Semester Project Protocol
Milestone 3

DevOps und Cloud Computing
Lukas Aichbauer

Team 1

Alexander Nachtmann
Markus Rösner
Max Sinnl
Stephanie Rauscher

Table of Contents

Infrastructure Overview

In contrast to our Infrastructure Specification Protocol for Milestone 2, we have made a few changes to our infrastructure. All servers use Ubuntu version 22.04 as their operating system.

Instance Name	Container Type	IP	Domain	Packages	Subnet
Gitlab Server	T2.medium	Private IP: 10.0.1.49 Public IP (elastic): 34.201.193.255	git.team01.at	GitLab 16.6.1	Public
Bastion Host	T2.micro	Private IP: 10.0.2.61 Public IP always changes after restart	bastionhost.team01.at	-	Public
Gitlab Runner	T2.small	Private IP: 10.0.2.70	gitrunner.team01.at		Private
Main DNS-Server	T2.micro	Private IP: 10.0.2.49	dnsmain.team01.at	BIND 9	Private
Backup DNS Server	T2.micro	Private IP: 10.0.2.48	dnsbackup.team01.at	BIND 9	Private
LDAP Server	T2.micro	Private IP: 10.0.2.80	ldap.team01.at	OpenLDAP	Private

Routing Tables

Public Subnet Route Table

Destination IP	Target name
10.0.0.0/16	VPC
0.0.0.0/0	Internet Gateway

Destination IP 10.0.0.0/16 - Target: VPC
This entry indicates that any traffic destined for an IP address within the 10.0.0.0/16 range) should be routed internally within the VPC. It essentially means that all IPs in this range are part of the VPC network. This is a standard entry for internal network routing within the VPC.

Destination IP 0.0.0.0/0 - Target: Internet Gateway
This entry is for routing all other traffic (not destined for the internal VPC network) to the Internet Gateway. The destination IP 0.0.0.0/0 represents all IP addresses not covered by more specific routes.

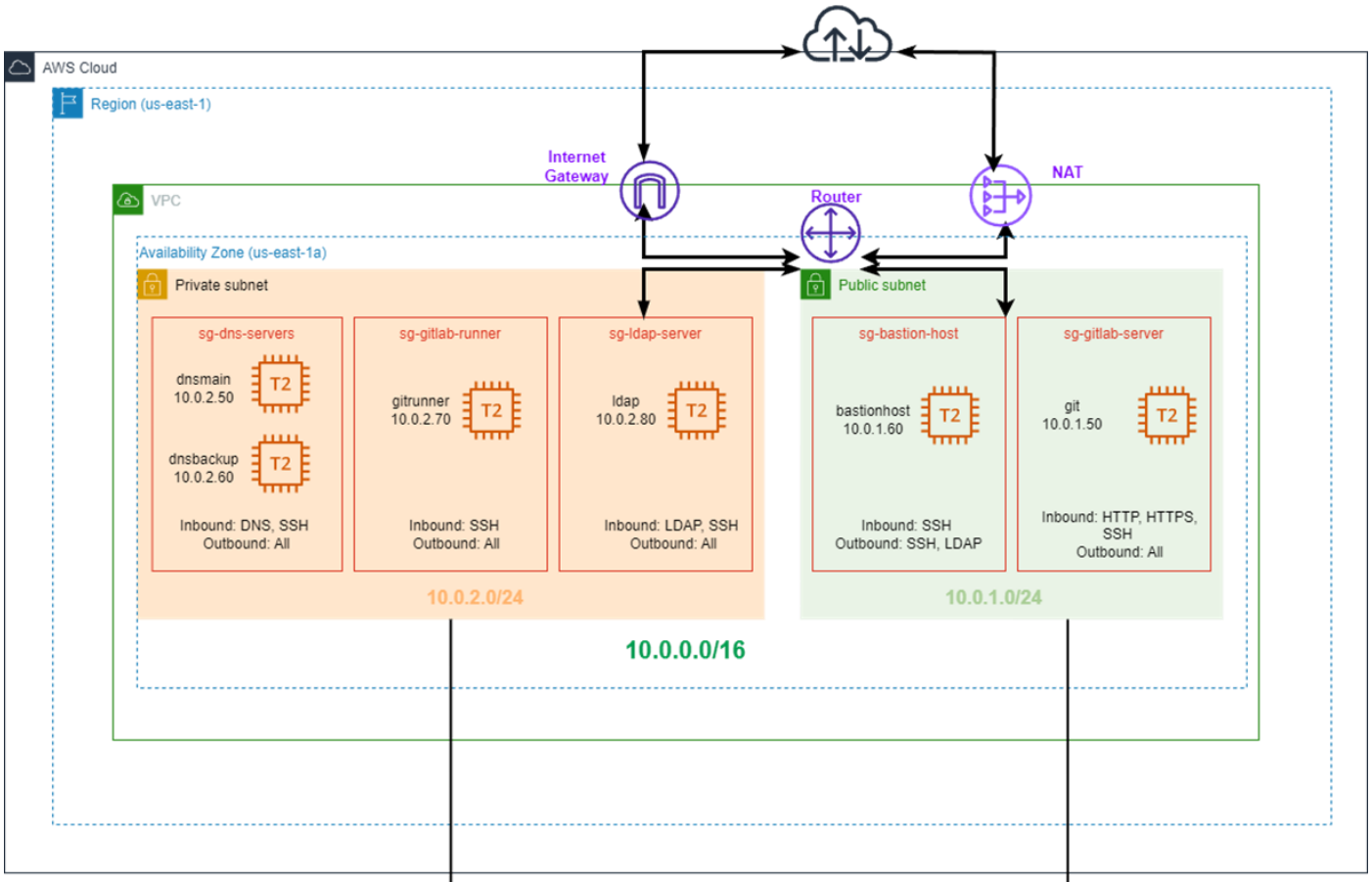
Private Subnet Route Table

Destination IP	Target name
0.0.0.0/0	NAT Gateway
10.0.0.0/16	VPC

Destination IP 0.0.0.0/0 - Target: NAT Gateway
This route directs all traffic that is not for local destinations (i.e., any destination not within the VPC) to a Network Address Translation (NAT) Gateway. It's used for allowing instances in the Private Subnet to access the internet for updates or downloads, but not allowing incoming internet traffic to initiate connections with those instances.

Destination IP 10.0.0.0/16 - Target: VPC
Similar to the Public Subnet, this entry ensures that traffic destined for the VPC's internal IP range is kept within the VPC network.

Network Topology Graph



Private Route Table	
Destination	*Target*
0.0.0.0/0	NAT
10.0.0.0/16	VPC

Public Route Table	
Destination	*Target*
10.0.0.0/16	VPC
0.0.0.0/0	igw

Security Groups

The (updated) Security Groups we used for our Project were:

Inbound Rules

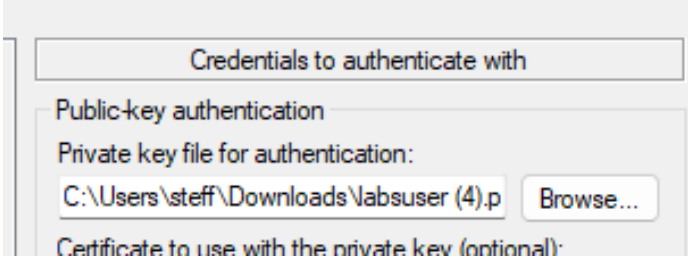
SG Name	Type	Port Range	Source
scg-gitlab-server	HTTP, HTTPS, SSH	TCP 80, TCP443 TCP 22	0.0.0.0/0 0.0.0.0/0 10.0.1.61/32
scg-bastion-host	SSH, SSH, SSH	TCP 22 TCP 22 TCP 22	10.0.1.0/24 10.0.2.0/24 84.115.XX.XX/32 (public IP of client that is connecting to Bastion Host, censored for privacy reasons)
scg-gitlab-runner	SSH	TCP 22	10.0.1.61/32
scg-dnsservers	DNS, DNS, SSH	TCP/UDP 53 TCP/UDP 53 TCP 22	10.0.2.0/24 10.0.1.0/24 10.0.1.61/32
scg-ldapserver	SSH, LDAP, LDAP	TCP 22 TCP 389 TCP 389	10.0.1.61/32 10.0.2.0/24 10.0.1.0/24

Outbound Rules

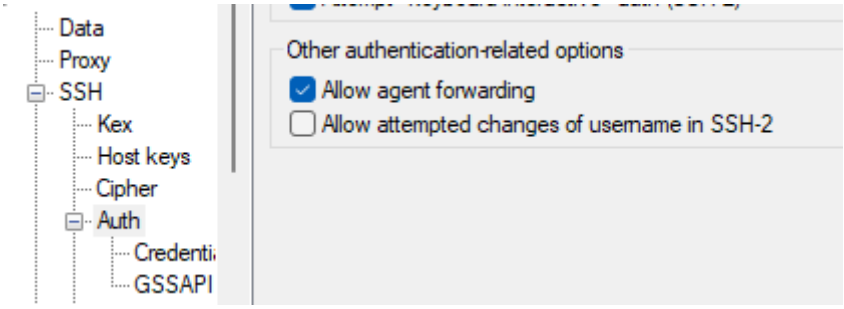
SG Name	Type	Port Range	Source
scg-gitlab-server	All Destinations	-	0.0.0.0/0
scg-bastion-host	SSH SSH	TCP 22 TCP 22	10.0.1.0/24 10.0.2.0/24
scg-gitlab-runner	All Destinations	-	0.0.0.0/0
scg-dns-servers	All Destinations	-	0.0.0.0/0
scg-ldapserver	All Destinations	-	0.0.0.0/0

Configuration of Services

First, we start by connecting to the Bastion Host (Jump Server) via SSH. We upload the .ppk file from AWS as our credentials in SSH, under 'Auth' -> 'Credentials'.



and tick the box labeled 'Allow Agent Forwarding' in SSH -> Auth.



