# 🚀 Proyecto Final SIS313: Sistema de Salas de Espera y Filas Virtuales (Virtual Queue)

Asignatura: SIS313: Infraestructura, Plataformas Tecnológicas y Redes

Semestre: 2/2025

Docente: Ing. Marcelo Quispe Ortega

## 👥 Miembros del Equipo (Grupo Virtual Queue)

|  |  |  |
| --- | --- | --- |
| **Nombre Completo** | **Rol en el Proyecto** | **Contacto (GitHub/Email)** |
| **Duran Chambi Benjamin Ricardo** | **Arquitecto de Backend & Proxy:** Encargado de la VM-PROXY y VM-APP (Nginx/Node.js). Diseño de la lógica de encolamiento. | **Ricardo** |
| **Escobar Moscoso Jorge Gabriel** | **Administrador de Datos:** Encargado de la VM-REDIS. Gestión de persistencia en memoria y optimización de consultas. | https://github.com/jogaesmo |
| **Onofre Alanoca Roy** | **Ingeniero de Observabilidad:** Encargado de la VM-MON (Prometheus/Grafana). Monitoreo y auditoría de métricas. | https://github.com/RoyOnofre |

## 🎯 I. Objetivo del Proyecto

**Objetivo:** Diseñar e implementar una arquitectura de **Cola Virtual (Virtual Waiting Room)** distribuida para el sistema universitario SUNiver, capaz de interceptar el 100% del tráfico entrante, aplicar **Rate Limiting** para limitar la concurrencia a un umbral seguro (ej. 100 usuarios/minuto) y redirigir el exceso de tráfico a una sala de espera estática, garantizando así la **disponibilidad del servicio crítico** bajo condiciones de saturación.

## 💡 II. Justificación e Importancia

**Justificación:** El problema recurrente durante las fechas de inscripción es la **Denegación de Servicio Involuntaria** (saturación de usuarios), lo que genera una **Falla de Continuidad Operacional (T1)**. Este proyecto es vital porque:

1. **Transforma el Fallo en Espera:** Convierte una caída del servidor (Error 503) en una **espera ordenada y controlada**.
2. **Protección Centralizada (T5):** Implementa una capa de protección intermedia (Nginx) que desacopla la carga masiva del servidor de aplicación, mitigando ataques de fuerza bruta y DDoS Capa 7 con **Rate Limiting**.
3. **Optimización Extrema (T4):** Utiliza **Redis (en RAM)** para la gestión del estado de la cola, resolviendo la concurrencia a latencias de sub-milisegundos, algo inviable con bases de datos tradicionales.

## 🛠️ III. Tecnologías y Conceptos Implementados

### 3.1. Tecnologías Clave

|  |  |  |
| --- | --- | --- |
| **Tecnología** | **Rol en el Proyecto (Componente)** | **Función Específica** |
| **Nginx (VM-PROXY)** | **Gateway y Reverse Proxy** | Protege la topología interna y aplica el módulo limit\_req para filtrar peticiones abusivas (Rate Limiting). |
| **Node.js (Express) (VM-APP)** | **Servidor de Aplicación / Lógica** | Ejecuta la lógica de Portero: consulta el estado de la cola a Redis y decide si servir la página de Login o la Sala de Espera HTML. |
| **Redis (VM-REDIS)** | **Gestión de Estado Global (Memoria)** | Base de Datos NoSQL en memoria RAM. Mantiene el contador atómico del aforo en tiempo real, garantizando la "Verdad Única" para la lógica de admisión. |
| **Prometheus / Grafana (VM-MON)** | **Observabilidad y Auditoría** | **Prometheus** hace *scraping* de métricas de la aplicación, y **Grafana** visualiza la saturación de tráfico y el comportamiento de la cola en Dashboards en tiempo real. |
| **Tailscale / Avahi (mDNS)** | **Networking Transparente** | Proporciona una interconexión de las VMs mediante nombres de dominio .local, facilitando la movilidad y el descubrimiento de servicios sin IPs estáticas. |

### 3.2. Conceptos de la Asignatura Puestos en Práctica (T1 - T6)

|  |  |
| --- | --- |
| **Concepto** | **Implementación en el Proyecto** |
| **Alta Disponibilidad (T2)** y Tolerancia a Fallos: | ✅ Implementación de un patrón **Circuit Breaker Lógico** que previene la caída en cascada del servidor principal mediante el desvío de tráfico a la sala de espera estática. |
| **Seguridad y Hardening (T5):** | ✅ Uso de **Rate Limiting** en el Proxy Inverso (Nginx) para mitigar ataques de fuerza bruta y denegación de servicio (DDoS) en Capa 7. |
| **Automatización y Gestión (T6):** | ✅ Configuración de servicios **systemd** para el arranque automático y la recuperación de servicios (Nginx, Redis, App) tras reinicios no programados. |
| **Balanceo de Carga/Proxy (T3/T4):** | ✅ Configuración de **Upstreams** en Nginx para abstraer la ubicación real de los servidores de aplicación y permitir escalabilidad horizontal futura. |
| **Monitoreo (T4/T1):** | ✅ Implementación de un stack de observabilidad completo (Métricas RED) para detectar cuellos de botella y auditar la estabilidad bajo picos de carga. |
| **Networking Avanzado (T3):** | ✅ Implementación de resolución de nombres interna (mDNS) y segmentación de servicios en distintas VMs/Hosts, simulando un entorno de producción real. |

