TP : Jenkins et Squah-TM

1. Préparation de l'environnement Docker

Nous allons utiliser **Docker Compose** pour déployer tous les services nécessaires : un serveur Jenkins préconfiguré (dans Docker) avec son agent Docker, Squash TM (version **gratuite** community), Squash Orchestrator (édition DEVOPS Community) et un service MailHog (facultatif, pour le courrier SMTP de test). Tout sera isolé dans des conteneurs et interconnecté via un réseau Docker Compose commun.

1.1 Dockerfile personnalisé pour Jenkins

Commencez par créer un fichier `Dockerfile` pour l'image Jenkins afin d'y préinstaller le client Docker. Ceci permettra à Jenkins (exécuté en container) de lancer des étapes dans des conteneurs Docker (via l'image Docker-in-Docker fournie). Utilisez le `Dockerfile` suivant :

```
FROM jenkins/jenkins:lts-jdk21
USER root
# Installer lsb-release (utilitaire requis) et ajouter les dépôts Docker
RUN apt-get update && apt-get install -y lsb-release
RUN curl -fsSLo /usr/share/keyrings/docker-archive-keyring.asc https://download.docker.com/linux
RUN echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-ke
  https://download.docker.com/linux/debian $(lsb_release -cs) stable" > /etc/apt/sources.list.d/
RUN apt-get update && apt-get install -y docker-ce-cli
USER jenkins
```

Ce Dockerfile part de l'image Jenkins LTS (Java 21) et y ajoute le client Docker CLI. Cela permettra au pipeline d'utiliser la commande `docker` à l'intérieur du conteneur Jenkins, en se connectant au démon Docker d'un conteneur **dind** (Docker-in-Docker) séparé.

1.2 Fichier docker-compose.yml

Créez ensuite un fichier `docker-compose.yml` pour définir l'ensemble des services. Vous pouvez vous baser sur le contenu ci-dessous, qui inclut les services Jenkins, Docker (dind), Squash TM, Squash Orchestrator et MailHog :

```yaml
version: '3'
services:
  jenkins-docker:
    image: docker:dind
    container_name: jenkins-docker
    privileged: true
    environment:
      - DOCKER_TLS_CERTDIR=/certs
    volumes:
      - jenkins-docker-certs:/certs/client
      - jenkins-data:/var/jenkins_home
    ports:
      - "2376:2376"
    networks:
      jenkins:
        aliases:
          - docker
    command: --storage-driver overlay2

  my-jenkins:
    build: .
    container_name: my-jenkins
    restart: on-failure
    environment:
      - DOCKER_HOST=tcp://docker:2376
      - DOCKER_CERT_PATH=/certs/client
      - DOCKER_TLS_VERIFY=1
    volumes:
      - jenkins-data:/var/jenkins_home
      - jenkins-docker-certs:/certs/client:ro
    ports:
      - "8080:8080"
      - "50000:50000"
    networks:
      - jenkins

  mailhog:
    image: mailhog/mailhog
    container_name: mailhog
    ports:
      - "8025:8025"   # Interface web (MailHog)
      - "1025:1025"   # Port SMTP
    networks:
```

```
      - jenkins

  squash-tm:
    image: squashtest/squash          # Image Squash TM Community (dernière version)
    container_name: squash-tm
    ports:
      - "8090:8080"                   # Squash TM accessible sur http://localhost:8090/squash
    networks:
      - jenkins
    volumes:
      - ./squash_plugins:/opt/squash-tm/plugins

  orchestrator:
    image: squashtest/squash-orchestrator:latest   # Image "all-in-one" Squash Orchestrator
    container_name: squash-orchestrator
    ports:
      - "7774:7774"   # Receptionist (création de workflows)
      - "7775:7775"   # Observer (suivi des workflows)
      - "7776:7776"   # Killswitch (annulation)
    networks:
      - jenkins

networks:
  jenkins:
    driver: bridge

volumes:
  jenkins-docker-certs:
  jenkins-data:
```