We use the public IP of the Bastion Host to connect to the instance. From there, we can connect to every other instance in our infrastructure using the command *ssh ubuntu@IP*.

Configuration of the Primary DNS Server

From the Jump Server, we connect to the Primary DNS Server using the command `ssh ubuntu@10.0.2.49` (10.0.2.49 being the IP of our Primary DNS Server). From there, we use the commands

```
sudo apt update
```

```
sudo apt install bind9
```

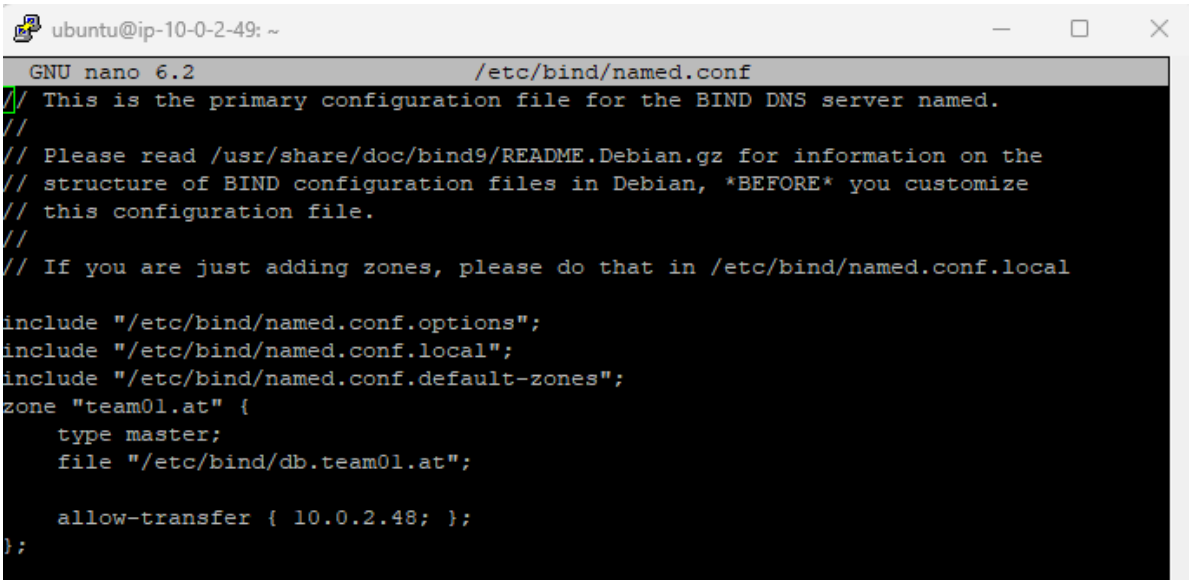
for installing BIND on our DNS Server.

To ensure BIND is running, we use:

```
sudo systemctl start named
sudo systemctl enable named
```

Then we edit the **/etc/bind/named.conf** file to define our zones:

```
sudo nano /etc/bind/named.conf
```



Next, we create our zone file **/etc/bind/db.team01.at** with all of our DNS Zones:

```
sudo nano /etc/bind/db.team01.at
```

```
ubuntu@ip-10-0-2-49: ~
  GNU nano 6.2                    /etc/bind/db.team01.at
;
; BIND data file for local loopback interface
;
$TTL    604800
@       IN      SOA     dnsmain.team01.at. admin.team01.at. (
                              6         ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                         604800 )       ; Negative Cache TTL
;
@       IN      NS      dnsmain.team01.at.
@       IN      NS      dnsbackup.team01.at.
dnsmain IN      A       10.0.2.49
dnsbackup       IN      A       10.0.2.48
bastionhost IN  A       10.0.1.61
git     IN      A       10.0.1.49
git     IN     A       54.88.228.103
gitrunner IN   A       10.0.2.70
ldap    IN     A       10.0.2.80
                                [ Read 21 lines ]
^G Help         ^O Write Out    ^W Where Is     ^K Cut          ^T Execute      ^C Location
^X Exit         ^R Read File    ^\ Replace      ^U Paste        ^J Justify      ^/ Go To Line
```

we check the configuration for errors:

```
sudo named-checkconf
sudo named-checkzone team01.at /etc/bind/zones/db.team01.at
```

Restart BIND to apply changes:

```
sudo systemctl restart named
```

To avoid local resolving Problems, we navigate to the File

```
ubuntu@ip-10-0-2-49: ~
  GNU nano 6.2                /etc/systemd/resolved.conf
#  This file is part of systemd.
#
#  systemd is free software; you can redistribute it and/or modify it under the
#  terms of the GNU Lesser General Public License as published by the Free
#  Software Foundation; either version 2.1 of the License, or (at your option)
#  any later version.
#
# Entries in this file show the compile time defaults. Local configuration
# should be created by either modifying this file, or by creating "drop-ins" in
# the resolved.conf.d/ subdirectory. The latter is generally recommended.
# Defaults can be restored by simply deleting this file and all drop-ins.
#
# Use 'systemd-analyze cat-config systemd/resolved.conf' to display the full config.
#
# See resolved.conf(5) for details.

[Resolve]
# Some examples of DNS servers which may be used for DNS= and FallbackDNS=:
# Cloudflare: 1.1.1.1#cloudflare-dns.com 1.0.0.1#cloudflare-dns.com 2606:4700:4700:>
# Google:     8.8.8.8#dns.google 8.8.4.4#dns.google 2001:4860:4860::8888#dns.google>
# Quad9:      9.9.9.9#dns.quad9.net 149.112.112.112#dns.quad9.net 2620:fe::fe#dns.q>
DNS=10.0.2.49 10.0.2.48
#FallbackDNS=
#Domains=
#DNSSEC=no
#DNSOverTLS=no
```

```
sudo nano /etc/systemd/resolved.conf
```

We will edit the file and, under the 'DNS=' section, insert the IPs of both our DNS servers.

We also create the File `sudo nano /etc/netplan/99-custom-dns.yaml` with the following content

```
ubuntu@ip-10-0-2-49: ~
  GNU nano 6.2                /etc/netplan/99-custom-dns.yaml
network:
  version: 2
  ethernets:
    eth0:
      nameservers:
        addresses: [10.0.2.49, 10.0.2.48]
      dhcp4-overrides:
        use-dns: false
        use-domains: false
```

use these commands to apply changes

```
sudo netplan generate
sudo netplan apply
```

Restart BIND

```
sudo systemctl restart bind9
```

Testing

To test if our zones are set up correctly, we use the following command from another server in our VPC (from our Gitlab Server with the IP 10.0.1.49):

```
nslookup bastionhost.team01.at 10.0.2.49
```

```
ubuntu@ip-10-0-1-49:~$ nslookup bastionhost.team01.at 10.0.2.49
Server:         10.0.2.49
Address:        10.0.2.49#53

Name:   bastionhost.team01.at
Address: 10.0.1.61
```

We also try it on our DNS Server

```
nslookup git.team01.at 10.0.2.49
```

```
ubuntu@ip-10-0-2-49:~$ nslookup git.team01.at 10.0.2.49
Server:         10.0.2.49
Address:        10.0.2.49#53

Name:   git.team01.at
Address: 54.88.228.103
Name:   git.team01.at
Address: 10.0.1.49
```

And we query our Main DNS Server one last time

```
dig bastionhost.team01.at @10.0.2.49
```

```
ubuntu@ip-10-0-2-49: ~

ubuntu@ip-10-0-2-49:~$ dig bastionhost.team01.at @10.0.2.49

; <<>> DiG 9.18.18-0ubuntu0.22.04.1-Ubuntu <<>> bastionhost.team01.at @10.0.2.49
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 46824
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
; COOKIE: 6a5d291cfc5c2c40010000006576da6dc874d4765e6e8c71 (good)
;; QUESTION SECTION:
;bastionhost.team01.at.         IN      A

;; ANSWER SECTION:
bastionhost.team01.at.  604800  IN      A       10.0.1.61

;; Query time: 0 msec
;; SERVER: 10.0.2.49#53(10.0.2.49) (UDP)
;; WHEN: Mon Dec 11 09:46:21 UTC 2023
;; MSG SIZE  rcvd: 94

ubuntu@ip-10-0-2-49:~$
```

Since we can see it returning the IP of our Bastion Host and Gitlab Server, our setup works as planned.

