Nutanix Database Service (NDB) -

Database-as-a-Service Across On-Premises & Public Cloud

Nutanix Database Service is the only hybrid cloud database-as-a-service for Microsoft SQL Server, Oracle Database, PostgreSQL, MongoDB, and MySQL. Efficiently manage hundreds to thousands of databases. Quickly and easily provision new databases, automate important, but tedious, administrative tasks like patching and backups, and choose the right operating systems, database versions, and database extensions to meet application and compliance requirements. Customers around the world have simplified their databases across on-premises, colocation sites, and public clouds and accelerated software development with Nutanix Database Service.

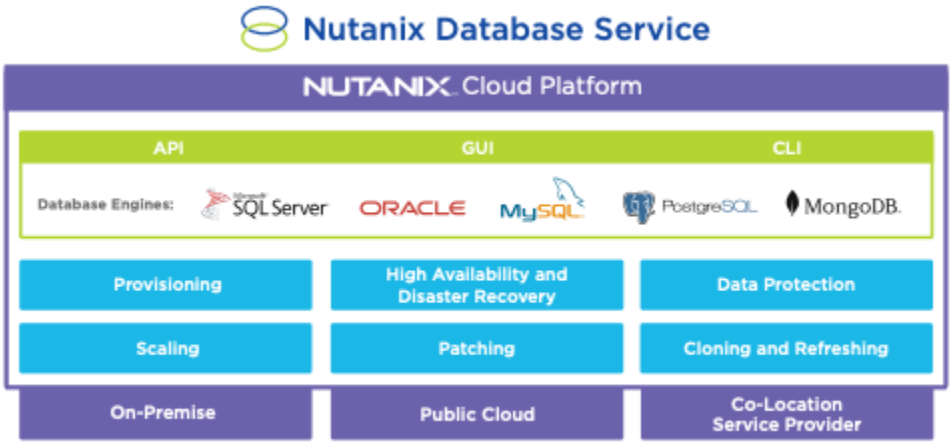
Features:

Lifecycle Management: Manage the entire database lifecycle, from database provisioning and scaling to version upgrades and patch automation.

Database Management at Scale: Manage hundreds to thousands of Microsoft SQL Server, Oracle, PostgreSQL, MySQL, and MongoDB databases across on-premises, colocation facilities, and one or more public clouds — all from a single control plane.

Self-Service Database: Provisioning Enables self-service database provisioning for both dev/test and production use via API integration with popular infrastructure management and development tool (e.g., ServiceNow).

Database Protection: Quickly roll out patches across some or all of your databases to protect against the latest security threats and restrict access to databases with role-based access controls to ensure compliance with regulatory requirements and best practices.



NDB Kubernetes Operator - Open Source

The NDB operator brings automated and simplified database administration, provisioning, and life-cycle management to Kubernetes.

Developers can now provision PostgreSQL, MySQL, and MongoDB databases directly from their K8s cluster, saving days to weeks of effort. They can use the opensource NDB Operator from their K8s platform of choice, while benefiting from the full database lifecycle management provided by NDB. Developers can get started with the NDB operator today at https://artifacthub.io/packages/helm/nutanix/ndb-operator.

Why do developers want to use the kubernetes operator?
Today, many developers use K8s to simplify deploying, managing, and scaling applications, both in development and production environments. However, databases like PostgreSQL, MySQL, or MongoDB are typically more complex to deploy and manage than other applications. Because of the added complexity, developers will typically need to choose one of two options to get the right database deployed into their environment. First, developers can reach out to a database administrator (DBAs) to get help. However, getting the help of a DBA may take hours to days and typically results in a one-time, manual, deployment of the database. Instead, if the developer wants an automated solution so they can easily repeat the database deployment for, for example, testing purposes, they will need to find the appropriate K8s operator that automates deployment of their database of choice. However, developers often use multiple database engines and will have to find the right K8s operator for each database engine. Finding, installing, configuring, and validating each K8s operator is a time consuming task that could take days to weeks per database engine. The databases provisioned by the developers are safeguarded by DBAs, who take care of backups, cloning, scaling and patching, and need to maintain multiple solutions for these.

