Eureka!

8 Developer Tricks for Running ClickHouse on Kubernetes

Robert Hodges
Altinity Engineering

Let's make some introductions

Robert Hodges

Database geek with 30+ years on DBMS. Kubernaut since 2018. Day job: Altinity CEO

Altinity Engineering

Database geeks with centuries of experience in DBMS and applications

ClickHouse support and services including Altinity.Cloud
Authors of Altinity Kubernetes Operator for ClickHouse
and other open source projects

ClickHouse is a real-time analytic database

Understands SQL

Runs on bare metal to cloud

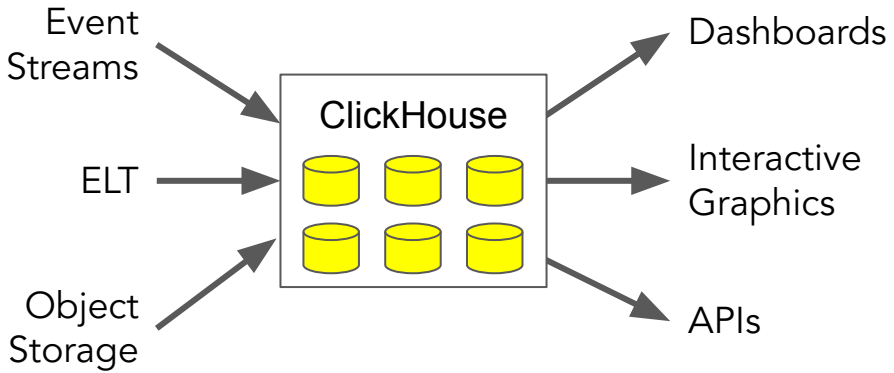
Shared nothing architecture

Stores data in columns

Parallel and vectorized execution

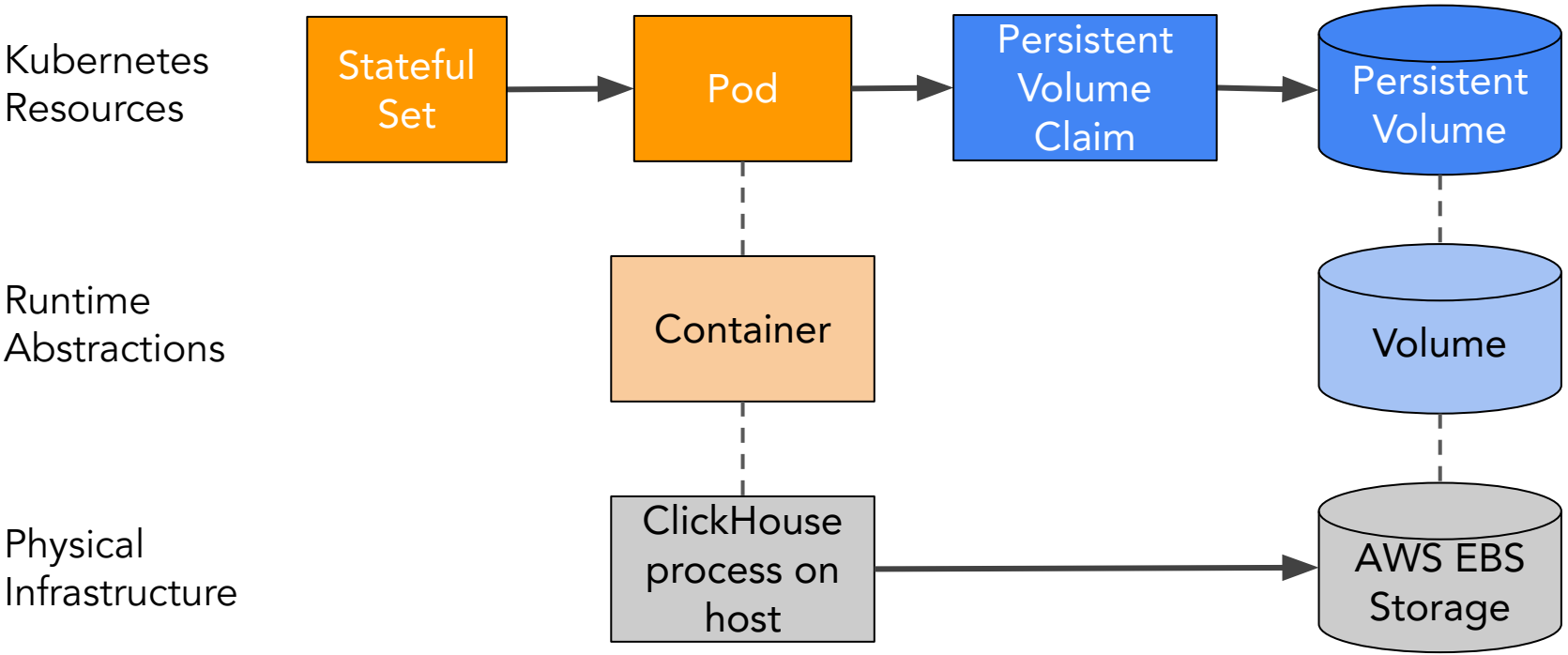
Scales to many petabytes

Is Open source (Apache 2.0)