For also making DNS reverse lookup work, we do the following:

We locate the **/etc/bind/named.conf.local** file

```
sudo nano /etc/bind/named.conf.local
```

and edit it like this:

```
GNU nano 6.2                    /etc/bind/named.conf.local
//
// Do any local configuration here
//

// Consider adding the 1918 zones here, if they are not used in your
// organization
//include "/etc/bind/zones.rfc1918";
//
// Do any local configuration here
zone "2.0.10.in-addr.arpa" {
    type master;
    file "/etc/bind/db.2.0.10";
    allow-transfer { 10.0.2.48; };
};

zone "1.0.10.in-addr.arpa" {
    type master;
    file "/etc/bind/db.1.0.10";
    allow-transfer { 10.0.2.48; };
};
```

The file **db.2.0.10** is used for defining reverse zone files in the private subnet, while the file **db.1.0.10** is for defining reverse zone files in the public subnet. We specify 'type master;' because this server is our main DNS, and we set 'allow-transfer' to enable the transfer of all zone files later to our backup DNS server.

We create both files and then edit them with the reverse zone files for our subnets:

```
sudo nano /etc/bind/db.2.0.10
```

Private Subnet

```
ubuntu@ip-10-0-2-49: ~
  GNU nano 6.2                    /etc/bind/db.2.0.10
; BIND reverse data file for local loopback interface
;
$TTL    604800
@       IN      SOA     dnsmain.team01.at. admin.team01.at. (
                            2           ; Serial
                       604800           ; Refresh
                        86400           ; Retry
                      2419200           ; Expire
                       604800 )         ; Negative Cache TTL
;
@       IN      NS      dnsmain.team01.at.
49      IN      PTR     dnsmain.team01.at.
48      IN      PTR     dnsbackup.team01.at.
70      IN     PTR     gitrunner.team01.at.
30      IN     PTR     ldap.team01.at.
```

```
sudo nano /etc/bind/db.1.0.10
```

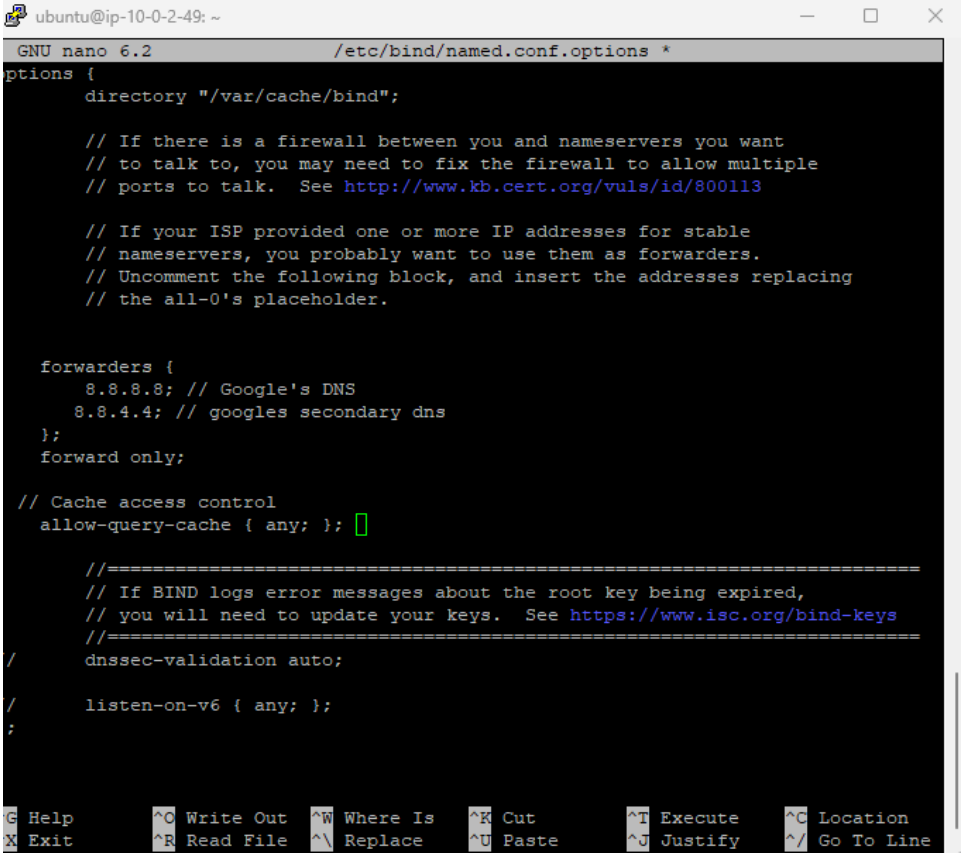
Public Subnet

```
ubuntu@ip-10-0-2-49: ~
  GNU nano 6.2                    /etc/bind/db.1.0.10
$TTL    604800
@       IN      SOA     dnsmain.team01.at. admin.team01.at. (
                            3           ; Serial
                       604800           ; Refresh
                        86400           ; Retry
                      2419200           ; Expire
                       604800 )         ; Negative Cache TTL
;
@       IN      NS      dnsmain.team01.at.
@       IN      NS      dnsbackup.team01.at.
51      IN      PTR     bastionhost.team01.at.
49      IN      PTR     git.team01.at.
```

We will also configure our DNS server to forward queries to external DNS servers. If there is no local record for the external domain, the query is forwarded to the configured forwarder DNS servers, which in our case are Google's DNS servers at 8.8.8.8 and 8.8.4.4. Once Google's DNS servers retrieve the response, they send this information back to our DNS server.

Additionally, we use 'allow-query-cache' to avoid DNS cache problems.

```
sudo nano /etc/bind/named.conf.options
```



Testing

We will now verify the functionality of reverse lookup and DNS forwarding.

Reverse Lookup:

`dig -x 10.0.2.70 @10.0.2.49` (from Bastion Host)

```
ubuntu@ip-10-0-1-61: ~
Connection to 10.0.2.49 closed.
ubuntu@ip-10-0-1-61:~$ dig -x 10.0.2.70 @10.0.2.49

; <<>> DiG 9.18.12-0ubuntu0.22.04.3-Ubuntu <<>> -x 10.0.2.70 @10.0.2.49
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 60145
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
; COOKIE: e48251b18ddd850b010000006576ddac7b4e7d796fd90ca7 (good)
;; QUESTION SECTION:
;70.2.0.10.in-addr.arpa.                IN      PTR

;; ANSWER SECTION:
70.2.0.10.in-addr.arpa. 604800  IN      PTR     gitrunner.team01.at.

;; Query time: 0 msec
;; SERVER: 10.0.2.49#53(10.0.2.49) (UDP)
;; WHEN: Mon Dec 11 10:00:12 UTC 2023
;; MSG SIZE  rcvd: 112

ubuntu@ip-10-0-1-61:~$
```

`dig +short -x 10.0.2.70 @10.0.2.49` (from Bastion Host)

```
ubuntu@ip-10-0-1-61:~$ dig +short -x 10.0.2.70 @10.0.2.49
gitrunner.team01.at.
ubuntu@ip-10-0-1-61:~$
```

`dig +short -x 10.0.2.70 @10.0.2.49` (from Gitlab Server)

`dig +short -x 10.0.1.61 @10.0.2.49` (from Gitlab Server)

```
ubuntu@ip-10-0-1-49:~$ dig +short -x 10.0.2.70 @10.0.2.49
gitrunner.team01.at.
ubuntu@ip-10-0-1-49:~$ dig +short -x 10.0.1.61 @10.0.2.49
bastionhost.team01.at.
ubuntu@ip-10-0-1-49:~$
```

Forward DNS Lookup for an External Domain

```
dig www.google.com @10.0.2.49
```

```
ubuntu@ip-10-0-1-49: ~
buntu@ip-10-0-1-49:~$ dig www.google.com @10.0.2.49

 <<>> DiG 9.18.12-0ubuntu0.22.04.3-Ubuntu <<>> www.google.com @10.0.2.49
; global options: +cmd
; Got answer:
; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 26840
; flags: qr rd ra; QUERY: 1, ANSWER: 6, AUTHORITY: 0, ADDITIONAL: 1

; OPT PSEUDOSECTION:
 EDNS: version: 0, flags:; udp: 1232
 COOKIE: 13a1b1b3d62f5751010000006576f3a1dfda1cc4ac21525d (good)
; QUESTION SECTION:
www.google.com.                         IN      A

; ANSWER SECTION:
ww.google.com.          247     IN      A       172.253.63.99
ww.google.com.          247     IN      A       172.253.63.103
ww.google.com.          247     IN      A       172.253.63.104
ww.google.com.          247     IN      A       172.253.63.105
ww.google.com.          247     IN      A       172.253.63.106
ww.google.com.          247     IN      A       172.253.63.147

; Query time: 7 msec
; SERVER: 10.0.2.49#53(10.0.2.49) (UDP)
; WHEN: Mon Dec 11 11:33:53 UTC 2023
; MSG SIZE  rcvd: 167
```

From these tests, we can see that the reverse lookup for our main DNS server is functioning properly.