## 🌐 IV. Diseño de la Infraestructura y Topología

### 4.1. Diseño Esquemático

El diseño se basa en la segmentación física de los servicios en **4 VMs** distribuidas en hosts distintos, comunicadas por nombre de dominio a través de la red LAN (mDNS).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **VM/Host** | **Rol** | **IP Lógica / Hostname** | **Software Principal** | **Capa** | **SO** |
| **VM-PROXY** | Gateway / Rate Limiter | proxy-server.local | Nginx | Acceso | Ubuntu 22.04 |
| **VM-APP** | Lógica de Negocio / Worker | app-server.local | Node.js v20 | Aplicación | Ubuntu 22.04 |
| **VM-REDIS** | Gestión de Estado (Cola) | redis-server.local | Redis Server 7 | Datos | Ubuntu 22.04 |
| **VM-MON** | Monitoreo y Alertas | monitor-server.local | Prometheus + Grafana | Gestión | Ubuntu 22.04 |

### Diseño de la Infraestructura

### 

### 4.2. Estrategia Adoptada

* **Estrategia de Desacoplamiento:** Se separó el **Proxy**, la **Lógica** y el **Estado** en máquinas virtuales distintas. Esto garantiza que la **VM-PROXY** pueda seguir respondiendo con páginas de Sala de Espera, incluso si la **VM-APP** se sobrecarga o falla, manteniendo el control del acceso.
* **Estrategia de Optimización (Redis):** Se eligió **Redis** sobre una base de datos SQL porque las operaciones de incremento/decremento de cola deben ser atómicas y de latencia cero. El uso de la RAM garantiza que el contador de cupos nunca sea el cuello de botella (T4).
* **Networking Híbrido:** La configuración del descubrimiento de servicios mediante **mDNS (.local)** permite que las máquinas se encuentren dinámicamente sin depender de IPs estáticas fijas, facilitando la movilidad del despliegue.

## 📋 V. Guía de Implementación y Puesta en Marcha

## Parte 0. Pre-Requisito General (Hacer en TODAS las VMs)

**Propósito:** Establecer la conectividad por nombre de host (hostname), crucial para que los servicios se encuentren entre sí sin depender de IPs estáticas.

**Máquinas Afectadas:** VM-REDIS, VM-APP, VM-PROXY, VM-MON.

1. Actualizar e Instalar herramientas básicas y Avahi (mDNS):  
   Instala herramientas de red y el servicio Avahi para el descubrimiento de servicios por nombre (.local).  
   sudo apt update  
   sudo apt install -y net-tools curl avahi-daemon libnss-mdns
2. Prueba de Conectividad:  
   Desde cualquier VM, verifica la conexión con otra usando su nombre lógico.  
   Ejemplo: ping app-server.local  
   Si la prueba es exitosa, el networking está listo.

## Parte 1. VM-REDIS (La Memoria - Capa de Datos)

Hostname: redis-server.local

Función: Almacenar el contador atómico de usuarios activos.

Sigue estos pasos para la configuración:

1. **Instalar el servicio Redis Server** (Se instala la base de datos ultrarrápida en RAM):  
   sudo apt install redis-server -y
2. **Editar la configuración para permitir conexiones remotas** (El servidor de la lógica VM-APP debe poder conectarse):  
   sudo nano /etc/redis/redis.conf
3. **CAMBIO CLAVE: Permitir conexiones externas.** Dentro del archivo, encuentra la línea bind 127.0.0.1 y cámbiala a:  
   bind 0.0.0.0

# Abrir el archivo de configuración

sudo nano /etc/redis/redis.conf

# Cambiar 'bind 127.0.0.1' por:

# bind 0.0.0.0

# Guardar y reiniciar

sudo systemctl restart redis-server

1. **Reiniciar el servicio** para aplicar cambios:  
   sudo systemctl restart redis-server
2. **Limpieza** (OPCIONAL, pero recomendado para demo):  
   redis-cli FLUSHALL

## Parte 2. VM-APP (El Cerebro - Capa de Aplicación)

Hostname: app-server.local

Función: Ejecutar la lógica de admisión (comparar cupo con Redis) y servir la aplicación en el puerto 3000.

