

Rapport de Stage de Tronc Commun EPITA

## CI/CD Plateforme TRUST

CHARTIER Abel

EPITA - PROMOTION 2027



**CEA Saclay - LCAN**

---

*Encadrants :*

Dr. Rémi BOURGEOIS — remi.BOURGEOIS@cea.fr — CEA/DES/ISAS/DM2S/LCAN

M. Adrien BRUNETON — adrien.BRUNETON@cea.fr — CEA/DES/ISAS/DM2S/LCAN

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Présentation de l'entreprise</b>	<b>2</b>
2.1	Le secteur d'activité . . . . .	2
2.2	L'entreprise . . . . .	2
2.3	Le service . . . . .	3
2.4	Le positionnement du stage dans les travaux de l'entreprise . . . . .	3
<b>3</b>	<b>Travail effectué</b>	<b>4</b>
3.1	Objectifs du stage . . . . .	4
3.2	Infrastructure d'intégration continue existante . . . . .	4
3.3	Infrastructure développée . . . . .	5
3.4	Ressources et accompagnement . . . . .	5
3.5	Compte-rendu d'activité . . . . .	6
3.5.1	Axes d'étude et de recherche choisis . . . . .	6
3.5.2	Déroulement concret des études, expérimentations et mises au point . . . . .	6
	Choix architectural : templates vs matrices . . . . .	9
	Stratégie de cache à deux niveaux . . . . .	9
	Mécanisme de matching de branches . . . . .	9
	Gestion des artefacts . . . . .	9
3.6	Développement, interprétation et critique des résultats . . . . .	11
3.6.1	Validation de l'infrastructure développée . . . . .	11
3.6.2	Limitations identifiées et perspectives d'amélioration . . . . .	11
	Support GPU AMD . . . . .	11
	Tests sur supercalculateurs . . . . .	11
	Optimisation des temps de tests . . . . .	12
	Observabilité avancée . . . . .	12
3.6.3	Comparaison avec l'infrastructure précédente . . . . .	12
3.6.4	Comparaison avec l'infrastructure précédente . . . . .	12
3.6.5	Bilan et perspectives . . . . .	12
<b>4</b>	<b>Conclusion générale</b>	<b>13</b>

# 1 Introduction

Le présent rapport expose les travaux réalisés au cours du stage de fin de tronc commun de l'*Ecole Pour l'Informatique et les Techniques Avancées* (EPITA) à Paris, du 1 septembre 2025 au 30 janvier 2026. Il a été réalisé au *Commissariat à l'Energie Atomique et aux énergies alternatives* (CEA) Paris-Saclay, plus particulièrement dans le *Service de Génie Logiciel pour la Simulation* (SGLS), au sein du *Laboratoire de Calcul Avancé Neutronique* (LCAN), sous la tutelle de M. Bourgeois et M. Bruneton.

L'objectif principal du stage a été de moderniser et d'industrialiser l'infrastructure d'intégration continue (CI) de TRUST, un logiciel open-source de thermohydraulique développé par le CEA, ainsi que de l'ensemble de l'écosystème de projets qui en dépendent. Mon travail visait à mettre en place des pipelines de CI robustes et automatisés, capables de gérer les spécificités des architectures GPU et des environnements de calcul haute performance. Le but était d'assurer la fiabilité et la reproductibilité des tests, tout en respectant les contraintes de sécurité élevées imposées par le contexte du CEA. Cette approche permettrait non seulement d'améliorer la qualité du code et de détecter les régressions plus rapidement, mais aussi de faciliter le déploiement continu des nouvelles fonctionnalités, rendant ainsi possible l'évolution rapide et sécurisée de l'écosystème TRUST. S'inscrivant dans le développement des machines exascale en Europe et en France, la mise en place de tests automatisés et l'orchestration de campagnes de validation constituèrent une partie importante de ma mission. Ces tests permettaient de garantir la stabilité des logiciels sur des architectures hybrides CPU/GPU et de s'assurer que les performances et la précision des simulations restaient optimales.

Mon choix pour ce stage au CEA s'appuie aussi sur la réputation de l'établissement en matière de recherche appliquée et sur la possibilité de contribuer à un projet open-source, s'appuyant sur des technologies DevOps de pointe dans un contexte exigeant. Le CEA est reconnu pour son excellence dans le domaine de la recherche scientifique et technologique, et travailler dans un tel environnement représentait une opportunité unique d'acquérir des compétences avancées en DevOps et en gestion d'infrastructures de calcul haute performance, tout en collaborant avec des experts. Enfin, la perspective de travailler dans un environnement de haute sécurité et de gérer des infrastructures complexes a fortement encouragé ma décision. Contribuer à un projet open-source d'envergure utilisé par la communauté scientifique représentait une opportunité unique pour approfondir mes connaissances et pour participer à des projets à la pointe de la technologie.

Les travaux effectués au cours de ce stage se sont structurés en quatre phases. Chacune de ces parties a fait l'objet d'un rapport qui détaille mon cheminement dans la mise en place et l'optimisation de l'infrastructure DevOps.

1. Découverte de l'écosystème TRUST et de son architecture logicielle. Familiarisation avec les contraintes de sécurité et les spécificités des environnements de calcul haute performance du CEA.
2. Modernisation de la CI de TRUST avec mise en place de pipelines automatisés pour les tests sur architectures CPU et GPU. Intégration des mécanismes de validation et de tests de non-régression.
3. Extension de l'infrastructure de CI à l'ensemble des projets dépendants de TRUST. Optimisation des temps de build et de test. Mise en place de mécanismes de monitoring et de reporting.
4. Rédaction d'un rapport et d'une documentation technique sur l'infrastructure mise en place et les procédures de maintenance.

