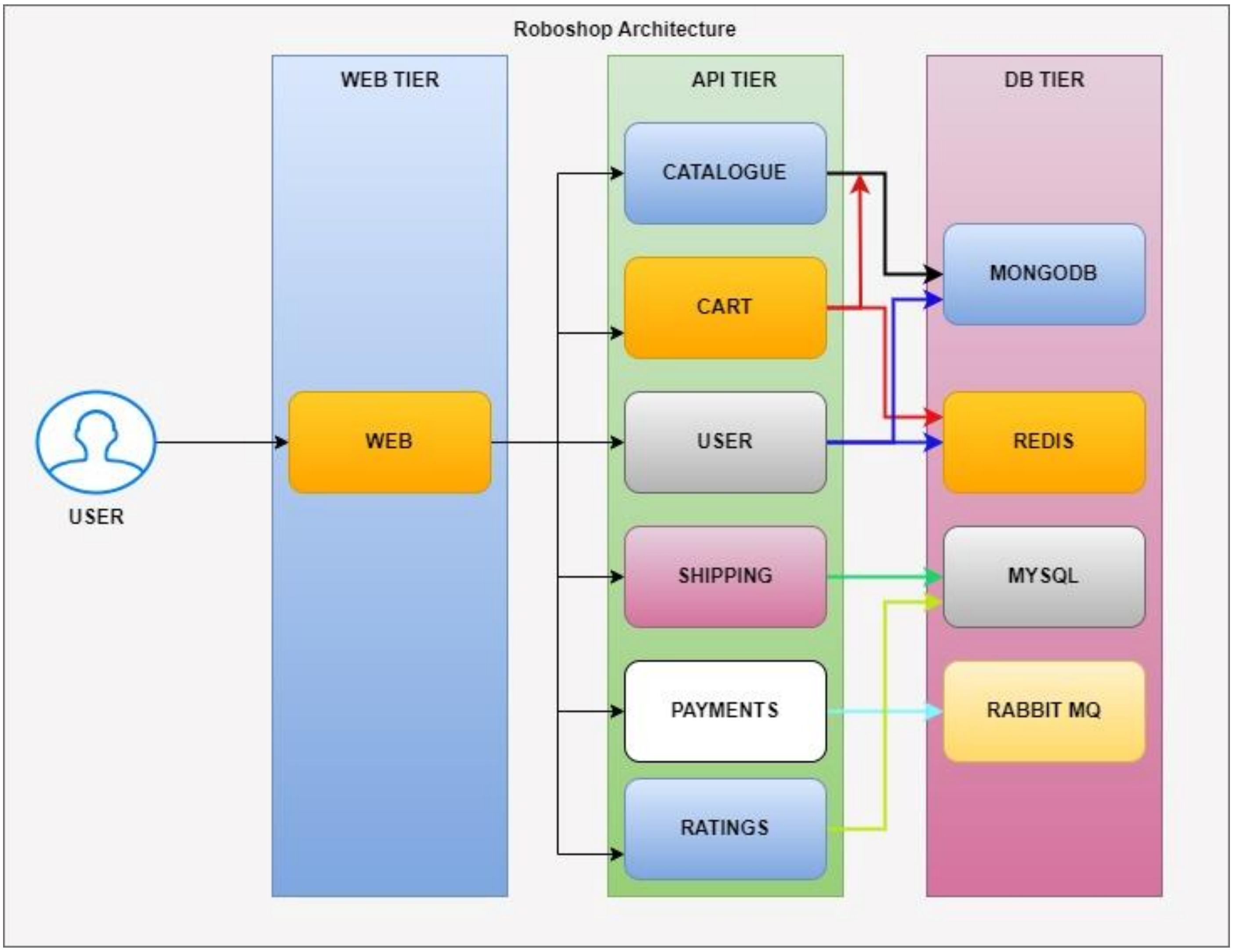
roboshop-documentation:

Below is the communication between components and dependency. This dependency comes from **Development team**. Architects decide that, DevOps has no scope in this.



WEB TIER:

- Usually web tier is the one which has frontend technologies like HTML, CSS, Java Script (React/Angular/Node).
- We use web server to deploy these kind of applications.
- Earlier Apache Server was popular, Now Nginx is the most popular web server.

APP TIER:

- APP/API Tier is the one which has backend technologies like Java, .NET, Python, Go, Php, etc.

