BONIN — Documentation technique

Procédure — Bot Discord de la team Ronin : planning, récaps de game et compos

Développeur	COUR Cyrille
Organisation	Projet personnel — team Ronin
Formation	BTS SIO SLAM — Alternance 2025/2027
Stack	Python 3.13 / discord.py 2.7 / SQLite 3
Date	04/11/2026
Version	1.0

Sommaire

Sommaire 2
1. Contexte 3
1.1 Les features 3
1.2 Périmètre 3
2. Environnement 3
2.1 Prérequis 3
Dépendances 3
2.2 Installation 4
3. Procédure 4
3.1 Étape 1 — Créer le bot sur Discord 4
Détail technique 4
3.2 Étape 2 — Configurer le fichier .env 5
Détail technique 5
3.3 Étape 3 — Lancer le bot et vérifier le statut 5
Détail technique 5
3.4 Étape 4 — Saisir les dispos et détecter les créneaux 6
Détail technique 6
3.5 Étape 5 — Suivre les pings et le récap 7
Détail technique 7
3.6 Étape 6 — Publier le calendrier et gérer la semaine type 8
Détail technique 8
3.7 Étape 7 — Poster le récap des games jouées à 5 8
Détail technique 8
3.8 Étape 8 — Afficher les meilleures compos 9
Détail technique 9
3.9 Étape 9 — Vérifier avec les tests 10
Détail technique 10
4. Structure / Données 10
4.1 Arborescence utile 11
5. Déploiement / CI-CD 11
6. Points de vigilance 12

1. Contexte

BONIN (pour Bot Ronin) est le bot Discord de l'équipe Ronin. Il est découpé en features, une par fonctionnalité, toutes chargées par le même bot. Sa fonction principale évite de demander « qui est dispo quand ? » dans le salon : chaque joueur écrit ses heures dans un calendrier, et le bot trouve le soir où toute l'équipe est disponible.

1.1 Les features

Feature	Dossier	Rôle
Planning (ex-RAP)	features/planning/	Planning hebdomadaire des dispos et pings des créneaux communs
Status	features/status/	Prévient l'équipe quand le bot se connecte et quand il s'arrête
Game	features/game/	Poste le récap de chaque game jouée à 5, calculé par RAID
Compo	features/compo/	/compocoeur, /compotype, /compoplein : meilleurs groupes de 3, 4 ou 5 champions joués ensemble

1.2 Périmètre

- Outil fait pour un petit groupe fixe (le rôle @Ronin) sur un seul serveur Discord : ce n'est pas un agenda généraliste.
- Le bot tourne en continu sur un VPS, sans interface web : tout se passe dans Discord.
- BONIN ne calcule aucune stat de game : il lance RAID et lit son fichier data/site.json.

Note — Cette procédure s'adresse à quiconque reprend le dépôt : elle explique comment créer et installer le bot, comment fonctionnent ses features, et comment le déployer sur le VPS.

2. Environnement

2.1 Prérequis

Outil	Version	Rôle
Python	≥ 3.10 (testé en 3.13.7)	Tout le bot (la syntaxe int | None impose 3.10 minimum)
SQLite	3.50.4	Stockage, via le module sqlite3 de la bibliothèque standard
pip	25.2	Installation des paquets dans .venv
Serveur Discord	—	Permission « Gérer le serveur » pour installer le bot
RAID	Dossier ../RAID par défaut	Avec sa clé Riot et un premier python main.py lancé

Dépendances

Paquet	Contrainte	Installée	Rôle
discord.py	>=2.4,<3	2.7.1	Connexion, boutons persistants, formulaires, commandes slash, tâches planifiées
python-dotenv	>=1.0	1.2.4	Lecture du fichier .env

Paquet	Contrainte	Installée	Rôle
tzdata	—	2026.6	Fuseaux horaires pour zoneinfo (indispensable sous Windows)
aiohttp	>=3.7.4,<4	3.14.5	API Riot (feature game) et icônes Data Dragon (feature compo)
Pillow	>=10,<13	12.3.0	Bandeau des icônes de champions sous chaque compo

Note — Aucune dépendance de développement : les tests utilisent unittest et unittest.mock de la bibliothèque standard. Le projet n'a pas de fichier de verrouillage : une nouvelle installation prend la dernière version compatible.