## 2 Présentation de l'entreprise

### 2.1 Le secteur d'activité

Le CEA est un acteur clef de la recherche en France. La recherche scientifique est un secteur dans lequel les travaux menés sont caractérisés par leur ambitions, leur complexité et leur innovation technologique. Ils sont entrepris pour répondre à de grands défis industriels (exploitation de l' énergie nucléaire dans le civil et le militaire) et sociétaux (transition écologique, stockage de l'énergie) et impliquent de nombreuses collaborations avec des partenaires académiques ou industriels. Les missions du CEA s'articulent autour de différents axes :

- Défense et sécurité nationale : Il est chargé de répondre aux enjeux de la dissuasion nucléaire (renouvellement des armes nucléaires et des chaufferies nucléaires de propulsion navale, lutte contre la prolifération nucléaire).
- Energies : Le CEA est au premier plan de la transition énergétique. Des recherches sur les façons de produire de l'énergie bas carbone y sont menées.
- Transition numérique : L'expertise du CEA dans les domaines de l' électronique et du numérique lui permette de concevoir des plateformes technologiques innovantes en micro et nano-électronique, en robotique, en intelligence artificielle et en technologies quantiques.
- Recherche fondamentale : L'institution investit dans la recherche théorique d'excellence, aussi bien dans les domaines de la physique, de l' astrophysique, des sciences des matériaux, de la chimie, de la biologie et de la santé.
- Assainissement et démantèlement des installations nucléaires en fin de vie : Le CEA est un expert de la gestion des déchets nucléaires.
- Technologies de la santé : L'organisme est particulièrement actif dans le domaine de l'imagerie médicale, avec entre autre le développement d'IRM à très haut champ.

### 2.2 L'entreprise

Le CEA est un établissement à caractère scientifique, technique et industriel. Il est placé sous la tutelle des ministres chargés de l'énergie, de la recherche, de l'industrie et de la défense. Fin 2022, il emploie plus de 21 000 salariés, pour un budget annuel de 5,8 milliards d'euros. Le CEA a pour mission principale de développer les applications de l'énergie nucléaire dans les domaines scientifique, industriel et de la sécurité nationale.

Avec ces 9 centres de recherche, le CEA valorise les technologies qu'il développe et les transfère vers l'industrie en soutenant ainsi la compétitivité et la souveraineté des entreprises technologiques françaises. Il est ainsi un véritable moteur de l'innovation. Le CEA en quelques chiffres :

- 1<sup>er</sup> déposant français de brevets en Europe
- 6 980 familles de brevets actives
- 700 partenaires industriels
- > 5000 publications

Afin de répondre à ses missions, le CEA se décompose en quatre directions opérationnelles :

- Direction des énergies (DES)
- Direction des applications militaire (DAM)
- Direction de la recherche technologique (DRT)
- Direction de la recherche fondamentale (DRF)

## 2.3 Le service

Le SGLS est un service de la DES, plus précisément de l'*Institut des Sciences Appliquées et de la Simulation pour les énergies bas carbone* (ISAS), dans le *Département de Modélisation des Systèmes et des Structures* (DM2S). Cet entité du CEA se compose de 40 collaborateurs et 20 étudiants. Il conçoit, développe, distribue et maintient des outils informatiques et plateformes logicielles dont les logiciels open-source SALOME, TRUST et Uranie, qui représentent plus de 100 000 téléchargement/an. Ses domaines de recherche sont :

- Conception Assistée par Ordinateur
- Calcul Haute Performance
- Intelligence Artificielle
- Science des données
- Maillages
- Interfaces Homme Machine
- Méthodes numériques
- Modélisation physique
- Quantification des incertitudes

Le SGLS fournit des outils open-source utilisés par les équipes métiers du CEA pouvant s'appliquer en neutronique, thermohydraulique, mécanique, science des matériaux que ce soit dans le cadre du nucléaire, du spatial ou des nouvelles technologies de l'énergie. Sur l'ensemble de ces activités, le SGLS ambitionne de proposer une interface entre les communautés scientifiques académiques et le monde de l'industrie. Grâce à la présence permanente de doctorants et de chercheurs habilités à diriger des recherches, le SGLS est alimenté par une veille technique active pour un suivi de l'état de l'art.

## 2.4 Le positionnement du stage dans les travaux de l'entreprise

Le déploiement des machines exascale en Europe ouvre une nouvelle ère dans le calcul haute performance, avec des capacités de calcul inédites. Acteur de premier plan dans ce domaine, le CEA accompagne cette évolution en intégrant les GPU au sein des architectures de calcul, en complément des CPU multi-cœurs traditionnellement utilisés. Cette architecture hybride, alliant CPU pour leur polyvalence et GPU pour leur puissance de calcul parallèle, impose de nouveaux défis, notamment en matière de gestion énergétique, de consommation mémoire et d'optimisation des échanges entre ces processeurs.

Les codes de calcul scientifique, qui occupent une place centrale dans des applications comme la thermohydraulique ou la mécanique des fluides, doivent s'adapter à cette transformation vers le calcul hybride. Néanmoins, cette évolution impose des contraintes strictes en matière de validation, de tests et de déploiement continu, d'autant plus dans un environnement où la sécurité et la fiabilité des résultats sont primordiales. C'est précisément dans ce contexte que s'inscrit mon stage, où j'ai pour mission d'assurer l'intégration et le déploiement continu de ces outils de simulation sur des infrastructures de calcul avancées.

Au sein du LCAN, mon travail se concentre sur la gestion de l'intégration continue (CI) de TRUST, un logiciel open-source de thermohydraulique développé par le CEA. Ce dernier est un environnement dédié aux calculs intensifs et à la simulation scientifique. Mon rôle consiste à mettre en place et maintenir des pipelines de CI adaptés aux architectures GPU, à orchestrer les tests sur supercalculateurs dans un contexte de haute sécurité, et à assurer la CI de l'ensemble des projets dépendant de TRUST. Ce travail contribue directement aux recherches du CEA pour garantir la fiabilité et les performances des outils de simulation tout en répondant aux exigences de sécurité des infrastructures de calcul de prochaine génération.