Configuring the Backup DNS Server

We disconnect from our Primary DNS using `exit` and connect to our Backup DNS using:

```
ssh ubuntu@10.0.2.48
```

We start configuring the Backup DNS Server like we did with the primary Server previously with the following commands

```
sudo apt update
```

and

```
sudo apt install bind9
```

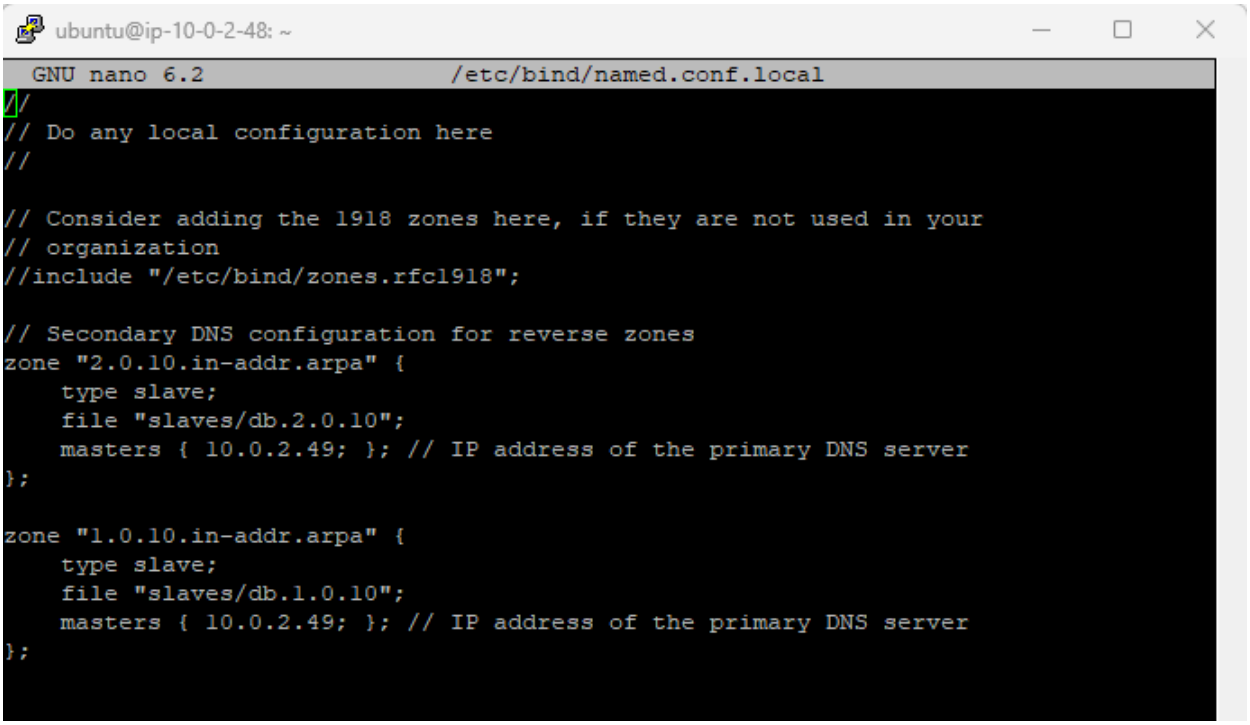
for installing BIND on our DNS Server.

We create a new directory

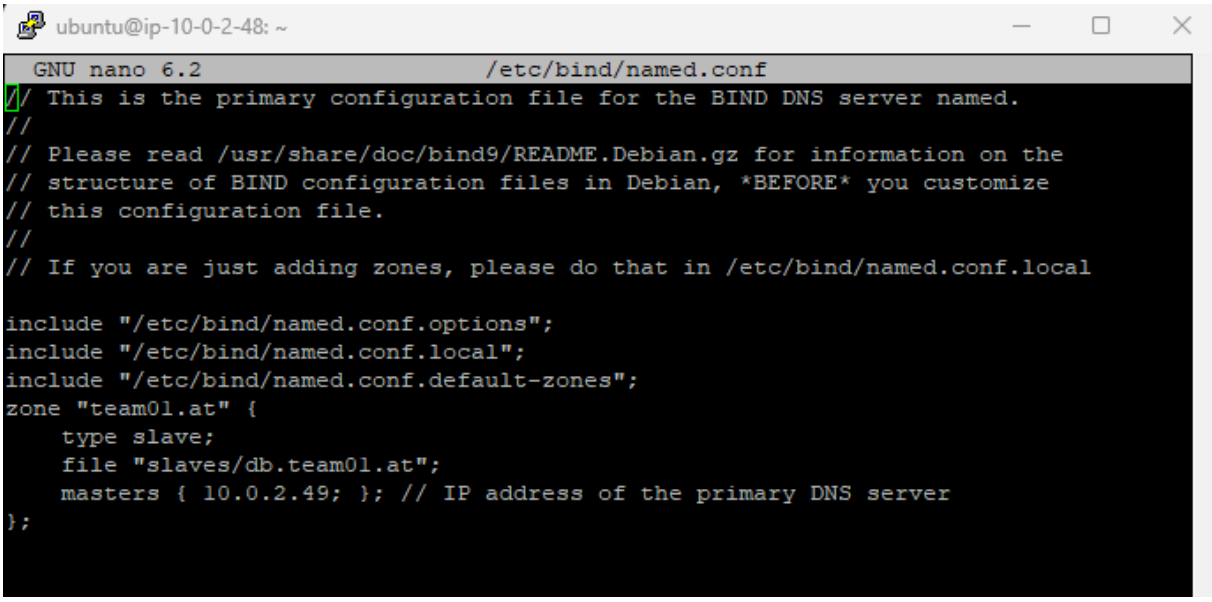
```
sudo mkdir /var/cache/bind/slaves
sudo chown bind:bind /var/cache/bind/slaves
```

Then, we modify our **named.conf.local** file to match the configuration shown in the screenshot. By setting 'type slave;', we designate this server as our Backup DNS (Slave) Server.

```
sudo nano /etc/bind/named.conf.local
```



We also edit the **named.conf** file




```
sudo nano /etc/bind/named.conf
```

There is no need to copy paste the zone files, since they will transfer automatically. After editing both files, we restart BIND

```
sudo systemctl restart bind9
```

If the zone transfer worked, the files on the secondary server should look similar to this

```
ubuntu@ip-10-0-2-48: ~
  GNU nano 6.2                    /var/cache/bind/slaves/db.team01.at
^@^@^@^B^@^@^@^Aes□□^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@Y^@^A^@^F^@^@^@       :□^@
^@^A=^@^@^@/^@^A^@^A^@^@^@      :□^@^@^@^A^@^U  dnsbackup^Fteam01^Bat^@^@^D
^@^B0^@^@^@-^@^A^@^A^@^@^@      :□^@^@^@^A^@^S^Gdnsmain^Fteam01^Bat^@^@^D
^@^B1^@^@^@)^@^A^@^A^@^@^@      :□^@^@^@^A^@^O^Cgit^Fteam01^Bat^@^@^D
^@^A1^@^@^@/^@^A^@^A^@^@^@      :□^@^@^@^A^@^U  gitrunner^Fteam01^Bat^@^@^D
^@^BF^@^@^@*^@^A^@^A^@^@^@      :□^@^@^@^A^@^P^Dldap^Fteam01^Bat^@^@^D
^@^BP
```

Testing

We test if our Backup DNS Server (10.0.2.48) can take queries as well.

```
nslookup git.team01.at 10.0.2.48
```

```
ubuntu@ip-10-0-1-49:~$ nslookup git.team01.at 10.0.2.48
Server:         10.0.2.48
Address:        10.0.2.48#53

Name:   git.team01.at
Address: 10.0.1.49

ubuntu@ip-10-0-1-49:~$
```

```
dig -x 10.0.1.61 @10.0.2.48
```

```
ubuntu@ip-10-0-1-49:~$ dig -x 10.0.1.61 @10.0.2.48

; <<>> DiG 9.18.12-0ubuntu0.22.04.3-Ubuntu <<>> -x 10.0.1.61 @10.0.2.48
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 7576
;; flags: qr aa rd; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
; COOKIE: 477b96f4dcacf39c010000006576f57322e3f32253fb9200 (good)
;; QUESTION SECTION:
;61.1.0.10.in-addr.arpa.                IN      PTR

;; ANSWER SECTION:
61.1.0.10.in-addr.arpa. 604800  IN      PTR     bastionhost.team01.at.

;; Query time: 0 msec
;; SERVER: 10.0.2.48#53(10.0.2.48) (UDP)
;; WHEN: Mon Dec 11 11:41:39 UTC 2023
;; MSG SIZE  rcvd: 114

ubuntu@ip-10-0-1-49:~$
```

As evidenced by these outputs, the Backup Server also responds to DNS queries, confirming that the zone transfer was successful.