2.2 Installation

```
$ git clone <url-du-depot> BONIN
$ cd BONIN
$ python -m venv .venv
$ .venv/Scripts/activate       # Linux : source .venv/bin/activate
$ pip install -r requirements.txt
$ cp .env.example .env
```

3. Procédure

Le bot est un seul processus asynchrone, connecté en permanence à Discord. Au démarrage, bot.py crée un commands.Bot et charge chaque feature listée dans FEATURES : chaque feature y ajoute son cog (ses commandes, ses boutons, ses tâches planifiées).

```
Joueur clique sur « Mar. 10/11 »
  └─ DayButton.callback            views.py (rôle et date)
      └─ DayModal                  formulaire visible par le joueur
          └─ DayModal.on_submit
              ├─ parse_ranges      slots.py (lecture, arrondi)
              ├─ set_user_ranges   db.py
              ├─ réponse à Discord  moins de 3 s
              └─ refresh_day       cog.py (sous verrou)
                  ├─ find_slots    slots.py (créneaux)
                  ├─ sync_pings    nouveau / modifié / annulé
                  ├─ refresh_calendar
                  └─ refresh_recap
```

3.1 Étape 1 — Créer le bot sur Discord

Détail technique

Sur le portail développeur (discord.com/developers/applications) :

- Cliquer sur New Application.
- Dans l'onglet Bot, cliquer sur Reset Token, copier le token et téléverser BONIN.png dans Icon.
- Dans OAuth2 → URL Generator, cocher les scopes bot et applications.commands.
- Cocher les permissions View Channels, Send Messages, Embed Links, Read Message History et Mention @everyone, @here and All Roles, puis ouvrir l'URL générée pour inviter le bot.
- Dans Discord, activer le mode développeur (Paramètres → Avancés), puis copier par clic droit les identifiants du serveur, des salons et du rôle @Ronin.

Attention — Sans la permission de mentionner les rôles, ou si @Ronin n'est pas « mentionnable », le rôle s'affichera dans le ping sans notifier personne.

3.2 Étape 2 — Configurer le fichier .env

Détail technique

Les réglages obligatoires du bot se placent dans .env :

```
DISCORD_TOKEN=le-token-copié
GUILD_ID=identifiant-du-serveur
PLANNING_CHANNEL_ID=identifiant-du-salon
PLANNING_RECAP_CHANNEL_ID=identifiant-du-salon-du-récap
PLANNING_ROLE_ID=identifiant-du-rôle-ronin
STATUS_CHANNEL_ID=identifiant-du-salon-des-annonces
GAME_CHANNEL_ID=identifiant-du-salon-des-récaps-de-game
```

Les autres réglages ont une valeur par défaut :

Variable	Défaut	Rôle
TIMEZONE	Europe/Paris	Fuseau horaire
DB_PATH	bonin.db	Fichier SQLite partagé par les features
RAID_DIR	../RAID	Dossier de RAID (celui qui contient main.py)
PLANNING_RECAP_CHANNEL_ID	(vide)	Salon du récap ; vide = pas de récap
PLANNING_MIN_PLAYERS	5	Joueurs nécessaires pour valider un créneau
PLANNING_MIN_DURATION	2	Durée minimale du créneau, en heures
PLANNING_FIRST_HOUR / LAST_HOUR	0 / 24	Heures acceptées et affichées (24 = minuit)
PLANNING_STEP	2	Largeur d'une colonne de la grille, en heures
PLANNING_POST_WEEKDAY / POST_HOUR	6 / 16	Publication automatique (6 = dimanche)
PLANNING_REMINDER_HOUR	18	Heure du rappel au rôle (0 à 23)
STATUS_ONLINE_MESSAGE	Bonin flash IN	Message posté à la connexion
STATUS_OFFLINE_MESSAGE	Bonin flash OUT	Message posté à l'arrêt
GAME_RAID_PYTHON	python	Python qui a les dépendances de RAID
GAME_POLL_MINUTES	3	Intervalle entre deux vérifications des games
COMPO_COEUR_MIN_GAMES	5	Games minimum d'un cœur de compo
COMPO_TYPE_MIN_GAMES	3	Games minimum d'une compo type
COMPO_PLEIN_MIN_GAMES	2	Games minimum d'une compo pleine

