AWS re:Invent

CON422-R

Building high-availability apps on EKS/Kubernetes

Jeremy Cowan
Principle Container Specialist
Amazon Web Services

Yuval Dovrat
Manager, Compute Specialists
Amazon Web Services

AWS re:Invent

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws

Agenda

What is high availability?

Service-level objectives and metrics

Error budgets

Designing for high availability

Stateless application

Stateful application

High availability

There are three principles of systems design in reliability engineering that can help achieve high availability:

1. Elimination of single points of failure: This means adding redundancy to the system so that failure of a component does not mean failure of the entire system.
2. Reliable crossover: In redundant systems, the crossover point itself tends to become a single point of failure. Reliable systems must provide for reliable crossover.
3. Detection of failures as they occur. If the two principles above are observed, then a user may never see a failure — but the maintenance activity must.

Does your application need to be highly available?

- 100% availability is not realistic and may be detrimental (except for things like medical devices)
  - Leaves less time for innovation
  - Users will not be able to tell the difference
- Applications are only as reliable as the infrastructure they run on
  - Can increase reliability by distributing the application across fault domains

Service-level objectives

- Sets expectations for how services will perform, e.g., 99.9% available
  - Controlled outages
- Differences between SLO & SLA
  - SLA usually includes a legal obligation or a financial penalty for missing the published SLA
  - No such consequences for SLO

SLO Examples

- Uptime/Availability
  - measured by a health check that approximate a user interacting with a website
- Latency
- Volume of trouble tickets

Measuring availability

- Based on a set of metrics
  - Latency
  - Request rates
  - Errors
- Avoid using averages
  - Mask issues
  - Miss outliers or the long tail
- Use distributions
  - p99, p90, etc.
- Use metrics that your users care about
- Start with loose SLOs and review periodically

Service metrics

Examples:

- 90% of requests are completed in 100ms or less
- 8% of requests are completed between 100ms and 500ms
- 2% of requests are completed in greater than 500ms
- Less than 5% of requests return 5xx errors

Error budgets

- Accounts for errors
- A quantifiable amount that an SLO can be missed by
- Calculating your error budget, a simple example:
  - (1 - your SLO)
  - 1 – 99.9 = .01% error budget
  - 1,000 requests per week
  - 10 500 errors per week
- Use budget to get max feature velocity
- Helps alleviate tension between development and ops
- Tells how much “un-reliability” is left, when it’s used up, no further code releases

Designing for HA (stateless)

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Practice Chaos

Be better prepared for outages

- Conduct game days

Build resilient software and infrastructure in a safe way

Implement Autoscaling

Cluster autoscaler

Horizontal pod autoscaling (HPA)

Replica Controller

- ReplicaSets
- Scheduler will attempt to deploy replicas across multiple fault domains/availability zones
  - Kubernetes will automatically spread the pods in a replication controller or service across zones (to reduce the impact of zone failures). (This is achieved via SelectorSpreadPriority). This is a **best-effort placement**, and so if the zones in your cluster are heterogeneous (e.g. different numbers of nodes, different types of nodes, or different pod resource requirements), this **might prevent perfectly even spreading of your pods across zones**.

When good isn't good enough

Topology Spread Constraints to the rescue

- Controls how pods are spread across failure domains!
- Alpha in 1.16

Pod Anti-affinity

- Better than best-effort
- Soft anti-affinity is obeyed during down-scaling
- When number of replicas exceed the number of AZs an even AZ distribution **is not guaranteed** for the replicas
- Soft anti-affinity is preferred over hard

Kubernetes Descheduler

- RemoveDuplicates & other strategies
  - De-schedules pods from a ReplicaSet so they're no longer running on the same node

ReplicaSet example

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: jicowan/frontend:v3
```

Health checks

- Liveness probes
  - Don’t use unless you fully understand the consequences
  - Don’t use the same settings as the readiness probe
  - Don’t have external dependencies, e.g. a database connection
  - Avoid using exec probes
- Readiness probes
  - Use with microservices that provide an HTTP endpoint
  - Use the traffic port for the readiness probe
- Used together with Kubernetes services to determine the set of healthy endpoints

Liveness example

```
spec:
  containers:
  - image: my-app:1.0
  livenessProbe:
    httpGet:
      path: /healthz
      port: 8080
  initialDelaySeconds: 30
```

Readiness examples

```
spec:
  containers:
  - image: my-app:1.0
  readinessProbe:
    exec:
      Command: [“stat”, “/var/run/file”]
```

Anti-affinity

- Avoid running mutually dependent services on the same worker node
- Schedule pods with respect to other pods
- Spread pods to achieve HA
- Use `topologyKey` field to enforce find-grained rules

Anti-affinity example

```
spec:
  affinity:
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 100
        podAffinityTerm:
          labelSelector:
            matchLabels:
              app: frontend
            topologyKey: kubernetes.io/hostname
```

Spread placement example

