**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

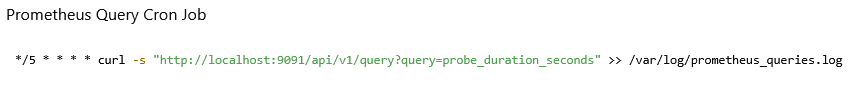
**[Fichier JSON prêt à importer sera généré sur demande, selon version Grafana]**

**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 :



1. Ajouter un horodatage dans le log :

Une image contenant texte, Police, blanc, capture d’écran

Le contenu généré par l’IA peut être incorrect.

1. Vérifier l’évolution avec :

Une image contenant texte, Police, blanc, reçu

Le contenu généré par l’IA peut être incorrect.

**3. Script Bash – Export CSV/JSON pour interprétation**

Une image contenant texte, capture d’écran, Police, algèbre

Le contenu généré par l’IA peut être incorrect.

**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{}** etprob**e\_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 :**

**Une image contenant texte, Police, capture d’écran, algèbre

Le contenu généré par l’IA peut être incorrect.**

**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**](http://192.168.1.50:9090) **ou :** [**http://prometheus.internal:9090**](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.**

1. **Teste depuis Grafana (curl ou navigateur) :**
2. **curl http://<ip\_prometheus>:9091/-/ready | curl http://47.156.46.238:9091/-/ready**

**Une image contenant texte, capture d’écran, logiciel, Police

Le contenu généré par l’IA peut être incorrect.**

**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**](http://localhost:9091/-/healthy)

**Une image contenant texte, capture d’écran, Police, ligne

Le contenu généré par l’IA peut être incorrect.**