Attention — Le token reste uniquement dans .env, ignoré par Git. En cas de fuite : Reset Token sur le portail développeur.

3.3 Étape 3 — Lancer le bot et vérifier le statut

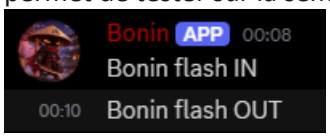
Détail technique

```
$ python bot.py
```

Le bot se connecte, enregistre ses commandes sur le serveur, crée bonin.db au premier lancement et poste « Bonin flash IN » dans STATUS_CHANNEL_ID. Ctrl + C l'arrête après avoir posté « Bonin flash OUT ».

- Une seule annonce par lancement : une simple reconnexion à Discord ne renvoie pas « flash IN ».
- Arrêt propre uniquement : bot.py transforme le SIGTERM de systemd en close(). Un plantage, un kill -9 ou une coupure du VPS ne laissent pas le temps de prévenir.
- Une erreur Discord est journalisée sans bloquer le bot, et le message d'arrêt est abandonné après 5 s.

Sur Discord, /calendrier poste un calendrier sans attendre dimanche ; l'option Semaine en cours permet de tester sur la semaine actuelle.



3.4 Étape 4 — Saisir les dispos et détecter les créneaux

Détail technique

Chaque joueur clique sur un jour du calendrier et écrit ses heures dans un formulaire visible par lui seul. parse_ranges lit la saisie et arrondit à la demi-heure sans jamais compter un joueur dispo quand il ne l'est pas : le début vers le haut, la fin vers le bas.

Saisie	Enregistré
20h-23h59, 20h-0h, 20h-24h	20h–23h59
8h00-13h15	08h–13h
8h10-13h45	08h30–13h30
14h-18h, 20h30-23h	14h–18h, 20h30–23h
14h-16h et 15h-18h	14h–18h (plages fusionnées)
(vide)	Pas dispo ce jour
14h-12h, 20h, 25h-26h, 8h10-8h50	Erreur expliquée, rien n'est enregistré

find_slots valide un créneau quand le même groupe d'au moins PLANNING_MIN_PLAYERS joueurs (5) est dispo pendant au moins PLANNING_MIN_DURATION heures d'affilée (2 h). La journée est découpée à chaque début ou fin de dispo, puis le bot cherche les morceaux consécutifs où le groupe reste le même.

Situation	Résultat
5 joueurs de 20h à 22h	Créneau 20h–22h
Un joueur de 8h à 13h, les 4 autres de 9h à 13h	Créneau 9h–13h
4 joueurs de 20h30 à 23h59, le 5e de 19h à 22h30	Créneau 20h30–22h30
4 joueurs de 20h à 23h, le 5e de 21h30 à 23h59	Refusé : ensemble seulement 1h30
5 joueurs de 14h à 16h, puis de 18h à 20h	Deux créneaux, deux pings

Note — slots.py ne dépend ni de Discord ni de la base : les heures y sont en minutes depuis minuit (20h30 = 1230, minuit = 1440, affiché 23h59), début inclus et fin exclue.