## 3 Travail effectué

### 3.1 Objectifs du stage

L'évolution des pratiques de développement logiciel dans le domaine du calcul scientifique a été marquée ces dernières années par l'adoption généralisée des approches GitOps. Ces méthodologies, qui s'appuient sur des systèmes de contrôle de version et d'intégration continue, permettent d'automatiser les processus de test, de validation et de déploiement des logiciels. L'arrivée de plateformes comme GitLab a considérablement facilité la mise en œuvre de ces pratiques en proposant des outils intégrés pour orchestrer l'ensemble du cycle de vie du développement logiciel.

Dans ce contexte, la Direction de la Recherche Technologique (DRT), une autre direction opérationnelle du CEA, a lancé via son laboratoire DeepLab une instance GitLab privée, interne au CEA. Cette infrastructure offre aux équipes de recherche un environnement sécurisé pour héberger leur code source et mettre en place des pipelines d'intégration continue adaptés aux exigences de sécurité de l'institution.

Cependant, TRUST, le logiciel de thermohydraulique développé par le CEA depuis plus de vingt ans, n'a pas bénéficié de ces évolutions récentes. Son code source, bien que continuellement amélioré sur le plan scientifique, accuse un retard significatif en matière de pratiques GitOps. Cette situation s'explique en partie par l'historique du projet et la complexité de ses contraintes techniques. En effet, TRUST doit être testé sur de multiples architectures matérielles afin de garantir sa portabilité : architectures CPU traditionnelles, architectures hybrides CPU-GPU, supercalculateurs, et différents systèmes d'exploitation. Ces tests de portabilité sont essentiels pour assurer que le logiciel fonctionne de manière fiable dans tous les environnements où il est déployé.

Au-delà du code source de TRUST lui-même, l'écosystème du logiciel comprend également les Baltiks, un ensemble de projets satellites qui dépendent de TRUST et qui étendent ses fonctionnalités pour des applications spécifiques. Ces projets nécessitent eux aussi d'être testés régulièrement pour détecter d'éventuelles régressions introduites par les évolutions de TRUST.

L'objectif principal de mon stage est donc de concevoir et de déployer une nouvelle infrastructure d'intégration continue pour TRUST et ses projets associés. Cette CI doit permettre de tester automatiquement le code source à chaque modification, de valider la portabilité sur différentes architectures, de vérifier le bon fonctionnement sur GPU, et d'assurer la non-régression des Baltiks. Ce travail s'inscrit pleinement dans la stratégie de modernisation des outils de simulation du CEA et dans l'adaptation aux nouvelles pratiques de développement logiciel.

### 3.2 Infrastructure d'intégration continue existante

Avant le lancement de ce projet, TRUST disposait déjà d'un système de tests automatisés, bien que celui-ci ne repose pas sur les standards modernes d'intégration continue. Ce système, baptisé l'Atelier, est constitué d'un ensemble de scripts Bash développés au fil des années par les équipes du laboratoire. Ces scripts s'exécutent quotidiennement sur les stations de travail des chercheurs et ingénieurs du LCAN durant la nuit, profitant ainsi des périodes d'inactivité des machines pour réaliser les tests sans perturber les activités de développement.

L'Atelier orchestre l'exécution d'une batterie de tests unitaires et d'intégration qui permettent de détecter les régressions introduites par les modifications du code. Avant chaque release officielle de TRUST, des tests plus exhaustifs et plus longs sont lancés afin de garantir la stabilité de la version publiée. Ces tests comprennent notamment des simulations complètes qui peuvent nécessiter plusieurs heures de calcul et qui sollicitent intensivement les ressources matérielles disponibles.

Bien que l'Atelier ait rendu service au projet pendant de nombreuses années, ce système présente plusieurs limitations importantes. Premièrement, sa dépendance aux machines personnelles des chercheurs pose des

problèmes de disponibilité et de reproductibilité : si une machine est éteinte ou indisponible, les tests ne sont pas exécutés. Deuxièmement, l'absence de centralisation rend difficile la supervision de l'ensemble des tests et la détection rapide des problèmes. Troisièmement, l'architecture basée sur des scripts Bash devient de plus en plus difficile à maintenir et à faire évoluer à mesure que les besoins se complexifient. Enfin, ce système ne permet pas d'intégrer facilement les nouvelles pratiques de développement collaboratif, notamment les tests automatiques déclenchés à chaque commit ou lors de l'ouverture de merge requests.

Ces constats ont motivé la décision de concevoir une nouvelle infrastructure d'intégration continue, plus moderne, plus robuste et mieux adaptée aux besoins actuels du projet TRUST.

### 3.3 Infrastructure développée

Mon travail a consisté à concevoir et déployer une infrastructure complète d'intégration continue pour TRUST, tirant parti des capacités offertes par GitLab CI/CD. Cette nouvelle infrastructure repose sur des pipelines automatisés qui se déclenchent à chaque modification du code source et qui orchestrent l'exécution de différentes catégories de tests.

La CI développée couvre plusieurs axes de validation. Premièrement, elle implémente des tests de portabilité permettant de vérifier que TRUST compile et s'exécute correctement sur diverses architectures matérielles et systèmes d'exploitation. Deuxièmement, elle intègre des tests spécifiques pour les architectures GPU, essentiels dans le contexte du passage aux supercalculateurs exascale. Troisièmement, elle assure la validation des Baltiks en testant automatiquement ces projets satellites contre la dernière version de TRUST, détectant ainsi immédiatement toute incompatibilité introduite par les évolutions du code principal.

La mise en œuvre de cette CI a nécessité le déploiement de runners GitLab sur l'infrastructure physique du laboratoire. Cette étape s'est révélée particulièrement complexe en raison des contraintes de sécurité strictes imposées par le CEA. En effet, l'ensemble de l'infrastructure doit fonctionner en mode rootless, c'est-à-dire sans privilèges administrateur, afin de minimiser les risques de sécurité. Cette contrainte a imposé des choix techniques spécifiques et a nécessité de résoudre plusieurs problèmes d'architecture réseau et de gestion des conteneurs.