Explications :

- Le service `jenkins-docker` est le daemon Docker en mode *dind*. Il tourne en mode privilégié et expose le port `2376` (Docker TLS). Les volumes `jenkins-docker-certs` et `jenkins-data` sont utilisés pour partager les certificats TLS et le répertoire Jenkins Home entre les conteneurs (permettant à Jenkins d'accéder au démon Docker et de conserver son état).
- Le service `my-jenkins` est construit à partir du Dockerfile ci-dessus. Il monte les volumes nécessaires, et grâce aux variables d'environnement `DOCKER_HOST`, `DOCKER_CERT_PATH` et `DOCKER_TLS_VERIFY`, il est configuré pour utiliser le daemon Docker distant (le conteneur

`jenkins-docker`). Les ports 8080 et 50000 sont exposés pour accéder à l'UI Jenkins et au port des agents Jenkins.

- Le service `mailhog` (optionnel) fournit une interface pour intercepter les emails (accessible sur http://localhost:8025). Ici, il n'est pas indispensable pour Squash mais peut être utile pour recevoir des notifications émail de Jenkins ou Squash.
- Le service `squash-tm` utilise l'image Docker officielle de Squash TM (community). Nous exposons le port 8080 interne sur le port hôte 8090 pour éviter le conflit avec Jenkins (qui utilise 8080). **Note :** par défaut, cette image utilise une base de données embarquée H2 pour la démo, suffisante pour nos tests. La première exécution peut prendre quelques minutes le temps d'initialiser la base de données, il est conseillé de suivre les logs pour vérifier l'avancement. Une fois démarré, l'interface web Squash TM sera disponible sur **http://localhost:8090/squash**. Les identifiants par défaut sont `admin` / `admin`.
- Le service `orchestrator` utilise l'image **Squash Orchestrator** « all-in-one » community (dernière version). Cette image contient l'ensemble des micro-services nécessaires (receptionist, observer, eventbus, etc.). Nous publions les ports principaux : `7774` (service *Receptionist* pour soumettre les workflows), `7775` (service *Observer* pour consulter l'état des workflows) et `7776` (service *Killswitch* pour annuler un workflow en cours). (Le port `38368` de l'event bus est utilisé en interne entre micro-services, et n'a pas besoin d'être exposé hors du réseau Docker).
- **Token d'authentification JWT :** Par défaut, l'orchestrateur génère à chaque démarrage un **jeton JWT temporaire** utilisé pour authentifier les requêtes. Ce token est valide uniquement pour la durée de vie du conteneur (solution acceptable en environnement de test, mais déconseillée en production). L'orchestrateur va afficher ce token dans ses logs au démarrage (recherchez une ligne mentionnant **"JWT token"** dans `docker logs squash-orchestrator`). **Notez bien ce token**, car il sera nécessaire pour configurer la connexion depuis Jenkins et Squash TM. (Il est possible de configurer un token statique signé via une clé, mais cela sort du cadre de ce tutoriel).

2. 📦 Télécharger le plugin "SCM Connector"

Rendez-vous sur la page officielle de téléchargement des plugins Squash TM :
👉 https://tm-fr.doc.squashtest.com/latest/downloads.html

Téléchargez le plugin nommé :

> **Git connector**
> Version : `10.0.0` (compatible avec Squash TM 10.2.0)

Ce plugin inclut le **connecteur SCM nécessaire à la déclaration des dépôts Git**.

3. 📁 Installer le plugin dans Squash TM

À la racine du répertoire `lab` créer un répertoire `squash_plugins`

Copiez les fichiers `.jar` dans `squash_plugins`

4. Lancement des services

- Une fois les fichiers préparés, ouvrez un terminal à la racine de `Lab` et executer `docker build -t my-jenkins .`
- Puis une fois le build terminé : `docker-compose up -d` . Patientez quelques minutes que tout se lance correctement. Vérifiez notamment :
- Les logs de **squash-tm** (`docker logs -f squash-tm`) pour voir quand l'application est prête (message *"Started SquashTmApplication"*). La première initialisation peut durer 2-3 minutes.
- Les logs de **squash-orchestrator** (`docker logs -f squash-orchestrator`) pour repérer le **token JWT** généré (copiez cette valeur).