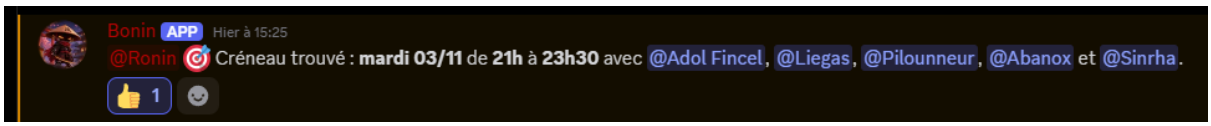
3.5 Étape 5 — Suivre les pings et le récap

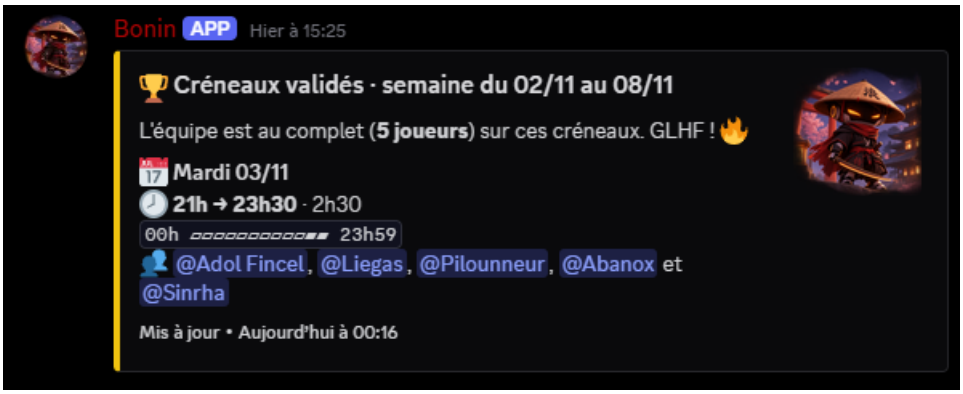
Détail technique

Après chaque saisie, sync_pings compare les créneaux trouvés aux pings déjà envoyés, puis refresh_recap met à jour le récap si PLANNING_RECAP_CHANNEL_ID est renseigné. L'ensemble s'exécute sous un asyncio.Lock : si deux joueurs valident en même temps, un seul ping part.

Événement	Ping	Récap
Nouveau créneau	Envoie un ping au rôle et aux joueurs	Posté, sans ping, s'il s'agit du premier de la semaine
Le créneau bouge ou change de groupe	Modifié, marqué « (mis à jour) », sans renotifier	Même message modifié
Désistement	Ping barré, joueurs prévenus en réponse	Modifié, ou supprimé s'il ne reste aucun créneau
Le créneau réapparaît	Envoie un nouveau ping	Reposté ou modifié

Note — Un jour passé ne peut plus être rempli, et ses pings ne sont plus modifiés. Un récap effacé à la main est reposté à la prochaine saisie s'il reste des créneaux.





3.6 Étape 6 — Publier le calendrier et gérer la semaine type

Détail technique

La tâche weekly_post se déclenche tous les jours à 16h et 18h (Europe/Paris). Le dimanche, elle publie le calendrier de la semaine suivante à 16h, puis mentionne @Ronin à 18h en réponse au calendrier. Si le bot était arrêté, les deux sont rattrapés au redémarrage tant qu'on est dimanche, sans jamais être envoyés deux fois.

Avec /semainetype, un joueur remplit une seule fois ses heures habituelles, qui pré-remplissent chaque nouveau calendrier.