L'infrastructure déployée s'appuie sur du matériel informatique de haute performance mis à disposition par le laboratoire, comprenant notamment des serveurs rackés et des stations de travail équipées de GPU consumer puissants. Cette puissance de calcul permet d'exécuter rapidement les tests, même les plus intensifs, et de fournir un retour rapide aux développeurs.

Au-delà de l'infrastructure technique, j'ai également produit la documentation nécessaire à la maintenance et à l'évolution de cette CI, ainsi que des guides d'utilisation pour les développeurs de TRUST. Cette documentation est essentielle pour assurer la pérennité du système et faciliter son appropriation par l'équipe.

### 3.4 Ressources et accompagnement

La réalisation de ce projet s'est effectuée avec le soutien de mes deux encadrants, Rémi Bougeois et Adrien Bruneton, qui m'ont accompagné tout au long du stage en apportant leur expertise technique et leur connaissance approfondie de l'écosystème TRUST. Leur disponibilité et leurs conseils ont été précieux pour orienter mes choix techniques et résoudre les problèmes complexes rencontrés.

Au-delà de mes encadrants directs, j'ai pu m'appuyer sur l'ensemble de l'équipe du LCAN, dont l'expérience collective en matière de calcul scientifique et de développement logiciel a constitué une ressource inestimable. Les échanges réguliers avec les chercheurs et ingénieurs du laboratoire m'ont permis de mieux comprendre les besoins opérationnels et les contraintes spécifiques du domaine.

Pour les aspects techniques plus spécialisés, notamment concernant l'infrastructure réseau et les questions de sécurité informatique, j'ai bénéficié du support de l'équipe RSI (Réseaux et Systèmes d'Information)

de l'ISAS. Cette équipe, joignable via un système de tickets, a apporté son expertise pour résoudre les problèmes d'infrastructure et valider la conformité de mes développements avec les politiques de sécurité du CEA.

Enfin, j'ai disposé d'un environnement matériel particulièrement favorable, avec un accès à une infrastructure physique puissante comprenant des serveurs rackés dédiés, des stations de travail équipées de GPU consumer de dernière génération, et un accès à différents environnements de calcul permettant de tester la portabilité du code dans des conditions réalistes. Cette richesse en ressources matérielles a été un atout majeur pour mener à bien les missions qui m'ont été confiées.

### 3.5 Compte-rendu d'activité

#### 3.5.1 Axes d'étude et de recherche choisis

Les axes d'étude et de développement se sont appuyés sur un cadre initial fourni par mes encadrants. Les principaux axes de travail sont les suivants :

1. Documentation et appropriation de l'écosystème TRUST. L'étude s'est concentrée sur la compréhension du fonctionnement de TRUST, de son processus de compilation et de son écosystème de tests existants (l'Atelier). Réalisation des tutoriels utilisateurs et analyse des scripts Bash de l'infrastructure existante.
2. Étude des technologies d'intégration continue et de conteneurisation. Exploration approfondie de GitLab CI/CD, des pipelines et des bonnes pratiques DevOps. Étude comparative des solutions de conteneurisation (Docker vs Podman) et des architectures de déploiement (Docker Compose, Usernetes, scripts personnalisés).
3. Exploration des contraintes de sécurité CEA. Compréhension des exigences de sécurité imposant un fonctionnement en mode rootless. Étude des implications sur l'architecture réseau (absence d'IPv4 forwarding, utilisation de slirp4netns) et sur la gestion des conteneurs.
4. Lecture de la documentation GitLab CI/CD et des runners. Étude approfondie des mécanismes de cache, des artifacts, des stratégies de parallélisation et des templates réutilisables. Exploration des patterns de configuration avancés (includes, anchors, matrices).

Ces activités de recherche et d'appropriation ont impliqué une phase intensive de documentation pendant les premières semaines du stage. Le livrable associé à ces travaux est une infrastructure complète documentée et déployée.

#### 3.5.2 Déroulement concret des études, expérimentations et mises au point

##### 3.5.2.1 Chronologie du stage

Le stage s'est déroulé sur une période de six mois, selon la chronologie suivante :

- **Semaines 1–2** : Arrivée au CEA, formalités administratives et mesures de sécurité. Découverte de l'environnement TRUST, réalisation des tutoriels utilisateurs.
- **Semaines 3–5** : Premières tentatives de compilation de TRUST dans des conteneurs Docker. Déploiement de Docker en mode rootless sur les serveurs du laboratoire.
- **Semaines 6–8** : Migration de TRUST et de ses dépendances (ExternalPackages) vers GitLab. Exploration de différentes infrastructures et logiciels de CI/CD.
- **Semaines 9–12** : Mise en place des runners GitLab via Docker Compose. Déploiement de la stack de monitoring (Prometheus, Grafana).



- **Semaines 13–16** : Développement de la CI/CD pour les tests CPU sur multiples distributions Linux. Configuration des runners GPU et intégration des tests GPU.
- **Semaines 17–20** : Déploiement du serveur de stockage distribué (MinIO) pour optimiser le partage de cache entre runners. Intégration des dépôts Baltiks dans la pipeline.
- **Semaines 21–24** : Finalisation de la documentation technique, guides utilisateurs et de maintenance. Tests de validation finale de l'infrastructure complète.

### 3.5.2.2 Déploiement de Docker en mode rootless sur l'infrastructure du laboratoire

Le premier défi technique a consisté à déployer Docker en mode rootless sur les serveurs du laboratoire. Ce choix s'est imposé pour respecter les contraintes de sécurité strictes du CEA, qui interdisent l'utilisation de privilèges administrateur pour les services de développement.