Gitlab Server

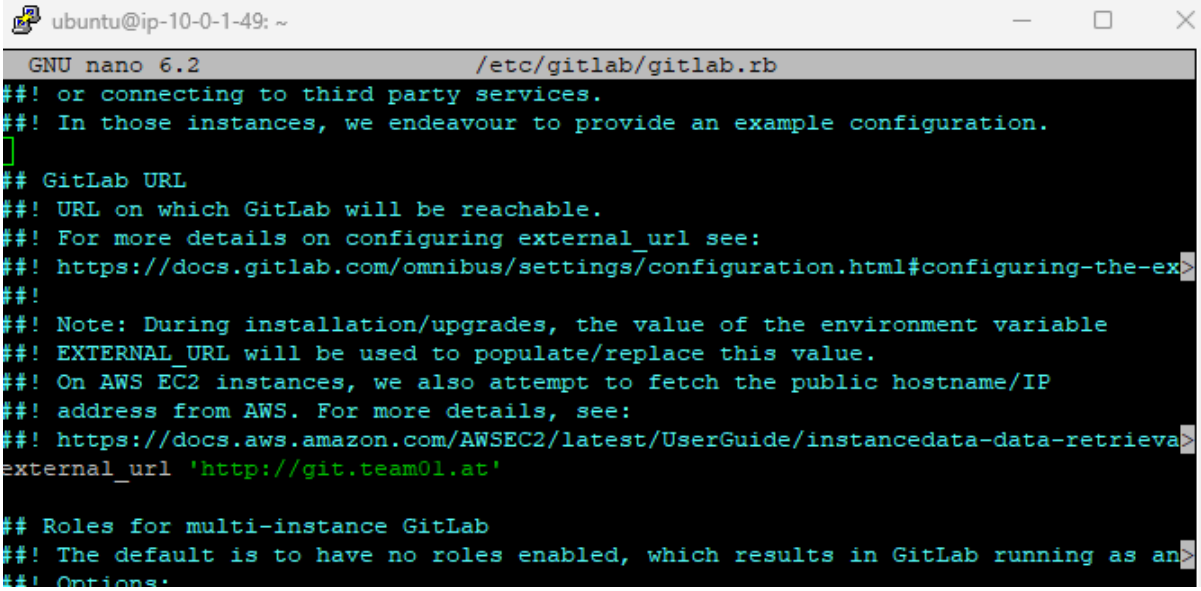
Add the GitLab package repository and install the package:

```
curl https://packages.gitlab.com/install/repositories/gitlab/gitlab-ce/script.deb.sh | sudo bash
sudo apt-get install gitlab-ce
```

Locate to

```
sudo nano /etc/gitlab/gitlab.rb
```

And edit the file at the section where you are prompted to enter your URL (Your GitLab domain)



Now GitLab is installed, and the interface should already be visible at your public IP for the GitLab Server.

To change the password for the root user, enabling login access, follow these steps:

Open the GitLab Rails Console

```
sudo gitlab-rails console -e production
```

Change the root Password

```
user = User.find_by(username: 'root')
user.password = 'new_password'
user.password_confirmation = 'new_password'
user.save!
```

Exit the Rails Console

```
exit
```

Gitlab Runner

First, run server Updates:

```
sudo apt update
```

Install Repository & GitLab Runner

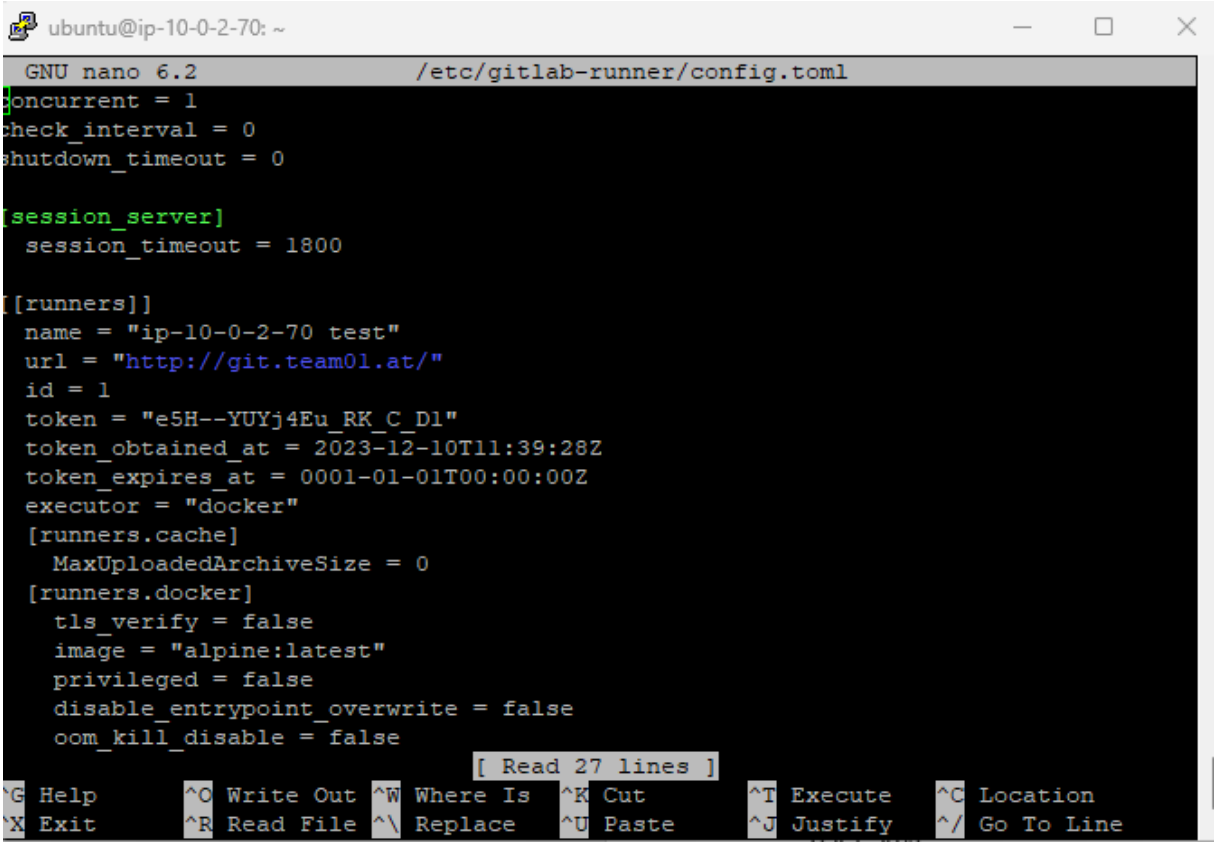
```
curl -L https://packages.gitlab.com/install/repositories/runner/gitlab-runner/script.deb.sh | sudo bash
sudo apt-get install gitlab-runner
```

Register the GitLab Runner. You will be asked for the Domain of your External URL, your GitRunner Token and optional Tags.

```
sudo gitlab-runner register
```

If everything went well, you will see your Configuration of the Runner in the **config.toml** File.

```
sudo nano /etc/gitlab-runner/config.toml
```



Creating a Repository on GitLab Server

Go to http://34.201.193.255// and log in.
Navigate to "Projects" and click "New project" or "Create project".
Enter the project name (e.g., "video").
Set Visibility Level to "Private".
Click "Create project".

Setting Up Your Local Repository:
Create a new directory and initialize it as a Git repository:

```
mkdir video
cd video
git init
```

Rename the default branch to "video" (optional):

```
git branch -m video
```

Add the remote GitLab repository:

```
git remote add origin http://34.201.193.255/root/video.git
```

Create a new file, stage, and commit it:

```
echo "Hello World" > hello.txt
git add hello.txt
git commit -m "Add video hello.txt"
```

Push the commit to the GitLab repository:

```
git push -u origin video
```

Cloning the repository

```
git clone http://34.201.193.255/root/video.git <name of folder>
```

LDAP Server

Ensure your package lists are up to date:

```
sudo apt-get update
```

Install the OpenLDAP server and the administrative utilities

```
sudo apt-get install slapd ldap-utils
```

Follow the instructions and set the administrative password for the LDAP Server. During reconfiguration, you'll be prompted for various settings:

```
Omit OpenLDAP server configuration? No
DNS domain name: Set this to match your domain (it will be used to create
the base DN).
Organization name: Your organization's name.
Administrator password: Set a password for the admin account.
Database backend: MDB
Remove the database when slapd is purged? No
Move old database? Yes
Allow LDAPv2 protocol? No
```

In order to configure the LDAP Server with our DNS Services, insert the IP Adresses of both DNS Servers on the resolved.conf file like stated on page 9.

```
sudo nano /etc/systemd/resolved.conf
```

Create a file (e.g., ou.ldif) to add organizational units like Users and Groups.

```
sudo nano ou.ldif
```

with the following content:

```
dn: ou=ldap,dc=team01,dc=at
objectClass: organizationalUnit
ou: ldap
```

now our organisational unit “ldap” is created.
Add it to the ldap directory:

```
ldapadd -x -D "cn=admin,dc=team01,dc=at" -W -f /home/ubuntu/ou.ldif
```

Enter the admin password when prompted.

