Plateforme de fouille de données à hautes performances sur Kubernetes

La plateforme "Eli"

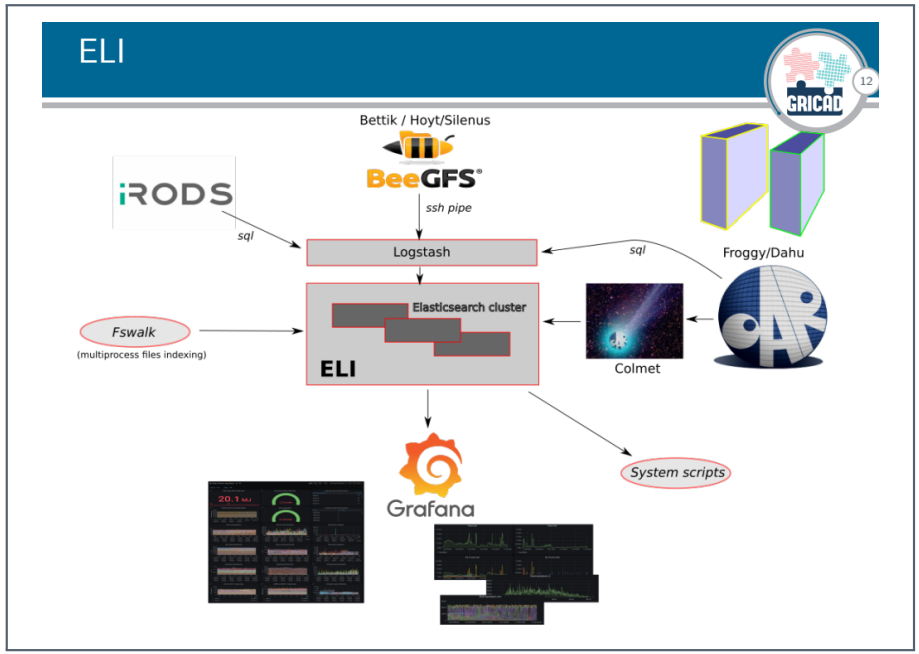
Bruno Bzeznik (GRICAD)
Oliver Henriot (GRICAD)

15 décembre 2021

GRICAD
GRENOBLE ALPES RECHERCHE
INFRASTRUCTURE DE CALCUL INTENSIF ET DE DONNÉES

il y a un an...

JCAD 2020: "Profiling, suivi et statistiques des jobs de calcul et des accès aux stockages distribués avec ElasticSearch et Grafana"



il y a un an...

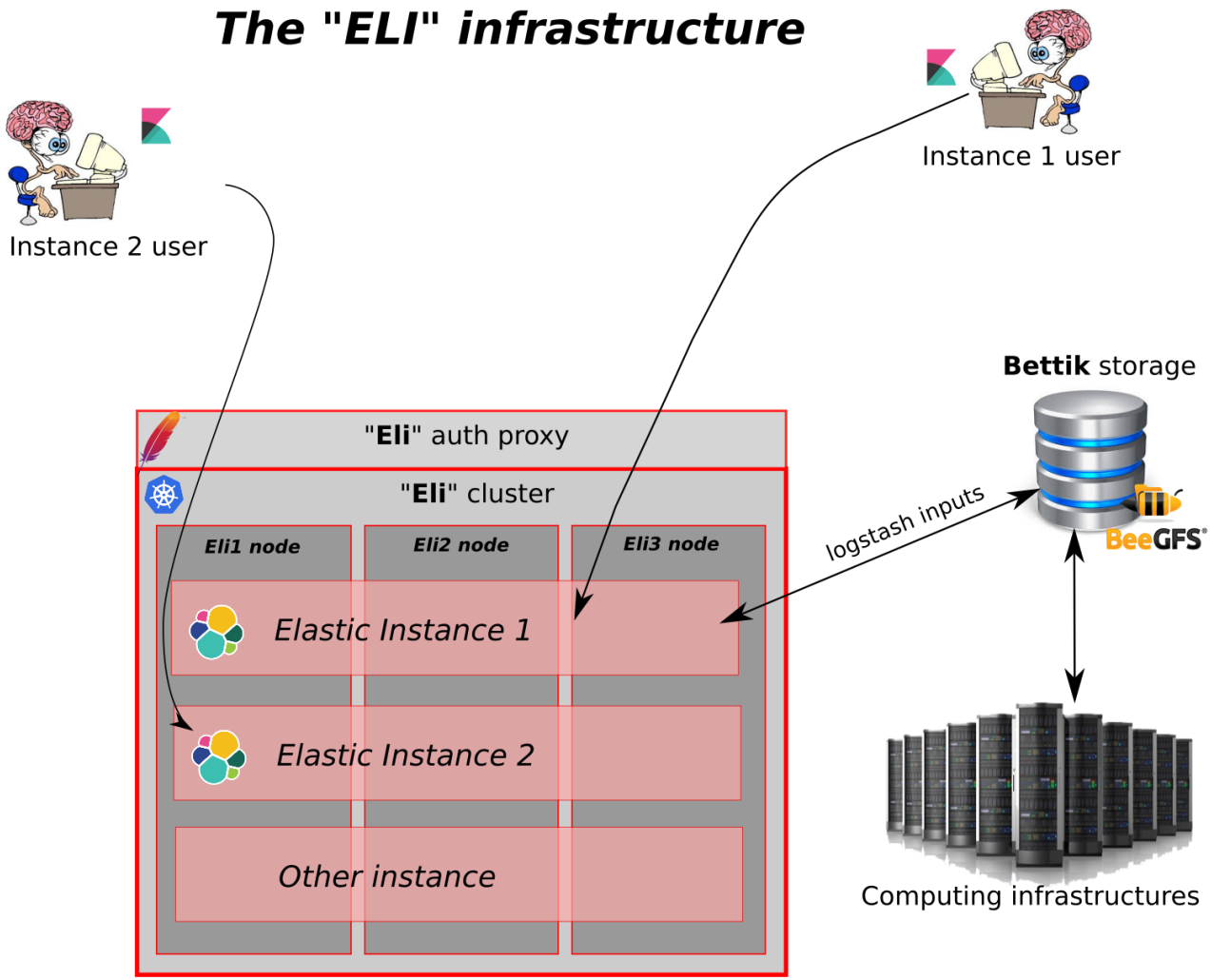
... aujourd'hui, on l'a fait!