Le processus de déploiement a nécessité de nombreuses interactions avec l'équipe RSI (Réseaux et Systèmes d'Information) de l'ISAS afin d'obtenir les configurations nécessaires, notamment l'attribution de plages de subuid et subgid pour chaque utilisateur devant exécuter Docker. Ces identifiants subordonnés sont essentiels au fonctionnement du mode rootless, car ils permettent de mapper les utilisateurs conteneurisés à des plages d'UID non privilégiées sur l'hôte.

Plusieurs obstacles techniques ont émergé lors du déploiement. L'absence d'IPv4 forwarding sur les serveurs a rendu impossible l'utilisation du bridge réseau standard de Docker. La solution retenue a été d'utiliser `slirp4netns`, un outil qui implémente un réseau en espace utilisateur permettant aux conteneurs de communiquer sans nécessiter de privilèges réseau. Cette configuration particulière a été documentée dans les fichiers de configuration Docker du projet.

Un autre défi concernait les images de base pour TRUST. Les premiers essais avec des images standards se sont heurtés à deux problèmes : d'une part, TRUST refuse de compiler en tant que root pour des raisons de sécurité, nécessitant la création d'utilisateurs non privilégiés dans les conteneurs ; d'autre part, les dépendances de TRUST (regroupées dans le paquet `ExternalPackages`) représentent plusieurs gigaoctets de données, rendant les images initiales excessivement volumineuses (plus de 5 Go). Une refonte complète de la stratégie de construction des images a permis de réduire significativement leur taille en séparant les dépendances système des artefacts de compilation.

Le choix entre Docker et Podman a également été étudié. Bien que Podman soit conceptuellement plus adapté à un environnement rootless, Docker a été retenu pour sa maturité, sa documentation plus extensive et sa meilleure intégration avec GitLab Runner.

### 3.5.2.3 Migration de TRUST et de ses dépendances vers GitLab

Bien que le code source principal de TRUST ait déjà été migré vers GitLab au début du stage, le dépôt `ExternalPackages`, contenant l'ensemble des dépendances de TRUST sous forme d'archives compressées, posait un problème majeur. Ce dépôt est nécessaire car TRUST doit pouvoir se compiler en environnement isolé sans accès Internet, notamment dans les zones sécurisées du CEA. Il contient plusieurs dizaines de gigaoctets d'archives (fichiers `.tar.gz` des bibliothèques tierces).

Les premiers essais de migration directe se sont heurtés aux limitations de taille imposées par le serveur GitLab. Plusieurs tentatives de contournement par l'envoi fragmenté des données se sont révélées infructueuses et chronophages.

La solution retenue a été d'utiliser Git LFS (Large File Storage) pour gérer les archives volumineuses. Cette approche n'avait pas été envisagée initialement afin de maintenir une transition aussi fluide que possible pour les développeurs, mais elle s'est finalement imposée comme la seule option viable. La migration

a nécessité une réécriture complète de l'historique Git du dépôt ExternalPackages pour intégrer Git LFS de manière transparente, en préservant l'historique des commits.

De nombreux tests ont été effectués pour s'assurer que l'accès aux dépendances restait fonctionnel dans tous les environnements d'utilisation, notamment pour les workflows existants des développeurs. L'ensemble de ces travaux est documenté dans la pull-request de migration et dans le guide de maintenance du dépôt.

#### 3.5.2.4 Déploiement de l'infrastructure via Docker Compose

Le choix de Docker Compose pour orchestrer l'infrastructure de CI/CD s'est imposé après l'évaluation de plusieurs alternatives. Docker Compose offre un équilibre optimal entre flexibilité et simplicité : il est suffisamment permissif pour avoir des configurations avancées tout en restant accessible aux membres de l'équipe qui devront maintenir l'infrastructure.

D'autres solutions ont été considérées, notamment Usernetes (une distribution Kubernetes en mode utilisateur) et l'utilisation de scripts shell personnalisés pour déployer les conteneurs. Usernetes a été écarté en raison de sa complexité excessive pour les besoins du projet, tandis que l'approche par scripts aurait posé des problèmes de maintenabilité à long terme.

L'infrastructure déployée via Docker Compose comprend deux composants principaux : les runners GitLab proprement dits, et une stack de monitoring complète. Cette dernière repose sur Prometheus pour la collecte et le stockage des métriques, Grafana pour la visualisation, Node Exporter pour les métriques système de l'hôte, et cAdvisor pour les métriques des conteneurs.

La configuration a été conçue pour exposer les métriques de manière structurée, permettant une surveillance en temps réel de l'état des runners, de la charge système et des performances des jobs CI/CD. Plusieurs tableaux de bord Grafana préconfigurés ont été développés pendant le stage : vue d'ensemble complète de la stack, monitoring détaillé des conteneurs, métriques spécifiques aux runners GitLab et statistiques des jobs, ainsi que le monitoring système de l'hôte (CPU, mémoire, disque, réseau).

- **Prometheus** (port 9090) : collecte et stockage des métriques
- **Grafana** (port 3000) : visualisation via tableaux de bord
- **Node Exporter** (port 9100) : métriques système de l'hôte
- **cAdvisor** (port 8080) : métriques des conteneurs de l'hôte

#### 3.5.2.5 Conception et implémentation de la CI/CD pour les tests CPU

La conception de la pipeline d'intégration continue a constitué le cœur technique du stage. L'objectif était de créer une infrastructure capable de tester automatiquement TRUST sur l'ensemble des distributions Linux supportées, en parallélisant au maximum les jobs tout en optimisant l'utilisation des ressources.

La pipeline s'articule autour de six stages principaux :

1. `.pre` : détermination automatique de la branche ExternalPackages à utiliser
2. `setup` : clonage et mise à jour du dépôt ExternalPackages
3. `configure` : exécution de `./configure` pour chaque distribution
4. `build` : compilation de TRUST
5. `test` : exécution de la suite de tests
6. `.post` : sessions de debug interactives (manuelles)

Un système de nommage cohérent a été adopté pour les jobs : `<stage>:<config_name>`, facilitant l'identification rapide des problèmes (exemples : `configure:cpu_fedora40`, `build:cpu_ubuntu22`, `test:cpu_ubi9`).