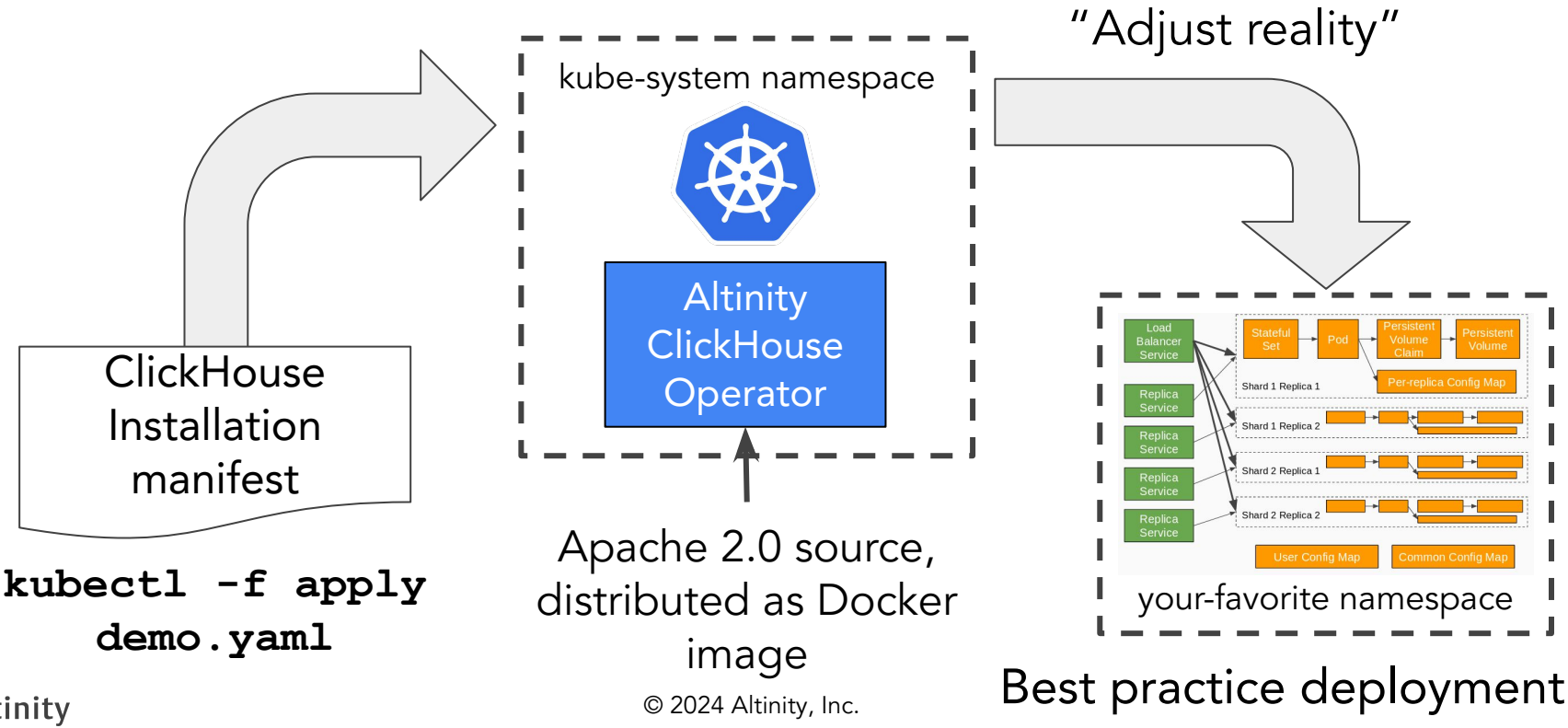


It's a popular engine for real-time analytics

Kubernetes manages container-based applications

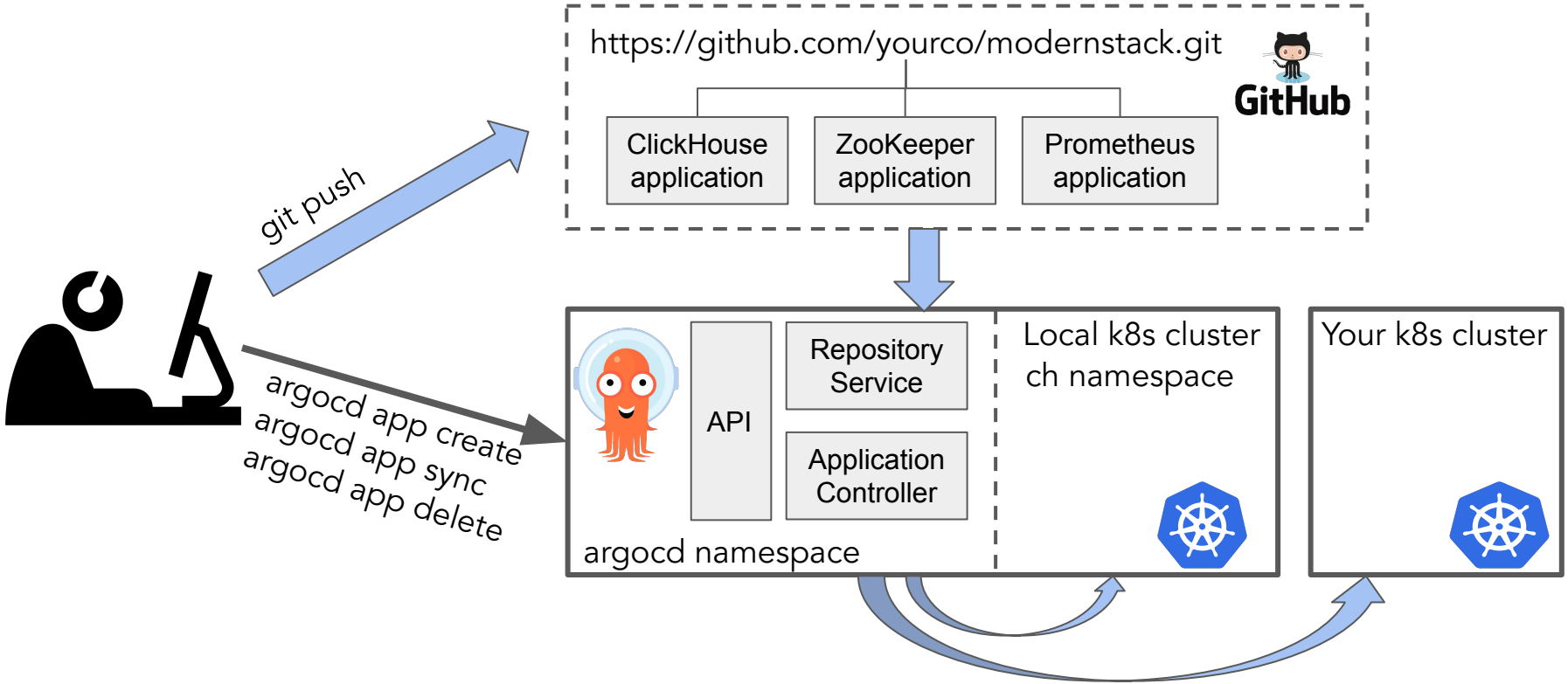


Altinity operator installs and runs ClickHouse



Tip #1: Manage ClickHouse manifests using Argo CD

Implementing GitOps with Argo CD



Define your cluster (cluster configuration, page 1)

```
apiVersion: "clickhouse.altinity.com/v1"
kind: "ClickHouseInstallation"
metadata:
  name: "argocd"
spec:
  configuration:
    clusters:
      - name: "demo"
        layout:
          shardsCount: 1
          replicasCount: 2
        templates:
          podTemplate: server
          volumeClaimTemplate: storage
    zookeeper:
      nodes:
        - host: keeper
```

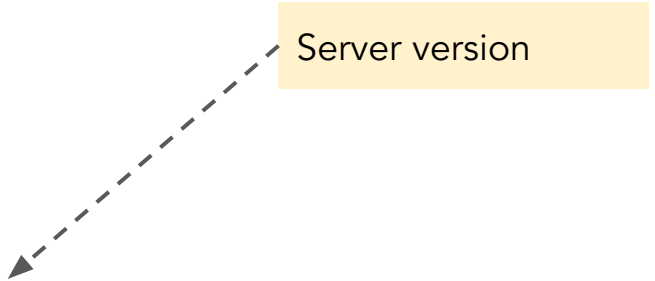