5. Configuration de Squash TM (serveur de test)

Une fois Squash TM démarré, accédez à l'interface web sur **http://localhost:8090/squash**.
Connectez-vous avec **login** `admin` et **mot de passe** `admin` (identifiants par défaut).

5. Configuration de Squash-TM

🛠 Étape 1 – Ajouter un serveur SCM (Git)

1. Connectez-vous à Squash TM avec un compte admin (par défaut `admin` / `admin`)
2. Allez dans le menu **Administration > Servers > SCM servers**
3. Cliquez sur **"Add server"**
4. Renseignez les champs :

Champ	Valeur
Name	`GitHub JUnit5`
Type	`Git`
URL	`https://github.com/junit-team/junit5-samples.git`
Login	*(laissez vide si public)*
Password	*(laissez vide si public)*

5. **Toujours dans Administration > Servers > SCM servers**
6. Cliquez sur votre serveur `Git` que vous venez de créer
7. En bas, dans la section **Repositories**, cliquez sur **“Add Repository”**
8. Renseignez les champs suivants :

Champ	Valeur
Name	`JUnit5 Demo`
Branch	`main`

9. Cliquez **Save**

✅ Étape 2 – Créer un projet et un cas de test

1. Allez dans l'onglet **"Administration" > "Gestion des projets"**
2. Cliquez sur **Créer un projet** :
   - **Nom** : `Demo Squash`
   - Laissez les autres champs par défaut

Ajouter un cas de test automatisé

1. Dans le projet `Demo Squash` , allez à l’onglet **"Cas de test"**
2. Cliquez sur **Créer un cas de test**
   - **Nom** : `Hello World`
   - Laissez les autres champs par défaut
3. Cliquez sur le bouton **Éditer** le cas de test
4. Allez à l’onglet **"Automatisation"**
   - **Automated test technology** : `JUnit`

- Le champ **“URL of the Source code repository”** est maintenant un menu déroulant :
  - Cliquez dessus, vous verrez apparaître votre dépôt

    `https://github.com/junit-team/junit5-samples.git/JUnit5 Demo`
- Une fois le référentiel ajouté, sélectionnez-le
- **Automated test reference** :

  `junit5-samples/junit5-jupiter-starter-gradle/src/test/java/com/example/project/Calculat`

✅ Étape 3 – Préparer le plan d'exécution

1. Allez dans **"Plans d’exécution"** du projet
2. Cliquez sur **Nouvelle campagne** :
   - Nom : `Campagne Demo`
3. Dans la campagne, **Execution plan** cliquez **Ajouter une itération**
   - Nom : `Iteration 1`
4. Ouvrez l’itération créée, cliquez sur **Associate test cases**
   - Sélectionnez votre cas “Hello World” via drag and drop

💡 **Récupérer l’UUID de l’itération** :

- Cliquez sur l’itération
- L’URL du navigateur contient un identifiant comme :

  `http://localhost:8090/squash/campaign-workspace/campaign/<UUID>/test-plan?anchor=plan-exec`

- Copiez cet UUID pour l’étape **6.5**

✅ Étape 4 – Ajouter l’Orchestrateur dans Squash TM

1. Allez dans **Administration > Serveurs > Serveurs d'automatisation**
2. Cliquez sur **Ajouter un serveur**
   - **Nom** : `MyOrchestrator`
   - **Type** : `Squash Orchestrator`
   - **URL** : `http://squash-orchestrator:7774`
3. Cliquez sur **Add**
4. Éditer les paramètres

- **Observer URL** : `http://squash-orchestrator:7775`
- **Event bus URL** : `http://squash-orchestrator:38368`
- **Killswitch URL** : `http://squash-orchestrator:7776`
- **Token JWT** : récupéré dans les logs via :

```
docker logs squash-orchestrator
```