Distribution	Image Docker	Gestionnaire
Fedora 40	fedora:40	dnf
Rocky Linux 9	rockylinux:9	dnf
CentOS Stream 9	centos:stream9	dnf
Ubuntu 22.04	ubuntu:22.04	apt
UBI 9	redhat/ubi9	dnf

TABLE 1 – Distributions CPU intégrées à la CI

Cinq distributions CPU ont été intégrées (table 1). Chaque distribution est définie via une ancre YAML contenant tous ses paramètres : image Docker, gestionnaire de paquets, options de configuration, cibles de tests et tags de runners.

**Choix architectural : templates vs matrices** Une décision importante a concerné la structure de la configuration. GitLab CI/CD propose deux approches principales : les matrices (permettant de générer automatiquement des jobs variés à partir d’une définition unique) et les templates réutilisables avec `include`.

La solution retenue utilise des templates avec `include`, car elle offre une meilleure lisibilité et une plus grande flexibilité. Chaque distribution est définie dans `gitlab-ci.yml` puis incluse via un template générique `build-config.yml` qui génère automatiquement les trois jobs (`configure`, `build`, `test`). Cette approche facilite l’ajout de nouvelles distributions et permet des personnalisations fines par distribution si nécessaire.

**Stratégie de cache à deux niveaux** Un système de cache sophistiqué a été mis en place pour optimiser les temps d’exécution :

- **Cache ExternalPackages** : clé `extpkgs-${EXTPKGS_BRANCH}`, partagé entre toutes les configurations. Ce cache n’est mis à jour que lorsque le dépôt `ExternalPackages` change, évitant des clonages répétés coûteux.
- **Cache de build** : clé `build-${CONFIG_NAME}-${CI_PIPELINE_ID}`, spécifique à chaque configuration et isolé par pipeline. Cette isolation prévient les conflits entre pipelines concurrents.

**Mécanisme de matching de branches** Un système intelligent de détection de branches a été implémenté pour `ExternalPackages`. Lors du job `determine_branch` en stage `.pre`, la pipeline vérifie si une branche du même nom que celle du dépôt principal existe dans `ExternalPackages`. Si oui, elle est utilisée ; sinon, la branche `next` est sélectionnée par défaut. Ce mécanisme permet un développement coordonné multi-dépôts, facilitant les évolutions nécessitant des modifications synchronisées de `TRUST` et de ses dépendances.

**Gestion des artefacts** Les artefacts sont conservés selon leur utilité : logs de configuration et compilation (30 jours), résultats de tests au format JUnit avec listes de tests échoués (30 jours), et fichiers d’environnement indiquant les branches utilisées et le statut du cache (1 jour).

Les jobs utilisent le mot-clé `needs` pour optimiser la parallélisation, avec `artifacts: false` lorsque seule la complétion du job importe, réduisant ainsi les transferts inutiles. Cette optimisation s’est révélée particulièrement efficace pour les jobs de test, qui n’ont pas besoin de télécharger les logs de compilation.

### 3.5.2.6 Extension de la CI pour les tests GPU

L’extension de la pipeline aux architectures GPU a nécessité des adaptations spécifiques. Trois configurations GPU ont été ajoutées, basées sur les images NVIDIA CUDA officielles : CUDA 12.4 sur UBI 9, CUDA 12.5 sur UBI 9, et CUDA 12.4 sur Ubuntu 22.04.

Ces configurations utilisent des tags de runners spécifiques (`["docker", "gpu", "nvidia"]`) pour s’exécuter sur les stations de travail équipées de GPU consumer puissants mis à disposition par le laboratoire. Le passage de l’option `-cuda` à la commande `./configure` active le support GPU dans `TRUST`.

Les jobs GPU partagent la même structure que les jobs CPU (configure, build, test) mais nécessitent des timeouts légèrement plus longs en raison de la complexité accrue des tests. La stratégie de cache reste identique, garantissant que les builds GPU bénéficient également de l'optimisation ExternalPackages. Des tests spécifiques ont été ajoutés pour valider le bon fonctionnement sur GPU, notamment via la cible `ctest_optim` qui exécute la suite de tests complète avec accélération GPU.

### 3.5.2.7 *Serveur de stockage distribué pour le partage de cache*

Afin d'améliorer encore les performances de la CI, un serveur de stockage distribué basé sur MinIO a été déployé. MinIO est un serveur de stockage objet compatible S3, open-source et auto-hébergeable.

Cette infrastructure permet aux runners de partager efficacement leurs caches, notamment le volumineux cache ExternalPackages. Sans ce système, chaque runner devait maintenir sa propre copie locale du cache, entraînant une duplication importante des données et des temps de restauration de cache variables selon le runner sélectionné.

Avec MinIO, le cache est centralisé et accessible via le protocole S3. GitLab CI est configuré pour utiliser ce stockage distribué comme backend de cache, garantissant que tous les runners accèdent à la même source de vérité. Cela se traduit par des temps de restauration de cache plus prévisibles et une utilisation plus efficace de l'espace disque disponible.

Le déploiement de MinIO s'est intégré naturellement dans le Docker Compose existant, avec une configuration adaptée au mode rootless et aux contraintes réseau du CEA. Cette solution s'est révélée particulièrement efficace pour réduire les temps de setup des jobs de 5–10 minutes à moins d'une minute dans le cas où le cache ExternalPackages est déjà présent.

### 3.5.2.8 *Intégration des dépôts Baltiks*

Les Baltiks constituent des projets satellites de TRUST qui étendent ses fonctionnalités pour des applications spécifiques. Leur intégration dans la CI était essentielle pour détecter automatiquement les régressions introduites par les modifications de TRUST.