Shards and replicas

Definitions for pods and storage

Where is Zookeeper?

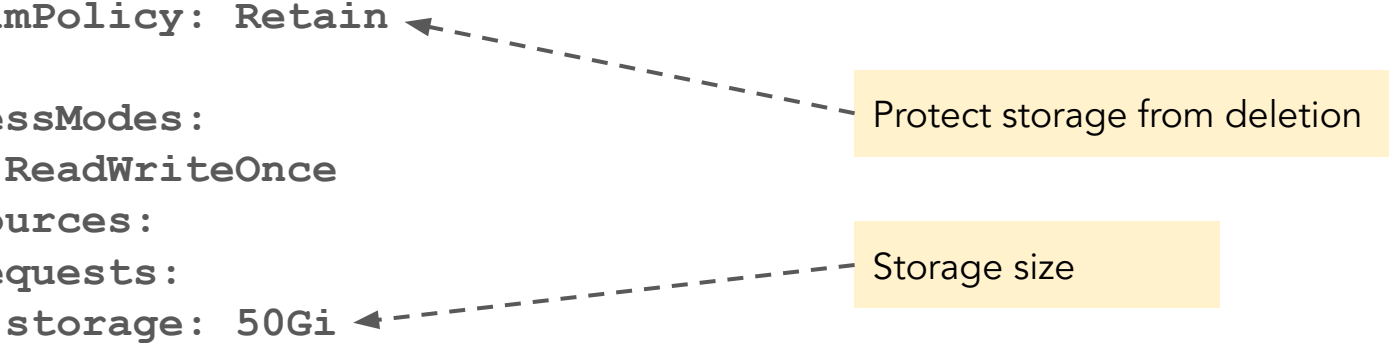
Define your cluster (pod definition, page 2)

```
templates:
  podTemplates:
    - name: server
      spec:
        containers:
        - name: clickhouse
          image: altinity/clickhouse-server:23.3.8.22.altinitystable
  volumeClaimTemplates:
    - name: storage
```



Define your cluster (storage definition, page 3)

```
volumeClaimTemplates:
  - name: storage
    # Do not delete PVC if installation is dropped.
    reclaimPolicy: Retain
    spec:
      accessModes:
        - ReadWriteOnce
      resources:
        requests:
          storage: 50Gi
```



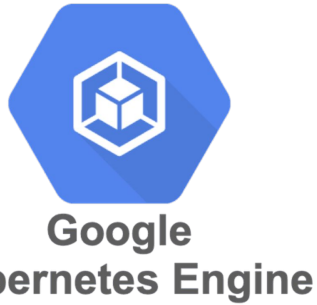
Run it with Argo CD!!