```
spec:
    affinity:
    podAntiAffinity:
        preferredDuringSchedulingIgnoredDuringExecution:
        - podAffinityTerm:
            labelSelector:
            matchExpressions:
            - key: app
                operator: In
                values:
                - redash
            topologyKey: failure-domain.beta.kubernetes.io/zone
        weight: 100
```

Designing for HA (stateful)

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

StatefulSets

For applications that require

- Stable, unique network identifiers
- Stable, persistent storage
- Ordered, graceful deployment and scaling
- Ordered, automated rolling updates

StatefulSets continued

- Run applications that handle data replication at the application level, e.g., Mongo, Kafka
- Use read replicas with RDBMS applications, e.g., MySQL
- Use an operator to manage the lifecycle of stateful applications:
  - https://github.com/oracle/mysql-operator
  - https://github.com/percona/percona-xtradb-cluster-operator
  - https://operatorhub.io/?category=Database

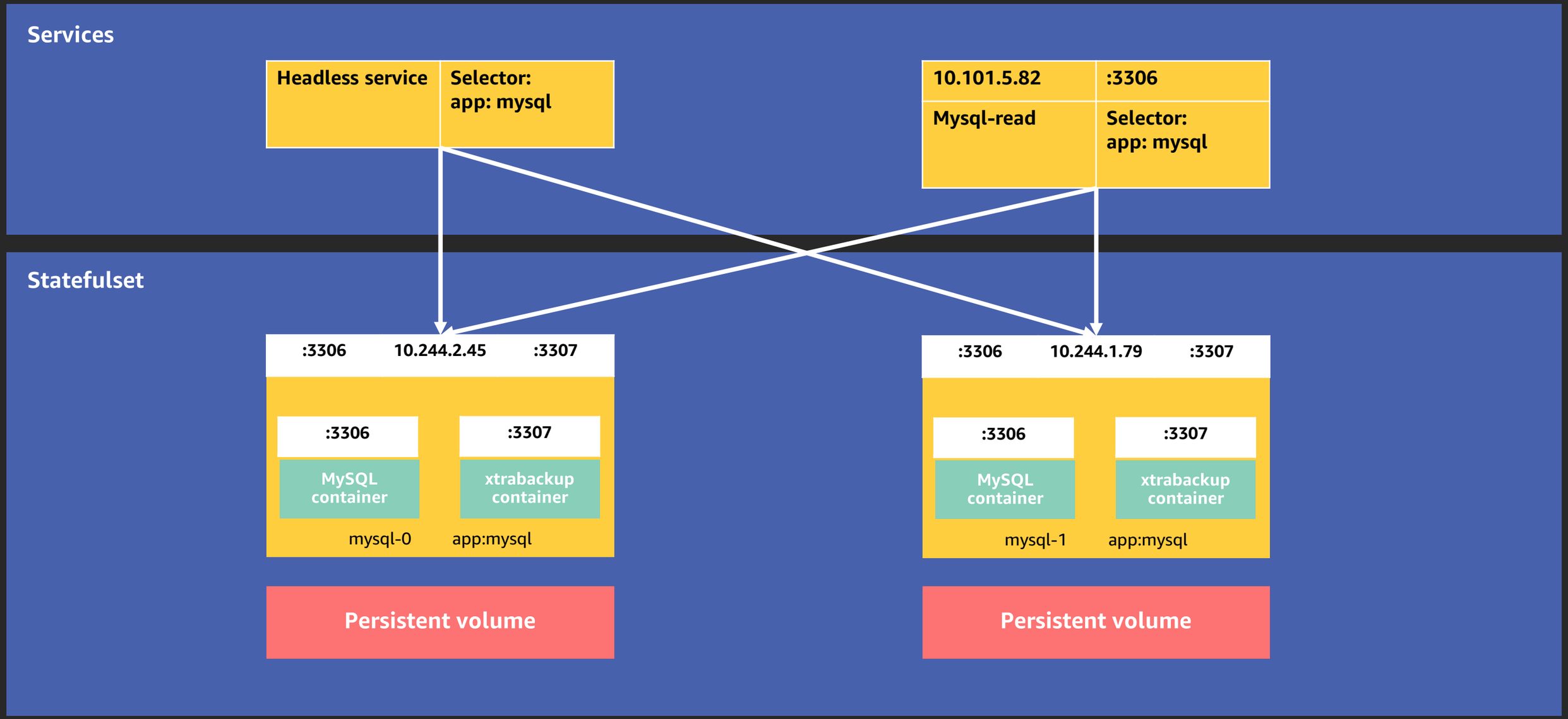
Kubernetes Operators

- Aims to capture the goals of a human operator who is managing a service or set of services
