Opensource Git Server with Gitea

Clone and Push Source Code

To clone and push code to your self-hosted Gitea instance running on Kubernetes, you'll need to use the SSH service exposed by Gitea. Here's how to do it:

1. Setup SSH Access

First, ensure your SSH key is added to Gitea so that you can use it for Git operations (clone, push, etc.).

Steps to Add SSH Key in Gitea:

1. Access the Gitea web interface at your custom domain.
2. Log in to your Gitea account.
3. Go to Settings > SSH Keys > Add Key.
4. Paste your public SSH key in the provided field and save.

2. Clone a Repository

Once your SSH key is added, you can clone repositories using SSH.

- Get the SSH URL for your repository:
  - Navigate to the repository in Gitea.
  - Click the Clone button.
  - Copy the SSH URL (e.g., git@git.introvesia.com:username/repository.git).
- Clone the repository: On your local machine, run:

```
git clone git@<domain>:username/repository.git
```

Replace `username/repository.git` with the actual repository path.

3. Push Code to the Repository

After cloning the repository, you can push your local changes to it.

1. Navigate to the cloned repository directory

```
cd repository
```

2. Make changes to your code and commit them:

```
git add .
git commit -m "Your commit message"
```

3. Push the changes to Gitea:

```
git push origin master
```

This will push your changes to the master branch (replace with the appropriate branch name if needed).

4. SSH Configuration (Optional)

If you want to avoid typing the full SSH URL every time, you can set up an SSH config file:

1. Open or create the SSH config file:

```
nano ~/.ssh/config
```

2. Add the following to it:

```
Host <domain>
  User git
  HostName <domain>
  Port 22
  IdentityFile ~/.ssh/id_rsa  # Path to your private key
```

Now you can clone and push using a shorter URL, like:

```
git clone git:username/repository.git
```

Accessing SSH via Kubernetes (if needed)

If you're accessing Gitea's SSH service through Kubernetes, ensure you're using the correct NodeIP and port, as exposed by the NodePort service:

- The SSH port will be Node-IP:32222 (or the port you set in the service definition).

Create Public SSH Key

1. Open Git Bash
2. Paste the text below, replacing the email used in the example with your GitHub email address.

```
ssh-keygen -t ed25519 -C "your_email@example.com"
```

If you are using a legacy system that doesn't support the Ed25519 algorithm, use:

```
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

This creates a new SSH key, using the provided email as a label.

When you're prompted to "Enter a file in which to save the key", you can press Enter to accept the default file location. Please note that if you created SSH keys previously, ssh-keygen may ask you to rewrite another key, in which case we recommend creating a custom-named SSH key. To do so, type the default file location and replace id_ALGORITHM with your custom key name.

3. At the prompt, type a secure passphrase.

```
Generating public/private ed25519 key pair.
Enter file in which to save the key (/c/Users/<user>/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/<user>/.ssh/id_ed25519
Your public key has been saved in /c/Users/<user>/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:<code> <email>
The key's randomart image is:
+--[ED25519 256]--+
|    ..+.. oo.    |
|     =E=  .o .   |
|    oo=o... o    |
|     ..Bo=.+ .   |
|      +.S.* +    |
|     . + * = .   |
|       o * +     |
|        +o= .    |
```

```
|       ..o++.  |
+----[SHA256]-----+
```

Error when Push Source Code

The error message

```
> git push origin main:main
error: RPC failed; HTTP 413 curl 22 The requested URL returned error: 413
send-pack: unexpected disconnect while reading sideband packet
fatal: the remote end hung up unexpectedly
Everything up-to-date
```

The error you're seeing is related to a Git push issue, where the data being pushed exceeds the allowed size for HTTP requests. Specifically, the HTTP 413 error indicates that the request is too large, and it can happen when pushing large commits or files.

Solution Steps

1. Increase Git HTTP Buffer Size

The most common fix is to increase the buffer size for Git's HTTP operations. You can do this by configuring Git with a larger buffer size on your local machine.

Run this command to increase the Git HTTP buffer size:

```
git config --global http.postBuffer 524288000  # 500MB
```

2. Configure Gitea to Allow Larger Pushes (Optional)

If increasing the buffer on your client doesn't work, you may need to adjust the settings in Gitea to allow larger pushes.

1. Edit the Gitea configuration (app.ini):

- If you're using the default Gitea deployment, this file is located in the Gitea container or in the /data/gitea/conf/ directory (depending on your setup).
- Look for the [server] section and set the following options:

```
[server]
; Max size for HTTP upload, in MB
MAX_UPLOAD_SIZE = 1024
```

2. Restart Gitea to apply the configuration changes:

```
kubectl rollout restart deployment gitea -n git
```

3. Increase Kubernetes Resource Limits (Optional)

If your Gitea pod has limited CPU or memory, it might be throttling large requests. You can increase the resource limits for the Gitea pod in your Kubernetes deployment configuration.

For example, add resource limits in the deployment YAML:

```
resources:
  requests:
    memory: "512Mi"
    cpu: "500m"
  limits:
    memory: "2Gi"
    cpu: "1"
```

Apply the changes with:

```
kubectl apply -f gitea-deployment.yaml
```

4. Push in Smaller Batches

If the file you're pushing is very large, you can try pushing smaller commits or files in batches to avoid hitting the request size limit.

After Applying Changes

- Run the git push origin main:main command again.
- If the problem persists, check if you are pushing very large files. You can use Git LFS (Large File Storage) for handling large binary files in your repository.

Directory Permission

The "Permission denied" error you're encountering indicates that the user running the Gitea service does not have sufficient permissions to create directories or files in the specified location (`/data/gitea/log`). This issue typically occurs due to incorrect file or directory permissions.

To resolve this issue, follow these steps:

Option 1: Adjust Directory Ownership and Permissions

1. **Check Current Ownership and Permissions**: First, verify the current ownership and permissions of the `/data/gitea` directory and its subdirectories:

```
ls -ld /data/gitea
ls -l /data/gitea/log
```

2. **Change Ownership to Gitea User**: If Gitea is running under a specific user (e.g., `gitea`), change the ownership of the directory and its contents to this user:

```
sudo chown -R gitea:gitea /data/gitea
```

3. **Adjust Permissions**: Set appropriate permissions to allow the `gitea` user (or the user running Gitea) to read, write, and execute in the directories:

```
sudo chmod -R 755 /data/gitea
```

4. **Retry Gitea Operations**: Restart or retry the operation that caused the permission denied error in Gitea.

Option 2: Use ACLs (Access Control Lists)

If you need more granular permissions control, you can use ACLs to grant specific permissions to the Gitea user:

1. **Install ACL Package** (if not already installed):

```
sudo apt update
sudo apt install acl
```

2. **Set ACLs on the Directory**:

   ```
   sudo setfacl -R -m u:gitea:rwx /data/gitea
   sudo setfacl -d -R -m u:gitea:rwx /data/gitea
   ```

   This command grants read, write, and execute permissions recursively (`-R`) to the `gitea` user (`u:gitea:rwx`) on `/data/gitea` and sets the default ACL (`-d`) so that any new files or directories inherit these permissions.
3. **Verify ACLs**: Check the ACLs to ensure they are correctly applied:

   ```
   getfacl /data/gitea
   ```

4. **Retry Gitea Operations**: Restart or retry the operation that caused the permission denied error in Gitea.

Additional Considerations

- **SELinux or AppArmor**: If your system uses SELinux or AppArmor, ensure that their policies allow Gitea to access the necessary directories.
- **Gitea Configuration**: Double-check the configuration files (`app.ini` or similar) to ensure they are correctly pointing to `/data/gitea/log` for logging.

By following these steps, you should be able to resolve the "Permission denied" issue for Gitea and allow it to create directories and files as required. Adjust the ownership, permissions, or ACLs according to your specific security and access requirements.

Create Dedicated User for Gitea

Benefits of Creating a Dedicated User and Group (`gitea`):

1. **Isolation**: By creating a separate user and group (`gitea`), you isolate Gitea's files and processes from other system users and applications. This isolation enhances security by limiting access to Gitea-related resources only to the `gitea` user and group.
2. **Security**: Assigning specific permissions to the `gitea` user and group allows you to control exactly what Gitea can access and modify on your system. This reduces the risk of unintended modifications or security breaches.
3. **Standardization**: Using a dedicated user and group (`gitea`) for Gitea installations promotes consistency and standardization across different deployments. It simplifies management and troubleshooting, especially in environments with multiple applications and users.
4. **Compatibility**: Many applications and services, including Gitea, are designed to run under a specific user and group for optimal compatibility and security configurations.

Steps to Create `gitea` User and Group:

Here’s how you can create the `gitea` user and group on your system:

1. Create the `gitea` Group:

```
sudo groupadd -r gitea
```

- `-r` flag creates a system group, which is often preferred for service-related accounts.

2. Create the `gitea` User:

```
sudo useradd -r -g gitea -d /var/lib/gitea -s /bin/bash gitea
```

- `-r` flag creates a system user, suitable for service accounts.
- `-g gitea` assigns the `gitea` group as the primary group for the user.

- `-d /var/lib/gitea` specifies the home directory for the `gitea` user (adjust as needed).
- `-s /bin/bash` sets the default shell for the `gitea` user (adjust as needed).

3. Set Permissions:

Ensure that directories and files relevant to Gitea (e.g., `/var/lib/gitea`, `/data/gitea`, or specific paths you use) are owned by the `gitea` user and group (`gitea:gitea`). Adjust permissions as necessary to allow Gitea to read, write, and execute where required.

Example Usage in NFS Setup:

- **NFS Server**: Set permissions and ownership (`chown` and `chmod`) for directories shared with NFS to be accessible by the `gitea` user and group.
- **NFS Client (Gitea Server)**: Mount NFS shares using options (`uid`, `gid`) that correspond to the `gitea` user and group (`gitea:gitea`), ensuring that Gitea has proper access to the shared directories.

Conclusion:

Creating a dedicated user and group (`gitea`) specifically for Gitea installations enhances security, isolation, and compatibility with other services. It’s a recommended practice to follow when setting up applications like Gitea on your system, ensuring clear separation of privileges and streamlined management. Adjust configurations based on your specific deployment needs and security policies to achieve optimal performance and security for Gitea and other applications.

Deploying Gitea on Docker

Deploying Gitea on Docker involves setting up the Gitea application along with a database backend. Here’s a step-by-step guide to deploy Gitea using Docker Compose:

Step 1: Prepare Docker Compose File

Create a `docker-compose.yml` file to define the services (Gitea and MySQL in this case):

```
version: '3'

services:
  server:
    image: gitea/gitea:latest
    environment:
      - USER_UID=1000  # Replace with your host user UID if necessary
      - USER_GID=1000  # Replace with your host group GID if necessary
    restart: always
    volumes:
      - ./gitea:/data
    ports:
      - "3000:3000"
      - "2222:22"  # SSH port (optional, adjust as needed)
    depends_on:
      - db
    networks:
      - gitea_network

  db:
    image: mysql:5.7
    environment:
      MYSQL_DATABASE: gitea
      MYSQL_USER: gitea
      MYSQL_PASSWORD: password
      MYSQL_ROOT_PASSWORD: root_password
    volumes:
```

```
      - ./mysql:/var/lib/mysql
    restart: always
    networks:
      - gitea_network

networks:
  gitea_network:
    driver: bridge
```

Step 2: Configure Docker Compose File

- **Gitea (`server` service)**:
  - `image`: Pulls the latest Gitea Docker image.
  - `environment`: Sets user UID and GID for file permissions (adjust if necessary).
  - `volumes`: Mounts a local directory `./gitea` to persist Gitea data.
  - `ports`: Maps container ports `3000` (HTTP) and `2222` (SSH) to host ports (adjust as needed).
  - `depends_on`: Ensures the `db` service starts before Gitea.
  - `networks`: Connects to a custom bridge network `gitea_network`.
- **MySQL (`db` service)**:
  - `image`: Pulls the MySQL 5.7 Docker image.
  - `environment`: Sets MySQL database name, user, and passwords.
  - `volumes`: Mounts a local directory `./mysql` to persist MySQL data.
  - `restart`: Always restarts the MySQL container if it stops unexpectedly.
  - `networks`: Connects to the `gitea_network` for communication with Gitea.

Step 3: Deploy Gitea

Run the following command in the directory containing your `docker-compose.yml` file:

```
docker-compose up -d
```

This command starts both Gitea and MySQL containers in detached mode (`-d`), allowing them to run in the background.

Step 4: Access Gitea

After deployment, access Gitea through your browser at `http://localhost:3000`. Replace `localhost` with your server's IP address or domain name if accessing remotely.

Step 5: Initial Setup

Follow the on-screen instructions to complete the initial setup of Gitea, including setting up an admin account and configuring basic settings.

Notes:

- **Data Persistence**: Data for Gitea and MySQL will persist in local directories `./gitea` and `./mysql`. Adjust paths and volumes as needed.
- **Customization**: Customize environment variables and ports according to your specific requirements.
- **Security**: Consider securing Gitea with HTTPS using Nginx or another reverse proxy and obtaining SSL/TLS certificates.

This setup provides a basic deployment of Gitea using Docker Compose. Modify as per your environment and security requirements for production use.