Next, log in into your GitLab Server via ssh and go to the file /etc/gitlab/gitlab.rb.

```
sudo nano /etc/gitlab/gitlab.rb
```

Scroll down all the way until you find the stage like provided in the screenshot below and edit it like this:

```
ubuntu@ip-10-0-1-49: ~
  GNU nano 6.2                    /etc/gitlab/gitlab.rb
#     admin_group: ''
#     sync_ssh_keys: false
gitlab_rails['ldap_enabled'] = true
gitlab_rails['ldap_servers'] = YAML.load <<-EOS
  main:
    label: 'LDAP'
    host: 'ldap.team01.at'
    port: 389
    uid: 'uid'
    bind_dn: 'cn=admin,dc=team01,dc=at'
    password: 'admin_password'
    encryption: 'plain'
    verify_certificates: true
    base: 'ou=ldap,dc=team01,dc=at'
    user_filter: ''
    ## EE only
    group_base: ''
    admin_group: ''
    sync_ssh_keys: false
EOS

^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J
```

Then, reconfigure gitlab with the following commands:

```
sudo gitlab-ctl reconfigure
sudo gitlab-ctl restart
```

After that, log into your LDAP Server again.
Next, we create a file to create a user:

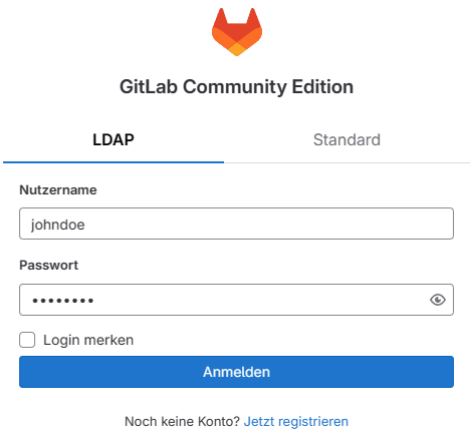
```
sudo nano user.ldif
```

```
ubuntu@ip-10-0-2-80: ~
  GNU nano 6.2                        user.ldif
dn: uid=johndoe,ou=ldap,dc=team01,dc=at
objectClass: inetOrgPerson
objectClass: posixAccount
objectClass: shadowAccount
uid: johndoe
sn: Doe
givenName: John
cn: John Doe
displayName: John Doe
uidNumber: 1000
gidNumber: 1001
userPassword: password
homeDirectory: /home/johndoe
mail: johndoe@example.com
##
```

Here we put in all information about our user.
Then we add the user with

```
ldapadd -x -D "cn=admin,dc=team01,dc=at" -W -f /home/ubuntu/user.ldif
```

After that go to the GitLab Website (http://34.201.193.255//). Select “LDAP” and log in with the credentials written in the file.



CI/CD Pipeline Project Creation

First, we push an existing project to on a repository on our gitlab server. In our case, it’s a simple calculator project written in javascript.

Next, we add Unittests and a Linter to our Project. In our case, we used jest and EsLint.

In the terminal of VSCode (you need to have node.js installed), type

```
npm install --save-dev jest
```

and follow the instructions to create a config file. Your configuration file should then look something like this:

```
{} package.json > {} repository
1  {
2    "name": "taschenrechner",
3    "version": "1.0.0",
4    "description": "",
5    "main": "scriptCalc.js",
       Debug
6    "scripts": {
7      "lint": "eslint . --ext .js",
8      "test": "jest"
9    },
10   "repository": {
11     "type": "git",
12     "url": "http://34.201.193.255/root/taschenrechner.git"
13   },
14   "author": "",
15   "license": "ISC",
16   "devDependencies": {
17     "eslint": "^8.56.0",
18     "jest": "^29.7.0"
19   }
20 }
```

Next, we install EsLint.

```
npm install --save-dev eslint
```

Again, follow the guide to create a config file. It should look similar to this:

.eslintrc.yml

```yaml
env:
  browser: true
  commonjs: true
  es2021: true
  node: true
  jest: true
extends: eslint:recommended
parserOptions:
  ecmaVersion: latest
rules: {}
```

We also create a Dockerfile with an entrypoint and all needed specifications for our project.

Dockerfile 167 B

```dockerfile
FROM node:14-alpine


WORKDIR /usr/src/app


COPY package*.json ./


RUN npm install --production && npm cache clean --force


COPY . .

CMD ["node", "scriptCalc.js"]
```

If you want to run your project in a docker container later on, you need to add a HTTP Server in your Dockerfile and specify the port it should listen to.

Dockerfile 224 B

```
FROM node:14-alpine


WORKDIR /usr/src/app


COPY package*.json ./


RUN npm install --production && npm cache clean --force


RUN npm install -g http-server


COPY . .


EXPOSE 5000


CMD ["http-server", ".", "-p", "5000"]

```

Next, we write our unit tests. In your main file (scriptCalc.js), use the following function to refer to the function that should be used for testing.

```
module.exports = { calculateOperation }; // Export for testing
```

We create a scriptCalc.test.js file to write our unit tests.

scriptCalc.test.js > ...

```
const { calculateOperation } = require('./scriptCalc');

test('adds 1 + 2 to equal 3', () => {
    expect(calculateOperation(1, 2, '+')).toBe(3);
});

test('subtracts 5 - 2 to equal 3', () => {
    expect(calculateOperation(5, 2, '-')).toBe(3);
});

test('multiplies 2 * 3 to equal 6', () => {
    expect(calculateOperation(2, 3, '*')).toBe(6);
});

test('divides 8 / 4 to equal 2', () => {
    expect(calculateOperation(8, 4, '/')).toBe(2);
});

test('returns 0 for invalid operation', () => {
    expect(calculateOperation(4, 2, 'invalid')).toBe(0);
});

```

Afterwards, we check if our unit tests and linter work correctly:

```
PS C:\Users\steff\Downloads\taschenrechner> npm test

> taschenrechner@1.0.0 test
> jest

 PASS  ./scriptCalc.test.js
  √ adds 1 + 2 to equal 3 (2 ms)
  √ subtracts 5 - 2 to equal 3 (1 ms)
  √ multiplies 2 * 3 to equal 6
  √ divides 8 / 4 to equal 2 (1 ms)
  √ returns 0 for invalid operation (1 ms)

Test Suites: 1 passed, 1 total
Tests:       5 passed, 5 total
Snapshots:   0 total
Time:        0.534 s, estimated 1 s
Ran all test suites.
PS C:\Users\steff\Downloads\taschenrechner>
```

```
PS C:\Users\steff\Downloads\taschenrechner> npm run lint

> taschenrechner@1.0.0 lint
> eslint . --ext .js

PS C:\Users\steff\Downloads\taschenrechner>
```

Everything seems to be fine, so we can push our project with all of its configurations to our GitLab repository.

For the last step, you should add .gitignore and .dockerignore files to prevent unnecessary files and directories (like dependencies, local configuration files, and build outputs) from being included in your Git repository and Docker build context. This practice helps to minimize the size of your repository and Docker images, ensuring faster build times and more efficient deployment of your application.

```
git remote add origin <REPOSITORY_URL>
git add .
git commit -m "Commit message"
git push -u origin master
```

After pushing to your repo, go to your repo and go to Build -> Pipeline editor. Here you can edit your CI/CD Pipeline.

To make your pipeline work with your runner, you first need to install docker on your GitLab Runner Server. Connect to your server via ssh and install docker:

Install Required Packages:
```
sudo apt-get install apt-transport-https ca-certificates curl software-properties-common
```

Update the Package Database
```
sudo apt-get update
```

Install Docker CE
```
sudo apt-get install docker-ce
```

Start and Enable Docker
```
sudo systemctl start docker
sudo systemctl enable docker
```

Verify Docker Installation
```
sudo docker run hello-world
```

Understanding the .gitlab-ci.yml File

Overview

The .gitlab-ci.yml file defines the structure and order of the pipeline. Our pipeline includes the following stages:

1. Lint (lint-job)
2. Test (test-job)
3. Build (build-job)
4. Deploy (deploy-job)

At the start of the file, we specify Docker services and set environment variables:

```
services:
  - name: docker:24.0.7-dind
    command: ["--host=tcp://0.0.0.0:2375"]

variables:
  DOCKER_HOST: tcp://docker:2375
  DOCKER_DRIVER: overlay2
  DOCKER_TLS_CERTDIR: ""
```

Docker-in-Docker (DinD): `docker:24.0.7-dind service` is used for Docker-in-Docker. This allows us to run Docker inside our GitLab CI jobs. The command parameter configures the Docker daemon to listen on TCP port 2375.

Variables: We define `DOCKER_HOST, DOCKER_DRIVER`, and `DOCKER_TLS_CERTDIR` to configure Docker within the CI environment.

Lint Stage (lint-job)

```
lint-job:
  image: node:latest
  tags:
    - devops
    - team01
  stage: lint
  script:
    - npm install
    - npm run lint
```

Purpose: Checks the code for stylistic and programming errors.

`image: node:latest` - This job uses the latest Node.js image.
`tags:` Specify runner tags like devops, team01.
`script:` Executes commands to install dependencies and run the linting process.