More details are available on Public Github Repository -
https://github.com/nutanix-cloud-native/ndb-operator

This project aims to follow the Kubernetes Operator pattern

It uses Controllers which provides a reconcile function responsible for synchronizing resources until the desired state is reached on the cluster.

Pre-requisites:

Golang: All of the projects will require reading and writing GoLang code; see language guide for an introduction.
Kubernetes: All of the projects will require installing a local kubernetes distribution and basic cluster administration knowledge. Please see the kubernetes guide for an introduction.

Operator Framework: All of the projects will require using the Operator framework. Please read the go operator SDK documentation for an introduction.

Student Projects :

NTNX-1:- Support provisioning MongoDb via NDB Kubernetes Operator

The NDB Kubernetes operator currently supports only PostgreSQL database provisioning. In this project, add a new database type - MongoDB. Extend current interfaces to support NoSQL databases and test out provisioning and deprovisioning end-to-end.

NTNX-2:- Support provisioning MS SQL Server databases via NDB Kubernetes Operator
The NDB Kubernetes operator currently supports only PostgreSQL database provisioning. In this project, add a new database type - MS SQL. Extend current interfaces to support commercial databases like MS SQL and test out provisioning and deprovisioning end-to-end.

NTNX-3: Refactor models to keep profiles (software, compute, network, etc) as optional and use default if not specified
The NDB Kubernetes operator currently uses default compute, network and OS software profiles while provisioning the database. Refactor this module to include optional fields and only if absent, fall back to default.

NTNX-4: Extend Nutanix Database Service Kubernetes operator to provision Postgres Single Instance databases on AWS EC2
The NDB Kubernetes operator currently supports only PostgresQL database provisioning on Nutanix cloud infrastructure. Extend the support of PostgreSQL (and similar open source databases) to deploy databases on EC2 using NDB's public cloud APIs.

NDB Lab Access / Development Guide

Request NDB : https://www.nutanix.com/test-drive-database-operations

Mentor contact:
Manav Rajvanshi manav.rajvanshi@nutanix.com
Krunal Jhaveri krunal.jhaveri@nutanix.com

Development Guide

1. Operator installation
    a. Local Setup:
        i. Prerequisites: Access to a Kubernetes cluster. Make sure that the user is authorized with cluster-admin permissions
        ii. Install the operator-sdk
        iii. Clone the operator repository, run:
        git clone https://github.com/nutanix-cloud-native/ndb-operator
        iv. Run **cd ndb-operator**
        v. Run the operator outside the cluster as a go program by running:
            make install run
        Run the operator within the cluster as a deployment by running:
            make deploy
    b. RedHat OpenShift:
        i. Prerequisites: SSH access to an OpenShift environment with operator-sdk and olm installed. Public docker hub account.

        [Do this on local setup: Bundle and push the operator]

        ii. Clone the operator repository, run:
        git clone https://github.com/nutanix-cloud-native/ndb-operator
        iii. Run **cd ndb-operator**
        iv. Export environment variables, run:
        export DOCKER_USERNAME=your-dockerhub-username
        export VERSION=x.y.z (x,y,z are numbers)
        export IMG=docker.io/$DOCKER_USERNAME/ndb-operator:v$VERSION
        export BUNDLE_IMG=docker.io/$DOCKER_USERNAME/ndb-operator-bundle:v$VERSION
        v. Run **make docker-build docker-push**
        vi. Run **make bundle** (and follow the prompts)
        vii. Run **make bundle-build bundle-push**

        [Do this on the openshift cluster]

        viii. Export same environment variables as in step iv.
        ix. Run **operator-sdk run bundle $BUNDLE_IMG**