Événement	Effet
Première publication d'un calendrier	Chaque joueur qui a une semaine type est rempli avec, du jour même au dimanche ; les pings partent si l'équipe est complète
Calendrier reposté	Rien n'est recopié : les modifications de la semaine sont gardées
Jour modifié dans le calendrier	Seule cette semaine change ; ce jour ne sera plus écrasé
Semaine type modifiée	Reportée sur les calendriers déjà publiés des semaines à venir, sauf jours modifiés à la main
« Supprimer ma semaine type »	Les prochains calendriers ne sont plus pré-remplis

3.7 Étape 7 — Poster le récap des games jouées à 5

Détail technique

La feature game surveille les games de l'équipe et s'appuie sur RAID pour toutes les stats :

- Toutes les GAME_POLL_MINUTES minutes (3), le bot demande à l'API Riot la dernière game des joueurs suivis par RAID.
- Si l'un d'eux a une nouvelle game Flex, Clash ou Team 5v5, il lance python main.py dans le dossier de RAID.
- Il relit data/site.json : si RAID range la game parmi les games jouées à 5, il poste le récap dans GAME_CHANNEL_ID.

Situation	Action du bot
Premier lancement	Note la dernière game de chaque joueur, ne poste rien
Pas de nouvelle game	Rien (5 requêtes Riot, pas de RAID)
Nouvelle game Solo/Duo, ARAM...	Note la game, ne lance pas RAID

Situation	Action du bot
Nouvelle game Flex, Clash ou Team 5v5	Lance RAID une seule fois, même si les 5 joueurs l'ont jouée
RAID ne la range pas parmi les games à 5	Ne poste rien
Game ou timeline pas encore publiée	Attend le tour suivant
RAID échoue ou dépasse 15 min	Avertissement dans les logs, game retentée au tour suivant
Récap déjà posté	Jamais reposté

La clé Riot est lue dans le .env de RAID à chaque tour : une clé renouvelée est prise en compte sans relancer le bot. /game poste dans le salon courant le récap de la dernière game à 5 connue, sans relancer RAID.



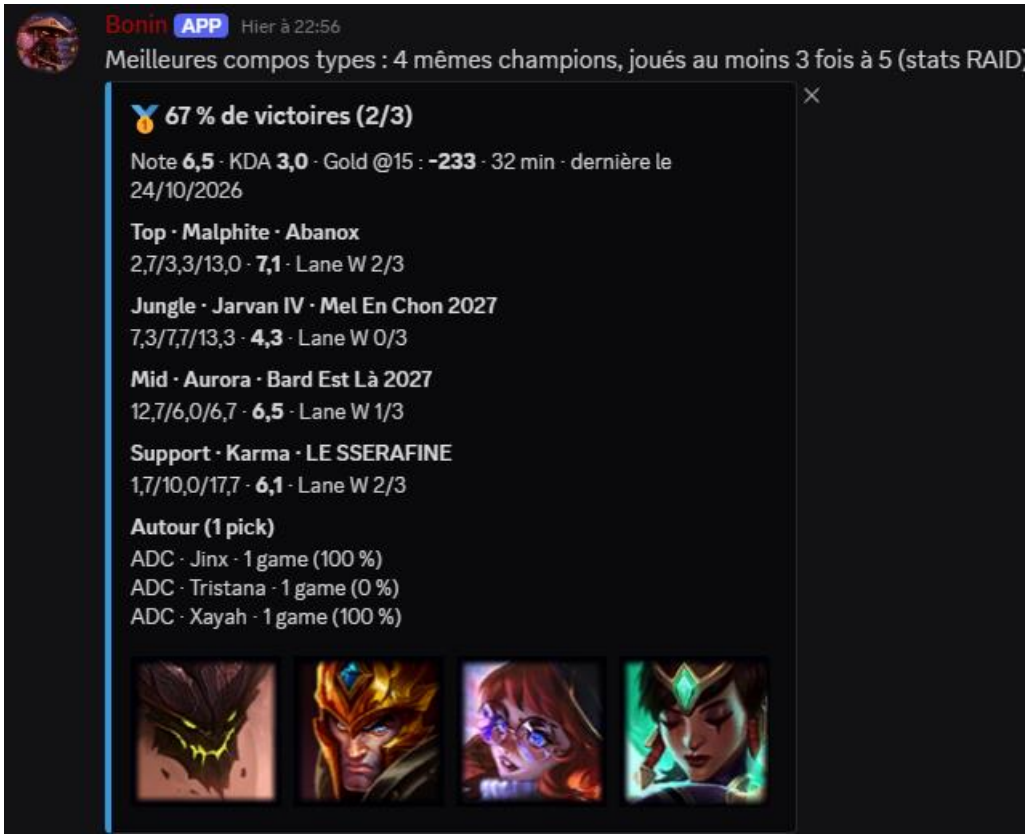
3.8 Étape 8 — Afficher les meilleures compos