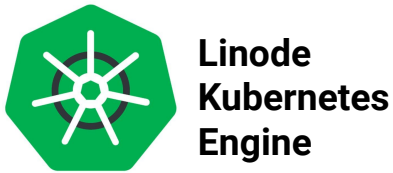
```
(Install Argo CD)

argocd app create clickhouse \
 --repo https://github.com/Altinity/argocd-examples-clickhouse.git \
 --path apps/clickhouse \
 --dest-server https://kubernetes.default.svc \
 --dest-namespace default

argocd app sync clickhouse
```

https://github.com/Altinity/argocd-examples-clickhouse

Tip #2: Use Terraform to set up managed Kubernetes

Choosing a managed Kubernetes distribution





Beta Altinity Terraform blueprint for AWS EKS cluster

```
provider "aws" {
  # https://registry.terraform.io/providers/hashicorp/aws/latest/docs
}
module "eks_clickhouse" {
  source  = "github.com/Altinity/terraform-aws-eks-clickhouse"
  cluster_name = "your-cluster"
  region       = "us-west-2"
  cidr         = "10.0.0.0/16"
  subnets      = [
    { cidr_block = "10.0.1.0/24", az = "us-west-2a" },
    { cidr_block = "10.0.2.0/24", az = "us-west-2b" },
    { cidr_block = "10.0.3.0/24", az = "us-west-2c" }
  ]
```

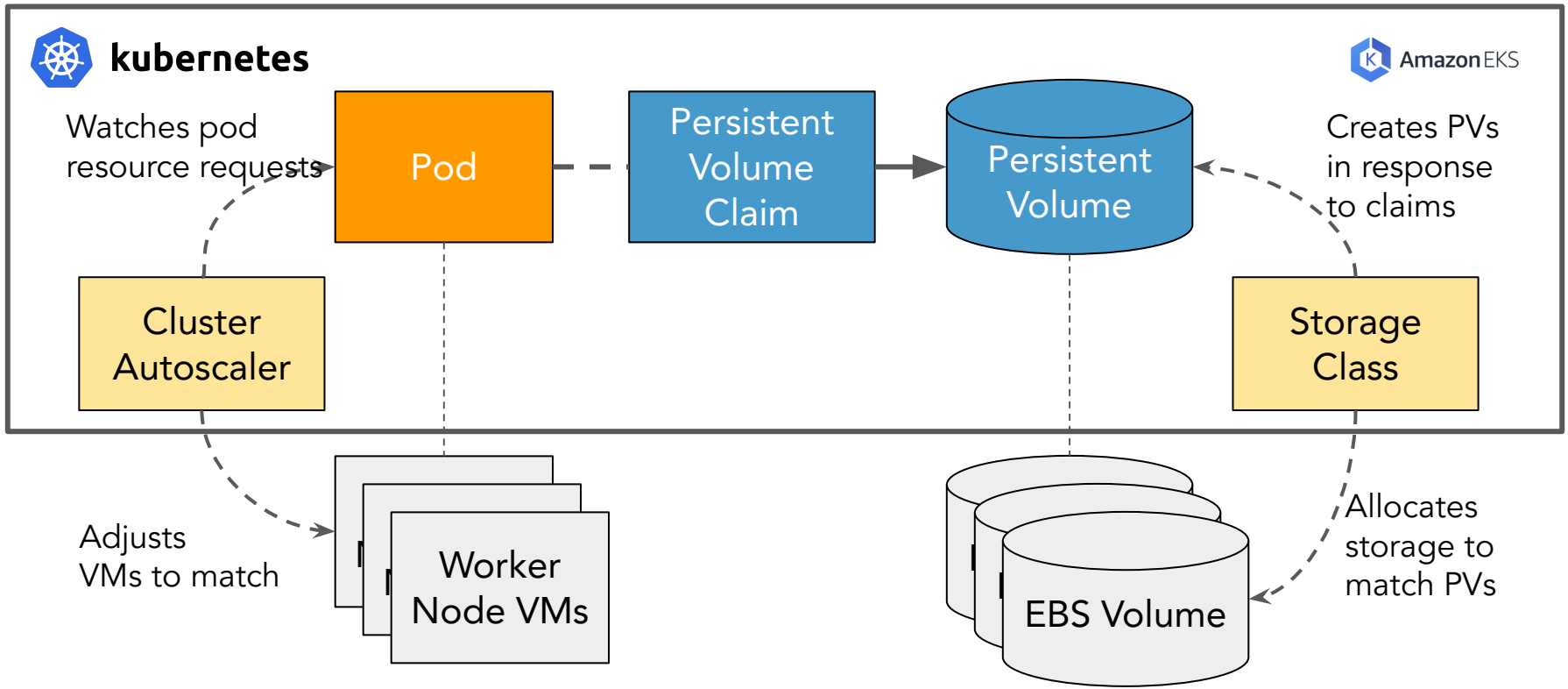
Beta Terraform blueprint, page 2

```
  # Continued…

  node_pools_config = {
    scaling_config = {
      desired_size = 2
      max_size     = 10
      min_size     = 0
    }
    disk_size      = 50
    instance_types = ["m6i.large", "m6i.xlarge"]
  }
}
```

https://github.com/Altinity/terraform-aws-eks-clickhouse

Tip #3: Scale compute using a VM auto scaler

How VM and storage provisioning work



Assign pod to a VM type

```
templates:
  podTemplates:
    - name: server
      spec:
        containers:
        - name: clickhouse
          image: altinity/clickhouse-server:23.3.13.7.altinitystable
        nodeSelector:
          node.kubernetes.io/instance-type: m6i.large
```