- Earlier Backend technologies had servers like tomcat, Jboss, IIS, etc.
- Now all backend technologies are coming up with in built servers.
- Usually API tier should not opened through internet, it should be only accessible through web tier.

DB TIER:

- Storage of the applications will be here like user data, products, orders data, etc.
- We can use RDBMS like MySQL, MSSQL, Postgress, etc for row and column based data.
- We can use NoSQL databases like MongoDB for storing the product information.
- We can use Cache servers like Redis to access the data with lightening speed.
- We can use MQ Servers like RabbitMQ, ActiveMQ, Kafka, etc for asynchronous communication.

01-WEB

- The Web/Frontend is the service in RoboShop to serve the web content over Nginx.
- This will have the web page for the web application.
- Developer has chosen Nginx as a web server and thus we will install Nginx Web Server.

Install Nginx

```
yum install nginx -y
```

Start & Enable Nginx service

```
systemctl enable nginx
systemctl start nginx
```

Try to access the service once over the browser and ensure you get some default content.

```
http://<public-IP>:80
```

Remove the default content that web server is serving.

```
rm -rf /usr/share/nginx/html/*
```

Download the frontend content

```
curl -o /tmp/web.zip https://roboshop-builds.s3.amazonaws.com/web.zip
```

Extract the frontend content.

```
cd /usr/share/nginx/html
unzip /tmp/web.zip
```

Try to access the nginx service once more over the browser and ensure you get roboshop content.

Create Nginx Reverse Proxy Configuration.

```
vim /etc/nginx/default.d/roboshop.conf
```

Add the following content

```
vim /etc/nginx/default.d/roboshop.conf
```

```
proxy_http_version 1.1;
location /images/ {
  expires 5s;
  root   /usr/share/nginx/html;
  try_files $uri /images/placeholder.jpg;
}
location /api/catalogue/ { proxy_pass http://localhost:8080/; }
location /api/user/ { proxy_pass http://localhost:8080/; }
location /api/cart/ { proxy_pass http://localhost:8080/; }
location /api/shipping/ { proxy_pass http://localhost:8080/; }
location /api/payment/ { proxy_pass http://localhost:8080/; }

location /health {
  stub_status on;
  access_log off;
}
```

Ensure you replace the `localhost` with the actual ip address of those component server.
Word `localhost` is just used to avoid the failures on the Nginx Server.
Restart Nginx Service to load the changes of the configuration.

```
systemctl restart nginx
```

MongoDB

- Developer has chosen the database MongoDB.
- Hence, we are trying to install it up and configure it.
  NOTE: Versions of the DB Software you will get context from the developer, Meaning we need to check with developer. Developer has shared the version information as MongoDB-4.x

Setup the MongoDB repo file

```
vim /etc/yum.repos.d/mongo.repo
[mongodb-org-4.2]
name=MongoDB Repository
baseurl=https://repo.mongodb.org/yum/redhat/$releasever/mongodb-org/4.2/x86_64/
gpgcheck=0
enabled=1
```

Install MongoDB

```
yum install mongodb-org -y
```

Start & Enable MongoDB Service

```
systemctl enable mongod
systemctl start mongod
```

Usually MongoDB opens the port only to localhost(127.0.0.1), meaning this service can be accessed by the application that is hosted on this server only. However, we need to access this service to be accessed by another server(remote server), So we need to change the config accordingly.

Update listen address from 127.0.0.1 to 0.0.0.0 in /etc/mongod.conf

You can edit file by using

```
vim /etc/mongod.conf
```

Restart the service

```
systemctl restart mongod
```

Catalogue

Catalogue is a microservice that is responsible for serving the list of products that displays in roboshop application.

Setup NodeJS repos. Vendor is providing a script to setup the repos.

```
curl -sL https://rpm.nodesource.com/setup_lts.x | bash
```

Install NodeJS

```
yum install nodejs -y
```

Configure the application.

Our application developed by the developer of our company and it is not having any RPM software just like other softwares. So we need to configure every step manually

Add application User

```
useradd roboshop
```

User roboshop is a function / daemon user to run the application. Apart from that we don't use this user to login to server.

Also, username roboshop has been picked because it more suits to our project name.

We keep application in one standard location. This is a usual practice that runs in the organization.