Détail technique

Trois commandes répondent dans le salon courant avec les 3 meilleurs groupes de champions joués ensemble à 5, en lisant data/site.json sans relancer RAID.

Commande	Groupe	Minimum (réglage)
/compocoeur	Cœur de compo : 3 mêmes champions, 2 picks construits autour	5 (COMPO_COEUR_MIN_GAMES)
/compotype	Compo type : 4 mêmes champions, un pick qui change	3 (COMPO_TYPE_MIN_GAMES)
/compoplein	Compo pleine : les 5 mêmes champions	2 (COMPO_PLEIN_MIN_GAMES)

- Même groupe : les mêmes champions, quel que soit le joueur ou le poste, à condition que ce soient les seuls communs à toutes ses games.
- Classement : meilleur taux de victoire, puis groupe le plus joué, puis groupe le mieux noté.
- Bandeau : les icônes de Data Dragon sont collées avec Pillow en une image PNG jointe à chaque embed. Sans aucune icône, le groupe est envoyé sans bandeau.
- Aucun groupe assez joué : le bot le dit en message éphémère, avec le nombre de games du groupe le plus joué.



3.9 Étape 9 — Vérifier avec les tests

Détail technique

26 suites unittest, 434 tests en un peu plus de deux secondes. Discord est entièrement simulé avec unittest.mock, la base tourne en mémoire et les réglages sont figés : ni token ni .env ne sont nécessaires. La feature game n'appelle ni Riot ni le vrai RAID (FakeRiot, FakeRaid).

```
$ python -m unittest                                  # toute la suite
$ python -m unittest tests.planning.test_pings -v  # une seule suite
```

Astuce — Pour ranger une nouvelle feature : suivre la section « Rangement des features » de CLAUDE.md. Chaque feature a son dossier, son cog et ses réglages ; ses tables sont préfixées par son nom.

4. Structure / Données

Une seule base SQLite (DB_PATH), ouverte par le bot et partagée par les features : chaque feature y crée ses propres tables, préfixées par son nom. Pas de comptes : les joueurs sont identifiés par leur identifiant Discord. Les features status et compo n'ont aucune table.

Élément	Description	Remarque
planning_availabilities	Une ligne par plage de dispo d'un joueur	Clé (user_id, day, start_min) ; plages fusionnées
planning_edits	Jours modifiés à la main par un joueur	Clé (user_id, day) ; la semaine type n'y touche plus
planning_templates	Semaine type : une ligne par plage et jour	Clé (user_id, weekday, start_min) ; 0 = lundi

Élément	Description	Remarque
planning_weeks	Un message de calendrier par semaine	Clé week_start, date ISO du lundi
planning_reminders	Un message de rappel par semaine	Évite de rappeler deux fois
planning_recaps	Un message de récap par semaine validée	Ligne supprimée avec le message
planning_pings	Un message de ping par créneau trouvé	active = 1 tant que le créneau tient
game_last_matches	Dernière game vue de chaque joueur	Clé riot_id ; PUUID non stocké
game_recaps	Un message par game à 5 postée	Clé match_id ; évite les doublons

Note — Le PUUID n'est pas stocké parce qu'il dépend de la clé API, qui change avec chaque clé de dev. Au premier démarrage, PlanningDB convertit l'ancien format en tranches horaires en une transaction (migrate_hours_to_minutes).

