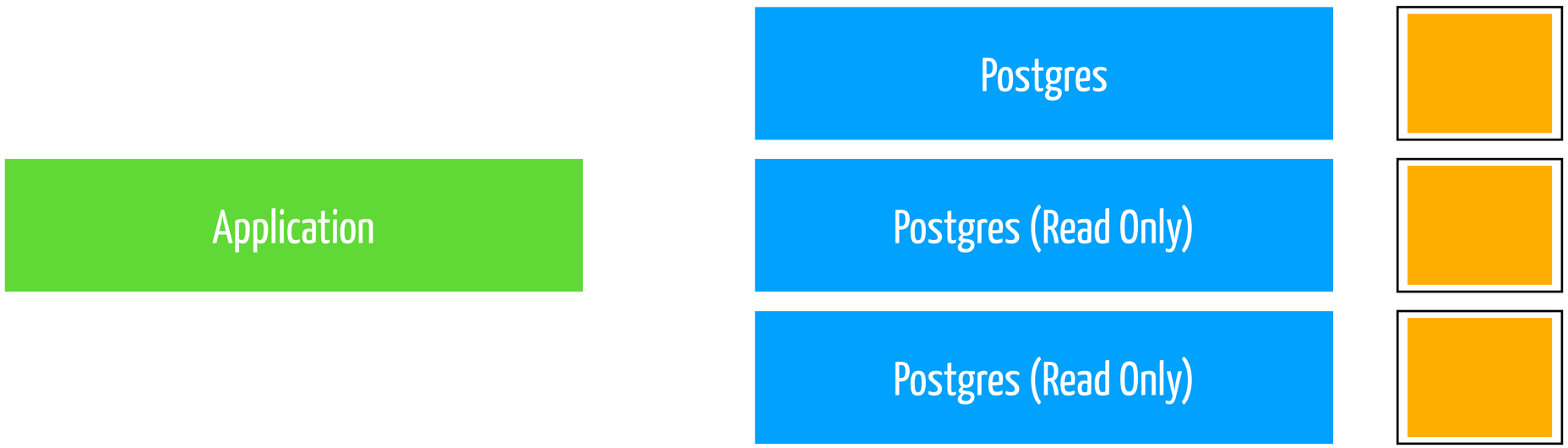
Monitoring

Connection Pooling

Scaling Capabilities

Scaling Capabilities

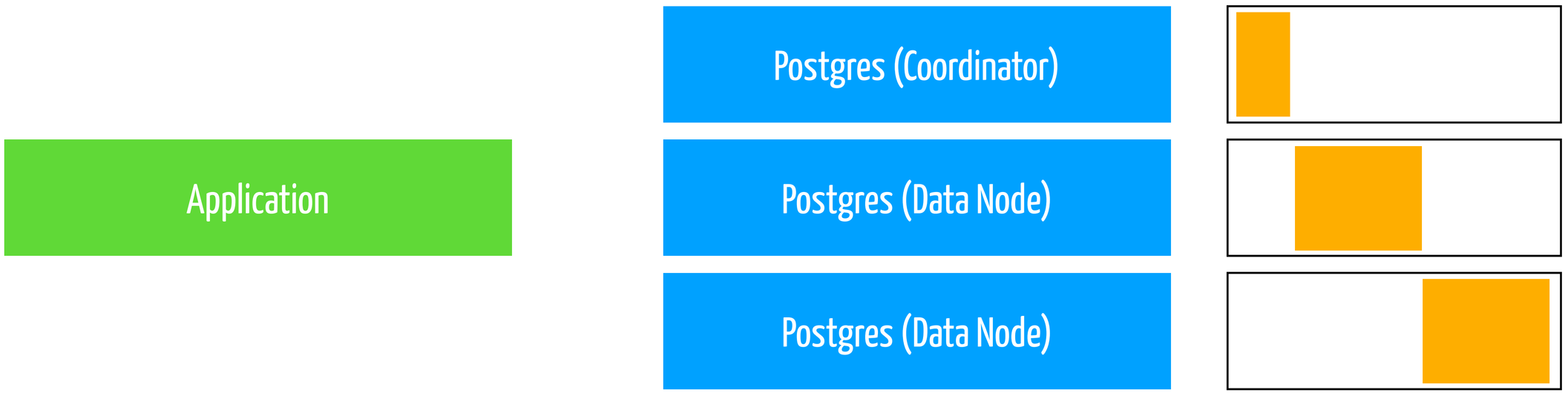
Read Replicas:



Help you scale the read performance; Max data size = max storage size per node

Scaling Capabilities

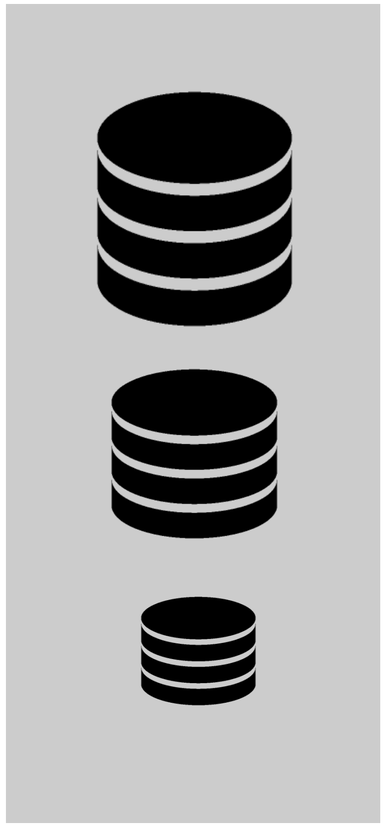
Hyperscale (Citus):



Scales both read and write performance; Max data size = # of data nodes * storage size per node

Scaling Up

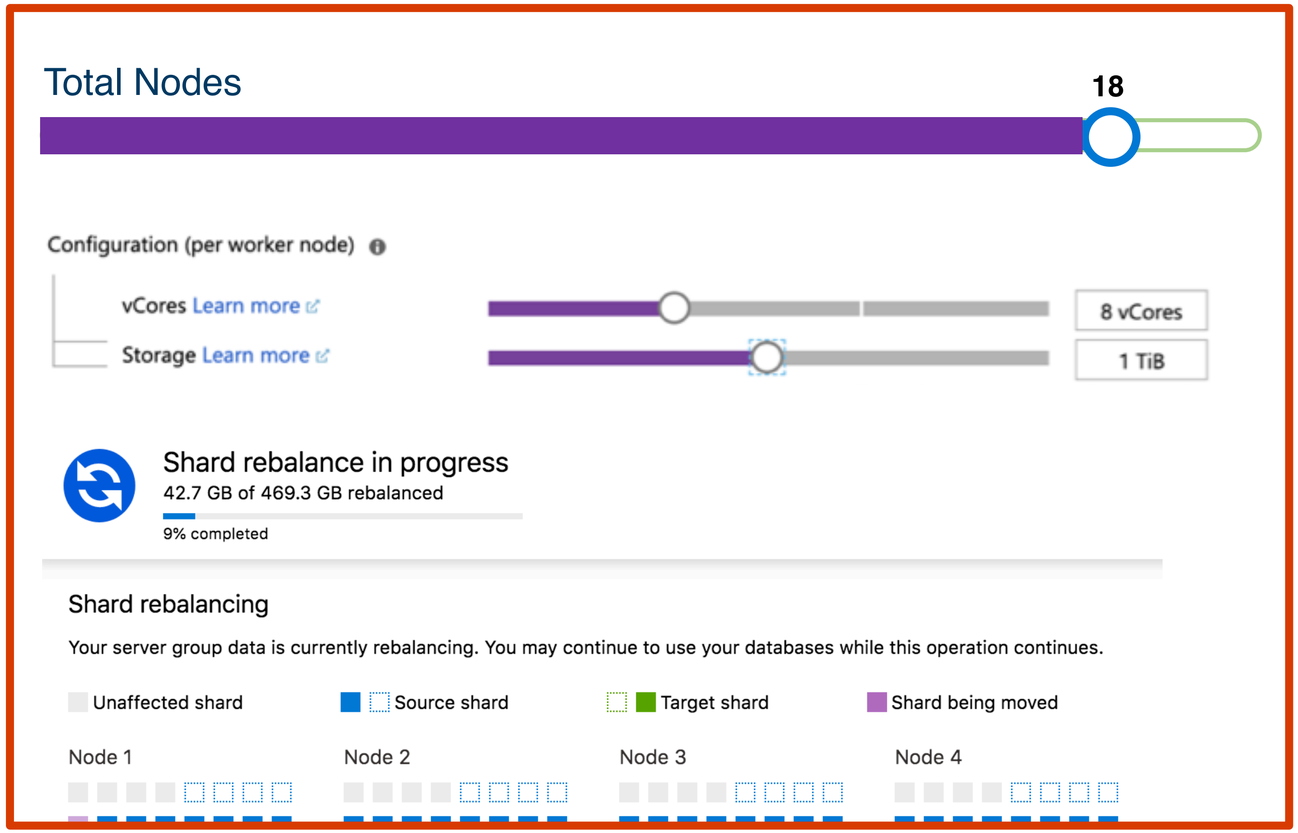
Block growth on 1
(monolithic) database



VS.

Scaling Out

Grow to 100's of database nodes,
without re-architecting your application



Demo

Deploying & Scaling out with
PostgreSQL Hyperscale - Azure Arc

Thank you!

lukas@fittl.com

@LukasFittl