(et cherchez la ligne contenant : `INFO in startup: eyJ...`)

5. Cliquez sur **Enregistrer**

✅ Étape 5 – Lier l'orchestrateur au projet

1. Allez dans **Projets > Demo Squash > Configuration**
2. Onglet **Automatisation**
3. Sélectionnez le serveur `MyOrchestrator` comme **serveur d'automatisation par défaut**
4. Enregistrez

6. Configuration de Jenkins et du plugin Squash DEVOPS

Avec Jenkins en place (accessible sur http://localhost:8080), connectez-vous à l'interface Jenkins.

> Si c'est la première exécution, récupérez le mot de passe administrateur initial : exécutez `docker exec my-jenkins cat /var/jenkins_home/secrets/initialAdminPassword` pour l'afficher, puis suivez l'assistant de démarrage Jenkins. Vous pouvez sélectionner les plugins suggérés de base. Le plugin Squash DevOps n'étant pas dans le catalogue public, inutile de le chercher dans la liste pour l'instant.

6.1 Installation du plugin Squash DEVOPS dans Jenkins

Téléchargez le fichier HPI du plugin Squash DEVOPS (version community) depuis le site Squashtest

Dans Jenkins, allez dans **Manage Jenkins > Manage Plugins > Advanced > Upload Plugin**. Uploadez le fichier `.hpi` du plugin Squash DEVOPS. Une fois installé, redémarrez Jenkins si nécessaire. (Le plugin ajoute une étape Pipeline et une section de configuration d'orchestrateur.)

➡️ *Vérification :* Après installation, vous devriez voir dans **Manage Jenkins > Configure System** une section **Squash Orchestrator servers**.

6.2 Installation du plugin Docker Pipeline dans Jenkins

Dans Jenkins, allez dans **Manage Jenkins > Manage Plugins**, Onglet **Available** (ou **Installed** pour vérifier).

Recherchez Docker Pipeline (nom officiel du plugin).

Installez-le, puis redémarrez Jenkins si nécessaire : `docker compose restart my-jenkins`

> 👉 Cela active la syntaxe docker.image(...).inside {} dans vos pipelines.

6.3 Configuration de la connexion à l'orchestrateur dans Jenkins

Au préalable, allez dans **Manage Jenkins > Credentials** et ajoutez une nouvelle entrée de type **Secret text** nommée par ex. `OrchToken` avec pour contenu le token JWT copié plus tôt

Dans **Manage Jenkins > Configure System**, localisez **Squash Orchestrator servers**. Cliquez **Add** pour ajouter la configuration de notre orchestrateur :

- **Server name:** donnez un nom identifiant ce serveur, par ex. `MyOrchestrator` . (Nous utiliserons ce nom dans le pipeline.)
- **Receptionist endpoint URL:** `http://orchestrator:7774` (depuis Jenkins, le conteneur orchestrator est accessible par le hostname `orchestrator` sur le réseau Docker).
- **Workflow Status endpoint URL:** `http://orchestrator:7775` (URL du service observer).
- **Credential:** sélectionnez ou ajoutez un **Secret Text** contenant le token JWT de l'orchestrateur dans la liste déroulante.
- Laissez les autres paramètres par défaut (intervalle de polling, timeout de création). Enregistrez la configuration.

Jenkins est maintenant capable de communiquer avec l'orchestrateur Squash.

6.4 Création du pipeline Jenkins

Créez un nouveau **Job** de type **Pipeline** dans Jenkins (nommez-le par ex. *Demo-Squash-Pipeline*).

Dans la section `Pipeline script` , nous allons écrire un Jenkinsfile qui réalise les étapes voulues :

- checkout du code,

- exécution des tests,
- publication des résultats JUnit,
- puis appel à l’orchestrateur via l’étape pipeline fournie par le plugin Squash DEVOPS.