2. Operator Usage Using a YML file

```yaml
apiVersion: ndb.nutanix.com/v1alpha1
kind: Database
metadata:
 name: db
spec:
 ndb:
   clusterId: "Nutanix Cluster Id"
   credentials:
     loginUser: admin
     password: "NDB Password"
     sshPublicKey: "SSH Key"
   server: https://[NDB IP]:8443/era/v0.9

 databaseInstance:
   databaseInstanceName: "Database Instance Name"
   databaseNames:
     - database_one
     - database_two
     - database_three
   password:
   size: 10
   timezone: "UTC"
   type: postgres
```

Architecture details:

Entities/Objects and Organization (Group, Version and Kind):

The idea is to group all the NDB related resources and APIs into the **ndb** group. The group will contain all the **kinds** (resources) along with the versions. For the initial stages, we have only a single kind - the 'database' kind, which refers to the database instances running on the VMs and carries all the information about the other related resources. Moving forward as the operator matures, we can decompose the database kind into multiple kinds like VMs, Clusters, Profiles, Time Machines etc.

DB and Application Connectivity Approaches:

The primary challenge in connecting the operator-provisioned database with an application lies in automatically sharing the connection details of the database instance provisioned in NDB. Sharing the username/password is relatively easier as the user is aware of the credentials while provisioning the database (and the app stack) and these can be easily injected into the application(s) by using configmaps/secrets/plain-text etc. We still **need a mechanism to share the database instance host IP** with the application pod once the database has been provisioned (IP is assigned after provisioning is done). Also, we **need a mechanism for the application pod to wait** for the database provisioning before starting up (This is not a strict necessity, we can design our apps to fail if DB connectivity fails and then kubernetes can restart the failed pods until it succeeds).

Sharing the database instance IP can be achieved in the following ways:

a. Creating a K8s service that maps to an external service [10][11] endpoint (NDB in this case) once the database instance has been provisioned, and making the application use the service instead of NDB directly.
b. Creating a configmap with the IP for the database once it is provisioned and referencing the configmap in the application pod for the IP.

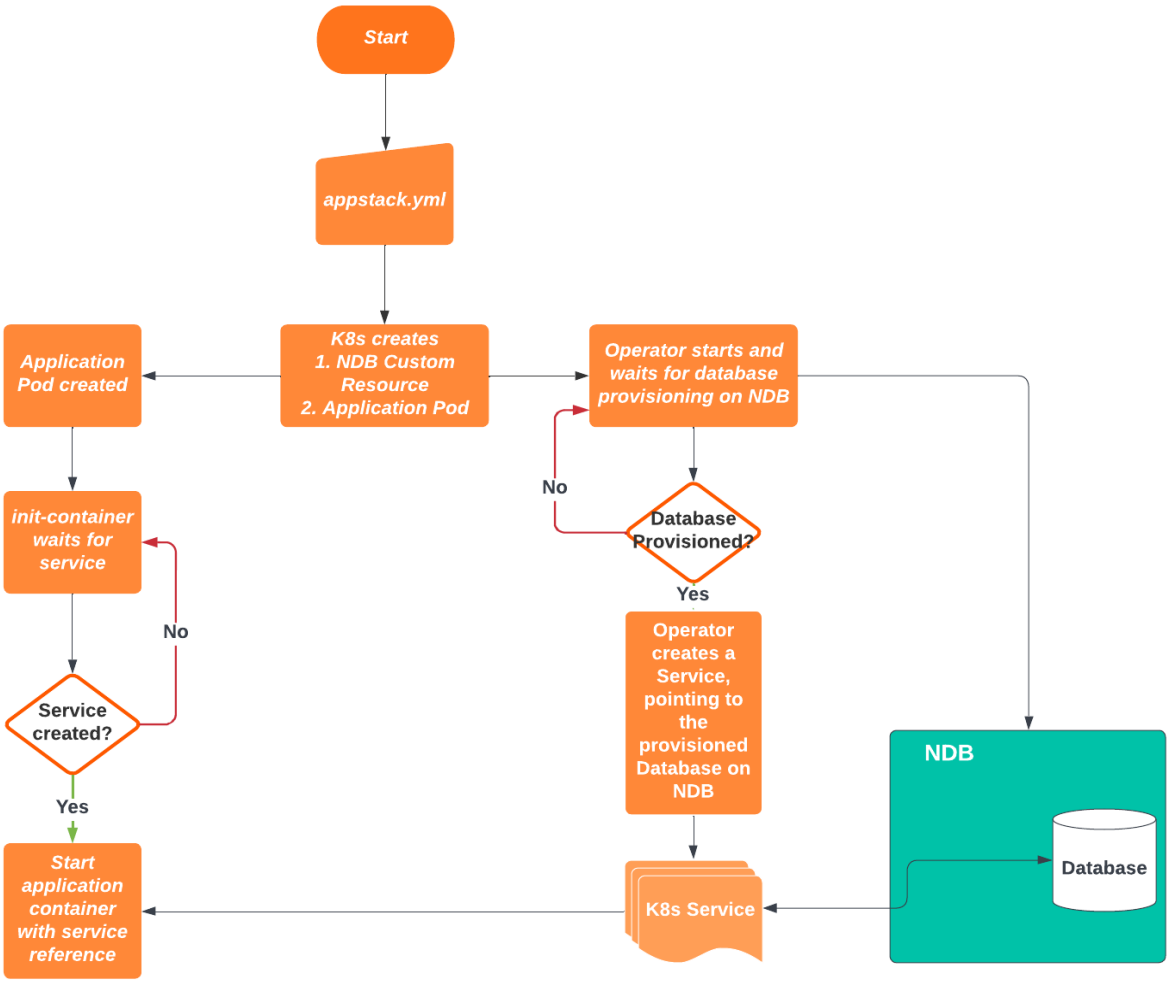
Making an application wait for DB before starting up can be done in the following ways:

a. Handling the DB connection wait in the application logic (Not recommended).
b. Failing the application (and hence the pod) in case of a DB connection failure so that Kubernetes can attempt to restart the application pod until it succeeds.
c. Using init-containers to wait on a Kubernetes pod / service.

Selected Approach:
Out of the enlisted methods, **creating a K8s service** that maps to the external NDB service endpoint is the real Kubernetes native method and is even recommended by Google Cloud in the series 'Kubernetes Best Practices'. This provides a decoupling between the database instance and the application pod. Also, an **init-container can wait for the service** to be ready and make our application container start up only after the service (and the underlying database

instance on NDB) is available. Using these two mechanisms in conjunction can help us achieve the automatic connectivity between the database instance and the application pod(s).

Flow Chart:



Best practices

Building an operator involves using Kubernetes API. Use of framework like Operator SDK is recommended [1]

- Design an operator in such a way that the application instance (in our case database instance and the connection of application to database) continues to run unaffected and effectively even if the operator is stopped or un-installed.
- Develop one operator per application (In our case one operator for whole of NDB) [7]
- Ensure backward compatibility and API interoperability

- Use built-in kubernetes primitives: replica set, services, field-export, etc.

References:

[1] Operator framework SDK: https://sdk.operatorframework.io/
[2] Operator Capability Levels:
https://sdk.operatorframework.io/docs/overview/operator-capabilities/
[3] Red hat openshift operators:
https://www.redhat.com/en/technologies/cloud-computing/openshift/what-are-openshift-operators
[4] Go Operator Tutorial:
https://sdk.operatorframework.io/docs/building-operators/golang/tutorial/
[5] CNCF White paper:
https://github.com/cncf/tag-app-delivery/blob/eece8f7307f2970f46f100f51932db106db46968/operator-wg/whitepaper/Operator-WhitePaper_v1-0.md
[6] Best practices for building kubernetes operator:
https://cloud.google.com/blog/products/containers-kubernetes/best-practices-for-building-kubernetes-operators-and-stateful-apps
[7] Top kubernetes Operators:
https://cloud.redhat.com/blog/top-kubernetes-operators-advancing-across-the-operator-capability-model
[8] Google Cloud's Kubernetes Best Practices
Mapping External Services
[9] Blog on Google Cloud's Kubernetes Best Practices
https://www.googblogs.com/kubernetes-best-practices-mapping-external-services/