Et si ça pouvait servir à nos utilisateurs?

"Ingestion d'un fichier de séquences ADN pour recherche d'intervalles (1.7 milliard de lignes)"

Sur la base d'un tel cas test, nous prévoyons de "Kubernet-iser" notre plateforme Eli afin d'offrir des slice ELG temporaires pour ce genre de problèmes.

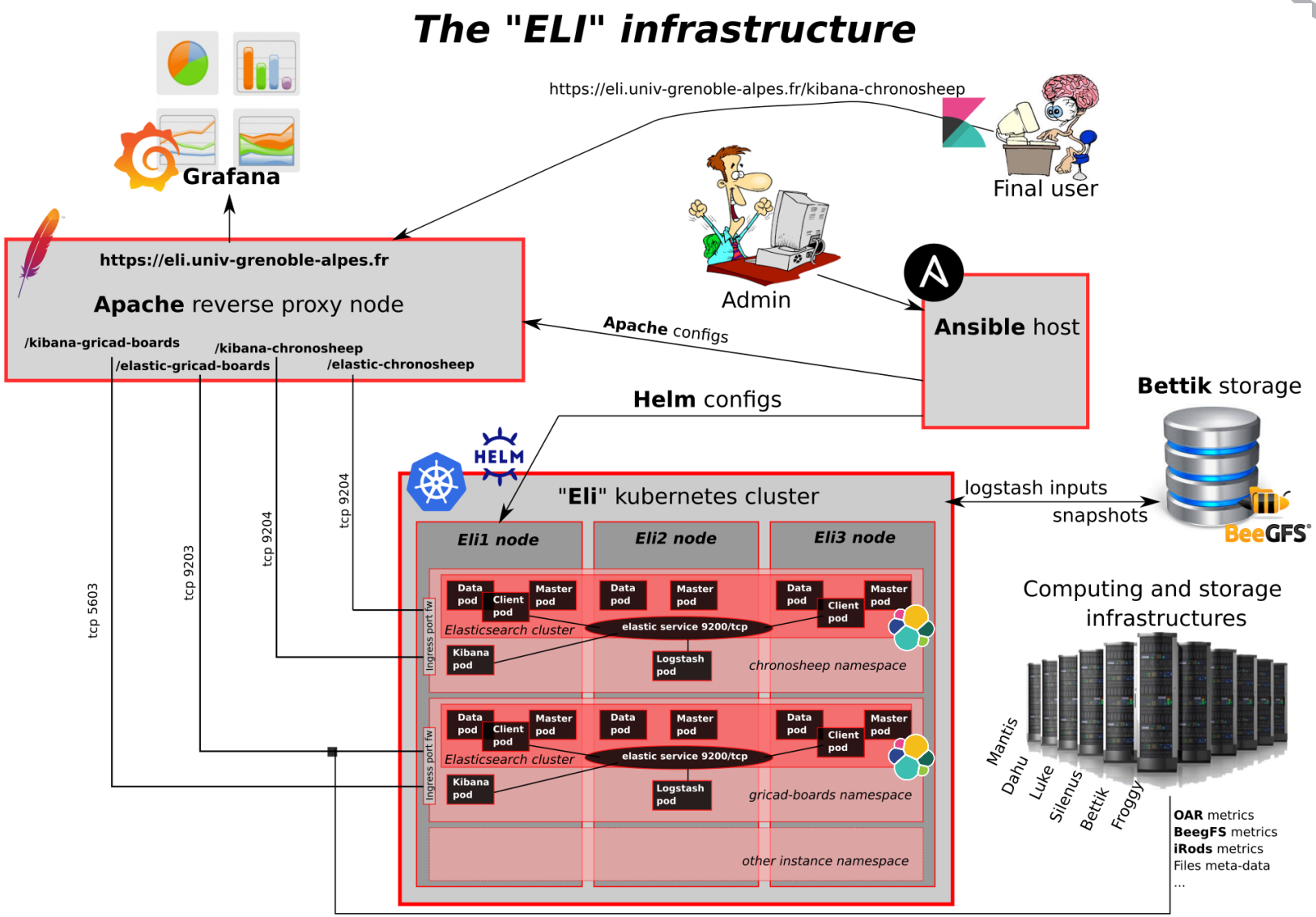
Eli: instances ELK à la demande



Exemples d'instances

- Accounting des systèmes (ce qui a été présenté l'an dernier)
- Ingestion d'un fichier de séquences ADN (1 milliard de lignes) pour recherche d'intervalles (ce qui a été fait cette année): thèse de Charlotte Her - **Apports de la datation d'évènements de mutation et de sélection à partir de génomes contemporains dans la compréhension d'un processus évolutif : la domestication du mouton.**
- Analyse d'un flux de tweets
- Stockage et analyse de données geospatiales
- ...

Eli: la solution technique



Eli: le hardware

- 3 Dell R640 nodes
  - 2 x Intel(R) Xeon(R) Silver 4214R CPU @ 2.40GHz (12 cores)
  - 192 GB RAM
  - 10 HD / RAID 5: 2,4 To, 10 000 tr/min, SAS 12 Gbit/s
- Exploitation de "Bettik", notre BeeGFS partagé en 10GbE pour les données d'entrée

ELK: Elasticsearch / Logstash / Kibana

La stack "Elastic":

- **Elasticsearch**: Base de données No-SQL, indexation et recherche très rapide. L'élément central de la stack. Tout le reste est d'ailleurs optionnnel. La gestion se fait via une API REST.
- **Logstash**: Facilite l'indexation via les 3 notions "input plugin", "filtre", "output plugin"
- **Kibana**: Outil graphique (web) pour la recherche, les statistiques et la réalisation de visualisations.
- D'autres outils sont disponibles, pour le monitoring, l'ingestion,...

La version "gratuite" ne fournit pas d'authentification (mais on peut en mettre une en place via un reverse proxy http)

Kubernetes

- Orchestrateur de containers
- Pour le cloud ou le bare-metal
- En mode bare-metal (celui qui nous intéresse, pour raisons de performance), la gestion du stockage est particulière et un peu moins bien documentée: nous avons besoin de donner un accès direct à nos containers sur les filesystèmes des noeuds physiques.
- Utilisation de **Helm** (package manager pour kubernetes: charts)

Elasticsearch clustering

- 4 types de services: master, data, ingest, client
- Facile de répartir les services sur les noeuds d'un Kubernetes, ce qui est moins évident sur une install directement bare-metal