Sigue estos pasos para la configuración:

1. **Instalar Node.js** (Se instala el entorno de ejecución para la aplicación de lógica):  
   sudo apt install nodejs -y
2. **Instalar las dependencias de la aplicación** (Necesitas Express para el servidor web y Redis para la conexión a la VM-REDIS):  
   npm install express redis
3. **Crear la lógica del portero** (Introduce el código de decisión en un nuevo archivo):  
   nano index.js

// Archivo index.js

const express = require('express');

const redis = require('redis');

const app = express();

const PORT = 3000;

const MAX\_CAPACITY = 5; // Límite de usuarios concurrentes permitidos

const REDIS\_KEY = 'suniver:active\_users';

// 1. CONEXIÓN AL HOST REMOTO (VM-REDIS)

const client = redis.createClient({

    socket: {

        host: 'redis-server.local', // Usamos el hostname de la VM-REDIS

        port: 6379

    }

});

client.connect();

client.on('error', err => console.error('Redis Client Error:', err));

// 2. LOGICA DEL SERVIDOR

app.get('/', async (req, res) => {

    try {

        // Incrementamos el contador: preguntamos a Redis cuantos hay Y sumamos 1

        const activeUsers = await client.incr(REDIS\_KEY);

        if (activeUsers <= MAX\_CAPACITY) {

            // ESCENARIO A: HAY CUPO (ENTRA - Pantalla VERDE)

            // Simular que el usuario se queda 10 segundos y luego se va (decr)

            setTimeout(() => client.decr(REDIS\_KEY), 10000);

            res.send(`

                <body style="background-color: #2ecc71; color: white; text-align: center; padding-top: 50px; font-family: sans-serif;">

                    <h1>✅ BIENVENIDO AL SISTEMA (Login)</h1>

                    <p>Usuarios Activos: ${activeUsers} / ${MAX\_CAPACITY}</p>

                    <p>Su sesión termina en 10 segundos.</p>

                </body>

            `);

        } else {

            // ESCENARIO B: NO HAY CUPO (SALA DE ESPERA - Pantalla NARANJA)

            // Devolvemos la cuenta ya que no se le permitió pasar

            await client.decr(REDIS\_KEY);

            res.send(`

                <body style="background-color: #e67e22; color: white; text-align: center; padding-top: 50px; font-family: sans-serif;">

                    <h1>🟠 SALA DE ESPERA - Capacidad Maxima Alcanzada</h1>

                    <p>El sistema esta saturado. Recargando en 5 segundos...</p>

                </body>

                <meta http-equiv="refresh" content="5">

            `);

        }

    } catch (error) {

        // En caso de fallo de Redis o lógica

        console.error(error);

        res.status(503).send('<h1>🔴 ERROR 503: Servidor temporalmente no disponible.</h1>');

    }

});

// 3. INICIO

app.listen(PORT, () => {

    console.log(`App corriendo en puerto ${PORT}`);

});

1. **Ejecutar el servidor de la aplicación** (Inicia la aplicación en el puerto 3000 en segundo plano):  
   nohup node index.js &

## Parte 3. VM-PROXY (El Portero - Capa de Acceso)

Hostname: proxy-server.local

Función: Punto de entrada único. Aplica Rate Limiting (T5) y enruta el tráfico al servidor de aplicación (Proxy Inverso T3/T4).

Sigue estos pasos para la configuración:

1. **Instalar el servidor Nginx** (Servidor proxy de alto rendimiento):  
   sudo apt install nginx -y
2. **Configurar Rate Limiting (T5)** (Edición del archivo de configuración principal de Nginx):  
   sudo nano /etc/nginx/nginx.conf
3. **CAMBIO CLAVE: Definir la regla de Rate Limit.** Debajo de la línea http {, añade la siguiente directiva (10 peticiones por segundo por IP de usuario):  
   limit\_req\_zone $binary\_remote\_addr zone=one:10m rate=10r/s;
4. **Configurar Proxy Inverso (T3/T4)** (Apuntar al servidor de la lógica VM-APP):  
   sudo nano /etc/nginx/sites-available/default

*Clave:* Definir el *upstream* hacia la VM-APP.