- Designed for automation

MySQL example

- statefulset
- configmap
- storageclass
- service

MySQL architecture



Demo: Running MySQL reliably on Amazon Elastic Kubernetes Service (Amazon EKS)

AWS re:Invent

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws

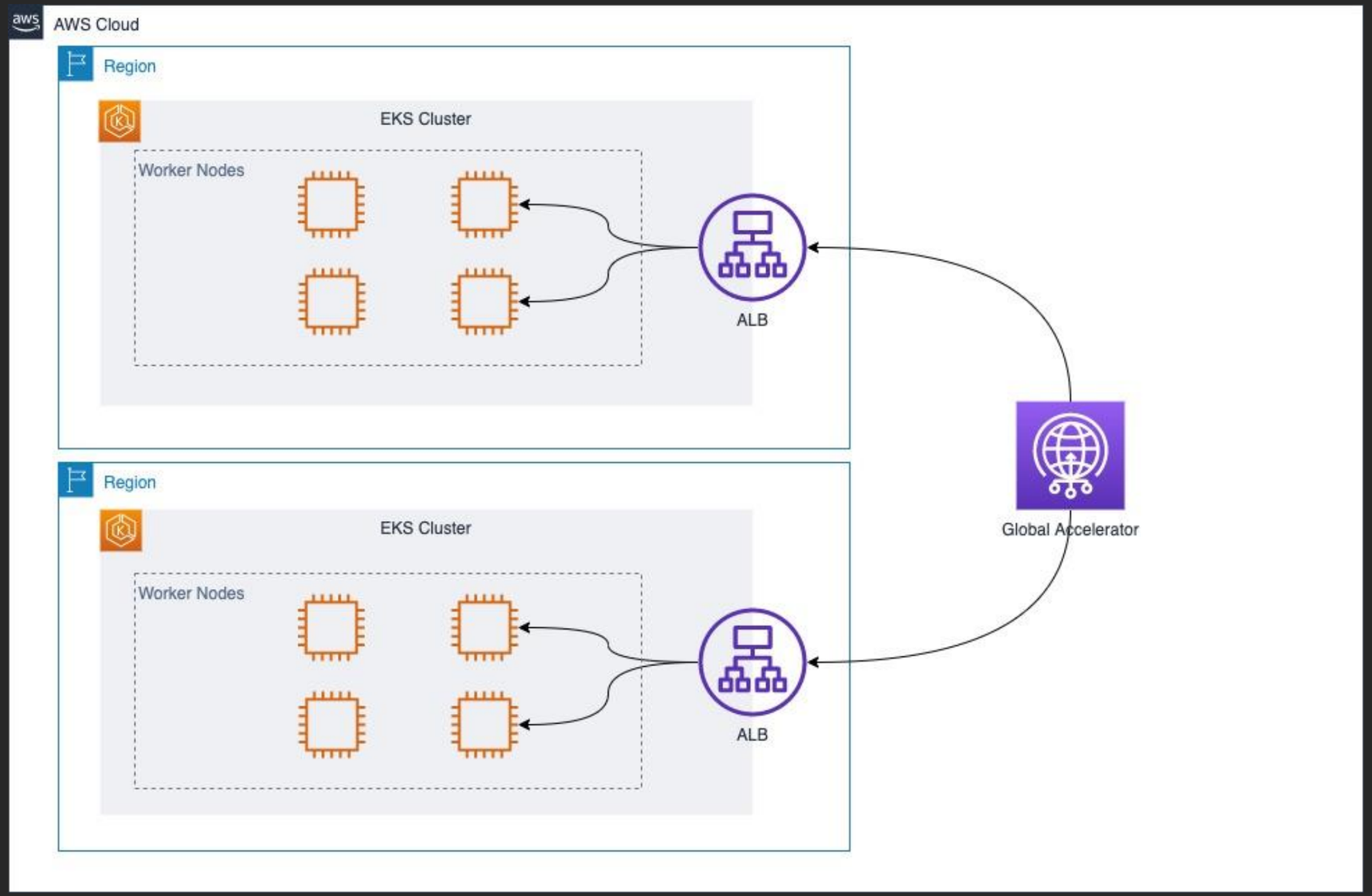
Multi-cluster HA

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Multi-cluster deployments

- Deploy application to multiple regions
- Manually register external endpoints in the respective cluster
- Use a CNI, e.g. Cilium, that utilizes eBPF for load balancing
  - Sandboxed VM running in the Linux kernel (Facebook and Cilium)
- Use a global load balancer to distribute traffic between deployments in different regions
  - Global Accelerator, Route 53, Heptio Gimbal and Heptio Contour
- Use a service mesh

Cross-region deployment architecture



Demo: Multi-cluster with GSLB demo

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Thank you!

AWS re:Invent

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.

aws



Please complete the session survey in the mobile app.

© 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.