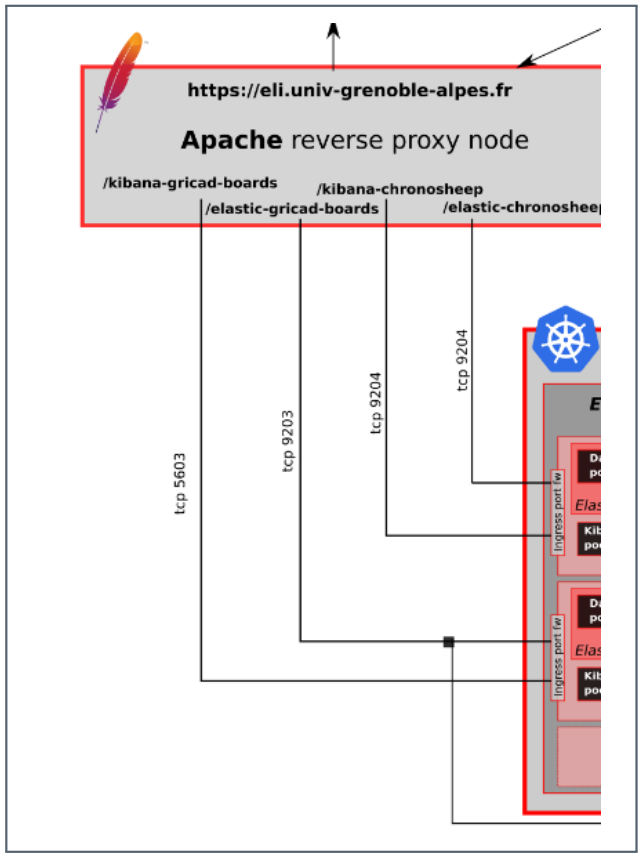
Persistent volumes

- Stockage des index: provisionnement **local** avec **node affinity** (voir ci-contre)
- Creation d'un "input" PV qui pointe sur le BeeGFS

```
kind: PersistentVolume
apiVersion: v1
metadata:
  name: eli1-chronosheep-data
  labels:
    rank: "data"
spec:
  capacity:
    storage: 5Ti
  accessModes:
    - ReadWriteOnce
  local:
    path: "/data/kubes/chronosheep_data"
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - eli1
```

Ingressing et Reverse proxy

- Exposition d'un port elasticsearch et un port kibana par instance
- Filtrage: seul le proxy peut joindre ces ports
- Proxy reverse: apache SSL avec authentification basic



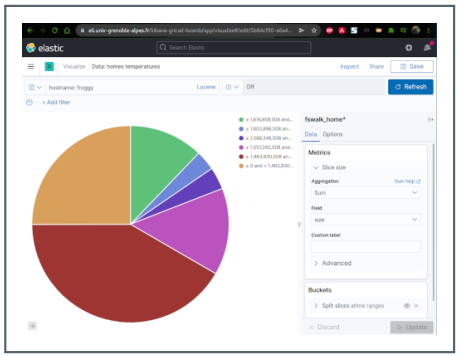
Ansible

- Ansible est utilisé pour configurer automatiquement les nouvelles instances: création des volumes peristants, déploiement d'un chart Helm, configuration Ingress et reverse proxy

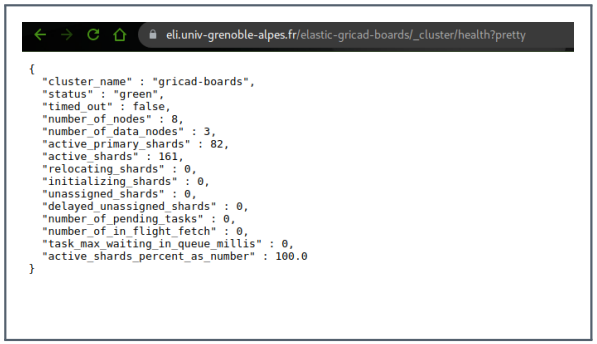
```
instance_name: chronosheep

# Note: "cpus" values are actually "threads" (hyperthreading)

# Kibana
kibana_ingress_port: 5604 # must be unique!
kibana_proxy_password: "XXXXXXXXXXXXX"

# Elastic
elastic_ingress_port: 9204 # must be unique!
elastic_proxy_password: "XXXXXXXXXXXXX"

# Master nodes
master_replicas: 3
master_cpus: 1
master_memory: 1500M
master_heap_size: 1000m
master_disk_size: 30Gi

# Data nodes
data_replicas: 3
data_cpus: 4
data_memory: 8000M
data_heap_size: 6500m
data_disk_size: 5Ti

# Client nodes
client_replicas: 2
client_cpus: 4
client_memory: 8000M
client_heap_size: 6500m
client_indice_breaker_limit: 6GB
client_max_buckets: 100000

# Logstash node
# You can customize logstash config into template/logstash.j2
  # Input volume
input_node: eli2
input_path: /bettik/bzizou/eli_inputs
input_vol_size: 2Ti
  # Logstash pod
logstash_cpus: 24
logstash_memory: 32G
logstash_heap_size: 30g
logstash_workers: 12
logstash_batch_size: 1800
logstash_index_name: seq_freq_tmrca_cat3
```