Voici un exemple de pipeline (Jenkinsfile) commenté que vous pouvez adapter :

```
pipeline {
  agent any

  environment {
    MAVEN_IMAGE = 'maven:3.8.7-eclipse-temurin-17'
  }

  stages {
    stage('Checkout') {
      steps {
        git 'https://github.com/francois-sa-semifir/java-maven-junit-helloworld'
      }
    }

    stage('Build & Test') {
      steps {
        script {
          docker.image("${env.MAVEN_IMAGE}").inside {
            sh 'mvn clean test -Dmaven.test.failure.ignore=true'
          }
        }
        junit '**/target/surefire-reports/*.xml'
      }
    }

    stage('Generate .otf.yml') {
      steps {
        script {
          def otfYaml = """\
metadata:
  name: Résultats Jenkins vers Squash TM
resources:
  files:
    - HelloAppTest.xml
    - HelloTest.xml
jobs:
  sendResults:
    runs-on: inception
    generator: tm.squashtest.org/tm.generator@v1
    with:
      squashTMUrl: http://squash-tm:8080/squash
      squashTMAutomatedServerLogin: taserver
      squashTMAutomatedServerPassword: taserver
```

```
      squashTMAutomatedServerName: "MyOrchestrator"
      testPlanUuid: "106"
      testPlanType: Iteration
"""
          writeFile file: '.otf.yml', text: otfYaml
          sh 'cat .otf.yml'
        }
      }
    }

    stage('Send to Squash Orchestrator') {
      steps {
        sh '''
          cp target/surefire-reports/TEST-com.example.javamavenjunithelloworld.HelloAppTest.xml
          cp target/surefire-reports/TEST-com.example.javamavenjunithelloworld.HelloTest.xml Hel
        '''

        runOTFWorkflow(
          serverName: 'MyOrchestrator',
          workflowPathName: '.otf.yml',
          workflowTimeout: '200S',
          fileResources: [
            [ name: 'HelloAppTest.xml', path: 'HelloAppTest.xml' ],
            [ name: 'HelloTest.xml',   path: 'HelloTest.xml' ]
          ]
        )
      }
    }
  }

  post {
    always {
      echo 'Pipeline terminé.'
    }
    failure {
      echo 'Le pipeline a échoué.'
    }
  }
}
```

Explications :

- On utilise un **agent Docker** pour la phase de build/test : le pipeline récupère une image Maven avec JDK approprié, puis exécute `mvn test`. Grâce à la configuration Docker-in-Docker, Jenkins peut lancer ce conteneur et Maven construira le projet et exécutera les tests unitaires JUnit.
- Les résultats des tests (fichiers XML JUnit dans `target/surefire-reports`) sont ensuite publiés avec la directive `junit`. Jenkins affichera le bilan des tests (succès/échecs) dans l'interface du job.
- La dernière étape prépare l'envoi des résultats vers Squash TM en s'appuyant sur Squash Orchestrator :
  - On construit dynamiquement un **workflow "EPAC"** en JSON. Ici on utilise la fonctionnalité *Inception* de Squash DEVOPS qui permet de **publier les résultats de tests déjà exécutés** plutôt que de faire exécuter les tests par l'orchestrateur. En effet, nos tests ont été lancés par Jenkins/Maven ; l'orchestrateur se chargera simplement de prendre les rapports JUnit et de les envoyer à Squash TM.
  - Dans l'EPAC JSON :
    - La section `resources.files` déclare une ressource nommée `"junit_report"` qui représentera le fichier de rapport JUnit à envoyer.
    - Le job `"prepare"` utilise l'action `actions/prepare-inception@v1` pour associer un fichier réel à la ressource. Nous indiquons le nom du fichier JUnit généré (par ex. `TEST-HelloWorldTest.xml` – adaptez ce nom au fichier réel dans `target/surefire-reports`), et on lie ce fichier à la ressource `${{ resources.files.junit_report }}`. Si plusieurs fichiers de rapports existaient, on pourrait en ajouter plusieurs dans `resources.files` et les mapper via plusieurs entrées `with` dans cette étape prepare.
    - Le job `"publish"` (générateur) est configuré pour *runs-on: inception* et *needs: [prepare]* (il attend que le job prepare ait associé les fichiers). Il utilise l'action `tm.squashtest.org/tm.generator@v1` – c'est le module de récupération de plan Squash TM (fourni par le plugin *Test plan retriever* installé sur TM). On passe dans `with` les paramètres de connexion à Squash TM :
      - `squashTMUrl` : l'URL interne de Squash TM (depuis l'orchestrateur). Ici, comme orchestrateur et TM sont sur le même réseau Docker, on peut utiliser `http://squash-tm:8080/squash`.