VM type

"Well-known" label

Pin pod to a specific VM

```
templates:
  podTemplates:
    - name: server
      spec:
        containers:
        - name: clickhouse
          image: altinity/clickhouse-server:23.3.13.7.altinitystable
        nodeSelector:
          node.kubernetes.io/instance-type: m6i.xlarge
```

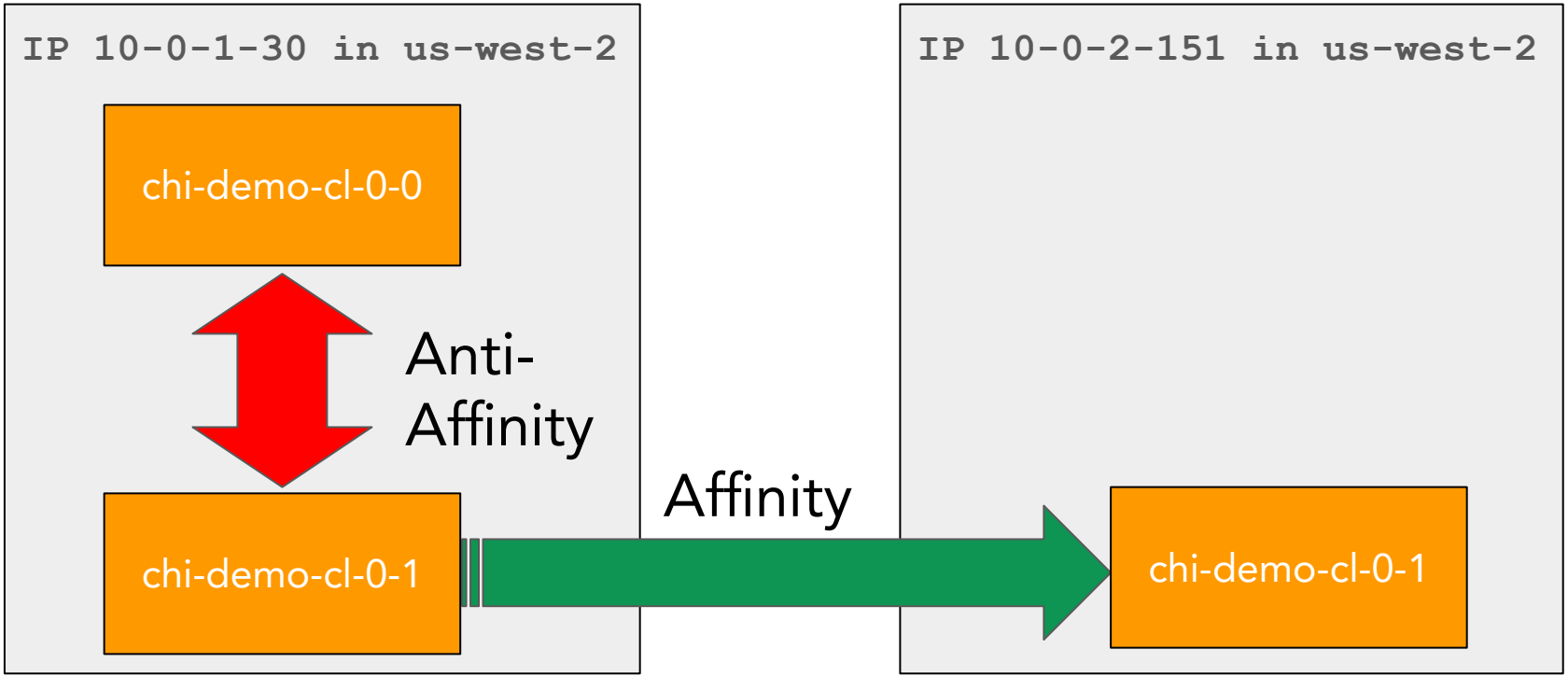
New VM type

Checking pod VM types

```
kubectl get node
-o=custom-columns=NODE:.metadata.name,ZONE:.metadata.labels.'topo
logy\.kubernetes\.io/zone',VM:.metadata.labels.'node\.kubernetes\
.io/instance-type'
NODE                                          ZONE         VM
ip-10-0-1-30.us-west-2.compute.internal       us-west-2a   m6i.large
ip-10-0-2-151.us-west-2.compute.internal      us-west-2b   m6i.large
ip-10-0-3-126.us-west-2.compute.internal      us-west-2c   m6i.large
```

Tip #4: Spread ClickHouse servers over AZs with affinity

Affinity vs. anti-affinity



Assign pod to a VM type

```
apiVersion: "clickhouse.altinity.com/v1"
kind: "ClickHouseInstallation"
metadata:
  name: "demo"
spec:
  configuration:
    clusters:
      - name: "cl"
        layout:
          shards:
            - replicas:
                - templates:
                    podTemplate: replica-in-zone-us-west-2a
                - templates:
                    podTemplate: replica-in-zone-us-west-2b
```