Du point de vue de l'utilisateur

- Kibana: *https://eli.univ-grenoble-alpes.fr/kibana-instance1*



- Elastic API: *https://eli.univ-grenoble-alpes.fr/elastic-instance1*



Exemple de requêtage via python

Use case "recherche de fréquences dans un fichiers de séquences d'ADN de mout


```
def search(freq_min,freq_max,tmrca_min,tmrca_max):
  f_min=str(freq_min)
  f_max=str(freq_max)
  t_min=str(tmrca_min)
  t_max=str(tmrca_max)
  body={"query": {
            "bool": {
              "must": [
                {
                  "query_string": {
                    "query": "freq:(>="+f_min+" AND <"+f_max+") AND tmrca:(>="+t_min+" AND <"+t_max+")"
                  }
                }
              ]
            }
          },
          "aggs": {
            "seq_agg": {
              "terms": {
                "field": "seq",
                "order": {
                  "_key": "asc"
                },
                "size": 5
              }
            }
          }
        }
  return es.search(index=index,body=body,request_timeout=timeout)

freq_=0
tmrca_=0
for tmrca in range(bin_tmrca_size,500000,bin_tmrca_size):
    for f in range(int(bin_freq_size*10000),10000,int(bin_freq_size*10000)):
        freq=float(f)/10000
        output={ "freq_min" : freq_, "freq_max" : freq,
                 "tmrca_min" : tmrca_, "tmrca_max": tmrca
                }
        result=search(freq_,freq,tmrca_,tmrca)
        counts={}
        for item in result['aggregations']['seq_agg']['buckets'] :
            counts[item["key"]]=item["doc_count"]
        output["counts"]=counts
        print(json.dumps(output)+",")
        sys.stdout.flush()
        freq_=freq
    tmrca_=tmrca
```

Du point de vue de l'admin

- Etude du besoin de l'utilisateur
- Création et déploiement d'une instance via Ansible
- Mise en place config spécifique des index et des PV input, avec une éventuelle config logstash ou autre moyen d'ingestion (nécessite compétences k8s de base, et compétences elastic/logstash)

Tests de performance
1/2

- Avant l'installation de la plateforme, nous devions valider les choix technologiques, en particulier celui de Kubernetes et des performances qu'on pouvait obtenir d'une instance ELK gérée de cette manière.
- Nous avons donc utilisé plusieurs benchs:
  - L'ingestion d'un gros fichier de fréquences d'ADN - (notre use case de départ)
  - Le comptage dans 50000 intervalles dans les fréquences précédemment indexées (toujours pour ce use-case)
  - Le benchmark Rally
  - Le ressenti de la navigation dans nos graphes de monitoring sous Grafana, en particulier lors du changement d'intervalles de dates dans les timeseries
- Référence sur les 3 serveurs en bare-metal
- Comparaison avec notre ancienne instance mono-serveur

Tests de performance
2/2

	Mono-server	Bare metal cluster	Kubernetes
Ingest rate	-	105k docs/s	136k docs/s
Ingest time	impossible	2h50	2h
Intervals search	impossible	7h55	8h
Rally "geopoints"	-	406k docs/s	301k docs/s
Grafana	slow	very reactive	very reactive

Tests de performance
conclusions

- Meilleures perfs avec notre use case principal (ingestion et recherche sur 1 milliards de seq ADN) sur Kubernetes!: dû au fait qu'on a séparé les différents services elastic sur différents containers, en allouant finement des ressources (ce qu'on ne peut pas faire facilement en clustering bare-metal). Le logstash a aussi été "placé" dans un container, augmentant probablement aussi sensiblement ses performances
- Le cas "geopoints" du bench Rally montre une faiblesse sur notre instance de test en k8s. Manque de temps pour mener plus d'essais, mais il est probable que ce test soit très sensible à la latence, avec de très petits documents (quelques entiers, à indexer, sans transformation)
- Suite du use case: Environ 15 fichiers de 400 millions de lignes, soit 6 milliards d'entrées

K8S + ELK = Succes!

Conclusion

Installer notre cluster Elasticsearch avec une couche Kubernetes nous permet de fournir des instances indépendantes de stacks complètes ELK, mais nous permet également dans certains cas de gagner en performances, par un meilleur contrôle et placement des différents services. L'ajout de noeuds supplémentaires pour augmenter les performances de la solution ne devrait pas poser de problèmes.

Perspectives

- Ouverture plus large à nos utilisateurs
- Test d'autres solutions NoSQL
- Augmentation des perfs globales par ajout de nouveaux noeuds

Merci!

Doc (privée) d'admin, rushs d'installation, benchs... à votre disposition:
Bruno.Bzeznik@univ-grenoble-alpes.fr