Un stage dédié `baltiks` a été créé, s'exécutant uniquement lors des merge requests pour ne pas surcharger la pipeline des commits de développement quotidien. Deux approches ont été implémentées :

- **Méthode rapide via `baltiks-distros.yml`** : permet de tester un Baltik sur toutes les distributions CPU en une seule déclaration `include`, générant automatiquement cinq jobs (un par distribution).
- **Méthode personnalisée via `baltiks-config.yml`** : offre un contrôle fin sur la configuration, permettant de tester un Baltik sur une distribution spécifique avec des options particulières (timeout personnalisé, options de configuration spéciales).

Le mécanisme de fonctionnement des jobs Baltiks suit cette séquence : (i) récupération du cache de build du job `test:${base_config_name}` correspondant, (ii) clonage du dépôt Baltik avec le même système de matching de branches que ExternalPackages, (iii) reconfiguration avec les options spécifiées, (iv) recompilation, et (v) exécution des tests du Baltik.

Les résultats sont conservés dans les artefacts pendant 30 jours, incluant les listes de tests échoués et les logs détaillés. Cette approche garantit que tout changement dans TRUST est immédiatement validé contre l'ensemble de son écosystème, réduisant drastiquement le risque de régressions non détectées.

### 3.5.2.9 *Jobs de debug interactifs*

Deux jobs de debug manuels ont été conçus pour faciliter l'investigation des problèmes :

- `debug:all` : environnement Fedora 40 avec les caches de plusieurs configurations chargés, idéal pour des investigations comparatives ou lorsque la configuration problématique n'est pas identifiée.
- `debug:config` : environnement configurable via des variables (`DEBUG_IMAGE`, `CONFIG_NAME`, `DEBUG_WORKER`), permettant de reproduire exactement l'environnement d'un job spécifique, y compris sur GPU.

Ces jobs lancent un conteneur en mode interactif pendant deux heures, donnant accès à un shell complet. Un script helper (`./scripts/debug-v2.sh`) simplifie la connexion aux sessions de debug via Docker.

Cette fonctionnalité s'est révélée indispensable lors du développement de la CI pour diagnostiquer les problèmes de configuration et de dépendances sans avoir à modifier constamment les jobs et relancer des pipelines complètes. Elle est également utilisable par les développeurs pour investiguer les échecs de tests dans l'environnement exact où ils se sont produits.

### 3.6 Développement, interprétation et critique des résultats

#### 3.6.1 Validation de l'infrastructure développée

L'infrastructure déployée a été validée par plusieurs semaines de tests intensifs. Les points suivants ont été confirmés :

1. **Reproductibilité** : les pipelines peuvent être relancées avec des résultats identiques, garantissant la fiabilité des tests.
2. **Performance** : le système de cache à deux niveaux réduit significativement les temps d'exécution. Un build complet sans cache prenait initialement 45–60 minutes ; avec cache, ce temps est réduit à 8–12 minutes.
3. **Scalabilité** : l'architecture supporte l'ajout de nouvelles distributions et de nouveaux Baltiks sans modification majeure de la structure. L'ajout d'une nouvelle distribution nécessite moins de 20 lignes de configuration YAML.
4. **Portabilité multi-architecture** : validation du fonctionnement sur CPU (Intel et AMD) et GPU (NVIDIA), confirmant la compatibilité avec l'infrastructure hétérogène du laboratoire.
5. **Intégration développeur** : les retours de l'équipe ont confirmé que la CI s'intègre naturellement dans le workflow de développement, avec des feedbacks rapides sur les merge requests (moins de 30 minutes pour une validation complète CPU).

#### 3.6.2 Limitations identifiées et perspectives d'amélioration

Plusieurs limitations ont été identifiées au cours du stage :

**Support GPU AMD** Bien que l'infrastructure supporte théoriquement les GPU AMD via ROCm, aucune configuration spécifique n'a été déployée faute de matériel disponible pendant le stage. L'ajout du support AMD MI250 (disponible sur le supercalculateur Adastra) constitue une évolution naturelle, nécessitant principalement l'ajout d'images Docker basées sur ROCm et l'accès à des runners équipés de GPU AMD.

**Tests sur supercalculateurs** Les tests intensifs sur architectures multi-nœuds (Topaze, Adastra) n'ont pas encore été intégrés à la CI automatique. Cette intégration nécessiterait de déployer des runners sur ces infrastructures, ce qui pose des défis de sécurité et de gestion des ressources partagées. Une approche par jobs manuels déclenchés avant les releases majeures pourrait constituer un premier pas.

**Optimisation des temps de tests** Certaines suites de tests prennent encore plus de deux heures. Une analyse fine permettrait d'identifier les tests les plus longs et de les paralléliser davantage ou de les exécuter de manière conditionnelle (par exemple, uniquement sur la branche principale ou avant les releases).

**Observabilité avancée** Bien que Grafana fournisse des métriques système, l'ajout de métriques applicatives spécifiques à TRUST (temps par étape de compilation, couverture de tests, évolution des temps d'exécution dans le temps) enrichirait l'analyse des performances et faciliterait la détection de régressions de performance.

### 3.6.3 Comparaison avec l'infrastructure précédente

### 3.6.4 Comparaison avec l'infrastructure précédente

Par rapport à l'Atelier (table 2), l'infrastructure GitLab CI/CD apporte plusieurs améliorations majeures : centralisation de tous les résultats via l'interface GitLab contre des résultats dispersés sur différentes machines, disponibilité garantie grâce à des runners dédiés indépendants des postes de développeurs, validation automatique sur chaque merge request contre des tests nocturnes déconnectés du workflow, traçabilité complète avec historique des exécutions et artefacts conservés facilitant le diagnostic de régressions anciennes, et extensibilité via quelques lignes de YAML contre modification de scripts Bash complexes.