Two replicas for 1 shard

Separate pod template for each AZ

Add affinity rules for each pod

```
templates:
  podTemplates:
    - name: replica-in-zone-us-west-2a
      zone:
        values:
          - "us-west-2a"
      podDistribution:
        - type: ClickHouseAntiAffinity
          scope: ClickHouseInstallation
      spec:
        containers:
        - name: clickhouse
          image: altinity/clickhouse-server:23.8.8.21.altinitystable
```

Pod must be scheduled in us-west-2a

Keep pods on different hosts

Where are my pods running?

```
kubectl get pod
-o=custom-columns=NAME:.metadata.name,STATUS:.status.phase,NODE:.spe
c.nodeName
NAME                    STATUS    NODE
chi-demo-cl-0-0-0   Running   ip-10-0-1-30.us-west-2.compute.internal
chi-demo-cl-0-1-0   Running   ip-10-0-2-151.us-west-2.compute.internal
```

Checking Kubernetes worker locations

```
kubectl get node -o=custom-columns=NODE:.metadata.name,ZONE:
.metadata.labels.'topology\.kubernetes\.io/zone'

NODE                                           ZONE
ip-10-0-1-30.us-west-2.compute.internal        us-west-2a
ip-10-0-2-151.us-west-2.compute.internal       us-west-2b
ip-10-0-3-126.us-west-2.compute.internal       us-west-2c
```

How does it work? A look inside the pod specification

```
spec:
  affinity:
    nodeAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        nodeSelectorTerms:
        - matchExpressions:
          - key: topology.kubernetes.io/zone
            operator: In
            values:
            - us-west-2a
    podAntiAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
      - labelSelector:
          matchLabels:
            clickhouse.altinity.com/app: chop
            clickhouse.altinity.com/chi: demo
            clickhouse.altinity.com/namespace: default
        topologyKey: kubernetes.io/hostname
```

Generated pod specification

Pod must be scheduled on a node in us-west-2a

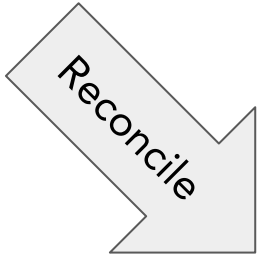
Keep servers from this cluster on different hosts

Tip #5: Turn off compute with stop: "yes"

Use stop: property to shut down all cluster pods

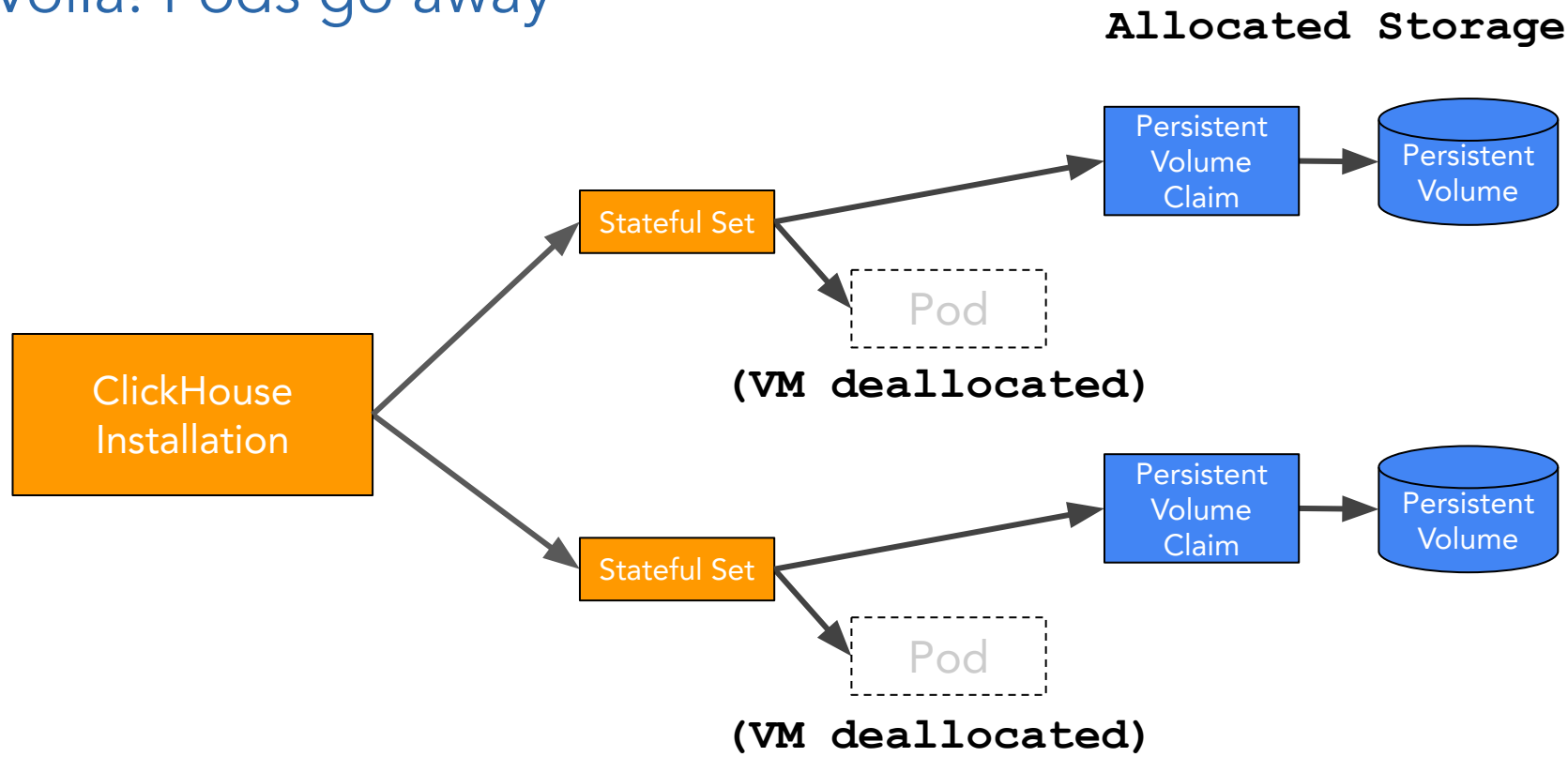
```
apiVersion: "clickhouse.altinity.com/v1"
kind: "ClickHouseInstallation"metadata:
  name: "prod"
spec:
  stop: "yes"
  configuration:
    clusters:
      - name: "ch"
```

