Conseils : T**rois éléments** pour structurer l'analyse du trafic et accompagner les tests de charge, dans un objectif de supervision active et de planification de la maintenance du SI.

1. Dashboard Grafana – Import JSON

Objectif : **Visualiser CPU, RAM, disque, latence réseau, disponibilité cible.**

Modules inclus :

- node_exporter (CPU, RAM, disque)
- blackbox_exporter (latence, reachability)
- probe_duration_seconds, probe_success
- rate(node_cpu_seconds_total{}), node_memory_MemAvailable_bytes

[Fichier JSON prêt à importer sera généré sur demande, selon version Grafana]

2. Procédure d'usage : Sauvegarde automatisée des requêtes Prometheus

Objectif : **Conserver un historique de requêtes exécutées (tests de charge, post-mortem, analyse comparative).**

Étapes :

1. Créer une tâche cron avec curl :

Prometheus Query Cron Job

```
*/5 * * * * curl -s "http://localhost:9091/api/v1/query?query=probe_duration_seconds" >> /var/log/prometheus_queries.log
```

2. Ajouter un horodatage dans le log :

Memory Calculation

```
node_memory_MemAvailable_bytes / 1024 / 1024
```

3. Vérifier l'évolution avec :

Filesystem Utilization Calculation

```
(node_filesystem_avail_bytes{fstype=~"ext4|xfs"} / node_filesystem_size_bytes{fstype=~"ext4|xfs"}) * 100
```

3. Script Bash – Export CSV/JSON pour interprétation

Prometheus Export

```bash
#!/bin/bash

# Prometheus API URL
API="http://localhost:9091/api/v1/query?query="

# Requête d'exemple
QUERY="rate(node_cpu_seconds_total{mode!='idle'}[1m])"

# Récupération JSON
curl -s "${API}${QUERY}" | jq > prometheus_output.json

# Conversion CSV simplifiée
jq -r '.data.result[] | [.metric.instance, .value[0], .value[1]] | @csv' prometheus_output.json > prometheus_export.csv

echo "[✓] Export terminé : prometheus_export.csv"
```

Interprétation et usage pour l'exploitation

Indicateur	Signification	Conséquences SI / Maintenance
rate(node_cpu_seconds_total)	**Charge CPU réelle par instance**	**Surconsommation → besoin de montée en charge ou migration**
node_memory_MemAvailable_bytes	**Mémoire disponible**	**Seuil critique = ajout RAM / containerisation**
node_filesystem_avail_bytes	**Espace disque libre**	**Alertes proactives avant saturation**
probe_success	**Accessibilité cible (0 ou 1)**	**Cibles instables → failover, ajout de redondance**
probe_duration_seconds	**Temps de réponse (latence réseau)**	**Surveiller latence sous test de charge**

En pratique :

- Ces indicateurs deviennent cruciaux pour **planifier les mises à jour**, prévoir la **dégradation progressive** d'un service, ou ajuster dynamiquement les **ressources (scale up/down)**.
- En situation de montée en charge, tu identifies où ça coince : CPU, IO, réseau, DNS, etc.
- L'historisation (via CSV ou dashboards Grafana avec annotations) te donne un socle d'analyse post-incident.

Phase 1 — Dashboard Grafana : Résolution DNS + Corrélations Blackbox

Objectif :

Créer un **dashboard Grafana** dédié à :

- **la résolution DNS** via **blackbox_exporter_dns**
- **la corrélation des alertes up{}** et probe**_success{}** pour affiner la détection d'anomalies
- **La latence de réponse** (**probe_duration_seconds**) pour détecter lenteurs et congestions réseau

Étapes à suivre

1. Création d'un Dashboard dédié dans Grafana

- **Nom : Monitoring DNS et Corrélations Réseau**
- **Dossier : Production / Maintenance SI**
- **Permissions : Lecture seule** pour **les viewers**, modification pour les **ops**
-

2. Panels à créer (extraits Grafana prêts à importer)

Panel	Requête PromQL	Objectif
Résolution DNS réussie	probe_success{job="blackbox_exporter_dns"} == 1	**Vérifie que le domaine se résout correctement**
Durée de résolution DNS	probe_dns_lookup_time_seconds{job="blackbox_exporter_dns"}	**Affiche la latence de lookup**
Latence totale (tous modules)	probe_duration_seconds	**Visualisation globale de la latence**
Corrélation up{} vs probe_success{}	(up == 1) and (probe_success == 0)	**Service UP mais pas joignable via Blackbox (potentiel bug réseau ou firewall applicatif)**
Top des latences > 2s	topk(5, probe_duration_seconds > 2)	**Identifier les lenteurs critiques**
Statut TCP/ICMP par cible	`probe_success{job=~"blackbox_exporter_(icmp	**tcp)"}`**

Phase 2 — Corrélation logique : up{} vs probe_success{}

Pourquoi cette corrélation ?