      > `squashTMAutomatedServerLogin` / `Password` : les identifiants du compte Squash TM à utiliser pour publier les résultats. Vous pouvez utiliser `admin` / `admin` pour simplifier (bien que ce ne soit pas recommandé en production). **Astuce :** il est conseillé en réalité de créer un utilisateur dédié dans Squash TM (ex: *automation* avec rôle d'automatisation) et d'utiliser ce login. Indiquez ici ce compte et son mot de passe. L'orchestrateur l'utilisera pour appeler l'API REST de Squash TM.

        - `testPlanUuid` : l'UUID de l'**itération** créée précédemment dans Squash TM, qui correspond au plan d'exécution à renseigner.
        - `testPlanType` : le type de plan – dans notre cas `"Iteration"` (car nous avons utilisé une itération de campagne). *(Si vous aviez un autre type de plan (suite de tests par exemple), indiquez la valeur appropriée, mais "Iteration" est le plus courant pour une exécution planifiée.)*
    - Lorsque ce workflow sera exécuté sur l'orchestrateur, voici ce qu'il fera : (a) grâce au **Test Plan Retriever**, il va contacter Squash TM et récupérer la liste des cas de test de l'itération spécifiée ; (b) il **n'exécutera pas de tests**, car on est en mode inception (pas d'environnement de test requis) ; à la place, (c) il passera directement à l'**étape de publication des résultats** : le module Result Publisher de Squash TM prendra le(s) fichier(s) JUnit fournis (attachés via `resources.files`) et **mettra à jour le statut des cas de test** dans Squash TM en fonction des résultats.
- Dans le pipeline Jenkins, on écrit ce contenu JSON dans un fichier `workflow.json` dans l'espace de travail, puis on appelle la méthode pipeline `runSquashTFWorkflow(...)` fournie par le plugin Squash DEVOPS. On lui passe le chemin du fichier JSON, un timeout (ici 60 secondes, ajustez selon la durée potentielle de vos tests) et le **serverName** que l'on a configuré plus tôt (nous avions nommé le serveur *MyOrchestrator* dans Jenkins).
- La méthode retourne immédiatement un **ID de workflow** si la soumission au receptionist a réussi. L'orchestrateur s'exécute ensuite en tâche de fond pour traiter le workflow. Le plugin Jenkins va interroger périodiquement l'orchestrateur (via l'Observer) pour connaître le statut jusqu'à complétion. Vous verrez ainsi le pipeline Jenkins rester en cours sur cette étape pendant quelques secondes, puis passer à success/failure selon le résultat. Nous affichons l'ID juste pour information.

Enregistrez ce pipeline dans Jenkins (n'oubliez pas de remplacer `<YOUR_ITERATION_UUID>` par l'UUID réel de votre itération Squash, et ajustez éventuellement les noms de fichiers de rapport et identifiants).