Turn off compute



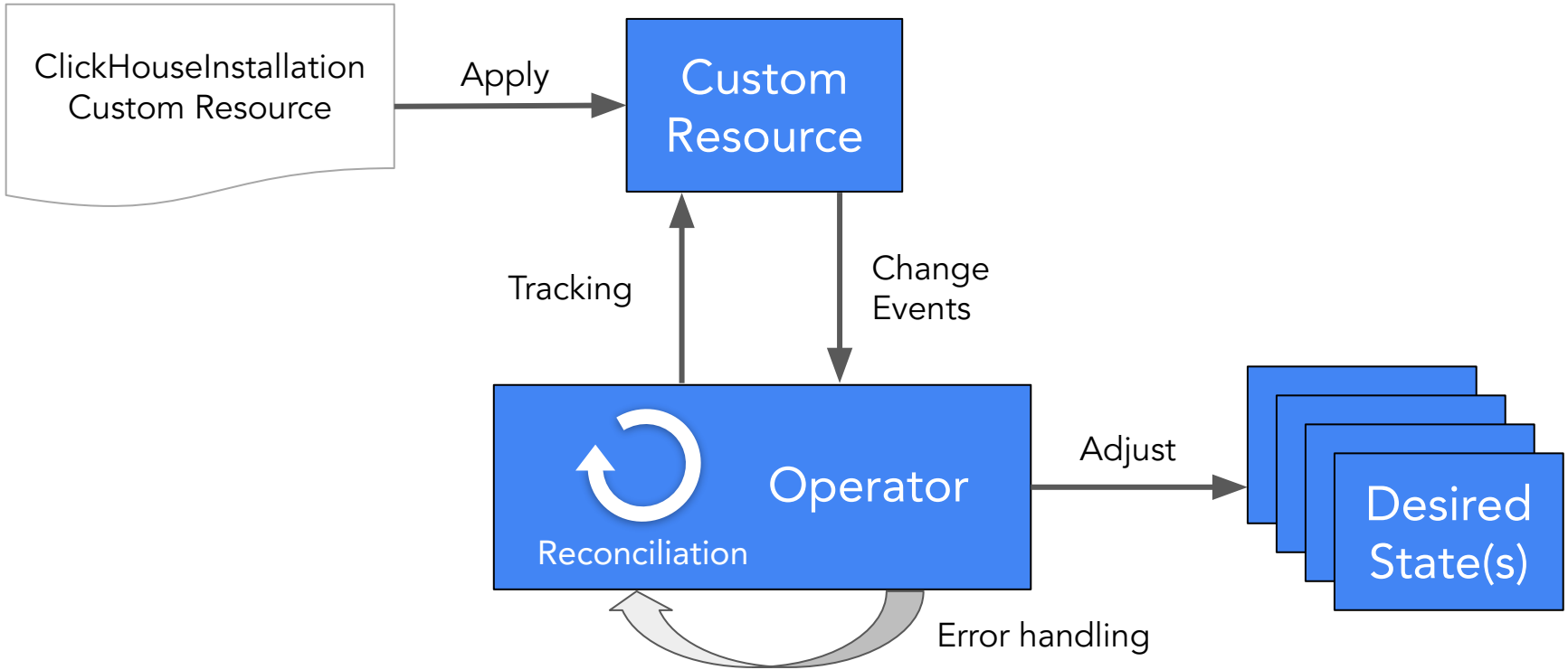

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: chi-argocd-demo-0-0
spec:
  podManagementPolicy:
OrderedReady
  replicas: 0
  revisionHistoryLimit: 10
```

Voila! Pods go away



Tip #6: Force operator to reconcile

Operators “reconcile” to apply changes



Handy use of reconciliation: refreshing passwords

```
apiVersion: "clickhouse.altinity.com/v1"
kind: "ClickHouseInstallation"
metadata:
  name: "secure"
spec:
  taskID: "1"
  configuration:
    users:
      default/password_sha256_hex:
        valueFrom:
          secretKeyRef:
            name: db-passwords
            key: default_password_sha256
```

Default user password

Pull from default_password_sha256 key in Secret named db-passwords

Operator does not automatically regen when Secret values change

Use the taskID property to force operator reconciliation