Test Stage (test-job)

```
test-job:
  image: node:latest
  tags:
    - devops
    - team01
  stage: test
  script:
    - npm install
    - npm run test
```

Purpose: Run automated tests to ensure code quality. Follows a similar structure to lint-job, but the script runs the implemented unit tests instead of linting.

Build Stage (build-job)

```
build-job:
  stage: build
  tags:
    - devops
    - team01
 script:
    - echo "$DOCKER_PASSWORD" | docker login -u "$DOCKER_USERNAME" --password-stdin
    - >
      if [ "$CI_COMMIT_BRANCH" = "develop" ]; then
        docker pull $DOCKER_USERNAME/taschenrechner:$CI_COMMIT_BRANCH || true
        docker build --cache-from $DOCKER_USERNAME/taschenrechner:$CI_COMMIT_BRANCH
-t $DOCKER_USERNAME/taschenrechner:$CI_COMMIT_BRANCH .
        docker save $DOCKER_USERNAME/taschenrechner:$CI_COMMIT_BRANCH >
taschenrechner_$CI_COMMIT_BRANCH.tar

      elif [ "$CI_COMMIT_BRANCH" = "main" ]; then
        docker pull $DOCKER_USERNAME/taschenrechner:latest || true
        docker build --cache-from $DOCKER_USERNAME/taschenrechner:latest -t
$DOCKER_USERNAME/taschenrechner:latest .
        docker save $DOCKER_USERNAME/taschenrechner:latest >
taschenrechner_latest.tar

      fi

  artifacts:
    paths:
      - taschenrechner_*.tar
```

Purpose

Build Docker images based on the branch (develop or main).

Docker Login

```
- echo "$DOCKER_PASSWORD" | docker login -u "$DOCKER_USERNAME" --password-
stdin
```

Authentication to the Docker registry is critical for pulling base images and pushing the built images. This is done using docker login, with credentials passed securely.

Docker Build with Branch Logic

```
if [ "$CI_COMMIT_BRANCH" = "develop" ]; then
    # Build steps for develop branch
elif [ "$CI_COMMIT_BRANCH" = "main" ]; then
    # Build steps for main branch
fi
```

The script performs a conditional build based on the Git branch.
For the develop branch, it tags the built image with develop.
For the main branch, it tags the image as latest.
This is handled using shell if-else statements.

```
docker pull $DOCKER_USERNAME/taschenrechner:$CI_COMMIT_BRANCH || true
```

Attempts to pull a Docker image from a Docker registry. The image is named taschenrechner and is tagged with the name of the Git branch that triggered the CI/CD pipeline `($CI_COMMIT_BRANCH)`. The image is associated with a Docker Hub account specified by the `$DOCKER_USERNAME` variable. The `|| true` part ensures that if the image pull fails (for instance, if the image doesn't exist for that branch), the pipeline doesn't stop, allowing subsequent commands to execute.

```
 docker build --cache-from $DOCKER_USERNAME/taschenrechner:$CI_COMMIT_BRANCH -t
$DOCKER_USERNAME/taschenrechner:$CI_COMMIT_BRANCH .
```

The `docker build --cache-from` command in this CI/CD pipeline is used to build a Docker image with efficient layer caching. It attempts to use previously built layers from the specified image `($DOCKER_USERNAME/taschenrechner:$CI_COMMIT_BRANCH)` to speed up the build process. The `-t` option tags the newly built image with the branch name for easy identification and retrieval in future steps of the pipeline.

```
docker save $DOCKER_USERNAME/taschenrechner:$CI_COMMIT_BRANCH >
taschenrechner_$CI_COMMIT_BRANCH.tar
```

The `docker save` command in this context is used to create a tarball (tar archive) of the specified Docker image. It targets the image `$DOCKER_USERNAME/taschenrechner:$CI_COMMIT_BRANCH`, which is dynamically named based on the Docker username and the branch that triggered the CI/CD pipeline. The resulting tar file, named `taschenrechner_$CI_COMMIT_BRANCH.tar,` contains all the layers of the Docker image and is used for transporting or storing the image outside of Docker environments.

```
  artifacts:
    paths:
      - taschenrechner_*.tar
```

The `artifacts` section is used to specify files created during the job that should be saved and made available after the job is finished. The paths directive under artifacts lists `taschenrechner_*.tar`, which means any tar files with names starting with taschenrechner_ and ending with the branch name will be saved as artifacts. These artifacts are typically used for sharing data between stages in a pipeline or for downloading after the pipeline completes.

Deploy Stage (deploy-job)

```
deploy-job:
  stage: deploy
  tags:
    - devops
    - team01
  dependencies:
    - build-job
  script:
    - echo "$DOCKER_PASSWORD" | docker login -u "$DOCKER_USERNAME" --password-stdin
    - >
      if [ "$CI_COMMIT_BRANCH" = "develop" ]; then
        docker load < taschenrechner_$CI_COMMIT_BRANCH.tar
        docker push $DOCKER_USERNAME/taschenrechner:$CI_COMMIT_BRANCH
      elif [ "$CI_COMMIT_BRANCH" = "main" ]; then
        docker load < taschenrechner_latest.tar
        docker push $DOCKER_USERNAME/taschenrechner:latest
      fi
```

Purpose: The deploy stage is responsible for pushing the built Docker images to a Docker registry, making them available for deployment.

`Dependencies`
Specifies that this job depends on the successful completion of the build-job.

`echo "$DOCKER_PASSWORD" | docker login -u "$DOCKER_USERNAME" --password-stdin`
Similar to the build-job, it logs in to the Docker registry.

```
if [ "$CI_COMMIT_BRANCH" = "develop" ]; then
elif [ "$CI_COMMIT_BRANCH" = "main" ]; then
fi
```

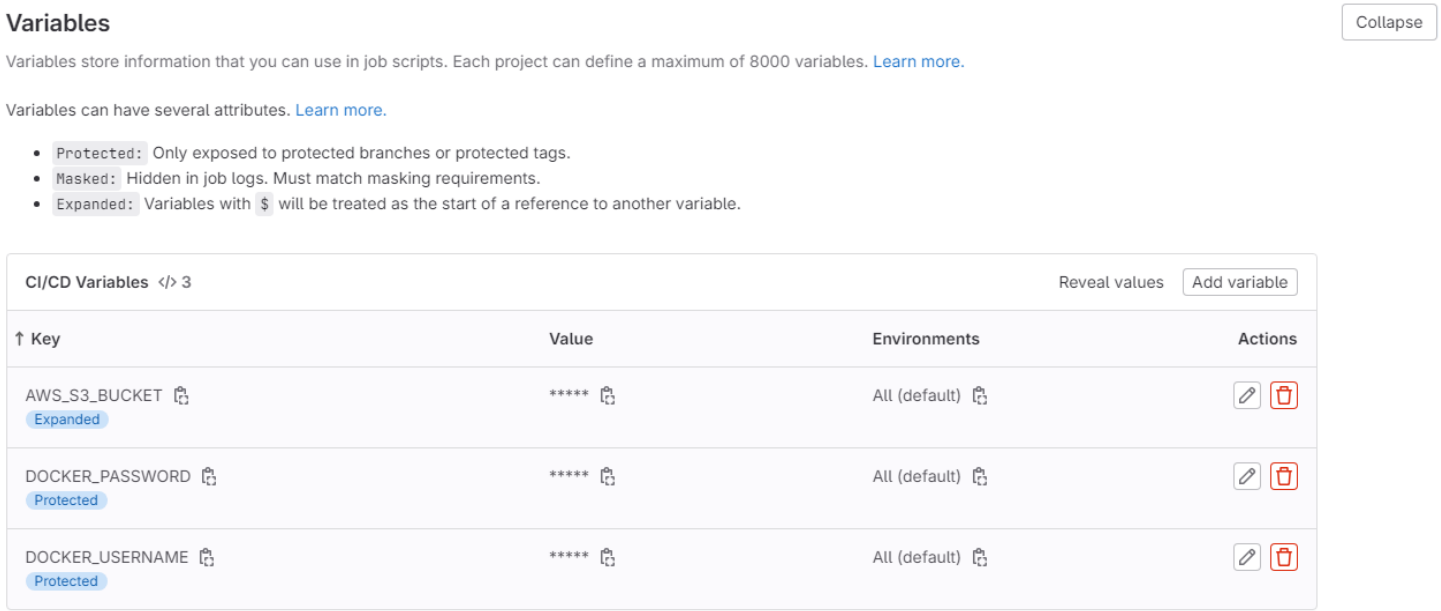
Also similar to the build job, this conditional statement checks the name of the Git branch that triggered the pipeline: if the branch is develop, it executes the commands in the first block; if the branch is main, it executes the commands in the second block. The fi marks the end of the conditional statement.