Lets setup an app directory.

```
mkdir /app
```

Download the application code to created app directory.

```
curl -o /tmp/catalogue.zip https://roboshop-builds.s3.amazonaws.com/catalogue.zip
cd /app
unzip /tmp/catalogue.zip
```

Every application is developed by development team will have some common softwares that they use as libraries. This application also have the same way of defined dependencies in the application configuration.

Lets download the dependencies.

```
cd /app
npm install
```

We need to setup a new service in systemd so systemctl can manage this service

Setup SystemD Catalogue Service

```
vim /etc/systemd/system/catalogue.service
[Unit]
Description = Catalogue Service

[Service]
User=roboshop
Environment=MONGO=true
Environment=MONGO_URL="mongodb://<MONGODB-SERVER-IPADDRESS>:27017/catalogue"
ExecStart=/bin/node /app/server.js
SyslogIdentifier=catalogue

[Install]
WantedBy=multi-user.target
```

Load the service.

```
systemctl daemon-reload
```

Start the service.

```
systemctl enable catalogue
systemctl start catalogue
```

- For the application to work fully functional we need to load schema to the Database.
- Schemas are usually part of application code and developer will provide them as part of development.

We need to load the schema. To load schema we need to install mongodb client.

To have it installed we can setup MongoDB repo and install mongodb-client

```
vim /etc/yum.repos.d/mongo.repo
[mongodb-org-4.2]
name=MongoDB Repository
baseurl=https://repo.mongodb.org/yum/redhat/$releasever/mongodb-org/4.2/x86_64/
```

```
gpgcheck=0
enabled=1
yum install mongodb-org-shell -y
```

Load Schema

```
mongo --host MONGODB-SERVER-IPADDRESS </app/schema/catalogue.js
```

NOTE: You need to update catalogue server ip address in frontend configuration. Configuration file is /etc/nginx/default.d/roboshop.conf

Redis

Redis is used for in-memory data storage(Caching) and allows users to access the data of database over API.

Redis is offering the repo file as a rpm. Lets install it

```
yum install https://rpms.remirepo.net/enterprise/remi-release-8.rpm -y
```

Enable Redis 6.2 from package streams.

```
yum module enable redis:remi-6.2 -y
```

Install Redis

```
yum install redis -y
```

Usually Redis opens the port only to localhost(127.0.0.1), meaning this service can be accessed by the application that is hosted on this server only. However, we need to access this service to be accessed by another server, So we need to change the config accordingly.

Update listen address from 127.0.0.1 to 0.0.0.0 in /etc/redis.conf & /etc/redis/redis.conf

```
vim /etc/redis.conf
```

Start & Enable Redis Service

```
systemctl enable redis
systemctl start redis
```

User

User is a microservice that is responsible for User Logins and Registrations Service in RobotShop e-commerce portal.

Setup NodeJS repos. Vendor is providing a script to setup the repos.

```
curl -sL https://rpm.nodesource.com/setup_lts.x | bash
```

Install NodeJS

```
yum install nodejs -y
```

Configure the application.

Add application User

```
useradd roboshop
```

Lets setup an app directory.

```
mkdir /app
```

Download the application code to created app directory.

```
curl -L -o /tmp/user.zip https://roboshop-builds.s3.amazonaws.com/user.zip
cd /app
unzip /tmp/user.zip
```

Every application is developed by development team will have some common softwares that they use as libraries. This application also have the same way of defined dependencies in the application configuration.

Lets download the dependencies.

```
cd /app
npm install
```

We need to setup a new service in systemd so systemctl can manage this service

Setup SystemD User Service

```
vim /etc/systemd/system/user.service
[Unit]
Description = User Service
[Service]
User=roboshop
Environment=MONGO=true
Environment=REDIS_HOST=<REDIS-SERVER-IP>
Environment=MONGO_URL="mongodb://<MONGODB-SERVER-IP-ADDRESS>:27017/users"
ExecStart=/bin/node /app/server.js
SyslogIdentifier=user

[Install]
WantedBy=multi-user.target
```

Load the service.

```
systemctl daemon-reload
systemctl enable user
systemctl start user
```

For the application to work fully functional we need to load schema to the Database. Then

NOTE: Schemas are usually part of application code and developer will provide them as part of development.

We need to load the schema. To load schema we need to install mongodb client.

