Guidance on High Availability

ESS makes use of Kubernetes for deployment so most guidiance on high-availability is tied directly with general Kubernetes guidance on high availability.

Kubernetes

Essential Links

- Options for Highly Available Topology
- Creating Highly Available Clusters with kubeadm
- Set up a High Availability etcd Cluster with kubeadm
- Production environment

High-Level Overview

It is strongly advised to make use of the Kubernetes documentation to ensure your environment is setup for high availability, see links above. At a high-level, Kubernetes achieves high availability through:

- **Cluster Architecture.**
  - **Multiple Masters:** In a highly available Kubernetes cluster, multiple master nodes (control plane nodes) are deployed. These nodes run the critical components such as `etcd`, the API server, scheduler, and controller-manager. By using multiple master nodes, the cluster can continue to operate even if one or more master nodes fail.
  - **Etcd Clustering:** `etcd` is the key-value store used by Kubernetes to store all cluster data. It can be configured as a cluster with multiple nodes to provide data redundancy and consistency. This ensures that if one etcd instance fails, the data remains available from other instances.
- **Pod and Node Management.**
  - **Replication Controllers and ReplicaSets:** Kubernetes uses replication controllers and ReplicaSets to ensure that a specified number of pod replicas are running at any given time. If a pod fails, the ReplicaSet automatically replaces it, ensuring continuous availability of the application.
  - **Deployments:** Deployments provide declarative updates to applications, allowing rolling updates and rollbacks. This ensures that application updates do not cause downtime and can be rolled back if issues occur.
  - **DaemonSets:** DaemonSets ensure that a copy of a pod runs on all (or a subset of) nodes. This is useful for deploying critical system services across the entire cluster.
- **Service Discovery and Load Balancing.**
  - **Services:** Kubernetes Services provide a stable IP and DNS name for accessing a set of pods. Services use built-in load balancing to distribute traffic among the pods, ensuring that traffic is not sent to failed pods.

    - **Ingress Controllers:** Ingress controllers manage external access to the services in a cluster, typically HTTP. They provide load balancing, SSL termination, and name-based virtual hosting, enhancing the availability and reliability of web applications.
- **Node Health Management.**
    - **Node Monitoring and Self-Healing:** Kubernetes continuously monitors the health of nodes and pods. If a node fails, Kubernetes can automatically reschedule the pods from the failed node onto healthy nodes. This self-healing capability ensures minimal disruption to the running applications.
    - **Pod Disruption Budgets (PDBs):** PDBs allow administrators to define the minimum number of pods that must be available during disruptions (such as during maintenance or upgrades), ensuring application availability even during planned outages.
- **Persistent Storage.**
    - **Persistent Volumes and Claims:** Kubernetes provides abstractions for managing persistent storage. Persistent Volumes (PVs) and Persistent Volume Claims (PVCs) decouple storage from the pod lifecycle, ensuring that data is preserved even if pods are rescheduled or nodes fail.
    - **Storage Classes and Dynamic Provisioning:** Storage classes allow administrators to define different storage types (e.g., SSDs, network-attached storage) and enable dynamic provisioning of storage resources, ensuring that applications always have access to the required storage.
- **Geographical Distribution.**
    - **Multi-Zone and Multi-Region Deployments:** Kubernetes supports deploying clusters across multiple availability zones and regions. This geographical distribution helps in maintaining high availability even in the event of data center or regional failures.
- **Network Policies and Security.**
    - **Network Policies:** These policies allow administrators to control the communication between pods, enhancing security and ensuring that only authorized traffic reaches critical applications.
    - **RBAC (Role-Based Access Control):** RBAC restricts access to cluster resources based on roles and permissions, reducing the risk of accidental or malicious disruptions to the cluster's operations.
- **Automated Upgrades and Rollbacks.**
    - **Cluster Upgrade Tools:** Tools like `kubeadm` and managed Kubernetes services (e.g., Google Kubernetes Engine, Amazon EKS, Azure AKS) provide automated upgrade capabilities, ensuring that clusters can be kept up-to-date with minimal downtime.
    - **Automated Rollbacks:** In the event of a failed update, Kubernetes can automatically roll back to a previous stable state, ensuring that applications remain available.

How does this tie into ESS

As ESS is deployed into a Kubernetes cluster, if you are looking for high availability you should ensure your environment is configured with that in mind. One important factor is to ensure you deploy using the Kubernetes deployment option, whilst Standalone mode will deploy to a Kubernetes cluster, by definition it exists solely on a single node so options for high availability will be limited.