6.5 Choix d'un projet de test GitHub

Si vous n'avez pas de projet GitHub à tester, vous pouvez utiliser l'exemple mentionné ci-dessus : **java-maven-junit-helloworld**. C'est un petit projet Java qui possède quelques tests JUnit 5 basiques (il s'agit d'un "Hello World" avec tests unitaires et d'intégration). Dans notre pipeline, nous avons pointé l'URL Git de ce projet. Vous pouvez forker ce dépôt sur votre compte GitHub ou utiliser directement l'URL publique en lecture seule comme dans l'exemple. Le pipeline va cloner le projet et exécuter `mvn test` dessus (ce qui lancera les tests unitaires JUnit).

> *Remarque :* Le projet example exécute uniquement les tests unitaires avec `mvn test` . Il contient aussi des tests d'intégration (suffixe IT.java) qui ne tournent qu'avec `mvn verify` . Ici on se limite aux tests unitaires pour l'exemple. Assurez-vous que le cas de test créé dans Squash TM correspond bien à l'un des tests unitaires (par exemple, créez un cas de test "HelloWorldTest" et associez le à la classe/méthode de test correspondante).

7. Exécution du pipeline et vérification des résultats

Tout est prêt à présent : Squash TM et l'orchestrateur tournent, Jenkins est configuré avec le plugin et le pipeline.

- **Lancez le job Jenkins** (Cliquez *Build Now* sur le pipeline). Sur la console Jenkins, vous verrez les étapes défiler : le code est cloné, Maven compile et exécute les tests. Si tout va bien, les tests réussissent (ou provoquez volontairement un échec pour tester la chaîne). Jenkins publie ensuite les résultats JUnit (par exemple "2 tests, 0 failures").
- À l'étape **"Send results to Squash TM"**, Jenkins transmet le workflow à l'orchestrateur. Le plugin affiche l'ID du workflow soumis (par exemple `workflow_12345`). En quelques instants, l'orchestrateur va traiter le workflow : il récupère le plan d'exécution auprès de Squash TM et envoie le rapport JUnit. Une fois terminé, Jenkins marquera le stage comme réussi (sauf si l'orchestrateur a rencontré une erreur).

Maintenant, vérifions dans **Squash TM** :

- Allez dans votre projet Squash, ouvrez la campagne/l'itération que vous aviez créée. Vous devriez y voir une **exécution automatique** associée à l'orchestrateur. Le statut du cas de test devrait être mis à jour ("Succès" si le test JUnit est passé, "Échec" sinon), avec éventuellement un lien vers le rapport. Squash TM a enregistré le résultat grâce au plugin Result Publisher. Vous pouvez consulter les détails de l'exécution (log, date/heure, etc.) dans l'onglet des exécutions de l'itération.
- Si quelque chose ne remonte pas : vérifiez les journaux de l'orchestrateur (`docker logs squash-orchestrator`) pour voir s'il y a eu une erreur (par ex. problème d'authentification à Squash TM ou nom de fichier introuvable). Le plugin Jenkins renvoie également les messages d'erreur du workflow s'il échoue. Assurez-vous que les identifiants API dans l'EPAC (login/password) sont corrects et que l'UUID est bon.

Si tout est correctement configuré, vous avez un pipeline CI/CD complet où Jenkins exécute les tests et **rapatrie automatiquement les résultats dans Squash TM** 🎉. Vous pouvez ainsi suivre dans Squash la traçabilité des exigences -> cas de test -> exécutions, même pour les tests automatisés, et bénéficier des rapports de campagne de Squash.

8. Conclusion

Nous avons installé et intégré avec succès **Squash TM (community)** avec Jenkins via **Squash Orchestrator DEVOPS**. L'environnement Docker Compose regroupe tous les composants nécessaires pour un laboratoire d'automatisation de tests : Jenkins (CI), Squash TM (gestion des tests) et Orchestrateur (liaison CI<>TM). Cette solution utilise uniquement des versions **gratuites/community** de Squash. En production, on veillera à renforcer la configuration (tokens JWT statiques, comptes de service dédiés, etc.), mais pour un TP ou une démonstration, l'approche ci-dessus offre une vue complète du processus.

References : Squashtest documentation and community resources were used to ensure correct setup and configurations. Enjoy testing with Squash and Jenkins!