Configurações Jitsi Docker

Procedimentos de configurações e customizações.

- Jitsi Meet Load Balancing
- Load Testing Jitsi Meet
- Self-Hosting Guide - Log Analyser

Jitsi Meet Load Balancing

Link: https://meetrix.io/blog/webrtc/jitsi/jitsi-meet-load-balancing.html

Setup multi server, load balanced videobriges

One of the amazing features in Jitsi Meet is the inbuilt horizontal scalability. When you want to cater large number of concurrent users, you can spin up multiple video bridges to handle the load. If we setup mulple video bridges and connect them to the same shard, Jicofo, the conference manager selects the least loaded Videobridge for the next new conference.

Run Prosody on all interfaces

By default prosody runs only on local interface (127.0.0.0), to allow the xmpp connections from external servers, we have to run prosody on all interfaces. To do that, add the following line at the beginning of `/etc/prosody/prosody.cfg.lua`

```
component_interface = "0.0.0.0"
```

Allow inbound traffic for port 5222

Open inbound traffic from JVB server on port `5222` (TCP) of Prosody server. But DO NOT open this publicly.

Install JVB on a seperate server

Install a Jitsi Video Bridge on a different server.

Copy JVB configurations from Jitsi Meet server

Replace `/etc/jitsi/videbridge/config` and `/etc/jitsi/video-bridge/sip-communicator.properties` of JVB server with the same files from the original Jitsi Meet server

Update JVB config file

In `/etc/jitsi/videbridge/config` set the `XMPP_HOST` to the ip address/domain of the prosody server

```
# Jitsi Videobridge settings

# sets the XMPP domain (default: none)
JVB_HOSTNAME=example.com

# sets the hostname of the XMPP server (default: domain if set, localhost otherwise)
JVB_HOST=<XMPP_HOST>

# sets the port of the XMPP server (default: 5275)
JVB_PORT=5347

# sets the shared secret used to authenticate to the XMPP server
JVB_SECRET=<JVB_SECRET>

# extra options to pass to the JVB daemon
JVB_OPTS="--apis=rest,xmpp"



# adds java system props that are passed to jvb (default are for home and logging config file)
JAVA_SYS_PROPS="-Dnet.java.sip.communicator.SC_HOME_DIR_LOCATION=/etc/jitsi -
Dnet.java.sip.communicator.SC_HOME_DIR_NAME=videobridge -
Dnet.java.sip.communicator.SC_LOG_DIR_LOCATION=/var/log/jitsi -
Djava.util.logging.config.file=/etc/jitsi/videobridge/logging.properties"
```

In `/etc/jitsi/video-bridge/sip-communicator.properties` file update the following properties

1. `<XMPP_HOST>`: The ip address of the prosody server. Better if you can use the private IP address if that can be accessed from JVB server.
2. `<JVB_NICKNAME>`: This should be a unique string used by Jicofo to identify each JVB

```
org.ice4j.ice.harvest.DISABLE_AWS_HARVESTER=true
org.ice4j.ice.harvest.STUN_MAPPING_HARVESTER_ADDRESSES=meet-jit-si-turnrelay.jitsi.net:443
org.jitsi.videobridge.ENABLE_STATISTICS=true
org.jitsi.videobridge.STATISTICS_TRANSPORT=muc
org.jitsi.videobridge.xmpp.user.shard.HOSTNAME=<XMPP_HOST>
org.jitsi.videobridge.xmpp.user.shard.DOMAIN=auth.example.com
org.jitsi.videobridge.xmpp.user.shard.USERNAME=jvb
org.jitsi.videobridge.xmpp.user.shard.PASSWORD=<JVB_SECRET>
org.jitsi.videobridge.xmpp.user.shard.MUC_JIDS=JvbBrewery@internal.auth.example.com
org.jitsi.videobridge.xmpp.user.shard.MUC_NICKNAME=<JVB_NICKNAME>
org.jitsi.videobridge.xmpp.user.shard.DISABLE_CERTIFICATE_VERIFICATION=true
```

You can add the following line at the beginning of `/usr/share/jitsi/jvb/jvb.sh` to generate a unique nickname for the JVB at each startup. This might be useful if you are using an auto-scaling mechanism.

```
sed -i
"s/org.jitsi.videobridge.xmpp.user.shard.MUC_NICKNAME=.*/org.jitsi.videobridge.xmpp.user.shard
.MUC_NICKNAME=$(cat /proc/sys/kernel/random/uuid)/g" /etc/jitsi/videobridge/sip-
communicator.properties
```