To have it installed we can setup MongoDB repo and install mongodb-client

```
vim /etc/yum.repos.d/mongo.repo
[mongodb-org-4.2]
name=MongoDB Repository
baseurl=https://repo.mongodb.org/yum/redhat/$releasever/mongodb-org/4.2/x86_64/
gpgcheck=0
enabled=1
yum install mongodb-org-shell -y
```

Load Schema

```
mongo --host MONGODB-SERVER-IPADDRESS </app/schema/user.js
```

Cart

Cart is a microservice that is responsible for Cart Service in RobotShop e-commerce portal.

Setup NodeJS repos. Vendor is providing a script to setup the repos.

```
curl -sL https://rpm.nodesource.com/setup_lts.x | bash
```

Install NodeJS

```
yum install nodejs -y
```

Configure the application.

Add application User

```
useradd roboshop
```

Lets setup an app directory.

```
mkdir /app
```

Download the application code to created app directory.

```
curl -L -o /tmp/cart.zip https://roboshop-builds.s3.amazonaws.com/cart.zip
cd /app
unzip /tmp/cart.zip
```

Every application is developed by development team will have some common softwares that they use as libraries. This application also have the same way of defined dependencies in the application configuration.

Lets download the dependencies.

```
cd /app
npm install
```

We need to setup a new service in systemd so systemctl can manage this service

Setup SystemD Cart Service

```
vim /etc/systemd/system/cart.service
[Unit]
Description = Cart Service
```

```
[Service]
User=roboshop
Environment=REDIS_HOST=<REDIS-SERVER-IP>
Environment=CATALOGUE_HOST=<CATALOGUE-SERVER-IP>
Environment=CATALOGUE_PORT=8080
ExecStart=/bin/node /app/server.js
SyslogIdentifier=cart

[Install]
WantedBy=multi-user.target
```

Load the service.

```
systemctl daemon-reload
systemctl enable cart
systemctl start cart
```

MySQL

Developer has chosen the database MySQL. Hence, we are trying to install it up and configure it.

CentOS-8 Comes with MySQL 8 Version by default, However our application needs MySQL 5.7. So lets disable MySQL 8 version.

```
yum module disable mysql -y
```

Setup the MySQL5.7 repo file

```
vim /etc/yum.repos.d/mysql.repo
[mysql]
name=MySQL 5.7 Community Server
baseurl=http://repo.mysql.com/yum/mysql-5.7-community/el/7/$basearch/
enabled=1
gpgcheck=0
```

Install MySQL Server

```
yum install mysql-community-server -y
```

Start MySQL Service

```
systemctl enable mysqld
systemctl start mysqld
```

Next, We need to change the default root password in order to start using the database service. Use password RoboShop@1 or any other as per your choice.

```
mysql_secure_installation --set-root-pass RoboShop@1
```

You can check the new password working or not using the following command in MySQL.

```
mysql -uroot -pRoboShop@1
```

Shipping

Shipping service is responsible for finding the distance of the package to be shipped and calculate the price based on that.

Shipping service is written in Java, Hence we need to install Java.

Maven is a Java Packaging software, Hence we are going to install maven, This indeed takes care of java installation.

```
yum install maven -y
```

Configure the application.

Add application User

```
useradd roboshop
```

Lets setup an app directory.

```
mkdir /app
```

Download the application code to created app directory.

```
curl -L -o /tmp/shipping.zip https://roboshop-builds.s3.amazonaws.com/shipping.zip
cd /app
unzip /tmp/shipping.zip
```

Every application is developed by development team will have some common softwares that they use as libraries. This application also have the same way of defined dependencies in the application configuration.

Lets download the dependencies & build the application

```
cd /app
mvn clean package
mv target/shipping-1.0.jar shipping.jar
```

We need to setup a new service in systemd so systemctl can manage this service

Setup SystemD Shipping Service

```
/etc/systemd/system/shipping.service
[Unit]
Description=Shipping Service

[Service]
User=roboshop
Environment=CART_ENDPOINT=<CART-SERVER-IPADDRESS>:8080
Environment=DB_HOST=<MYSQL-SERVER-IPADDRESS>
ExecStart=/bin/java -jar /app/shipping.jar
SyslogIdentifier=shipping

[Install]
WantedBy=multi-user.target
```

Load the service.

```
systemctl daemon-reload
```

Start the service.

```
systemctl enable shipping
systemctl start shipping
```

For this application to work fully functional we need to load schema to the Database.