PostgreSQL

Essential links

- PostgreSQL - High Availability, Load Balancing, and Replication

- PostgreSQL - Different replication solutions

High-Level Overview

To ensure a smooth failover process for ESS, it is crucial to prepare a robust database topology. The following list outline the necessary element to take into consideration:

- **Database replicas**
    - **Location**: Deploy the database replicas in a separate data center from the primary database to provide geographical redundancy.
    - **Replication**: Configure continuous replication from the primary database to the s econdary database. This ensures that the secondary database has an up-to-date copy of all data.
- **Synchronization and Monitoring**
    - **Synchronization**: Ensure that the secondary database is consistently synchronized with the primary database. Use reliable replication technologies and monitor for any lag or synchronization issues.
    - **Monitoring Tools**: Implement monitoring tools to keep track of the replication status and performance metrics of both databases. Set up alerts for any discrepancies or failures in the replication process.
- **Data Integrity and Consistency**
    - **Consistency Checks**: Periodically perform consistency checks between the primary and secondary databases to ensure data integrity. -**Backups**: Maintain regular backups of both the primary and secondary databases. Store backups in a secure, redundant location to prevent data loss.
- **Testing and Validation**
    - **Failover Testing**: Conduct regular failover drills to test the transition from the primary to the secondary database. Validate that the secondary database can handle the load and that the failover process works seamlessly.
    - **Performance Testing**: Evaluate the performance of the secondary database under expected load conditions to ensure it can maintain the required service levels.

By carefully preparing the database topology as described, you can ensure that the failover process for ESS is efficient and reliable, minimizing downtime and maintaining data integrity.

How does this tie into ESS

As ESS relies on PostgreSQL for its database if you are looking for high availability you should ensure your environment is configured with that in mind. The database replicas can be achieved the same way in both Kubernetes and Standalone deployment, as the database is **not** managed by ESS.

ESS failover plan

This document outlines a high-level, semi-automatic, failover plan for ESS. The plan ensures continuity of service by switching to a secondary data center (DC) in the event of a failure in the primary data center.

Prerequisites

- **Database Replica**: A replica of the main database, located in a secondary data center, continuously reading from the primary database.
- **Secondary ESS Deployment**: An instance of the ESS deployment, configured in a secondary data center.
- **Signing Keys Synchronization**: The signing keys stored in ESS secrets need to be kept synchronized between the primary and secondary data centers.
- **Media Repository**: Media files are stored on a redundant S3 bucket accessible from both data centers.

ESS Architecture for failover capabilities based on 3 datacenters

DC1 (Primary)

- **ElementDeployment Manifest**:
  - Manifest points to addresses in DC1.
  - TLS Secrets managed by ACME.
- **TLS Secrets**:
  - Replicated to DC2 and DC3.
- **Operator**:
  - 1 replica.
- **Updater**:
  - 1 replica.
- **PostgreSQL**:
  - Primary database.

DC2

- **ElementDeployment Manifest**:
  - Manifest points to addresses in DC2.
  - TLS Secrets pointing to existing secrets, replicated locally from DC1.
- **Operator**:
  - 0 replica, it prevents the deployment of the kubernetes workloads
- **Updater**:
  - 1 replica, the base element manifest are ready for the operator to deploy the workloads
- **PostgreSQL**:
  - Hot-Standby, replicating from DC1.

DC3

- **ElementDeployment Manifest:**
  - Manifest points to addresses in DC3.
  - TLS Secrets pointing to existing secrets, replicated locally from DC1.
- **Operator**:

    - 0 replica, it prevents the deployment of the kubernetes workloads
- **Updater**:
    - 1 replica, the base element manifest are ready for the operator to deploy the workloads
- **PostgreSQL**:
    - Hot-Standby, replicating from DC1.

Failover Process

When DC1 experiences downtime and needs to be failed over to DC2, follow these steps:

- **Disable DC1**:
    - Firewall outbound traffic to prevent federation/outbound requests such as push notifications.
    - Scale down the Operator to 0 replicas and remove workloads from DC1.
- **Activate DC2**:
    - Promote the PostgreSQL instance in DC2 to the primary role.
    - Set Operator Replicas:
        - Increase the Operator replicas to 1.
        - This starts the Synapse workloads in DC2.
    - Update the DNS to point the ingress to DC2.
    - Open the firewall if it was closed to ensure proper network access.
- **Synchronize DC3**:
    - Ensure PostgreSQL Replication:
        - Make sure that the PostgreSQL in DC3 is properly replicating from the new primary in DC2.
        - Adjust the PostgreSQL topology if necessary to ensure proper synchronization.

You should decline your own failover procedure based on this high-level failover overview. By doing so, you can ensure that ESS continues to operate smoothly and with minimal downtime, maintaining service availability even when the primary data center goes down.

Revision #11
Created 10 June 2024 09:23:39 by Kieran Mitchell Lane
Updated 6 November 2024 13:20:36 by Kieran Mitchell Lane