4.1 Arborescence utile

Chemin	Rôle
bot.py	Crée BoninBot, charge les features, synchronise les commandes
config.py	Réglages communs (token, serveur, fuseau, base, dossier de RAID) et leur contrôle
raid.py	Lecture de data/site.json, commune aux features game et compo
features/planning/	Calendrier, formulaires, créneaux (slots.py), tables planning_*
features/status/	Messages de connexion et d'arrêt
features/game/	Surveillance, lancement de RAID, client Riot, récap, tables game_*
features/compo/	Regroupement et classement (compos.py), bandeau d'icônes (icons.py)
tests/	Tests unittest, même découpage que le code
bonin.service	Service systemd pour le VPS

5. Déploiement / CI-CD

BONIN se déploie sur un VPS, en service systemd sous un utilisateur système dédié (bonin), pas en root.

```
$ sudo useradd -r -m -d /opt/bonin bonin
$ sudo -u bonin git clone <url-du-depot> /opt/bonin
$ cd /opt/bonin
$ sudo -u bonin python3 -m venv .venv
$ sudo -u bonin .venv/bin/pip install -r requirements.txt
$ sudo -u bonin cp .env.example .env
$ sudo -u bonin nano .env
$ sudo cp bonin.service /etc/systemd/system/
$ sudo systemctl daemon-reload
$ sudo systemctl enable --now bonin
```

Pour la feature game, installer aussi RAID sur le VPS (par exemple dans /opt/raid, avec ses dépendances) et renseigner RAID_DIR et GAME_RAID_PYTHON avec des chemins absolus. L'utilisateur bonin doit pouvoir écrire dans le dossier de RAID (cache, Excel, data/site.json).

Le service redémarre le bot en cas de plantage et au démarrage du VPS. systemctl stop et restart envoient SIGTERM : le bot poste « Bonin flash OUT » avant de se couper.

```
$ sudo systemctl restart bonin   # appliquer une mise à jour du code
$ sudo systemctl status bonin    # vérifier que le bot tourne
$ journalctl -u bonin -f         # suivre les logs en direct
```

Note — Pipeline CI/CD : [NON VÉRIFIABLE : aucun workflow décrit dans le README]. La mise à jour décrite passe par le redémarrage du service.

6. Points de vigilance

Attention — Sur le VPS, une clé Riot Personal, qui n'expire pas, est indispensable : une clé de dev expire au bout de 24 h. Une clé expirée est signalée une fois dans les logs, puis la surveillance reprend avec la nouvelle clé.

Attention — La clé Riot n'est jamais dans BONIN : elle reste dans le .env de RAID, et le bot ne l'écrit jamais dans les logs.

Attention — Le bot a besoin de RAID à côté de lui, avec un premier python main.py lancé pour créer data/site.json : la feature game le lance et le lit, la feature compo le lit.

Attention — Les permissions d'invitation listées dans le README ne citent pas Attach Files, alors que /compocoeur, /compotype et /compoplein envoient leur bandeau en pièce jointe. À vérifier si le bandeau n'apparaît pas.

Attention — useradd -m peut créer des fichiers d'environnement dans /opt/bonin, ce qui ferait échouer le git clone dans un dossier non vide. Dans ce cas, copier les fichiers par scp, comme le prévoit le README.

Attention — Le récap d'une game arrive quelques minutes après la fin de la game : le temps que Riot publie la game et sa timeline (1 à 5 min), puis le prochain tour de surveillance et le calcul de RAID.

Attention — Un Excel de RAID resté ouvert fait échouer son lancement : la game est retentée à chaque tour jusqu'à ce que RAID réussisse.

Attention — @everyone et @here sont toujours bloqués. Seuls le ping et le rappel autorisent le rôle et les joueurs (PING_MENTIONS).

Attention — Si PLANNING_ROLE_ID est renseigné, seuls les membres du rôle remplissent le calendrier et utilisent /semainetype. /calendrier reste aussi ouvert à la permission « Gérer le serveur ».