Cas	up{}	probe_success{}	Interprétation
OK	1	1	Tout fonctionne
OK interne, KO externe	1	0	Service en ligne, mais bloqué à l'extérieur (firewall, DNS, proxy)
DOWN complet	0	0	Service indisponible, crash ou cible inaccessible
Cas rare	0	1	Anomalie de scrape Prometheus, mais Blackbox répond — à surveiller

Règle Prometheus possible à ajouter :

ServiceUpButProbeFails

```yaml
- alert: ServiceUpButProbeFails
  expr: up == 1 and probe_success == 0
  for: 1m
  labels:
    severity: warning
  annotations:
    summary: "Service up mais inaccessible via Blackbox"
    description: "L'instance {{ $labels.instance }} répond au scrape Prometheus mais échoue via Blackbox ({{ $labels.job }})."
```

1. Contenu du Dashboard "Telegraf System Dashboard (Prometheus)"

Voici les **quatre panels principaux** actuellement intégrés :

Panel	Expression PromQL utilisée	Interprétation
CPU Usage %	100 - (avg by (instance) (irate(cpu_usage_idle{job="telegraf"}[5m])) * 100)	Charge CPU globale, tous cœurs.
Memory Usage %	(1 - (mem_available_percent{job="telegraf"} / 100)) * 100	Pourcentage de RAM utilisée.
Disk Usage %	(1 - (disk_free_percent{job="telegraf"} / 100)) * 100	Taux d'occupation du disque.
Network RX/TX	rate(net_bytes_recv{job="telegraf"}[5m]) et rate(net_bytes_sent{job="telegraf"}[5m])	Débit entrant/sortant par interface.

Ces panels couvrent les **indicateurs clés** nécessaires pour :

- Détecter une **saturation CPU ou mémoire**,
- Anticiper un **manque d'espace disque**,
- Analyser la **charge réseau** (utile en test de montée en charge ou en cas de DoS).

🔍 2. Usage pour analyse de charge et compréhension SI

Lors d'un **test de charge**, voici comment exploiter ces données :

Étape	Analyse à mener	Conséquence pour le SI
1. **Montée CPU**	Si >90% sur plusieurs minutes : surcharge probable d'un service.	Revoir dimensionnement ou scaling vertical/horizontal.
2. **Mémoire disponible < 500 Mo**	Risque de swap / crash.	Augmenter la RAM ou optimiser les services.
3. **Disk usage > 90%**	Journalisation excessive ? Logs mal gérés ?	Nettoyage / logrotate / ajout disque.
4. **Débit réseau élevé / burst**	Identifier les pics > normal.	Corréler avec l'heure du test ou attaque potentielle.

3. Suivi et corrélation recommandée

Pour les **corrélations**, crée un **dashboard complémentaire** :

- **Panel 1** : probe_success (Blackbox Exporter, par job)
- **Panel 2** : up{} (Prometheus global health check)
- **Panel 3** : rate(net_bytes_recv{job="telegraf"}[5m]) + rate(net_bytes_sent{job="telegraf"}[5m])
- **Panel 4** : probe_duration_seconds (latence applicative)

Cela permettra de détecter :

- Quand une instance tombe (up == 0)
- Si un service est lent (probe_duration_seconds > 2)
- Si le réseau est saturé
- Si la charge CPU/mémoire coïncide avec la baisse de disponibilité

4. Étapes suivantes

1. **Importer les dashboards JSON** dans Grafana si ce n'est pas encore fait.
2. **Configurer les alertes Prometheus** pour CPU, mémoire, disque, et latence.
3. Activer la **sauvegarde automatique de ces métriques** en JSON/CSV avec un script Prometheus API (/api/v1/query_range).
4. Préparer un **rapport écrit d'interprétation** pour la phase de production.

Conseils quand Grafana n'arrive pas à interroger Prometheus à l'adresse spécifiée dans la configuration.

Exemple de capture, l'URL de Prometheus est définie comme : http://localhost:9090

Or, cela ne fonctionnera que si Grafana et Prometheus sont sur la *même machine*.

Cas fréquents

- **Si Prometheus est sur un autre serveur (VM, conteneur, etc.), il faut remplacer localhost par l'IP réelle ou nom DNS du serveur Prometheus.**
- **Exemple : http://192.168.1.50:9090 ou : http://prometheus.internal:9090**

Du fait que Jitsi s'octroie le port 9090 l'on doit impérativement en changer. Ici l'on utilisera ici 9091.

***L'IP prometheus se cale sur l'IP de l'application visio.workeezconnect.fr : 47.156.46.238**

Étapes correctives immédiates

1. **Vérifie l'IP de Prometheus :**
   Si tu es sur une VM, taper dans la console de Prometheus : ip a

On repère l'adresse inet de l'interface active.

2. **Teste depuis Grafana (curl ou navigateur) :**
3. **curl http://<ip_prometheus>:9091/-/ready | curl http://47.156.46.238:9091/-/ready**

```
ubuntu@jitsi-tercium:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: enp3s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc prio state UP group default qlen 1000
    link/ether fa:16:3e:4e:22:87 brd ff:ff:ff:ff:ff:ff
    inet 37.156.46.238/24 metric 100 brd 37.156.46.255 scope global dynamic enp3s0
       valid_lft 66703sec preferred_lft 66703sec
    inet6 2001:1600:16:10::488/128 scope global dynamic noprefixroute
       valid_lft 66705sec preferred_lft 66705sec
    inet6 fe80::f816:3eff:fe4e:2287/64 scope link
       valid_lft forever preferred_lft forever
ubuntu@jitsi-tercium:~$
```

L'on doit obtenir : Prometheus is Ready.

Corrige dans Grafana > Data Sources > Prometheus > URL :
Exemple correct : http:// 47.156.46.238:9091

Cliquer sur "Save & test" dans Grafana.
Tu dois voir : Data source is working

Astuce complémentaire : vérifier Prometheus actif
cd : systemctl status prometheus ou dans Docker : docker ps | grep prometheus

Bonus — Vérification rapide du service Prometheus :
cd : curl http://localhost:9091/-/healthy