E2E Encrypted Chat

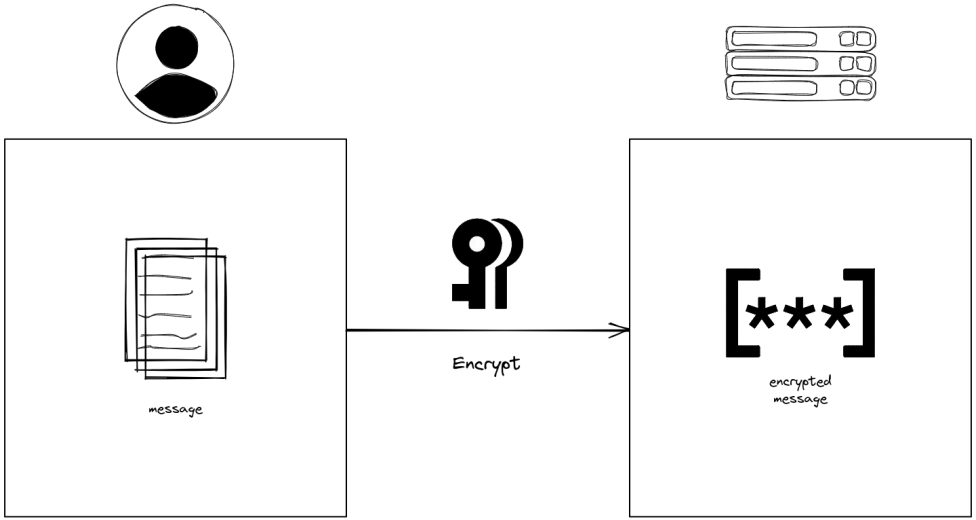
Proyecto Cloud Computing

Recapitulando

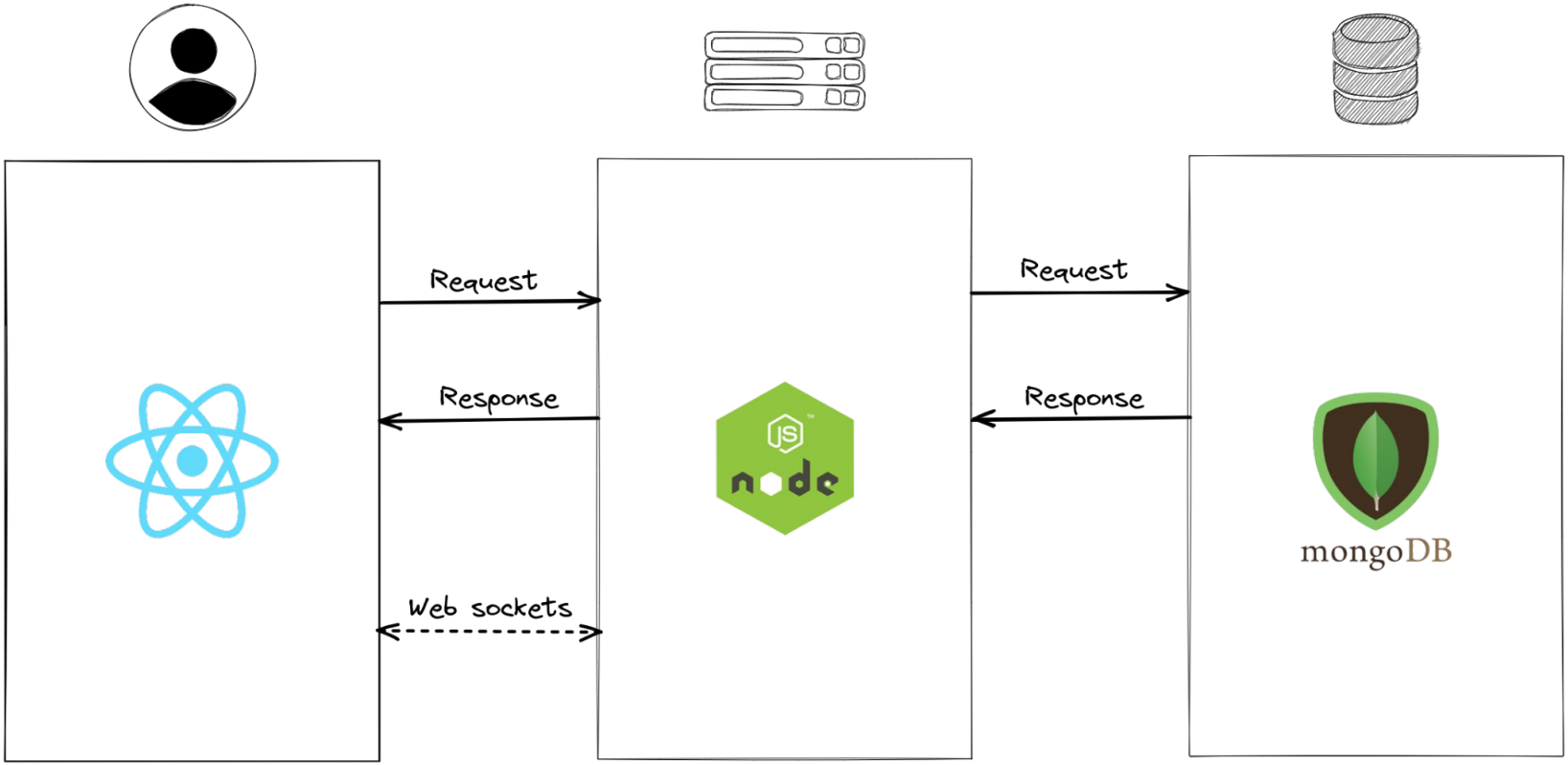
Aplicación

Features

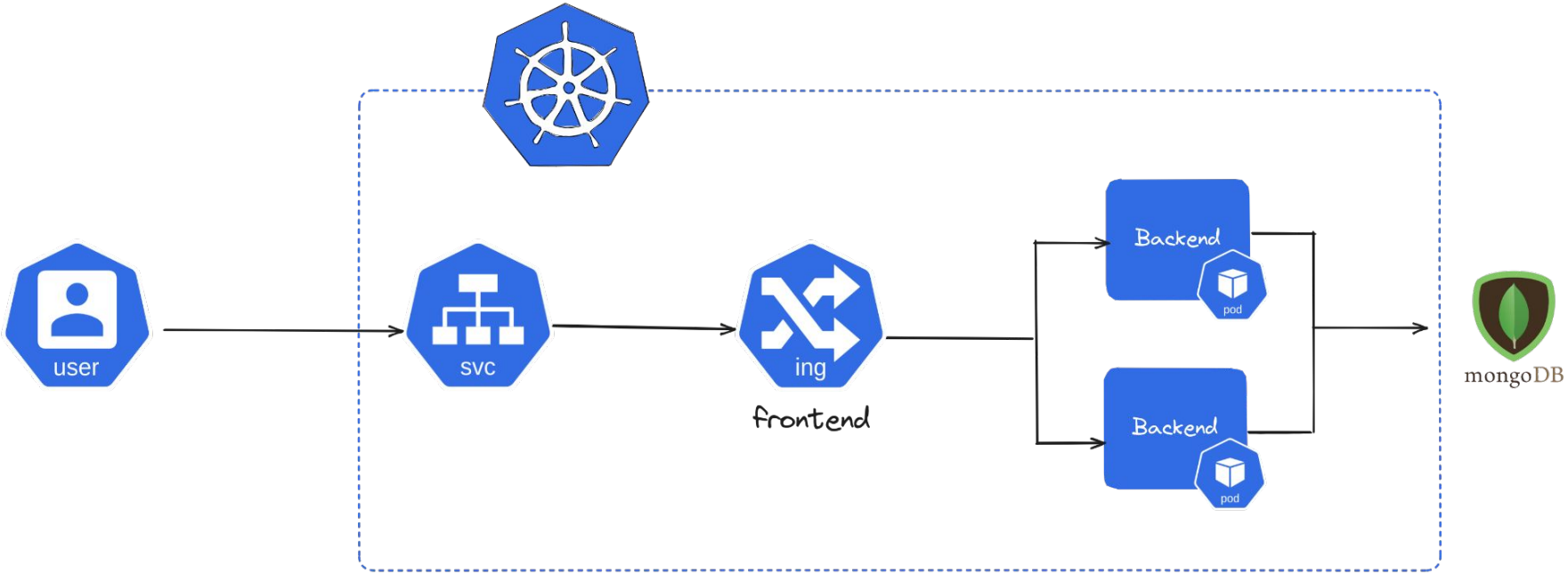
- Login y registro con 2FA.
- Mensajería instantánea.
- Mensajes con encriptación end-to-end.



Arquitectura App



Arquitectura Cloud



Beneficios con tecnologías cloud

Escalabilidad

De ser necesario, Kubernetes permite escalar la aplicación de forma sencilla aumentando el número de pods (réplicas).

Monitoreo

Para monitorear la aplicación se usará Prometheus, esto con el fin de verificar el performance, esto ayudará a prevenir y actuar a tiempo para escalar la aplicación.