```
apiVersion: "clickhouse.altinity.com/v1"
kind: "ClickHouseInstallation"metadata:
  name: "prod"
spec:
  taskID: 1
  configuration:
    clusters:
      - name: "ch"
```

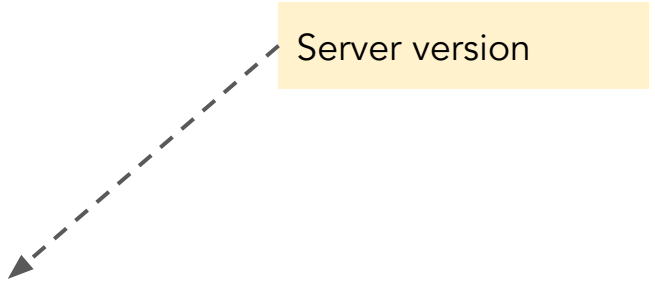
Reconcile when taskID changes

```
kubectl patch chi secure \
--type=merge -p '{"spec":{"taskID":"2"}}'
```

Tip #7: Upgrade ClickHouse automatically

Define your cluster (pod definition, page 2)

```
templates:
  podTemplates:
    - name: server
      spec:
        containers:
        - name: clickhouse
          image: altinity/clickhouse-server:23.8.8.21.altinitystable
```



Tip #8: Run ClickHouse Keeper with Altinity Operator

Introducing the ClickHouseKeeperInstallation resource!

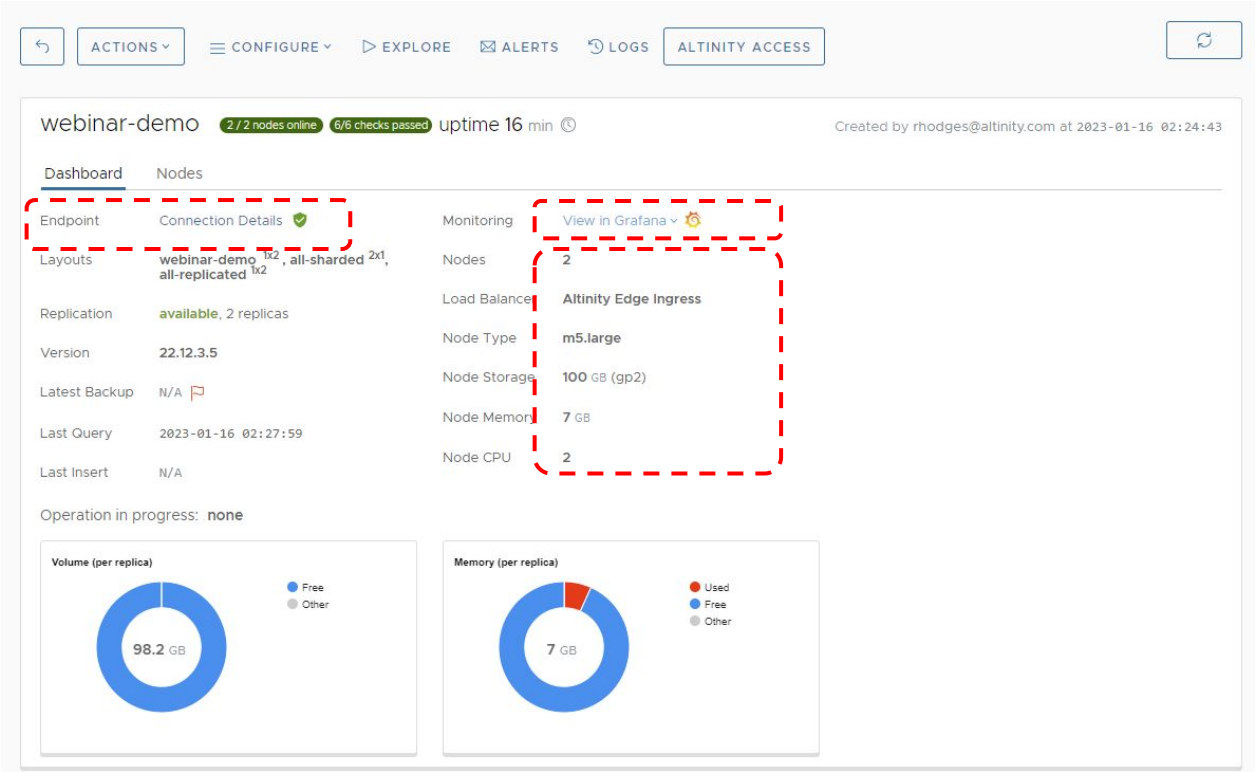
```
apiVersion: "clickhouse-keeper.altinity.com/v1"
kind: "ClickHouseKeeperInstallation"
metadata:
  name: clickhouse-keeper
spec:
  configuration:
    clusters:
      - name: "simple"
        layout:
          replicasCount: 3
    settings:
      keeper_server/storage_path: /var/lib/clickhouse-keeper
      keeper_server/tcp_port: "2181"
      keeper_server/four_letter_word_white_list: "*"
      keeper_server/raft_configuration/server/port: "9444"
```

Server version

Keeper settings

Help for managing ClickHouse on Kubernetes

Looking for an easier way? Check out Altinity.Cloud.



Typical Day 2 monitoring from Altinity.Cloud

Final thoughts

More information!

- Altinity Kubernetes Operator for ClickHouse on GitHub
  - https://github.com/Altinity/clickhouse-operator
- Altinity documention (https://docs.altinity.com)
- Altinity blog (https://altinity.com/blog)
- Kubernetes docs (https://kubernetes.io/docs/home/)
- EKS, GKE, and AKS documentation

Thank you!
Questions?

https://altinity.com
rhodges at altinity.com

Meet us at KubeCon EU in March!