Eine kurze Erklärung zum Diagramm

Namespace: WordPress (Installiert via Helm)

Komponenten:

- **WordPress Helm Release:** Managt alle zugehörigen Ressourcen inklusive Deployments, Services und PVCs für eine automatisierte und konsistente Installation.
- **WordPress Pods:** Enthalten die WordPress-Anwendung, die über das WordPress Deployment automatisch verwaltet wird.
- **WordPress Service:** Ein LoadBalancer, der externe Anfragen auf die WordPress Pods verteilt.
- **MariaDB Pods und Service:** Separate Datenbankinstanzen, die durch das Helm Chart bereitgestellt und verwaltet werden, spezifisch für WordPress-Daten.
- **Persistent Volume Claims (PVCs):** Für die Speicherung von Medien und anderen Dateien, die WordPress persistent speichern muss.

Beschreibung: In diesem Namespace orchestriert Helm die Installation und das Management von WordPress und seiner Datenbank. Der WordPress Service ermöglicht den externen Zugang zu den Anwendungspods, während MariaDB-Pods die notwendige Datenhaltung sicherstellen.

Namespace: MediaWiki (Installiert via Helm)

Komponenten:

- **MediaWiki Helm Release:** Verwaltet das MediaWiki-Deployment, zugehörige Services und Datenbankintegration über Helm.
- **MediaWiki Pods:** Führen die MediaWiki-Software innerhalb des Clusters aus.
- **MediaWiki Service:** Ein LoadBalancer, der den Zugriff von außerhalb des Clusters ermöglicht.
- **MariaDB Pods und Service:** Betreiben die Datenbankdienste speziell für MediaWiki.
- **Persistent Volume Claims (PVCs):** Gewährleisten dauerhaften Speicher für Inhalte und Datenbanken.

Beschreibung: Der MediaWiki-Namespace nutzt ebenfalls Helm zur Vereinfachung der Bereitstellung. MediaWiki und seine Datenbankkomponenten sind vollständig durch Helm integriert, was eine reibungslose Installation und Skalierbarkeit ermöglicht.

Namespace: Jira (Installiert via YAML-Dateien)

Komponenten:

- **Jira Deployment:** Managt die Jira-Pods direkt ohne die Nutzung von Helm.
- **Jira Pods:** Beherbergen die Jira-Softwareanwendung.
- **Jira Service:** Typischerweise als NodePort oder ClusterIP konfiguriert, um interne oder begrenzte externe Zugänglichkeit zu bieten.
- **Datenbank Deployment und Service:** (z.B. PostgreSQL) Für die Verwaltung der Jira-Daten.
- **Persistent Volume Claims (PVCs):** Stellen sicher, dass sowohl Jira als auch die Datenbank persistenten Speicher für ihre Daten haben.

Beschreibung: In diesem Namespace wird Jira über traditionelle YAML-Dateien eingerichtet, die eine manuellere Konfiguration von Ressourcen wie Deployments und Services erlauben. Dies bietet Flexibilität bei der Konfiguration und ist ideal für spezifische Anpassungen.

Hypothetische Ingress-Konfiguration bei Nutzung einer einzigen Cloud-Umgebung

Unter idealen Umständen, also wenn alle unsere Anwendungen wie WordPress, MediaWiki und Jira in einer einzigen Cloud-Umgebung gehostet wären, hätten wir einen zentralen Ingress-Controller eingesetzt. Dieser Ingress-Controller hätte als Hauptzugangspunkt für alle externen Anfragen gedient und hätte den Verkehr basierend auf URLs oder anderen Parametern zu den jeweiligen Anwendungen im Cluster geleitet.

Jede Anwendung hätte ihren eigenen spezifischen Pfad oder eine Subdomain gehabt, die durch den Ingress geregelt wird. Dies hätte es ermöglicht, den Netzwerkverkehr klar zu organisieren und zu sichern. Darüber hinaus hätten wir den Ingress mit Sicherheitszertifikaten für eine verschlüsselte und sichere Kommunikation ausgestattet.

Aufgrund von technischen Schwierigkeiten mit der Cloud-Konfiguration mussten wir jedoch auf mehrere Cloud-Umgebungen ausweichen, was eine solche zentralisierte Konfiguration verhinderte. Unter optimalen Bedingungen hätte unser Setup etwa so ausgesehen, wie hier beschrieben.

Externer Zugriff auf Kubernetes-Anwendungen über Ingress-Controller:

Wenn ein Benutzer eine Anwendung wie WordPress, MediaWiki oder Jira in einem Kubernetes-Cluster besuchen möchte, gibt er die entsprechende URL in seinen Browser ein. Diese Anfrage wird zuerst durch DNS in die IP-Adresse des Cloud-Load-Balancers aufgelöst, der den Eingangsverkehr an die Nodes im Kubernetes-Cluster verteilt, auf denen der Ingress-Controller läuft. Der Ingress-Controller prüft und leitet die Anfrage gemäß den festgelegten Regeln an den richtigen Kubernetes-Service weiter, der dann den Verkehr an die passenden Pods weitergibt, die die Anwendung ausführen. Dieser Prozess sorgt für eine effiziente, sichere und skalierbare Handhabung der Benutzeranfragen.