`docker load < taschenrechner_$CI_COMMIT_BRANCH.tar`
Loads a Docker image from a tar archive, named `taschenrechner_$CI_COMMIT_BRANCH.tar`. The tar file name dynamically changes based on the branch that triggered the pipeline.

```
docker push $DOCKER_USERNAME/taschenrechner:$CI_COMMIT_BRANCH
docker push $DOCKER_USERNAME/taschenrechner:latest
```

This command pushes the loaded Docker image to a Docker registry. The image is tagged with the branch name `($CI_COMMIT_BRANCH)` or `:latest` for the main branch and is under the Docker Hub account specified by `$DOCKER_USERNAME`.

Security and Best Practices: Sensitive information like `DOCKER_USERNAME` and `DOCKER_PASSWORD` should never be hardcoded in the .gitlab-ci.yml file. Instead, they should be stored as protected variables in the GitLab CI/CD settings.

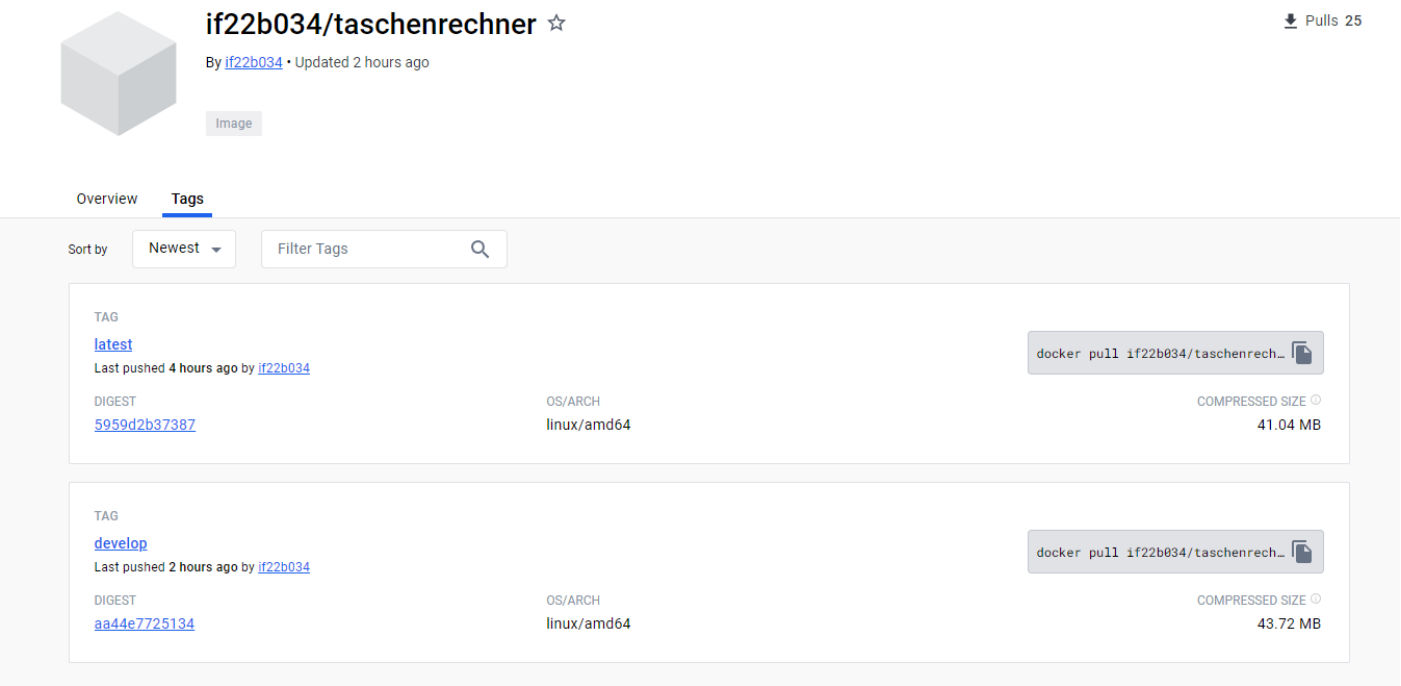


Usage in Jobs: These variables can be referenced in the script sections of the jobs. GitLab CI/CD replaces them with actual values during runtime.

How to run the pushed image in a docker container

Go to https://hub.docker.com/r/if22b034/taschenrechner/tags

Start Docker Desktop.



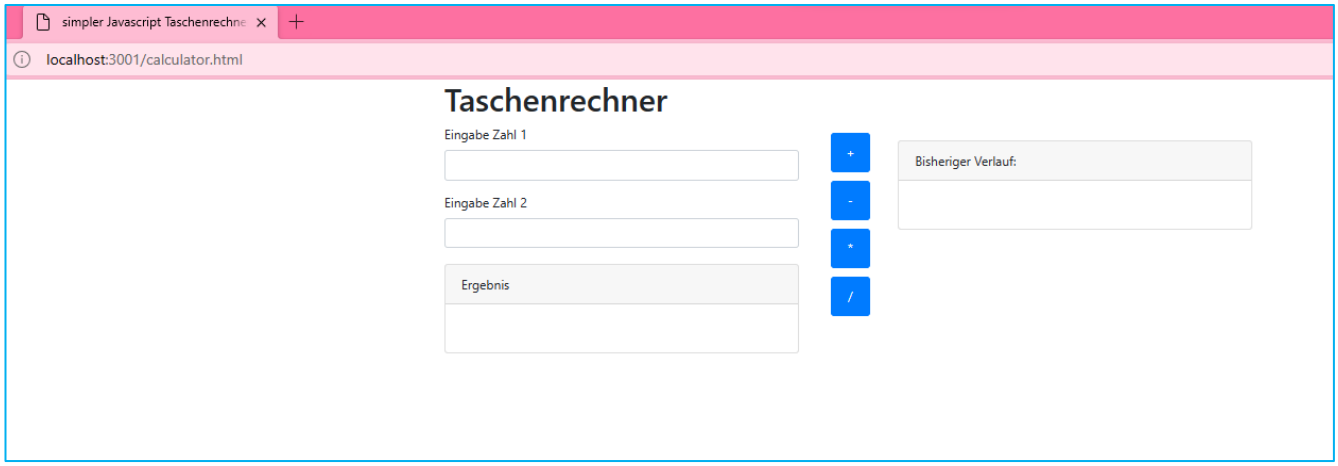
Copy the `docker pull if22b034/taschenrechner:develop` command and execute it in your terminal.

Execute the following command to run the image in a container.

```
docker run --detach --publish 3001:5000 if22b034/taschenrechner:develop
```

You will need to adjust the ports depending on your Dockerfile.

In this example, we have utilized the local server running on port 3001. You can view the image by navigating to http://localhost:3001/calculator.html



.gitlab-ci.yml File

.gitlab-ci.yml 1.95 KiB

Blame | Edit | Replace | Delete

```yaml
image: docker:24.0.7

services:
  - name: docker:24.0.7-dind
    command: ["--host=tcp://0.0.0.0:2375"]

variables:
  DOCKER_HOST: tcp://docker:2375
  DOCKER_DRIVER: overlay2
  DOCKER_TLS_CERTDIR: ""

stages:
  - lint
  - test
  - build
  - deploy

lint-job:
  image: node:latest
  tags:
    - devops
    - team01
  stage: lint
  script:
    - npm install
    - npm run lint

test-job:
  image: node:latest
  tags:
    - devops
    - team01
  stage: test
  script:
    - npm install
    - npm run test

build-job:
  stage: build
  tags:
    - devops
    - team01
  script:
    - echo "$DOCKER_PASSWORD" | docker login -u "$DOCKER_USERNAME" --password-stdin
    - >
      if [ "$CI_COMMIT_BRANCH" = "develop" ]; then
        docker pull $DOCKER_USERNAME/taschenrechner:$CI_COMMIT_BRANCH || true
        docker build --cache-from $DOCKER_USERNAME/taschenrechner:$CI_COMMIT_BRANCH -t $DOCKER_USERNAME/taschenrechner:$CI_COMMIT_BRANCH .
        docker save $DOCKER_USERNAME/taschenrechner:$CI_COMMIT_BRANCH > taschenrechner_$CI_COMMIT_BRANCH.tar
      elif [ "$CI_COMMIT_BRANCH" = "main" ]; then
        docker pull $DOCKER_USERNAME/taschenrechner:latest || true
        docker build --cache-from $DOCKER_USERNAME/taschenrechner:latest -t $DOCKER_USERNAME/taschenrechner:latest .
        docker save $DOCKER_USERNAME/taschenrechner:latest > taschenrechner_latest.tar
      fi
    - docker images
  artifacts:
    paths:
      - taschenrechner_*.tar

deploy-job:
  stage: deploy
  tags:
    - devops
    - team01
  dependencies:
    - build-job
  script:
    - echo "$DOCKER_PASSWORD" | docker login -u "$DOCKER_USERNAME" --password-stdin
    - >
      if [ "$CI_COMMIT_BRANCH" = "develop" ]; then
        docker load < taschenrechner_$CI_COMMIT_BRANCH.tar
        docker push $DOCKER_USERNAME/taschenrechner:$CI_COMMIT_BRANCH
      elif [ "$CI_COMMIT_BRANCH" = "main" ]; then
        docker load < taschenrechner_latest.tar
        docker push $DOCKER_USERNAME/taschenrechner:latest
      fi
```