Looking for commercial support for Jitsi Meet ? Please contact us via hello@meetrix.io

Updated: October 19, 2020

Load Testing Jitsi Meet

Link: https://meetrix.io/blog/webrtc/jitsi/jitsi-meet-load-testing.html

Make sure your Jitsi Meet infrastructure is ready for production

By deploying a horizontally scalable Jitsi Meet setup, you can scale your Jitsi Meet conferencing infrastructure to cater thousands of concurrent users. But, before you go live you might want to make sure that your infrastructure is capable of handling the desired number of concurrent users.

Jitsi Meet Torture is a tool that can be used to run tests against a Jitsi Instance and it is capable of load testing Jitsi Meet.

> Jitsi Meet Load Testing
> Jitsi Meet Load Testing with Torture

Selenium Grid

To simulate hundrads of concurrent users, we need to deploy Jitsi Meet Torture agains a selenium grid setup. There would be a `selenium hub` which accepts commands from Jitsi Meet Torture and pass them to `selenium nodes`. You can have hundrads of `selenium nodes` to simulate users.

> Jitsi Meet Load Testing
> Jitsi Meet Load Testing with Torture

Overwhelmed with managing
Jitsi Infrastructure?
Outsource full-time, high-cost Jitsi infrastructure management and maintenance

Talk to an expert

Minimal setup with docker-compose

1. Install Docker. You can use following scripts on Ubuntu 18.04

```
sudo apt-get update

sudo apt-get -y install apt-transport-https ca-certificates curl software-properties-common

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

sudo add-apt-repository -y \
"deb [arch=amd64] https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) \
stable"

sudo apt-get -y update
sudo apt-get -y install docker-ce

sudo usermod -a -G docker $(whoami)
```

2. Install the latest version of Docker Compose
3. Create a `docker-compose.yml` file with following content.

```
version: "3.3"
services:
  torture:
    image: meetrix/jitsi-meet-torture
  hub:
    image: selenium/hub:3.141.59
  node:
    build: ./node
    image: meetrix/jitsi-meet-torture-selenium-node
    volumes:
      - /dev/shm:/dev/shm
    depends_on:
      - hub
    environment:
      HUB_HOST: hub
```

4. Please note that `node` are configured to run only one chrome instance at a time. Run the setup with `docker-compose up -d --scale node=2` to run two nodes.
5. `docker-compose exec torture /bin/bash`

6. Change `<YOUR_JITSI_MEET_INSTANCE_URL>` to your Jitsi Meet installation (for example `https://meet.example.com`) in following script and run to perform the test.

```
./scripts/malleus.sh --conferences=1 --participants=2 --senders=2 --audio-senders=2 --
duration=300 --room-name-prefix=test --hub-url=http://hub:4444/wd/hub --instance-
url=<YOUR_JITSI_MEET_INSTANCE_URL>
```

Running the test with large number of nodes

You can scale the number of nodes with `docker-compose up --scale node=<NUMBER>` or you can create a VM using same docker images for `selenium hub` and `selenium-node`. Then you can use an autoscaling mechanism (Such as Autoscaling groups on AWS) to scale the number of nodes.

Looking for commercial support for Jitsi Meet ? Please contact us via hello@meetrix.io

Updated: June 19, 2020

Self-Hosting Guide - Log Analyser

Link: https://jitsi.github.io/handbook/docs/devops-guide/devops-guide-log-analyser Last updated on **Mar 7, 2025**

Welcome to the Jitsi Meet Log Analyser project! This integration leverages Grafana Loki and OpenTelemetry to enable effective log management and analysis for Jitsi Meet components.

Overview

This project offers a streamlined setup for collecting, parsing, and visualizing log data generated by Jitsi Meet components. The integration includes:

- A Docker Compose setup file (`log-analyser.yml`) for configuring Loki and OpenTelemetry Collector.
- A Docker Compose setup file (`grafana.yml`) for configuring Grafana.
- A unified Docker Compose command that initiates all services concurrently.
- Instructions on configuring Grafana with Loki as a log data source for enhanced visualization and analysis.

