Neptune: installation guide

Overview

Introduction

Neptune is composed of a set of microservices (Neptune Services) distributed as a Helm chart deployable on Kubernetes. If you don't have your own Kubernetes cluster deployed, our installer is able to set up a single-node cluster.

Aside from Kubernetes, Neptune requires the following components to function. All of them are delivered as part of the installation process so you don't need to worry about them.

- MySQL, version **5.7**
- Elasticsearch, version **6.5**

You can use the components delivered with Neptune, but in some cases it makes sense to use your own. E.g. in cloud deployments you can (and usually should) use a managed database.

Hardware requirements

Neptune services

Depending on how exactly users interact with Neptune, the hardware requirements can vary. A minimal recommended installation of Neptune requires 8 vCPUs and 32GB of RAM.

Storage

The requirement for storage is quite difficult to estimate, since it depends heavily on how much data users store in Neptune.

For simplest deployments, on a single VM, the recommended minimum is a single 1TB SSD.

Installation

Prerequisites

Neptune installer has the following prerequisites (some requirements may be optional, depending on the type of installation):

- A host with Ubuntu 18.04 installed (depending on the type of Neptune installation, it will serve as either a temporary host to run the installer on, or as the actual deployment host)
- Access to the Internet (required only for installation)
- Ansible (https://www.ansible.com/) in version **2.8.0** or higher. This is how to install it on Ubuntu 18.04

```
sudo apt-add-repository --yes ppa:ansible/ansible
sudo apt-get update
sudo apt-get install --yes 'ansible=2.9.*' --install-recommends
```

- `yq` (https://github.com/mikefarah/yq), installable with `pip install yq==2.10.1`
- Persistent storage for the data users push into Neptune:
    - When you configure the Neptune installer to deploy a single-node Kubernetes cluster, it needs an unformatted SSD attached to the VM where the installation takes place.
    - When you deploy Neptune to an existing Kubernetes cluster, it assumes that the cluster has the capability to provision storage. In particular, managed Kubernetes services (like Amazon's EKS, Google's GKE, Azure's AKS) usually have this capability out of the box.
- When deploying to an existing Kubernetes cluster, Neptune installer requires a configured `kubectl` executable available in `PATH`.

Installation artifacts

Neptune installer consists of two files.

The `neptune_installation_{version}.tgz` file needs to be unpacked. It contains a single directory `neptune_installation`.

The `configuration.yaml` file contains installation configuration. In particular, it can contain credentials to Neptune's docker registry and your license for using Neptune. **It should therefore be treated confidentially.**

Configuration

Before starting the installation you should modify the `configuration.yaml` file.

Aside from what was mentioned above, the `configuration.yaml` file may contain the following options (some are optional, depending on the installation type):

General configuration

This section describes some basic options that you need to fill before installing Neptune.

- `administrator_username` - the desired username of your administrator account. It can contain letters, numbers and hyphens. We recommend `administrator`.
- `administrator_password` - the password for the administrator account. Select a strong password here.
- `organization_name` - the name of your organization in Neptune. We suggest using something simple, related to your company. It can contain letters, numbers and hyphens.
- `deployment_type` - set to `local` in case of single-node deployment or `cluster` in case of deployment on existing Kubernetes cluster.

Kubernetes cluster configuration

Use options in this section to tell the Neptune installer how to interact with your Kubernetes cluster. Use these options only for `cluster` deployment.

- `kubeconfig_path` - path to an existing `kubectl` configuration file pointing to the cluster you want to deploy Neptune to (only for `cluster` deployment).
- `namespace` - Kubernetes namespace to which Neptune will be deployed. It's created if it doesn't exist. Defaults to `neptune`.
- `node_tolerations` - an array of node taint tolerations that Neptune installation can use. This is copied directly to Kubernetes manifests.

- `node_selector` - a key-value map that Neptune installation should use to select nodes. This is copied directly to Kubernetes manifests.

Storage configuration

Using options in this section you can configure how to provision storage for Neptune in your deployment.

Storage in `local` deployment

- `storage_device` - this should point to an SSD device. You can retrieve the value from the output of `lsblk` command. For a clean VM it's often `/dev/sdb`.

Storage in `cluster` deployment

If you decide to use MySQL or Elasticsearch delivered as part of Neptune installation or the default file storage, you need to specify the following options (they allow Neptune to provision disk space for the mentioned services):

- `ssd_storage_class` - name of the storage class to be used by MySQL and Elasticsearch services, and for file storage metadata.
- `hdd_storage_class` - name of the storage class to be used for file storage.

You can also choose to provide your own file storage by setting the following two options:

- `storage_type` - set to `Posix` to use your own Persistent Volume Claim specified in the `storage_pvc_name` option.
- `storage_pvc_name` - (obligatory if `storage_type` is set to `Posix`) the name of the Persistent Volume Claim present in the namespace to which Neptune will be installed. This claim needs to be of type `ReadWriteMany` and the underlying storage has to be POSIX-compliant.

Please note that changing the `storage_type` of an existing Neptune installation might cause Neptune to malfunction.