Critère	L'Atelier	GitLab CI
Centralisation	Non	Oui
Disponibilité	Variable	Garantie
Intégration MR	Non	Oui
Traçabilité	Limitée	Complète
Temps feedback	12-24h	30-60 min
Extensibilité	Difficile	Facile

TABLE 2 – Comparaison des infrastructures

Cependant, l'Atelier conserve certains avantages, notamment sa simplicité conceptuelle et son indépendance vis-à-vis d'infrastructures externes. Une approche hybride, où l'Atelier continuerait à exécuter les tests les plus lourds durant la nuit pendant que la CI GitLab valide rapidement les merge requests, pourrait être envisagée pour combiner les forces des deux systèmes.

### 3.6.5 Bilan et perspectives

Ce stage a permis de moderniser significativement l'infrastructure de tests de TRUST, alignant le projet sur les pratiques GitOps contemporaines. L'infrastructure développée est désormais opérationnelle, documentée et maintenue par l'équipe du LCAN.

Les perspectives d'évolution incluent :

- Extension du support GPU avec intégration de ROCm pour les architectures AMD.
- Intégration de tests de performance automatisés avec génération de rapports de régression.
- Déploiement de runners sur supercalculateurs pour les tests à très grande échelle.
- Mise en place de dashboards de santé projet (taux de réussite des merge requests, temps moyen de validation, évolution du nombre de tests).
- Intégration d'outils d'analyse statique de code et de mesure de couverture de tests.

L'expérience acquise sur les contraintes de sécurité du CEA, la conteneurisation rootless et l'orchestration de pipelines complexes constitue un acquis précieux, transférable à d'autres projets du laboratoire souhaitant moderniser leurs pratiques DevOps.

## 4 Conclusion générale

Ce stage avait pour objectif de moderniser l'infrastructure de tests de TRUST en mettant en place une solution d'intégration continue automatisée, capable de valider le code sur multiples architectures et de tester l'ensemble de l'écosystème des projets satellites (Baltiks). Les différents travaux effectués au cours du stage ont permis d'atteindre cet objectif.

Une infrastructure complète d'intégration continue a été développée et déployée, permettant d'automatiser l'ensemble du cycle de validation de TRUST, de la compilation aux tests en passant par la vérification de non-régression des Baltiks. Cette infrastructure a été pensée pour être modulaire, extensible et adaptable aux contraintes de sécurité strictes du CEA. Grâce à cette conception, elle prend en charge cinq distributions Linux pour les tests CPU et trois configurations GPU basées sur NVIDIA CUDA, tout en offrant une architecture facilement extensible pour l'ajout de nouvelles plateformes. L'ensemble du travail réalisé constitue donc une avancée importante vers l'adoption des pratiques GitOps modernes dans le développement de TRUST.

Les résultats obtenus sont très encourageants et démontrent l'intérêt indéniable de l'automatisation des tests dans un projet de cette envergure. Les pipelines mises en place ont permis de réduire drastiquement le temps de feedback pour les développeurs, passant de 12–24 heures avec l'Atelier à 30–60 minutes avec GitLab CI pour une validation complète CPU. Le système de cache à deux niveaux développé pendant le stage permet de diviser par cinq les temps de compilation (de 45–60 minutes à 8–12 minutes avec cache). Toutefois, l'infrastructure conçue nécessite encore des optimisations et un approfondissement au niveau de son déploiement, afin d'exploiter pleinement les capacités des architectures de calcul disponibles. L'un des axes d'amélioration futurs consistera à intégrer le support des architectures GPU AMD via ROCm et à déployer des runners sur les supercalculateurs Topaze et Adastras pour les tests à très grande échelle. De plus, l'expérience acquise au cours de ce stage fournit une base solide pour le déploiement d'infrastructures CI/CD similaires sur d'autres projets du laboratoire, en s'appuyant sur les solutions techniques validées (Docker rootless, Docker Compose, stratégies de cache). Par ailleurs, la nature open-source potentielle de cette infrastructure constitue un atout majeur, offrant un cadre propice au partage de bonnes pratiques DevOps au sein de la communauté du calcul scientifique. Ce projet permet également d'illustrer un cas concret d'intégration de GitLab CI/CD dans un environnement de développement scientifique hautement contraint, qui pourrait servir de référence pour d'autres équipes du CEA.

Ce stage a représenté une opportunité unique pour renforcer mes compétences, tant sur le plan théorique que pratique, dans des domaines variés tels que l'infrastructure DevOps, la conteneurisation, l'orchestration de pipelines CI/CD et l'administration système en environnement sécurisé. La possibilité de contribuer activement à un projet mêlant développement logiciel, optimisation d'infrastructure, automatisation et calcul haute performance correspondait parfaitement à mes attentes. Ce travail m'a permis de me positionner en tant qu'ingénieur pleinement impliqué dans la conception et le déploiement d'une infrastructure critique, et non comme un simple exécutant. J'ai particulièrement apprécié l'équilibre entre autonomie et encadrement, qui m'a permis d'explorer différentes solutions techniques (Docker vs Podman, Docker Compose vs Usernetes) et de prendre des décisions d'architecture en lien avec les objectifs du projet et les contraintes de sécurité du CEA. En outre, j'ai eu la chance d'être très bien accueilli par les équipes du LCAN et de bénéficier d'une intégration fluide dans les parcours proposés par le CEA aux nouveaux arrivants. J'ai également pu échanger avec plusieurs équipes techniques du site de Saclay, notamment l'équipe RSI de l'ISAS et l'équipe DeepLab, ce qui m'a ouvert sur d'autres problématiques d'infrastructure et de sécurité en environnement de recherche.

Je remercie particulièrement mes encadrants M. Rémi Bougeois et M. Adrien Bruneton pour leur disponibilité, leur soutien technique tout au long du stage et pour leurs orientations sur les choix d'architecture. Je tiens également à remercier l'ensemble de l'équipe du LCAN pour leur accueil et leur patience face à mes nombreuses questions sur l'écosystème TRUST, ainsi que l'équipe RSI de l'ISAS pour leur support technique sur les aspects réseau et sécurité. Enfin, je remercie l'équipe DeepLab pour la mise à disposition de l'infrastructure GitLab et leur réactivité sur les questions d'administration.