Getting Started

Prerequisites

Before you begin, ensure you have the following installed on your system:

- Docker
- Docker Compose

Setup

To set up the project, follow these steps:

1. **Clone the repository:**
   Clone the Jitsi Meet repository that contains the necessary Docker Compose files.

   ```
   git clone https://github.com/jitsi/docker-jitsi-meet.git
   cd docker-jitsi-meet
   ```

2. **Update Jitsi Meet Docker Compose Configuration:**
   To enable log collection and analysis, you need to modify the `docker-compose.yml` file for Jitsi Meet components. Add the following configuration to each Jitsi service within the `docker-compose.yml` file:

   ```
   logging:


     options:
       labels: "service"
   ```

   This configuration ensures that logs are collected in JSON format and tagged with service labels, which is essential for Loki to properly ingest and index the logs.
3. **Start the Docker containers:**
   Use the following command to start all required services, including the Jitsi Meet components, Grafana, Loki, and OpenTelemetry:

   ```
   docker-compose -f docker-compose.yml -f log-analyser.yml -f grafana.yml up -d
   ```

   - **Explanation:**
     - The command combines multiple Docker Compose files to launch the entire stack:
       - `docker-compose.yml` launches the Jitsi Meet components.
       - `log-analyser.yml` sets up the log analysis tools, including Loki and OpenTelemetry.
       - `grafana.yml` initializes Grafana for log visualization.
     - Logs generated by the Jitsi Meet components are automatically forwarded to Loki via the OpenTelemetry Collector, making them available for analysis in Grafana.
   - **Note:** If you only want to use Grafana for log visualization without the log analysis tools, you can run `grafana.yml` independently. However, for the complete log analysis setup, both `log-analyser.yml` and `grafana.yml` are required.

Access Grafana

After starting the services, follow these steps to access Grafana:

1. **Open Grafana in your web browser:**
   Navigate to http://localhost:3000 to access the Grafana interface.
2. **Log in to Grafana:**
   Use the default login credentials provided below:
   - **Username:** `admin`
   - **Password:** `admin`

   (It is recommended to change these credentials after the first login for security purposes.)

Pre-configured Dashboards

Grafana comes with several pre-configured dashboards specifically designed for monitoring different Jitsi Meet components. These dashboards will be automatically available once you log in to Grafana.

Important : For data to appear in these dashboards, logs need to be generated by the Jitsi Meet components. Here are the available dashboards:

- **Jicofo Dashboard:** Visualizes logs related to Jitsi Conference Focus (Jicofo), which handles media and signaling in Jitsi Meet.
- **JVB Dashboard:** Focuses on Jitsi Videobridge (JVB) logs, showing details on video streaming and performance metrics.
- **Prosody Dashboard:** Monitors Prosody, the XMPP server used by Jitsi Meet for signaling.
- **Web Dashboard:** Displays logs and metrics related to the web frontend of Jitsi Meet.
- **Jitsi All Dashboard:** A comprehensive dashboard that aggregates logs from all Jitsi Meet components.

Jitsi Meet Log Analyser Dashboard

Filtering Logs with LogQL

Beyond the pre-configured dashboards, you can explore and filter logs in Grafana's "Explore" section using LogQL, a powerful query language designed for Loki. Here’s how you can filter and analyze logs:

1. **Access the Explore section:**
   In Grafana, navigate to the **Explore** tab from the left sidebar. This section allows you to run queries on your logs in real-time.
2. **Select the Loki Data Source:**

In the Explore section, ensure that the **Data Source** is set to **Loki**. This is essential because Loki is the backend that stores and manages the log data collected from Jitsi Meet components.

3. **Using LogQL for Filtering:**
   LogQL enables you to create complex queries to filter specific logs. For example, the following LogQL query filters logs for the `jitsi-jicofo` component and parses them:

```
{exporter="OTLP"} | json | attributes_attrs_service="jitsi-jicofo"
```

   - **Explanation:**
     - `{exporter="OTLP"}`: Selects logs that are exported via OpenTelemetry.
     - `| json`: Parses the log data as JSON, making attributes accessible for filtering.
     - `attributes_attrs_service="jitsi-jicofo"`: Filters logs where the `attributes_attrs_service` field equals `"jitsi-jicofo"`.

   You can modify the query to filter logs from other components or adjust the criteria according to your needs.