Database configuration

Neptune installer can set up a MySQL database as part of the installation process, however, if you prefer to provide one yourself, this section describes how to do that. This may be desirable, especially in cloud environments, where the storage used by the database is automatically scaled.

- `db_host` - External MySQL database host for Neptune to use.

- `db_port` - External MySQL database port. Optional, defaults to `3306`.
- `db_username` - The username of a user with access to all schemes used by Neptune. Required if `db_host` is set.
- `db_password` - The password of the user specified in `db_username`. Required if `db_host` is set.

Required database schemas

Before running the Ansible installer you should create the following database schemas and grant the user defined in `db_username` access to those schemas:

- `neptune_instance`
- `neptune_notifications`
- `neptune_discussions`
- `neptune_leaderboard`
- `neptune_keycloak`

Elasticsearch configuration

Neptune installer can set up an Elasticsearch instance as part of the installation process, however, if you prefer to provide one yourself, this section describes how to do that.

- `elasticsearch_address` - An address of the external Elasticsearch server host in format `https://<address>[:<http api port>]`. The `<http api port>` is optional and defaults to `443` for https connections.
- `elasticsearch_cluster_name` - The name of your Elasticsearch cluster. Defaults to `elasticsearch`, which is the default cluster name in Elasticsearch.

Additional configuration

In some cases you want to connect Neptune to your own identity management system (like LDAP). If your identity management system uses a certificate issued by your own Certificate Authority, you need to provide a set of trusted certificates to Neptune (so that it's able to verify the security of the connectIon).

- `trusted_certificates` - A list of absolute paths to files containing certificates that are to be trusted
- `keycloak_java_opts` - A rarely used option that allows to override some default behaviours of Java. It's a string containing Java options that are to be added to Keycloak

(the component responsible for connecting to an external identity management system). Example value: `"-Djdk.tls.client.protocols=TLSv1.0,TLSv1.1,TLSv1.2"`

Exposing Neptune

This section deals with exposing Neptune from your cluster/VM to the outside.

The first thing you need to decide is whether to use an ingress controller embedded within Neptune installer, or provide your own.

For more information on ingress controllers, see
https://kubernetes.io/docs/concepts/services-networking/ingress-controllers

- `ingress_controller` - this option determines whether the installer should deploy an ingress controller, or use an ingress controller already present in the cluster. If you're installing Neptune within a VM or on a fresh cluster, chances are you should go with the default.
    - `embedded` (the default) - the installer will install an NGINX ingress controller
    - `provided` - the ingress controller will not be installed. It's assumed that you have an ingress controller working in the cluster.
- `service_exposition_type` - valid only if `ingress_controller` is set to `embedded` (which it is, by default). Determines the way Neptune (actually Neptune's embedded ingress controller) is exposed. Can take one of the following values:
    - `LoadBalancer` - usually the preferred option when running in a cloud environment. It provisions a load balancer that points to Neptune.

      For more information on LoadBalancer services, see
      https://kubernetes.io/docs/concepts/services-networking/service/#loadbalancer

    - `NodePort` (the default) - commonly used when deploying Neptune on your own hardware (especially when using the single VM setup). In this case, Neptune will be exposed on port `30080` on every node of your Kubernetes cluster.

      For more information on NodePort services, see
      https://kubernetes.io/docs/concepts/services-networking/service/#nodeport

      See the "Load balancer configuration" section below on how to set up a load balancer that directs traffic to Neptune exposed as a `NodePort` service.

The options below require some understanding of Kubernetes' concepts. You can skip them if you're installing Neptune within a VM and plan to add a separate load balancer to strip TLS.

- `ingress_host` - defines a domain at which ingress controller will listen for requests to Neptune. translates directly to `.spec.rules[].host` in Neptune's ingress. Note that if this option is set, Neptune will be available only through this domain.
- `ingress_annotations` - a dictionary, that translates directly to `.metadata.annotations` in Neptune's ingress. You can use this option to configure behaviour specific to your `provided` ingress controller, like forcing an SSL redirect.
- `ingress_labels` - a dictionary, that translates directly to `.metadata.labels` in Neptune's ingress. Please, note, that Neptune installer might add a few labels. In particular, label names `app`, `chart`, `release` and `heritage` are reserved and providing them will cause installation failure.
- `ingress_tls_secret` - a name of a secret in the target namespace containing a valid TLS certificate and key for ingress to use. If defined, values of `ingress_tls_cert`, `ingress_tls_key` are ignored.
- `ingress_tls_cert`, `ingress_tls_key` - a base64-encoded TLS certificate and private key. Using those two values the installer will build a secret and provide it to `.spec.tls.secretName` in Neptune's ingress.

Installation process

The actual installation can be performed with a single command executed from within the `neptune_installation` directory.

```
ansible-playbook neptune.yml --extra-vars @/path/to/configuration.yaml
```

Remember about the `@` character in the above command! It's a part of ansible's syntax.

If you're installing Neptune with `deployment_type` set to `local`, you need to run the above command as root.

The installation will take from 5 minutes up to an hour, depending on the cluster's Internet connection speed - installation downloads Neptune's docker images.