Docker

```
FROM node:16-alpine

RUN mkdir -p /app
WORKDIR /app

COPY package.json /app
COPY yarn.lock /app
RUN yarn install

COPY . /app
RUN yarn build

EXPOSE 8000

CMD ["yarn", "start"]
```

```
FROM node:16-alpine AS base

WORKDIR /app

COPY package.json ./
COPY yarn.lock ./

RUN yarn install

COPY . .

ENV NODE_ENV=production

RUN yarn build

EXPOSE 3000

ENV PORT=3000

CMD [ "yarn", "start" ]
```

Kubernetes

```yaml
apiVersion: v1
kind: Service
metadata:
  name: encrypted-chat-backend
spec:
  selector:
    app: encrypted-chat-backend
  ports:
  - port: 8000
    targetPort: 8000
    nodePort: 30001
  type: NodePort
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: encrypted-chat-backend
spec:
  replicas: 1
  selector:
    matchLabels:
      app: encrypted-chat-backend
  template:
    metadata:
      labels:
        app: encrypted-chat-backend
    spec:
      containers:
      - name: encrypted-chat-backend
        image: encrypted-chat-backend
        ports:
        - containerPort: 8000
        env:
        - name: MONGODB_URI
          value:
        - name: PORT
          value: "8000"
        - name: JWT_SECRET
          value:
        - name: CLIENT_URI
          value: http://encrypted-chat-frontend:3000
        imagePullPolicy: Never
```

```yaml
apiVersion: v1
kind: Service
metadata:
  name: encrypted-chat-frontend
spec:
  selector:
    app: encrypted-chat-frontend
  ports:
  - port: 3000
    targetPort: 3000
    nodePort: 30000
  type: NodePort
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: encrypted-chat-frontend
spec:
  replicas: 1
  selector:
    matchLabels:
      app: encrypted-chat-frontend
  template:
    metadata:
      labels:
        app: encrypted-chat-frontend
    spec:
      containers:
      - name: encrypted-chat-frontend
        image: encrypted-chat-frontend
        ports:
        - containerPort: 3000
        imagePullPolicy: Never
```

Testing

Testing
k6

Se probó:

- Login
- Obtener información de usuario
- Conexión por sockets

Parámetros:

- **500** *virtual users*
- **60** segundos

```
import http from 'k6/http';
import ws from 'k6/ws';
import { sleep } from 'k6';

const httpHostname = `http://192.168.49.2:30001`;
const wsHostname = `ws://192.168.49.2:30001`;
const DEFAULT_PASSWORD = 'Aa1#234567';

export const options = {
  vus: 500,
  duration: '60s',
};

const user = {
  username: 'alonso',
  password: DEFAULT_PASSWORD,
};

export default function() {
  const payload = JSON.stringify({
    username: user.username,
    password: user.password,
  });

  const params = {
    headers: {
      'Content-Type': 'application/json',
    },
  };

  const loginRes = http.post(`${httpHostname}/api/auth/login`, payload, params);

  const token = loginRes.json('token');

  if (token) {
    console.log(`User ${user.username} logged in successfully with token ${token}`);

    const res = http.get(`${httpHostname}/api/user/get-user-info`, {
      headers: {
        'Content-Type': 'application/json',
        'Authorization': `Bearer ${token}`,
      }
    });

    console.log(res)


    ws.connect(`${wsHostname}/socket.io/?token=${token}`, (socket) => {
      socket.on('open', () => {
        console.log('connected');
      });
    });

  }


  sleep(1);
}
```

Testing

Kubernetes

```yaml
resources:
  requests:
    cpu: "100m"
    memory: "128Mi"
  limits:
    cpu: "100m"
    memory: "128Mi"
```

Testing Grafana (23:13)

- Muchas veces el frontend no lo procesaba.

Escalamiento Vertical

Testing

Kubernetes

```yaml
resources:
  requests:
    cpu: "300m"
    memory: "512Mi"
  limits:
    cpu: "400m"
    memory: "1024Mi"
```

Testing

Grafana (23:08)

- Tuvo muchos *timeout error*.

Escalamiento Horizontal

Testing Kubernetes

replicas: 2

```yaml
resources:
  requests:
    cpu: "300m"
    memory: "512Mi"
  limits:
    cpu: "400m"
    memory: "1024Mi"
```

Testing

Grafana (23:04)

- No hubo errores.

Conclusiones

Conclusiones

- Según las pruebas de estrés, la escala horizontal ayudó a eliminar los *timeout errors*.
- Según las gráficas, el uso de recursos aumentó según se iba escalando.

Links

Frontend: https://github.com/alonso804/chat-project-frontend

Backend: https://github.com/alonso804/chat-project-backend

Devops: https://github.com/alonso804/chat-project-devops