   For more details and advanced usage of LogQL, you can refer to the official LogQL documentation from Grafana.
   LogQL Query Example

Usage

Once the setup is complete, you can start exploring your log data in Grafana. The pre-configured dashboards provide an insightful visualization of the logs collected from Jitsi Meet components. Use these dashboards to:

- **Parse Logs:** View detailed logs collected from various components.
- **Visualize Logs:** Analyze log data through various charts, graphs, and panels to gain insights into system performance and issues.

Customizing Dashboards

While the pre-configured dashboards provide a solid starting point, you may want to customize them or create new dashboards to suit your specific needs. Here's how you can do that:

1. **Create a New Dashboard:**
   - Go to the Grafana home page, click on the "+" icon on the left sidebar, and select "Dashboard."
   - Add panels to visualize different metrics or logs by selecting the appropriate data source (Loki) and using LogQL queries.

2. **Customize Existing Dashboards:**
   - Navigate to an existing dashboard and click on the "Edit" button (pencil icon) on any panel.
   - Adjust the LogQL query, visualization type, and panel settings to match your requirements.
3. **Save and Share Dashboards:**
   - After customizing, save the dashboard. You can also export it as a JSON file to share with others or for backup.
   - To export the dashboard:
     - Click on the dashboard title to open the options menu.
     - Select **"Dashboard settings"** > **"JSON Model"**.
     - Click **"Download JSON"** to save the file locally.
4. **Contribute to Jitsi Meet Dashboards:**
   - If you've created or customized a dashboard that could benefit the wider Jitsi community, you can contribute by updating the relevant dashboard JSON file.
   - Here’s how to do it:
     - Export the JSON file of your customized dashboard as described above.
     - Locate the corresponding Jitsi component dashboard JSON file in the repository (e.g., `jicofo-dashboard.json`, `jvb-dashboard.json`).
     - Update the existing JSON file with your changes.
     - Submit a pull request to the repository with the updated JSON file and a brief description of the changes you made.
   - This helps improve the pre-configured dashboards and makes your contributions available to all users.
5. **Grafana Provisioning:**
   - The Jitsi Meet Log Analyser uses Grafana provisioning to manage dashboards. When you update a dashboard JSON file in the repository, it will be automatically provisioned in Grafana when the stack is deployed.
   - This ensures that everyone using the repository gets the latest version of the dashboards without needing to manually import them.

By following these steps, you can not only customize your own monitoring setup but also contribute improvements back to the Jitsi community.

Troubleshooting

If you encounter issues while setting up or using the Jitsi Meet Log Analyser, here are some common problems and their solutions:

1. **Grafana Not Starting:**
   - Check if the Grafana container is running with `docker ps` and inspect logs using `docker logs grafana` for any errors.
2. **No Logs in Grafana Dashboards:**

- Ensure that Jitsi Meet components are generating logs. Clear browser cache, reload Grafana. Ensure OpenTelemetry, Loki, and Grafana containers are all running with `docker ps`, and inspect each container's logs for issues using `docker logs <container_name>`.

3. **OpenTelemetry Collector Not Forwarding Logs:**
   - Check OpenTelemetry's logs with `docker logs otel`, ensure it's connected to the correct endpoints, and verify the log format is correct.
4. **Authentication Failures in Grafana:**
   - Restart Grafana with `docker restart grafana otel`. If still unsuccessful, delete the data volume with `docker-compose down -v` and restart to reset to default credentials (admin/admin).
5. **Slow Queries:**
   - If LogQL queries are slow, try optimizing the query in Grafana.
6. **Permission Issues:**
   - If you encounter permission issues, make sure that Docker has the necessary access rights to the directories where logs are stored.
7. **Docker Network Issues:**
   - Verify Docker network connections, IP range, and restart the network if necessary.
8. **OpenTelemetry Collector Not Forwarding Logs:**
   - Check OpenTelemetry logs, verify configuration, and ensure log format compatibility.
9. **Docker Containers Failing to Start:**
   - Use `docker-compose logs` to view detailed startup errors, and check for common issues like incorrect configurations.

Acknowledgements

We appreciate your interest in the Jitsi Meet Log Analyser project! If you encounter any issues or have questions, feel free to reach out to the project maintainers or contribute to the repository.

Last updated on **Mar 7, 2025**