Read the next section to learn more about exposing and accessing Neptune.

Load balancer configuration

If you're deploying Neptune on your own hardware with `ingress_controller` set to `embedded` (the default) and `service_exposition_type` set to `NodePort`, Neptune will be exposed on port `30080` on each node of the cluster. An external load balancer has to be configured to terminate SSL/TLS connection and forward traffic to all nodes to port `30080`.

Obviously, in case of deployment on a single VM, there's only a single node.

Any SSL-stripping load balancer should work, provided it adds `x-forwarded-for`, `x-forwarded-port`, `x-forwarded-proto` and `x-forwarded-host` headers properly and allows for long-lasting websocket connections.

Note, that neptune-client can send large HTTP request bodies, so putting (for nginx) `client_max_body_size 30m;` is required to make sure client works properly

We recommend using NGINX - you can use the sample configuration below as an inspiration:

```
server {
  listen 80;
  return 302 https://$host$request_uri;
}

upstream kubernetes-nodes {
  server 127.0.0.1:30080;
}

server {
  listen 80;
  return 302 https://$host$request_uri;
}

map $http_upgrade $connection_upgrade {
  default upgrade;
  '' close;
}

map $http_x_forwarded_proto $new_x_forwarded_proto {
  default $http_x_forwarded_proto;
  "" $scheme;
}

map $http_x_forwarded_host $new_http_x_forwarded_host {
  default $http_x_forwarded_host;
  "" $host;
}

map $http_x_forwarded_port $new_http_x_forwarded_port {
  default $http_x_forwarded_port;
```

```
  "" $server_port;
}

server {
  listen 443;
  server_name _;

  ssl_certificate /etc/nginx/certs/cert.crt;
  ssl_certificate_key /etc/nginx/certs/cert.key;
  ssl on;
  ssl_session_cache builtin:1000 shared:SSL:10m;
  ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
  ssl_ciphers HIGH:!aNULL:!eNULL:!EXPORT:!CAMELLIA:!DES:!MD5:!PSK:!RC4;
  ssl_prefer_server_ciphers on;

  access_log /var/log/nginx/access.log;

  location / {

    client_max_body_size 10G;

    # Enable websocket connections; all 3 headers are required.
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection $connection_upgrade;
    proxy_set_header Host $host;

    proxy_set_header X-Real-IP $remote_addr;

    # Required to read client-side address of Neptune.
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $new_x_forwarded_proto;
    proxy_set_header X-Forwarded-Host $new_http_x_forwarded_host;
    proxy_set_header X-Forwarded-Port $new_http_x_forwarded_port;

    proxy_pass http://kubernetes-nodes$request_uri;

    # Enable long-lived connections (like websockets and large uploads)
    proxy_read_timeout 86400;
    proxy_send_timeout 86400;

    proxy_http_version 1.1;
  }
}
```

Upgrading

If instead of installing Neptune, you're upgrading a previous installation to the new version, this section describes how to do it. Make sure to read the section below on upgrading between particular versions.

Be advised that upgrading to version `1.X` is supported only from the previous version: `1.(X-1)` (so, for example, upgrading to version `1.2` is supported only from version `1.1`). Do not attempt to upgrade if you have an older version installed.

Standard upgrade procedure

A typical upgrade procedure consists of just a few steps (somewhat similar to a first-time installation):

1. Perform a backup (see section below). This step is optional, but strongly recommended.
2. Download the installer (`neptune_installation_{version}.tgz`) with the new version to a host where you will run the installer and unpack the archive.
3. From within the `neptune_installation` directory, run

   ```
   ansible-playbook neptune.yml --extra-vars @/path/to/configuration.yaml
   ```

   Remember about the `@` character in the above command! It's a part of ansible's syntax.

   If you're upgrading Neptune with `deployment_type` set to `local`, you need to run the above command as root.

   Remember that the configuration file used previously contains your licence, credentials for downloading Neptune images, your Neptune administrator credentials and a few others - they shouldn't be changed without consulting with Neptune team.
4. That's it! After a while, you should be happily using the upgraded version of Neptune.

Effect on end-user

An upgrade should usually take from 5 minutes to an hour, depending on your Internet connection.

Most upgrades do not require an upgrade of the client libraries (like `neptune-client` and `neptune-notebooks`), but the upgrade is always recommended. Read the “Client packages” section to learn more.

Upgrades are often imperceptible to the end-user (the data scientist running experiments), however there’s always a risk of Neptune being unavailable for some time. This risk is mitigated on the neptune-client library side - running experiments should not be affected provided the downtime lasts less than 30 minutes. For absolute certainty that no data is lost, schedule an upgrade during a window when no experiments are running.

Backup

Remember to back up your Neptune deployment before upgrading.

In particular, this means creating a snapshot of any persistent volume and/or database that you provide for Neptune.

Client packages

The version of the `neptune-client` library compatible with this release is `0.10.2`.

It can be installed with:

```
pip install neptune-client==0.10.2
```

The version of the `neptune-notebooks` package compatible with this release is `0.9.0`.

It can be installed with:

```
pip install neptune-notebooks==0.9.0
```