We need to load the schema. To load schema we need to install mysql client.

To have it installed we can use

```
yum install mysql -y
```

Load Schema

```
mysql -h <MYSQL-SERVER-IPADDRESS> -uroot -pRoboShop@1 < /app/schema/shipping.sql
```

This service needs a restart because it is dependent on schema, After loading schema only it will work as expected, Hence we are restarting this service. This

```
systemctl restart shipping
```

RabbitMQ

RabbitMQ is a messaging Queue which is used by some components of the applications.

Configure YUM Repos from the script provided by vendor.

```
curl -s https://packagecloud.io/install/repositories/rabbitmq/erlang/script.rpm.sh |
bash
```

Configure YUM Repos for RabbitMQ.

```
curl -s https://packagecloud.io/install/repositories/rabbitmq/rabbitmq-
server/script.rpm.sh | bash
```

Install RabbitMQ

```
yum install rabbitmq-server -y
```

Start RabbitMQ Service

```
systemctl enable rabbitmq-server
systemctl start rabbitmq-server
```

RabbitMQ comes with a default username / password as guest/guest. But this user cannot be used to connect. Hence, we need to create one user for the application.

```
rabbitmqctl add_user roboshop roboshop123
rabbitmqctl set_permissions -p / roboshop ".*" ".*" ".*"
```

Payment

This service is responsible for payments in RoboShop e-commerce app. This service is written on Python 3.6, So need it to run this app.

Install Python 3.6

```
yum install python36 gcc python3-devel -y
```

Configure the application.

Add application User

```
useradd roboshop
```

Lets setup an app directory.

```
mkdir /app
```

Download the application code to created app directory.

```
curl -L -o /tmp/payment.zip https://roboshop-builds.s3.amazonaws.com/payment.zip
cd /app
unzip /tmp/payment.zip
```

Every application is developed by development team will have some common softwares that they use as libraries. This application also have the same way of defined dependencies in the application configuration.

Lets download the dependencies.

```
cd /app
pip3.6 install -r requirements.txt
```

We need to setup a new service in systemd so systemctl can manage this service

Setup SystemD Payment Service

```
vim /etc/systemd/system/payment.service
[Unit]
Description=Payment Service
```

```
[Service]
User=root
WorkingDirectory=/app
Environment=CART_HOST=<CART-SERVER-IPADDRESS>
Environment=CART_PORT=8080
Environment=USER_HOST=<USER-SERVER-IPADDRESS>
Environment=USER_PORT=8080
Environment=AMQP_HOST=<RABBITMQ-SERVER-IPADDRESS>
Environment=AMQP_USER=roboshop
Environment=AMQP_PASS=roboshop123

ExecStart=/usr/local/bin/uwsgi --ini payment.ini
ExecStop=/bin/kill -9 $MAINPID
SyslogIdentifier=payment

[Install]
WantedBy=multi-user.target
```

Load the service.

```
systemctl daemon-reload
```

Start the service.

```
systemctl enable payment
systemctl start payment
```

Dispatch

Dispatch is the service which dispatches the product after purchase. It is written in GoLang, So wanted to install GoLang.

Install GoLang

```
yum install golang -y
```

Configure the application.

Add application User

```
useradd roboshop
```

Lets setup an app directory.

```
mkdir /app
```

Download the application code to created app directory.

```
curl -L -o /tmp/dispatch.zip https://roboshop-builds.s3.amazonaws.com/dispatch.zip
cd /app
unzip /tmp/dispatch.zip
```

Every application is developed by development team will have some common softwares that they use as libraries. This application also have the same way of defined dependencies in the application configuration.

Lets download the dependencies & build the software.

```
cd /app
go mod init dispatch
go get
go build
```

We need to setup a new service in systemd so systemctl can manage this service

Setup SystemD Dispatch Service

```
vim /etc/systemd/system/dispatch.service
[Unit]
Description = Dispatch Service
[Service]
User=roboshop
Environment=AMQP_HOST=RABBITMQ-IP
Environment=AMQP_USER=roboshop
Environment=AMQP_PASS=roboshop123
ExecStart=/app/dispatch
SyslogIdentifier=dispatch

[Install]
WantedBy=multi-user.target
```

Load the service.

```
systemctl daemon-reload
```

Start the service.

```
systemctl enable dispatch
systemctl start dispatch
```

Nginx

We officially pronounce it as **Engine-X**

- It is popular web server.
- It is popular reverse-proxy server.