# Bloque de configuracion de Nginx (dentro de sites-available/default)

# 1. Definicion del UPSTREAM (el destino real)

upstream backend\_app {

# Apunta a la VM-APP en su puerto interno 3000

server app-server.local:3000;

}

server {

listen 80 default\_server;

listen [::]:80 default\_server;

server\_name \_;

location / {

# 2. Aplicar el filtro de Rate Limiting

limit\_req zone=one burst=5 nodelay;

# 3. Proxy Pass (Enviar al Cerebro)

proxy\_pass http://backend\_app;

# Opcional: Agregar cabeceras para que el servidor de la app sepa la IP real del usuario

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

}

}

1. **CAMBIO CLAVE: Definir Upstream y aplicar Rate Limit.** Dentro del bloque server {, reemplaza la sección location / por el código que define el upstream a app-server.local:3000 y aplica la regla de Rate Limit. *(Ver el código Nginx de location / en tu entorno)*.
2. **Probar la sintaxis y reiniciar Nginx** (Asegura que la configuración no tenga errores):  
   sudo nginx -t && sudo systemctl restart nginx

## Parte 4. VM-MON (Los Ojos - Capa de Gestión)

Hostname: monitor-server.local

Función: Visualizar el estado de la cola y las métricas de carga en tiempo real (Auditoría T4).

Sigue estos pasos para la configuración:

1. Instalar Prometheus y Grafana (El stack de observabilidad):  
   (Se asume que los paquetes están instalados)
2. **Configurar el Scraper de Prometheus** (Indicar a Prometheus qué métricas recolectar):  
   sudo nano /etc/prometheus/prometheus.yml

# ... dentro de 'scrape\_configs'

- job\_name: 'node\_app'

scrape\_interval: 5s

static\_configs:

- targets: ['app-server.local:3000'] # Apunta a la VM-APP

1. **CAMBIO CLAVE: Añadir el objetivo de la VM-APP.** Dentro de scrape\_configs:, añade la configuración para recolectar datos de la ruta /metrics de la VM-APP cada 5 segundos. *(Ver el código Prometheus en tu entorno)*.
2. **Reiniciar Prometheus** (Inicia la recolección de datos desde la VM-APP):  
   sudo systemctl restart prometheus
3. **Configurar Grafana:** Abrir Grafana en el navegador (puerto 3000 o 80). Agregar Prometheus como Data Source y crear/importar un dashboard simple.

### 5.1. Ficheros de Configuración Clave

|  |  |  |
| --- | --- | --- |
| **Fichero** | **Ubicación / Rol** | **Función Crítica** |
| **index.js** | **VM-APP (Lógica)** | Algoritmo de decisión (If cupo -> Login, Else -> SalaEspera) y conexión remota a redis-server.local. |
| **nginx.conf** | **VM-PROXY (Proxy)** | Define el upstream a app-server.local y la directiva **Rate Limiting** para el control de acceso. |
| **redis.conf** | **VM-REDIS (Datos)** | Establece bind 0.0.0.0 para permitir la comunicación remota con la VM-APP. |
| **prometheus.yml** | **VM-MON (Monitoreo)** | Configura el *scraping* (recolección) de métricas de la VM-APP. |

## ⚠️ VI. Pruebas y Validación

|  |  |  |
| --- | --- | --- |
| **Prueba Realizada** | **Resultado Esperado** | **Resultado Obtenido** |
| **Test de Estrés (Saturación)** | Al superar el límite de N usuarios, el usuario N+1 debe ver la pantalla naranja de "Sala de Espera", no el Error 503. | OK: El sistema redirigió correctamente al usuario 6 a la cola. |
| **Test de Recuperación Automática** | Cuando un usuario activo abandona el sistema (timeout), la cola debe avanzar automáticamente. | OK: El usuario en espera ingresó automáticamente en el siguiente refresco. |
| **Prueba de Monitoreo en Vivo** | Grafana debe mostrar el pico de tráfico y el estancamiento de usuarios activos en el límite máximo. | OK: Dashboard reflejó la curva de saturación en tiempo real, validando la estabilidad (T1). |
| **Validación de Conectividad** | Las VMs deben comunicarse por hostname (.local) independientemente de la IP asignada por el DHCP. | OK: Pings y conexiones de base de datos exitosas por nombre. |

## Objetivo a llegar.

**Usuario en cola de espera**

**Usuario entrando a SUNiver**

## 

## 📚 VII. Conclusiones y Lecciones Aprendidas

**Logro Principal:** Demostramos que la **escalabilidad y la disponibilidad** se logran mediante una arquitectura de software inteligente (**Desacoplamiento**), no solo con la adición de más hardware. El sistema transformó un escenario de fallo (Error 500) en una experiencia de usuario controlada (Sala de Espera).

**Lección Aprendida:** La importancia de **Redis (T4)** es crítica. Intentar gestionar el estado de la cola en una base de datos tradicional hubiera introducido latencia, convirtiendo al contador en el principal cuello de botella. La gestión en RAM fue esencial.

**Mejora Futura:** Para un despliegue de producción real, implementaríamos un clúster de Redis (Sentinel) para evitar que la VM-REDIS sea un punto único de fallo, y aseguraríamos la capa de acceso con HTTPS y certificados SSL/TLS en Nginx.