Directory Structure:

1. Nginx configuration is available in

```
/etc/nginx
```

2. By default HTML files are available in

```
/usr/share/nginx/html
```

3. Logs are available in

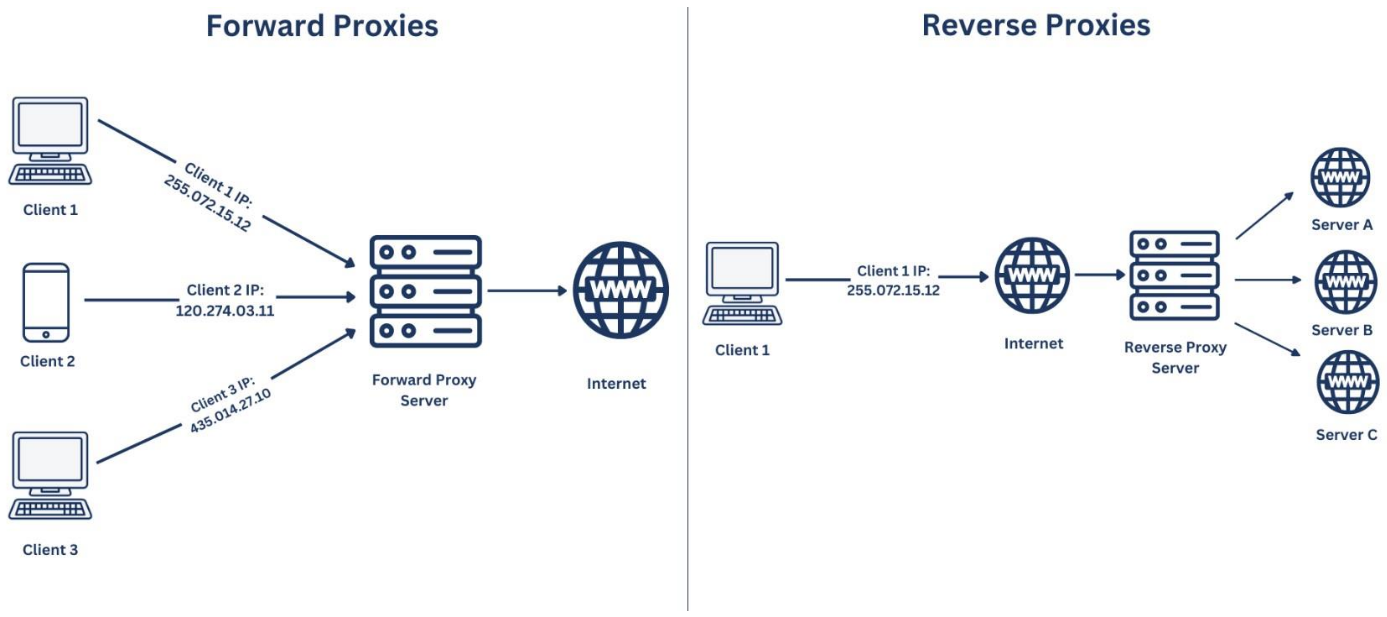
```
/var/log/nginx
```

4. Main configuration file is

```
/etc/nginx/nginx.conf
```

We have 2 type of proxies.

1. Forward Proxy
2. Reverse Proxy



Forward Proxy:

- Client-Centric: Client is aware of existence of proxy, we intentionally send the traffic through proxy server.
- Privacy and Anonymous: Forward proxy is used to hide the client IP address.
- Content Filtering and Access Control: We can restrict traffic out of the network through proxy.
- Caching: Frequently requested content can be cached here.

Reverse Proxy:

- Server-Centric: Clients are unaware of the existence of reverse proxy, for clients it the website they are connecting.
- Load Balancing: Reverse proxies have load balancing capabilities.

- Security: Reverse proxy servers are the best for security. They protect the backend server programming and their IP addresses.
- SSL Termination: Reverse proxies can handle SSL/TLS encryption and decryption, relieving the backend servers from this resource-intensive task.